PROJETO FINAL DE GRADUAÇÃO

# APLICAÇÃO PARA MONITORAMENTO DE FROTA DE ÔNIBUS DE TRANSPORTE COLETIVO NO DISTRITO FEDERAL

Hudson de Moraes Filho

Brasília, Dezembro de 2016

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

PROJETO FINAL DE GRADUAÇÃO

# APLICAÇÃO PARA MONITORAMENTO DE FROTA DE ÔNIBUS DE TRANSPORTE COLETIVO NO DISTRITO FEDERAL

Hudson de Moraes Filho

*Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação*

Banca Examinadora

Prof. Ugo Silva Dias, Dr., ENE/UnB _____
*Orientador*

Prof. Georges Daniel Amvame Nze, Dr., ENE/UnB _____
*Examinador interno*

Prof. Diego Martins de Oliveira, Esp., IFB _____
*Examinador externo*

Dedicatória

*Dedico este trabalho aos meus pais e irmã, Elair, Hudson e Marcella, pelo apoio e incentivo de sempre, e à minha companheira de todas as horas, Lilian Coutinho.*

*Hudson de Moraes Filho*

Agradecimentos

*É com muita alegria que finalizo este período tão importante na minha vida tendo a certeza que fiz grandes amigos e vivi experiências inesquecíveis. Agradeço aos amigos para a vida toda Guilherme Mangueira e Victor Oliveira, que, apesar de terem seguido caminhos diferentes, sempre me apoiaram a persistir. Agradeço aos amigos da Empresa Júnior que tanto me ensinaram sobre a vida pessoal e profissional, principalmente Ana Gabriela, Vitor Carazza e Arthur Antonoff. Agradeço aos fiéis parceiros de Unb Daniel Akira, Arthur Pacheco e Yuri Freitas, por todas as histórias e por sempre tornarem os dias muito mais agradáveis e memoráveis.*

*Ao professor Ugo Dias, a gratidão pelos ensinamentos das melhores disciplinas que tive e pela orientação neste projeto final. Ao professor Flávio Elias, o obrigado pelas dicas profissionais de sempre e pelas ajudas valiosas nos momentos de Empresa Júnior.*

*Ao Lucas Matos, o obrigado pela ajuda solícita em partes importantes do projeto.*

*À Lilian Coutinho, por ser a melhor companheira de vida que eu poderia encontrar. Sou infinitamente grato por tudo que você me ensina todos os dias e por todos os momentos juntos, sei que tenho enorme sorte por te ter ao meu lado. Espero genuinamente que continuemos sempre fazendo um ao outro mais feliz.*

*À minha irmã, Marcella, que apesar de ser tão diferente de mim, sempre me entendeu tão bem. Sou muito feliz por ter partilhado todos os momentos que já tivemos, e espero que momentos e histórias ainda melhores nos aguardem. Tenho muita sorte por tê-la como irmã.*

*Ao meu pai, a eterna gratidão por ser o maior exemplo de superação e empenho. Obrigado por ser sempre tão preocupado em dar sempre o melhor à família, e por ser o melhor pai que sempre pôde. Espero poder aprender muito com você ainda.*

*À minha mãe, a maior incentivadora e apoiadora de tudo na minha vida. Agradeço por sempre me ensinar, na prática e pelo seu exemplo, a ser respeitoso, humilde, comprometido e sempre, sempre, otimista quanto ao dia de amanhã. Agradeço, acima de tudo, por ter sempre me mostrado como alguém pode realmente ser melhor à cada dia. Sou seguidor das suas filosofias de vida para sempre.*

*Hudson de Moraes Filho*

RESUMO

O presente trabalho visa facilitar a utilização do sistema de transporte público por ônibus no Distrito Federal por meio da proposta de dois aplicativos para celular, sendo estes desenvolvidos utilizando a plataforma Android. O primeiro aplicativo deve ser utilizado pelo usuário final do sistema de transporte público por ônibus, e tem como objetivo principal a consulta das melhores rotas para se chegar a um destino final utilizando o referido sistema. O segundo aplicativo deve ser usado por funcionários do sistema de transporte público por ônibus, e tem por objetivo enviar a um banco de dados externo periodicamente a localização atual do ônibus em questão, atualizando a informação antiga do banco de dados acerca deste ônibus. Com ambos os aplicativos em funcionamento, o usuário será capaz de acompanhar, no mapa e em tempo real, o deslocamento dos ônibus de seu interesse, possibilitando que se programe de forma muito mais eficiente em seu dia-a-dia e, por consequência, tenha uma experiência muito melhor ao utilizar o sistema de transporte público por ônibus.

Palavras-chave: Aplicativo, Android, Sistema de Transporte Público, Ônibus, Localização, Tempo Real, Distrito Federal.

ABSTRACT

This study aims to ease the utilization of buses from the public transport system in Distrito Federal by proposing two mobile applications, both developed using Android platform. The first app should be used by the final user of the public transport system, and has as a main purpose the fetch of the best routes to travel to a final destination using this mentioned system. The second application should be used by the public transport system workers, and has the purpose of sending to an external server and database the real-time bus location coordinates, updating the old database information about this specific bus. With both apps working, the user will be capable of following, in real-time, the desired buses movement in the map, making it possible for him to schedule his appointments much more efficiently and, by consequence, having a much better experience while using the public transport system by bus.

Keywords: Application, Android, Public Transport System, Bus, Location, Real-time, Distrito Federal.

# SUMÁRIO

# LISTA DE FIGURAS

# Capítulo 1

# Introdução

## 1.1. Motivações do Projeto

O sistema público de transporte brasileiro é conhecidamente precário e carente de investimentos e fiscalização por parte das entidades responsáveis. O Distrito Federal, apesar da importância perante o país, não se encontra em situação diferente. Este, dentre outras necessidades, carece de sistemas tecnológicos de qualidade e precisão que trabalhem em prol da melhoria do sistema como um todo.

Um usuário do sistema público de transporte no DF tem basicamente duas opções: o sistema de transporte por metrô e o sistema de transporte por ônibus. O sistema de metrô, apesar de prestar um serviço de qualidade, tem uma abrangência bastante limitada, não atendendo à maioria das regiões do DF. Isso causa uma grande dependência da população ao sistema por ônibus. Apesar dessa dependência, o DF não possui um sistema por ônibus de qualidade. São vários os problemas que o usuário enfrenta, alguns deles são: greves frequentes de rodoviários, más condições dos ônibus, alterações frequentes nas linhas de ônibus, e o não cumprimento pelos ônibus dos horários de chegada nas paradas, horários estes estipulados para cada linha. Todos esses afetam de forma bastante impactante o usuário final do sistema de transporte público por ônibus.

## 1.2. Objetivos do Projeto

O último problema citado – o não cumprimento pelos ônibus dos horários de chegada nas paradas – é o que esse projeto visa dar uma resposta mais direta. Os outros problemas, no entanto, também devem ser considerados, principalmente no intuito de tornar o usuário do sistema cada vez mais informado dos eventuais problemas.

Uma tecnologia simples, porém de grande potencial de melhora no sistema, é o monitoriamento da frota de ônibus de transporte coletivo. Este monitoramento propicia, dentre algumas vantagens, transparência aos usuários deste sistema acerca do tempo que esperarão o ônibus com a rota desejada. O objetivo deste projeto é propiciar, por meio de

aplicativos celulares, funcionalidade semelhante a essa. Com a informação do tempo estimado da chegada do ônibus desejado, usuários poderão se planejar melhor, e, em caso de possíveis problemas com a frota, e consequentemente possíveis atrasos dos ônibus, poderão buscar formas de transporte alternativas.

## 1.3. Visão Geral

Para que o objetivo descrito seja atingido, foram criados dois aplicativos com funcionalidades e utilizações diferentes. O primeiro deles deve ser utilizado pelo usuário final do sistema de transporte público por ônibus, o qual necessita de informações precisas e atualizadas deste sistema, e o segundo deve ser utilizado pelos funcionários deste sistema, com a finalidades de prover as informações atualizadas acerca deste. Os aplicativos devem necessariamente ser de fácil e rápido manuseio, e, por isso, foram feitos de modo a possuir interfaces leves e simples. O usuário que se encontra próximo a um ponto de ônibus poderá, por meio do aplicativo, consultar os próximos ônibus que passarão no ponto em questão. O resultado da pesquisa contemplará ônibus de diversas rotas e trajetos que incluem o ponto em que o usuário se encontra, podendo este, com essa informação, optar pela melhor opção para o momento. Aliado a isso, há a informação do tempo estimado da chegada destes ônibus. Também nesta lista, são informados alguns outros detalhes de importância, como, por exemplo, a duração total do percurso, as distâncias a serem percorridas à pé, o valor que é taxado para o uso do ônibus específico, dentre outros.

Para o cálculo correto da estimativa de chegada do ônibus é necessária a informação da geoposição atual dos ônibus, de forma sempre atualizada e real. Aliando esta informação a algumas referentes ao histório de fluxo das vias a serem trafegadas pelos ônibus, o cálculo da estimativa de tempo pode ser feito. Como o aplicativo não possui as informações reais acerca das geolocalizações dos ônibus do sistema público de transporte, e para que simulações do funcionamento dos aplicativos possam ser feitas, as coordenadas que representam os ônibus foram aleatoriamente escolhidas.

## 1.4. Sistema Transmissor

Para que seja possível o rastreio dos ônibus em tempo-real, é necessária a instalação de transmissores independentes em cada um dos ônibus que façam parte da

frota, ou que estes ônibus possuam GPS. Estes dispositivos alimentarão periodicamente (por exemplo, de 3 em 3 segundos) um banco de dados com as informações referentes à geolocalização atual do ônibus em questão. Esse banco de dados armazenará as informações de todos os ônibus que compõem o sistema de transporte coletivo. Por meio de consultas ao banco de dados, o aplicativo saberá a posição atual dos ônibus. Unindo esta informação com informações referentes ao estado atual das vias em que o ônibus passará até chegar ao ponto de ônibus (quantidade de tráfego, acidentes, interdições), o aplicativo poderá estimar o tempo de chegada do ônibus.

## 1.5. Acordos Administrativos

Para que o aplicativo tenha sucesso em seus objetivos, o sistema transmissor é fundamental. No entanto, no Distrito Federal, não são todos os ônibus do sistema de transporte coletivo que possuem GPS ou transmissores instalados. Para um monitoramento correto da frota e, por consequência, um funcionamento correto dos aplicativos, a instalação deve ser feita em todos os ônibus pertencentes ao sistema. Além disso, para que os aplicativos possam fazer uso das informações de geolocalização capturadas pelos ônibus que já possuem a tecnologia instalada, deve ser firmado um acordo com as instituições responsáveis, de modo que estas autorizem a utilização destas informações e as disponibilizem de forma sempre atualizada.

## 1.6. Aplicações Semelhantes no Brasil

### 1.6.1. CittaMobi

O CittaMobi é um conjunto de soluções que visa disponibilizar informações sobre o transporte público aos usuários de ônibus. A aplicação fornece previsões em tempo real da chegada dos ônibus, as localizações dos pontos mais próximos, em conjunto com as linhas que passam por eles, e alguns detalhes relacionados a cada ônibus, como por exemplo, se o ônibus é adaptado à portadores de necessidades especiais ou não.

A página inicial do app mostra o mapa, a localização do usuário, e diversos pontos de ônibus próximos a ele. Caso o ponto de ônibus de interesse não esteja nos arredores do mapa, é possível deslocar o mapa até a área desejada por meio do campo de pesquisa, situado no topo. Ao se clicar em algum ponto de ônibus de interesse, o app mostra informações detalhadas acerca do ponto, separando-as em três abas diferentes:

- Visão geral: mostra fotos do local, bem como as linhas de ônibus que passam pelo ponto;
- Previsões: próximos ônibus que passarão no ponto, e as informações relacionadas a eles;
- Favoritas: lista de previsões resumida apenas com as linhas favoritas.

Em seguida, a Figura 1.1 retrata a página inicial do aplicativo, e a Figura 1.2 exibe a aba de 'Visão Geral'.

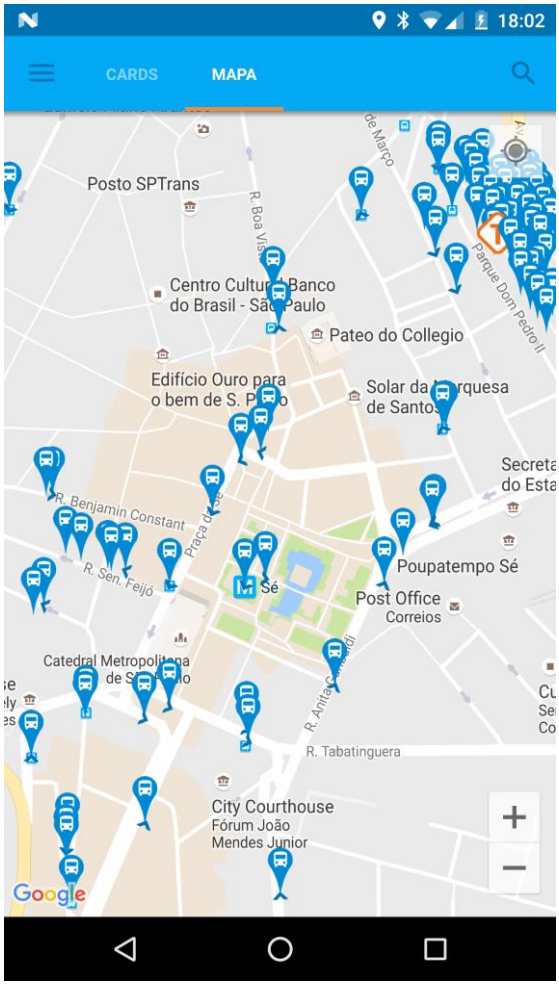


Figura 1.1: Tela Mapa CittaMobi.


Figura 1.2: Tela Visão Geral CittaMobi.

Na aba 'Previsões', ao se optar por um dos próximos ônibus, o app mostra os horários de chegada e alguns detalhes sobre os próximos ônibus desta mesma linha que passarão neste mesmo ponto. Também nessa página, é possível acompanhar no mapa, em tempo real, o deslocamento do ônibus em questão.

Em seguida, a Figura 1.3 exibe a aba ‘Previsões’ e a Figura 1.4 retrata a página com mais detalhes acerca da linha escolhida.

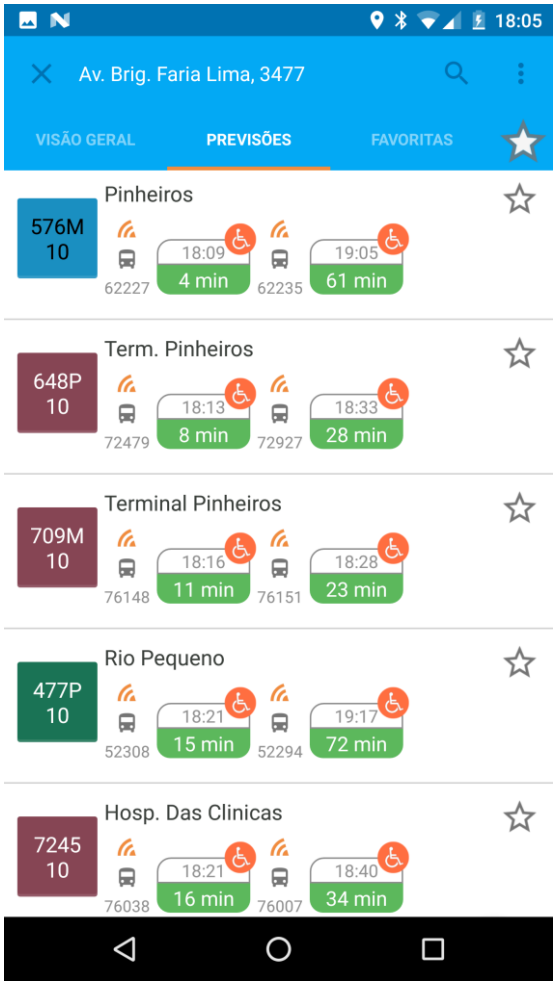


Figura 1.3: Tela Previsões CittaMobi.

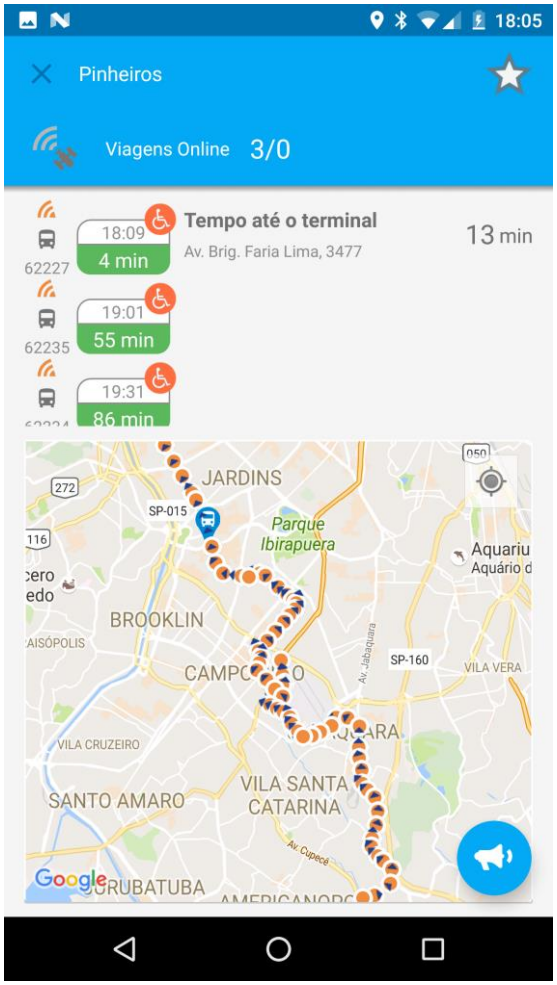


Figura 1.4: Tela Trajeto CittaMobi.

Na lista referente à linha, ao se clicar em algum ônibus, é possível fazer o acompanhamento em tempo real também visualizando o progresso dô ônibus à medida que passa pelas paradas. A Figura 1.5 demonstra esse acompanhamento por meio dos pontos de ônibus. Em quase todas as páginas é possível também enviar notificações/alertas sobre a linha ou o ônibus.

Na página inicial, por meio da aba lateral, é possível gerenciar os pontos e linhas favoritos, visualizar alertas e informativos e até gerenciar recargas de bilhete eletrônico. A Figura 1.6 mostra essa aba lateral e suas opções.

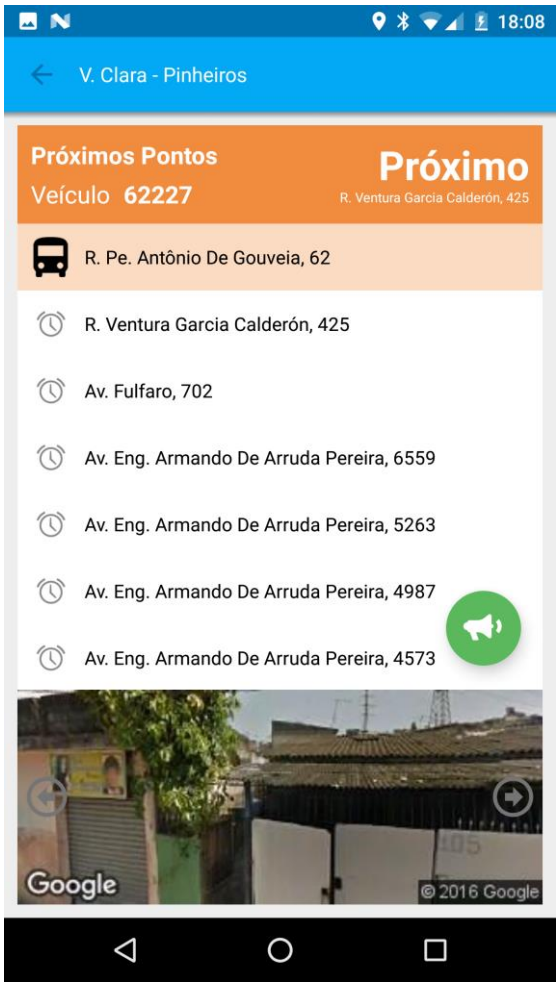


Figura 1.5: Tela Paradas CittaMobi.

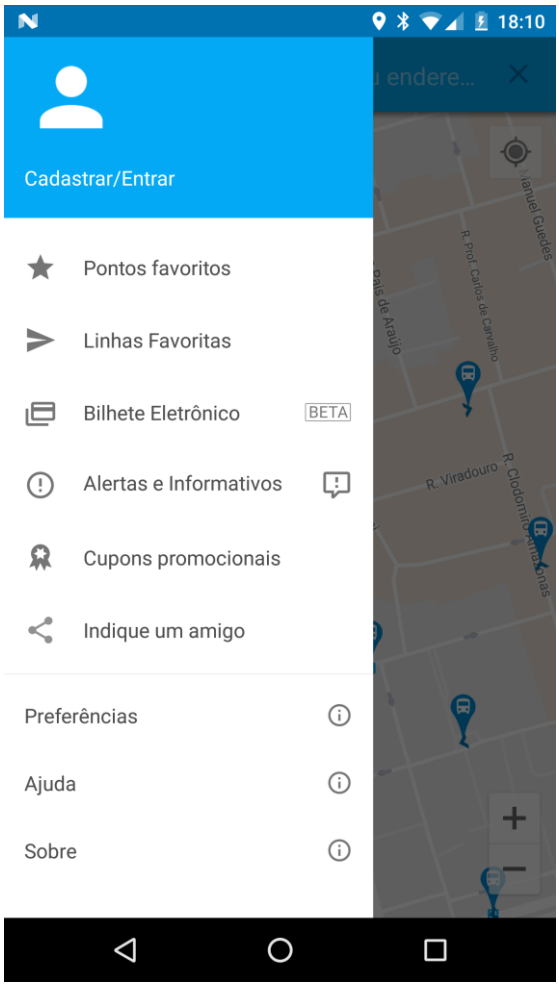


Figura 1.6: Tela Aba CittaMobi.

O CittaMobi, apesar de um ótimo aplicativo, não atende a região do Distrito Federal. Os aplicativos propostos nesse Projeto Final de Graduação visam propiciar ao público do DF um aplicativo semelhante ao CittaMobi, mas que inclua essa região e que contemple as suas particularidades. Funções bastante úteis desse app que podem futuramente ser incorporadas aos aplicativos propostos são: alertas e notificações de problemas em linhas ou ônibus, acompanhamento do deslocamento do ônibus pela lista de paradas e o detalhamento de quais ônibus são adaptados a portadores de necessidades especiais.

### 1.6.2. Moovit

O Moovit é um aplicativo gratuito de mobilidade urbana que fornece informações de transporte público em tempo real e de navegação GPS em diversos meios de transporte diferentes. Seus números são bastante impactantes: o aplicativo funciona em mais de 1.200 cidades (sendo 77 destas no Brasil), mais de 67 países e possui uma base de

mais de 45 milhões de usuários. Seguem em seguida algumas funcionalidades do aplicativo, em conjunto com algumas figuras deste:

1) Ao Redor

A página em questão possibilita que o usuário explore os arredores no mapa, identificando as paradas mais próximas, os próximos horários de chegada das linhas ativas, dentre outras infomações. A funcionalidade pode ser vista com mais detalhes na Figura 1.7.

2) Busca Inteligente

A funcionalidade permite ao usuário consultar as melhores rotas para os diversos locais da cidade. A busca pode ser feita pelo lugar, endereço, ou estação desejada. A página fornece diferentes alternativas de transporte público disponíveis, somadas a instruções detalhadas acerca do caminho até o destino. A funcionalidade é exibida na Figura 1.8.

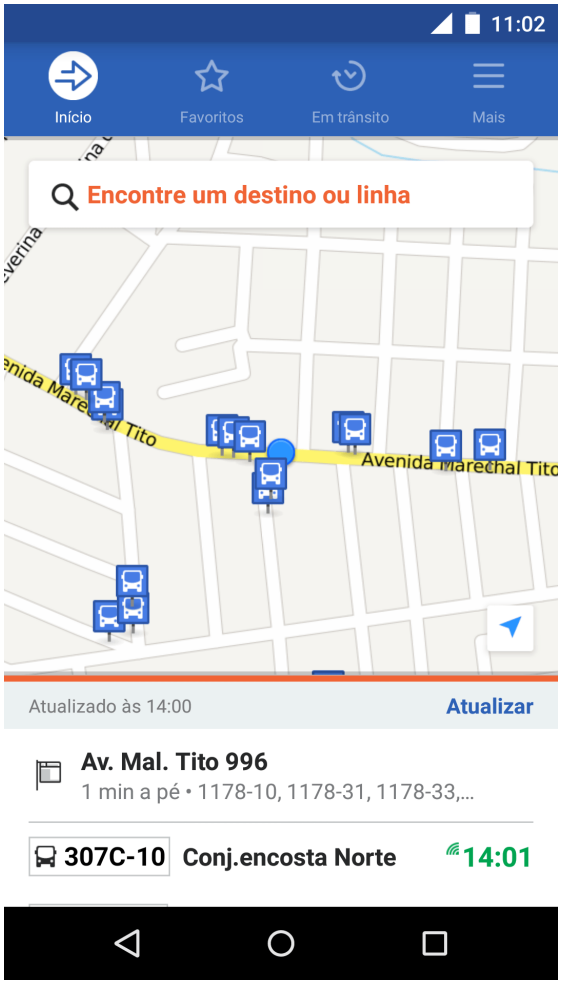


Figura 1.7: Tela Mapa Moovit.

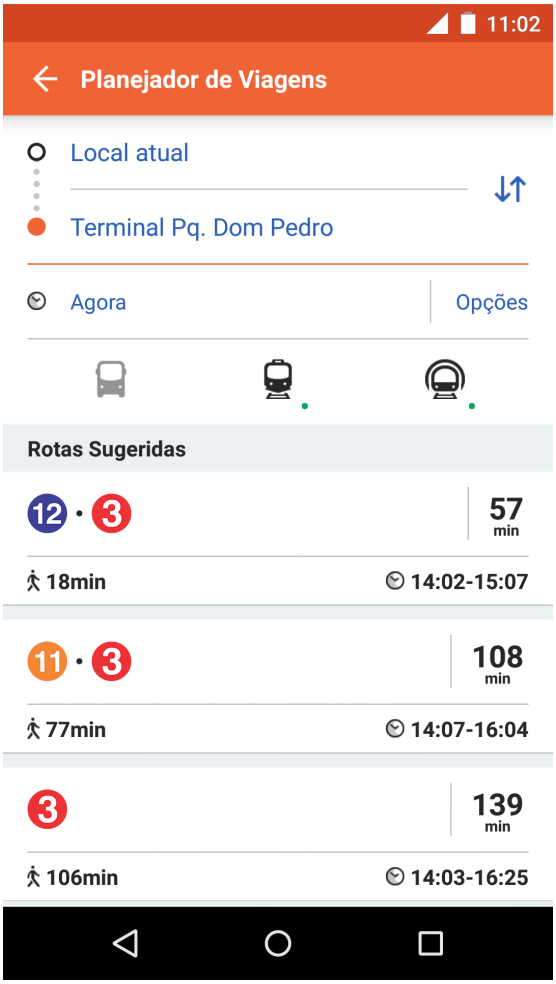


Figura 1.8: Tela Planejar Moovit.

3) Favoritos

Esta funcionalidade permite ao usuário registrar locais ou linhas favoritas para um acesso posterior mais rápido. Também é possível verificar o histórico de viagens já realizadas, podendo reutilizá-las mais facilmente. A página em questão é exibida na Figura 1.9.

4) Em Trânsito

Esta aba possibilita um controle maior da viagem, disponibilizando informações atualizadas acerca do tempo estimado de chegada, e também avisos nos momentos em que se deve desembarcar. A funcionalidade pode ser vista com mais detalhes na Figura 1.10.

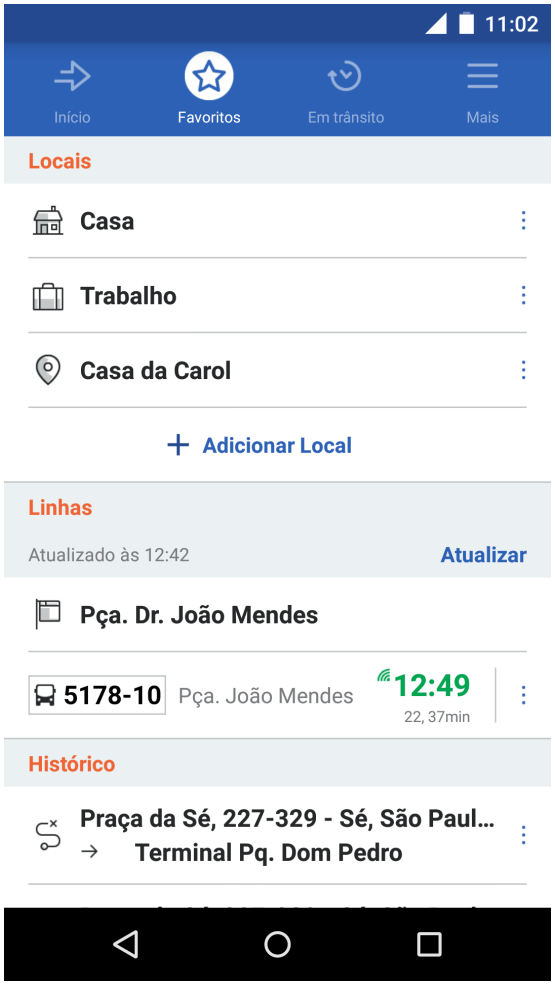


Figura 1.9: Tela Favoritos Moovit.

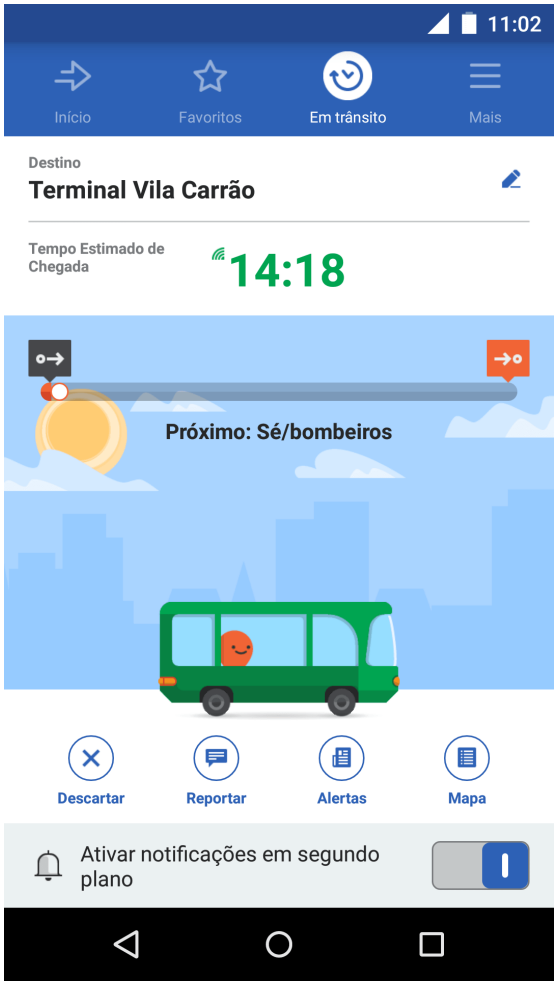


Figura 1.10: Tela Trânsito Moovit.

5) Notificações

Por meio de relatos feitos por usuários, de forma colaborativa, notificações úteis sobre a lotação das estações, incidentes, atrasos, linhas fora de serviço, dentre

outras, estão disponíveis em ‘Alertas’, possibilitando ao usuário final se planejar com antecedência. A página que disponibiliza essa opção é exibida na Figura 1.11.

Estas notificações são possíveis por meio da opção “Reportar”. Há vários tipos possíveis de relatos, alguns destes podem ser vistos com mais detalhes em seguida na Figura 1.12.

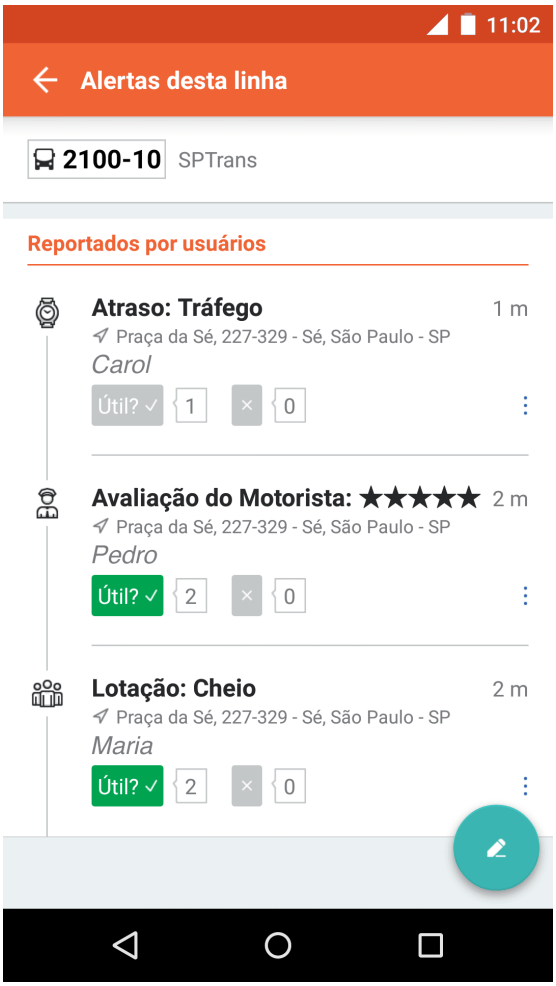


Figura 1.11: Tela Notificações Moovit.


Figura 1.12: Tela Relatos Moovit.

O Moovit atende a região do DF, porém, pelo fato de ser um aplicativo global, tende a criar uma interface padrão para todas as cidades, que atenda a todas estas de forma satisfatória. O objetivo dos aplicativos propostos neste projeto é de ter aplicativos focados no contexto do DF, e que estes entendam as particularidades da região. Além disso, estes também visam fornecer mais opções de apps ao usuário, para que este, então, possa optar pelos que mais lhe interessam. Algumas funções do aplicativo Moovit que podem futuramente ser implementadas nos aplicativos propostos nesse projeto são:

reportar situações-problema ou mudanças em linhas ou ônibus, favoritar rotas ou locais, consultar rotas partindo de outros locais, e não só o atual, dentre outros.

### 1.6.3. Cadê o Ônibus?

‘Cadê o Ônibus?’ foi o aplicativo vencedor da Hackatona realizada em 2013 pela SPTrans – gestora do sistema de transporte público por ônibus em São Paulo. O aplicativo é bastante completo, e apresenta funcionalidades semelhantes ao Moovit, dentre elas: posição geográfica dos ônibus em tempo real, localização dos pontos de ônibus próximos, itinerário das linhas, horário de partida dos ônibus, administração das linhas "favoritas" e criação de rotas. Cidades atendidas: São Paulo e Região (SPTrans e EMTU), Teresina (STrans), Rio de Janeiro (Capital) e Curitiba.

A página inicial do app é a que se encontra o mapa, estando este focado na localização do usuário, mas possibilitando navegação livre por parte do usuário. O mapa mostra também todas as paradas de ônibus dos arredores e, na parte inferior da tela, uma *toolbar* com algumas opções a serem escolhidas. A Figura 1.13 representa essa página inicial do aplicativo. Escolhendo a opção de pesquisa, representada pelo ícone da lupa, abre-se uma página com duas possibilidades de pesquisa diferentes: ou pela linha do ônibus, ou pelo ponto de ônibus desejado. Nesta mesma página, há duas abas diferentes, uma que mostra os resultados caso a pesquisa tenha sido feita pelo número da linha, e outra caso o usuário tenha optado por pesquisar o ponto. Há a opção de digitação por meio do teclado do celular, ou por meio de gravação de voz. A página já disponibiliza também os últimos resultados pesquisados e escolhidos para uma consulta mais rápida. A Figura 1.14 retrata a página em questão.

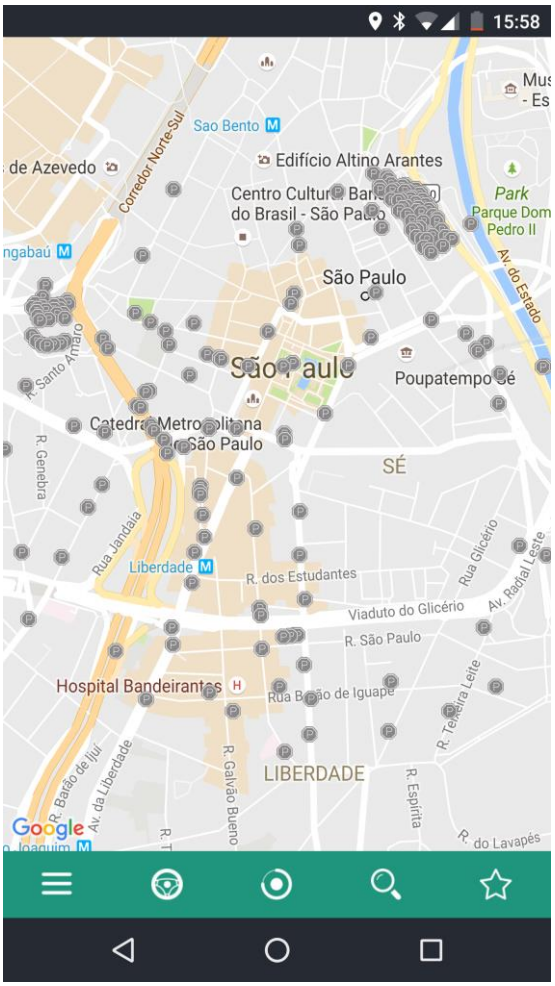


Figura 1.13: Tela Mapa Cadê o Ônibus.

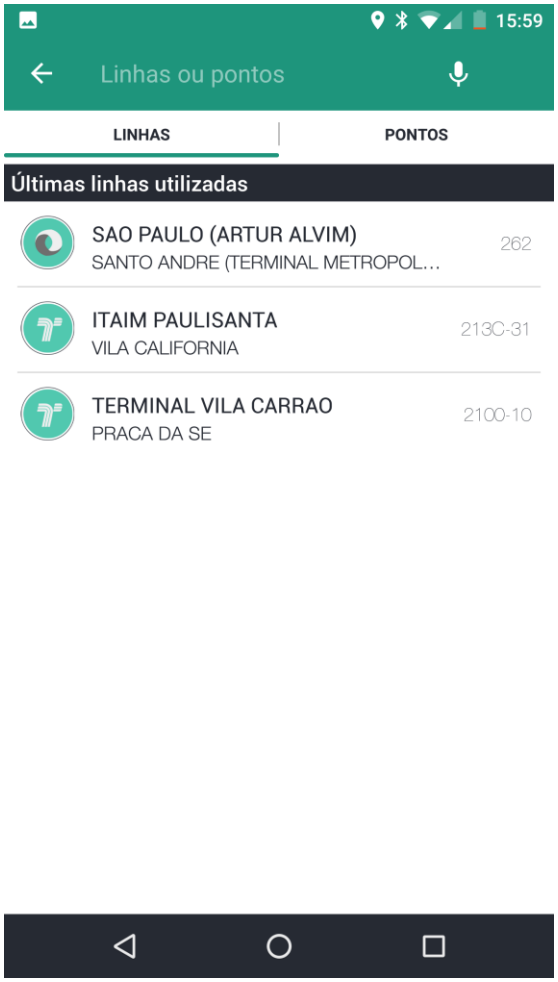


Figura 1.14: Tela Pesquisa Cadê o Ônibus.

Após a pesquisa, os resultados são exibidos, e o usuário deve escolher alguma das opções. Após a escolha, abrem-se algumas opções: “Tempo real”, “Itinerário”, “Horário” e “Favoritar”. Essa página com as funcionalidades mencionadas é exibida na Figura 1.15. Ao se optar pela opção “Tempo real”, o mapa da região a qual a linha de ônibus atende é mostrado, e neste é desenhado o trajeto exato que os ônibus da referida linha realizam. Em conjunto a essas informações, são adicionados ao mapa marcadores representando as localizações atuais exatas dos ônibus pertencentes à linha, possibilitando ao usuário o acompanhamento dos deslocamentos. Ao se clicar em algum dos marcadores dos ônibus, é mostrado na parte superior da tela alguns detalhes referentes a este ônibus, tais como: se o ônibus possui wifi ou ar-condicionado, o prefixo referente ao ônibus, a distância, e a possibilidade de seguir no mapa o ônibus. Essas funcionalidades podem ser vistas com mais detalhes em seguida na Figura 1.16.

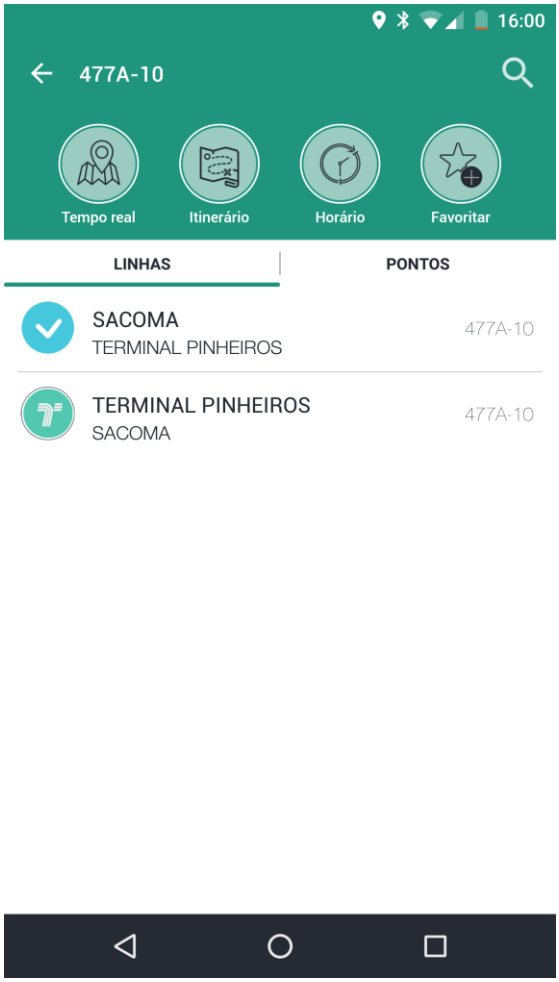


Figura 1.15: Tela Linhas Cadê o Ônibus.

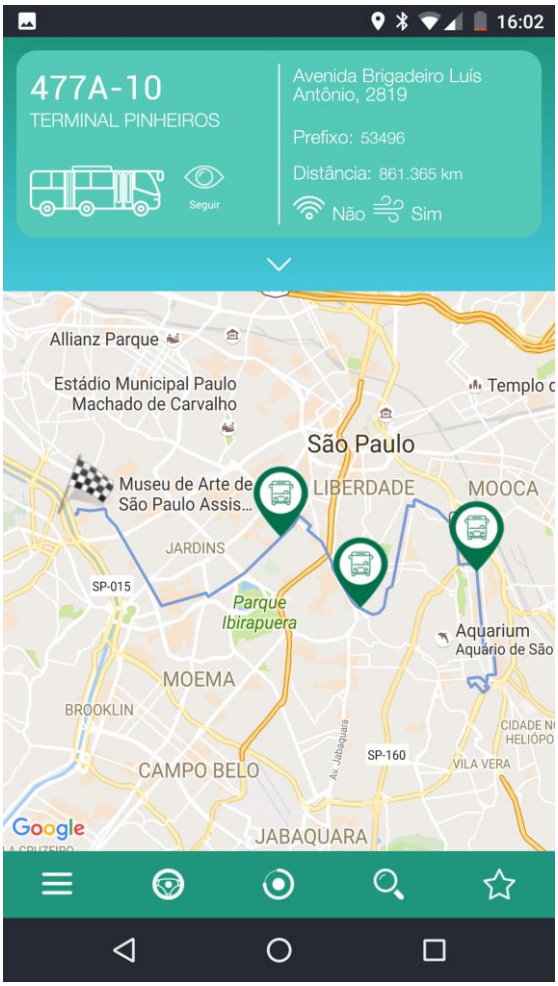


Figura 1.16: Tela Real Cadê o Ônibus.

A opção “Itinerário” mostra as paradas de ônibus contidas no trajeto, e é exibida com mais detalhes pela Figura 1.17. A opção “Horário” mostra detalhadamente os horários de partida dos ônibus, levando em consideração o ponto de partida inicial do trajeto. Esta opção mostra os hórarios para três cenários diferentes: dias úteis, sábados e domingos. Já a opção “Favoritar” possibilita ao usuário adicionar a linha ou o ponto escolhidos como favoritos. Desta forma, o usuário pode acessá-los mais rapidamente no futuro. A opção “Favoritar está exibida na Figura 1.18.

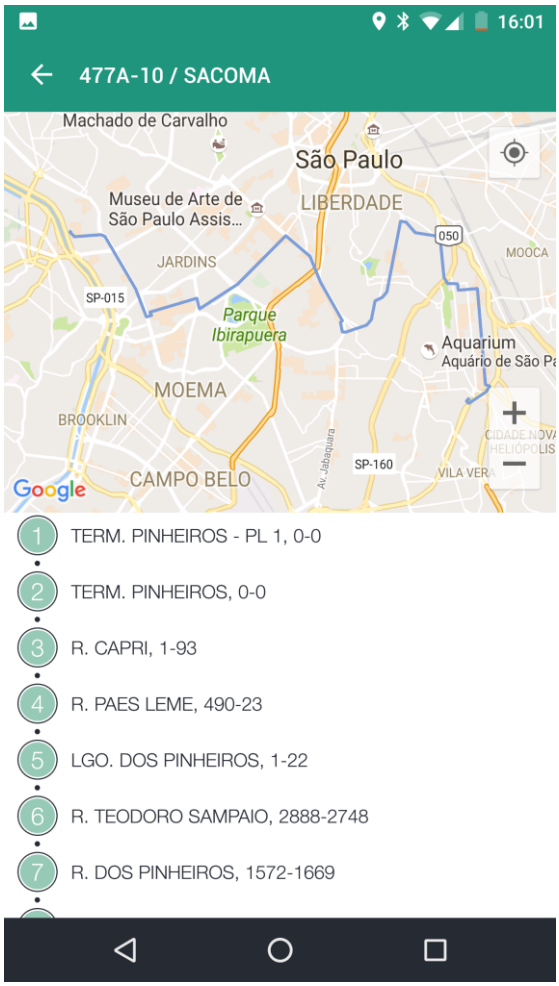


Figura 1.17: Tela Paradas Cadê o Ônibus.

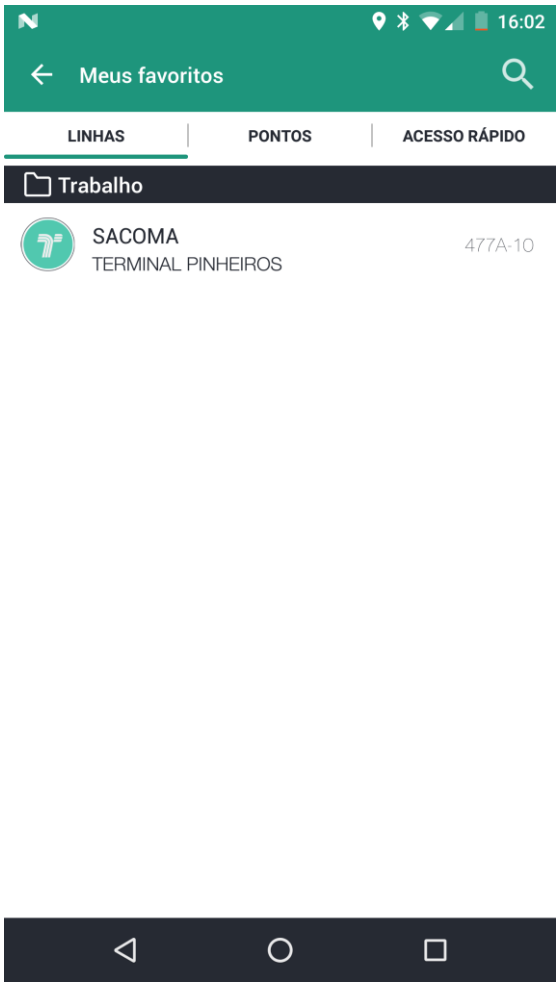


Figura 1.18: Tela Favoritos Cadê o Ônibus.

O aplicativo Cadê o Ônibus? não atende a região do Distrito Federal. De forma semelhante ao dito em 1.5.1, os aplicativos propostos nesse Projeto Final de Graduação visam propiciar ao público do DF um aplicativo que inclua essa região e que contemple as suas particularidades. Algumas funções do app em questão que podem ser incorporadas aos aplicativos propostos são: favoritar linhas ou ônibus para uso posterior, exibição da lista de paradas contidas no itinerário do ônibus e o fornecimento de informações detalhadas acerca de cada ônibus.

## 1.7. Metodologia

O projeto foi dividido em duas etapas:

1) Definições iniciais, aprendizado técnico e experiência inicial da linguagem

Nesta etapa, foram feitas reuniões de definições exatas do tema, objetivos e resultados finais esperados. Após definições iniciais e escolha da plataforma a ser usada,

iniciou-se um período de aprendizado técnico sobre a utilização da plataforma e sua linguagem. Posteriormente, iniciaram-se a experiências incipientes com a linguagem, a fim de fortalecer alguns dos conceitos estudados, e principalmente o uso de métodos e APIs que estão relacionadas ao aplicativo em questão.

2) Desenvolvimento do aplicativo

Esta etapa destinou-se ao desenvolvimento de fato dos aplicativos. As funcionalidades principais, pensadas para alcançar os objetivos traçados no início, foram todas implementadas. Algumas funcionalidades que tornariam os aplicativos ainda mais completos, porém, que demandariam mais tempo de programação, serão implementadas futuramente caso a necessidade destas se confirmem. Nesta etapa também foram implementados layouts condizentes com o objetivo inicial dos aplicativos serem de simples e de fácil utilização.

## 1.8. Estrutura do Projeto

A estrutura restante deste documento de Projeto Final de Graduação será a seguinte:

- Capítulo 2 – Ambiente de Programação: Espaço destinado à explicação da escolha da plataforma utilizada, ao detalhamento dos aplicativos criados e das principais funções utilizadas nestes, e às decrições das classes e métodos utilizados.

- Capítulo 3 – Funcionamento dos Aplicativos: capítulo destinado às descrições e demonstrações por meio de figuras do funcionamento dos aplicativos criados.

- Capítulo 4 – Conclusões e Trabalhos Futuros: segmento destinado às conclusões após o desenvolvimento dos aplicativos e do levantamento de possibilidades de melhorias nestes em trabalhos futuros.

# Capítulo 2

# Ambiente de Programação

## 2.1. Plataforma

Para a escolha da plataforma, uma análise de alguns fatores importantes foi feita, com a finalidade de que o protótipo final realmente cumprisse os objetivos listados inicialmente da forma mais eficaz possível. Um dos pontos mais importantes a serem satisfeitos é o maior alcance que o aplicativo e a tecnologia teriam na sociedade em geral, com foco na parte desta que realmente faz uso frequente do transporte coletivo por ônibus. Outro ponto levado em consideração é a quantidade de conhecimento disponível para estudo e pesquisa. Principalmente por essas duas questões, a plataforma escolhida para o desenvolvimento do projeto proposto foi Android. O *software* usado para a programação e a compilação dos códigos foi o Android Studio.

## 2.2. Aplicativos criados

Neste projeto foram criados dois aplicativos com objetivos diferentes e que devem ser utilizados por públicos diferentes, são eles:

- Aplicativo que deve ser utilizado pelo usuário final do sistema de transporte público por ônibus, e que tem como objetivo principal a consulta das melhores rotas para se chegar a um destino final utilizando o referido sistema. Este aplicativo foi denominado **LocalizaÔnibus**, e será doravante identificado dessa forma.

- Aplicativo que deve ser usado por funcionários do sistema de transporte público por ônibus, e tem por objetivo enviar ao servidor e banco de dados periodicamente a localização atual do ônibus em questão, atualizando a informação antiga do banco de dados acerca deste ônibus. A este app foi dado o nome **Rastreador**, e este será posteriormente identificado dessa forma.

## 2.3. Google Maps Directions API

A *Google Maps Directions API* é uma das principais APIs usadas no aplicativo principal (manuseado pelo usuário). É por meio dela que se fazem possíveis os cálculos das rotas que poderão ser utilizadas para o deslocamento necessário, sendo essas rotas do ponto inicial em que se encontra o usuário até o destino final escolhido por ele. A solicitação dessas possíveis rotas é feita por meio de uma solicitação HTTP.

A API possibilita cálculos de rotas para diferentes meios de transporte, sendo estes: transporte público, condução, caminhada ou bicicleta. Nestas rotas, há duas possibilidades diferentes de especificar origens, destinos e pontos de referência, podendo ser feitas como strings de texto ou como coordenadas de latitude/longitude. A Directions API pode retornar rotas em várias partes usando uma série de pontos de referência.

A solicitação HTTP necessária para o cálculo das rotas tem o seguinte formato:

*http://maps.googleapis.com/maps/api/directions/**output**?**parameters***

Como, no caso do aplicativo, a solicitação incluirá dados confidenciais ao usuário, como sua localização, será utilizado o HTTPS, em detrimento ao HTTP. Nesse contexto, a solicitação se dará da seguinte forma:

*https://maps.googleapis.com/maps/api/directions/**output**?**parameters***

Os valores de ***output*** e ***parameters*** devem ser escolhidos de acordo com as necessidades no cálculo da rota.

Para ***output***, há duas opções:

- json – a saída será em JavaScript Object Notation (JSON)
- xml – a saída será em XML

No aplicativo em questão, a opção adotada para ***output*** será a '***json***'.

Para ***parameters***, a lista de opções é mais extensa. Há dois tipos de parâmetros, os obrigatórios e os opcionais. Segue maior detalhamento:

- Parâmetros obrigatórios:
    - ***origin***: ponto de partida para o cálculo da rota. Pode ser informado por meio do endereço, do valor de latitude/longitude (em forma de texto) ou do ID de local.
    - ***destination***: ponto de chegada para cálculo das rotas. Pode ser informado por meio do endereço, do valor de latitude/longitude (em forma de texto) ou do ID de local.
    - ***key***: chave de API que identifica o aplicativo.
- Parâmetros opcionais:
    - ***mode***: modo de transporte a ser utilizado no cálculo da rota. As opções são as seguintes:
        - ***driving***: estabelece uma rota de condução comum fazendo uso das estradas. É a opção padrão para ***mode***.
        - ***walking***: indica uma rota de caminhada, considerando, quando disponíveis, vias para pedestres e calçadas.
        - ***bicycling***: indica uma rota para bicicleta, considerando, quando disponíveis, ciclovias e ruas preferencias.
        - ***transit***: estabelece, quando disponível, uma rota de transporte público.
    - ***waypoints***: indica pontos de referência, fazendo com que a rota passe por alguns locais específicos.
    - ***alternatives***: possibilita mais de uma alternativa de rota como resposta à requisição, caso o parâmetro seja definido como '***true***'.
    - ***avoid***: estabelece que as rotas calculadas devem evitar alguns componentes específicos nas vias, como pedágios, pontes, etc.
    - ***language***: especifica em que idioma devem estar os resultados .

- ***units***: indica o sistema de unidades que deve ser adotado na exibição dos resultados.
- ***region***: retorna resultados direcionados a uma região específica.
- ***arrival_time***: indica o horário desejado de chegada. Caso o parâmetro não seja especificado, o horário da solicitação será utilizado.
- ***departure_time***: indica o horário de saída desejado.
- ***traffic_model***: estabelece as suposições a serem usadas para o cálculo do tempo no trânsito.
- ***transit_mode***: indica os modos de transporte preferenciais no estabelecimento na rota. Esse parâmetro só pode ser especificado para rotas de transporte público. As opções são:
  - ***bus:*** indica preferência ao transporte por ônibus.
  - ***subway***: indica preferência ao transporte por metrô.
  - ***train***: indica preferência ao transporte por trem.
  - ***tram***: indica preferência ao transporte por bonde.
  - ***rail***: indica preferência ao transporte por trem, bonde e metrô.
- ***transit_routing_preference***: indica preferências para rotas de transporte público. As opções são:
  - ***less_walking***: indica preferência a uma quantidade reduzida de caminhada.
  - ***fewer_transfers***: indica preferência a uma quantidade reduzida de baldeações.

No contexto do aplicativo em questão, dois métodos diferentes utilizam a *Google Maps Directions API*. O primeiro deles tem como objetivo a solicitação de

possíveis rotas, considerando o transporte público, a serem utilizadas pelo usuário. A resposta contempla algumas alternativas de rotas, possibilitando o usuário a escolher qual alternativa mais lhe convém. Neste método, os parâmetros utilizados na requisição HTTPS, em conjunto com os valores passados a eles, foram:

- ***origin***: valor de latitude/longitude (em forma de texto) da localização em que o usuário se encontra.
- ***destination***: valor de latitude/longitude (em forma de texto) referentes ao local especificado pelo usuário como destino.
- ***key***: chave de API que identifica o aplicativo.
- ***mode***: opção '***transit***', indicando uma rota de transporte público.
- ***alternatives***: opção '***true***', requisitando mais de uma alternativa de rota como resposta.
- ***language***: opção '***pt-BR***', indicando português-Brasil como a língua escolhida.
- ***transit_mode***: opção '***bus***', indicando a preferência ao transporte por ônibus.

O segundo método tem por objetivo o cálculo da estimativa de tempo que o ônibus levará até chegar à parada de ônibus em que o usuário se encontra. Neste caso, para facilitação dos cálculos, o modo de transporte (***mode***, da lista acima) considerado é o '***driving'***. Seguem os parâmetros utilizados nesta requisição HTTPS:

- ***origin***: valor de latitude/longitude (em forma de texto) da localização do ônibus referente à alternativa escolhida pelo usuário.
- ***destination***: valor de latitude/longitude (em forma de texto) referentes ao ponto de ônibus o qual o usuário utilizará para entrar no ônibus.
- ***key***: chave de API que identifica o aplicativo.
- ***mode***: opção '***driving***', indicando uma rota de condução comum.

- ***alternatives***: opção '***false***', requisitando somente uma alternativa de rota como resposta.

- ***language***: opção '***pt-BR***', indicando português-Brasil como a língua escolhida.

Nas URLs a serem utilizadas nas requisições HTTPS, os parâmetros devem ser separados pelo caractere E comercial (&).

## 2.4. Google Places API

A *Google Places API* possibilita ao aplicativo o acesso aos dados do mesmo banco de dados utilizado pelo *Google Maps*, disponibilizando informações e localizações de mais de 100 milhões de empresas e pontos de interesse, sendo essas informações atualizadas de forma frequente. A funcionalidade da *API* utilizada neste projeto é a ***autocomplete***. Esse é um serviço que fornece previsões de lugares em resposta a uma solicitação HTTP. Esses lugares podem ser empresas, endereços e pontos de interesse, dependendo, então, de como a solicitação é feita.

A funcionalidade de ***autocomplete*** é utilizada no campo de pesquisa de destino final do trajeto. À medida que o usuário digita trechos desse destino desejado, o aplicativo apresenta, por meio de lista, opções de lugares que condizem com o trecho digitado. Requisições a esta *API* são feitas de forma bastante frequente, levando em consideração que o usuário pode alterar com facilidade o texto digitado.

A requisição HTTP enviada à Google Places API tem o seguinte formato:

*https://maps.googleapis.com/maps/api/place/autocomplete/**output**?**parameters***

Para ***output***, há duas opções:

- json – a saída será em JavaScript Object Notation (JSON)

- xml – a saída será em XML

No aplicativo em questão, a opção adotada para ***output*** foi a '***json***'.

Para ***parameters***, a lista de opções é mais extensa. Há dois tipos de parâmetros, os obrigatórios e os opcionais. Segue maior detalhamento:

- Parâmetros obrigatórios:
    - ***input***: o texto que deve ser pesquisado.
    - ***key***: chave de API que identifica o aplicativo.
- Parâmetros opcionais:
    - ***offset***: a posição do último caractere no termo de entrada que o serviço usa para corresponder às previsões,
    - ***location***: o ponto ao redor do qual deseja-se consultar informações de local.
    - ***radius***: a distância dentro da qual deve-se retornar resultados de local.
    - ***language***: o idioma em que os resultados serão retornados.
    - ***types***: os tipos de resultados de locais a retornar.
    - ***components***: um agrupamento de locais ao qual se deseja restringir os resultados. Por exemplo, é possível filtrar por país.

Nos aplicativos foram utilizados os seguintes parâmetros opcionais:

- ***language***: opção '***pt-BR***', indicando português-Brasil como a língua escolhida.
- ***types***: opção '***geocode***'.
- ***components***: opção '***country-br***', reduzindo a pesquisa a locais no Brasil.

## 2.5. Arquitetura Parse

Até recentemente, Parse era uma serviço da categoria BaaS (*Backend as a service*), e disponibilizava SDK's para o desenvolvimento de *backends* móveis para Windows 8, Windows Phone 8, iOS, Android, JavaScript, e OS X. Porém, em Janeiro de

2016, foi anunciado oficialmente que o serviço de hospedagem fornecido pelo Parse seria encerrado. A partir do anúncio, foi lançado o Parse Server, uma versão de código aberto do Parse *backend*. Esse servidor pode ser implantado em qualquer infraestrutura que possa rodar Node.js, e possibilita a execução da maior parte da Parse API. Node.js, por sua vez, é um interpretador de código JavaScript que funciona do lado do servidor. Com ele, é possível a criação de aplicações de rede rápidas e escaláveis.

Em seguida, encontram-se figuras com os esquemáticos que resumem o formato de conexão do cliente à infraestrutura de *backend*. No primeiro caso, representado pela Figura 2.1, o esquemático representa a estrutura anterior ao anúncio de encerramento do Parse. Neste contexto, toda a infraestrutura era hospedada pelo Parse. Os clientes se comunicavam com api.parse.com, e, dessa forma, podiam gerenciar seus projetos e banco de dados. Nos esquemáticos, a sigla 'BD' refere à banco de dados.


Figura 2.1: Estrutura antiga Parse.

Após as mudanças e o lançamento do Parse Server – versão de código aberto do Parse - a estrutura passa a não ser mais hospedada pelo Parse. Em outras palavras, o Parse Server não é dependente da infraestrutura de *backend* do Parse. Neste caso, há duas opções: pode-se criar uma infraestrutura local própria e executar o Parse Server localmente, ou implantar este servidor utilizando a infraestrutura de um provedor que ofereça serviços do tipo BaaS. Há várias opções de provedores que podem ser utilizados, alguns deles são: Heroku, Firebase e AWS (*Amazon Web Services)*.

Além da mudança do provedor, há também a alteração no banco de dados. Onde anteriormente usava-se um banco de dados próprio do Parse, agora há a utilização de uma ferramenta externa, chamada MongoDB. Essa é uma plataform de código aberto, sem custos, de banco de dados orientado a documentos. A plataforma é classificada como um programa de banco de dados '*NoSQL*', se referindo ao fato de evitar a estrutura tradicional de banco de dados baseados em tabelas, em favor do uso de documentos em formato JSON com esquemas dinânimos, os quais os denomina de BSON. Neste cenário,

o servidor se comunica com o banco de dados MongoDB, possibilitando com que a requisição ao banco de dados, feita pelo aplicativo do usuário, seja atendida com sucesso.

Em seguida, a Figura 2.2 exibe o esquemático que representa o novo cenário.

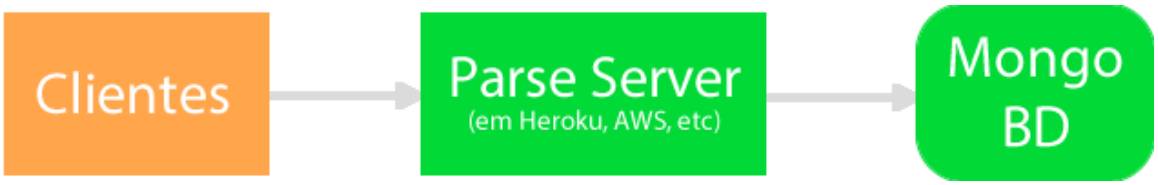


Figura 2.2: Estrutura nova Parse.

No contexto do aplicativo proposto, o provedor de infraestrutura escolhido para hospedar e gerenciar o Parse Server foi o Heroku. Essa é uma plataforma de aplicações em nuvem, a qual dá suporte ao Node.js. Para o gerenciamento do banco de dados, foi utilizado o mLab. Essa é uma plataforma de DaaS (*Database as a Service*), fornecendo serviço de banco de dados em nuvem, que disponibiliza provisionamento automático, gerenciamento e escalabilidade ao banco de dados MongoDB.

A Figura 2.3 em seguida exibe o esquemático que representa o cenário escolhido, com as devidas plataformas utilizadas.


Figura 2.3: Detalhamento da estrutura nova Parse.

## 2.6. Estrutura

### 2.5.1. Aplicativo LocalizaÔnibus

Conforme detalhado anteriormente, o app em questão será utilizado pelo usuário final, e tem por objetivo principal a consulta das melhores rotas para se chegar a um destino final utilizando o transporte público por ônibus.

Ao se abrir o aplicativo, a primeira classe a ser executada é a ‘StarterApplication’. Esta classe é responsável por inicializar o serviço do servidor Parse, e, para esta conexão, informa alguns parâmetros importantes, tais como:

- .*applicationId*: indica o ID da aplicação. Para estes aplicativos o ID é: ‘carro1dsauhdiy23h2i3huh’.
- .*clientKey*: indica a chave-cliente da aplicação, e é opcional. Não foi utilizada para os apps em questão.
- .*server*: indica a URL da aplicação no servidor. Neste caso é: ‘https://carro1.herokuapp.com/parse/’.

Após a inicialização do serviço, as demais classes do aplicativo podem enviar as requisições necessárias ao servidor sem que seja necessário iniciar novamente a conexão.

Em seguida, a principal classe, a ‘MainActivity’, é executada. Nessa *activity* estão contidas as principais estruturas e funcionalidades utilizadas no aplicativo. A principal estrutura desta classe é o mapa, inicializado já no método *onCreate()*. Após o mapa já pronto, o método *onMapReady()* é executado, e para que o mapa mostre apenas a região em que o usuário se encontra e seus arredores, *onMapReady()* aciona o método *getMyLocation()*. Neste, o aplicativo, por meio de alguns provedores, como GPS, Internet, etc., captura as coordenadas atuais do usuário e aplica um zoom nesta área. O método também torna esse processo periódico, fazendo com que o app esteja frequentemente atualizando a informação acerca da localização atual do usuário, mesmo que o zoom não seja novamente alterado. As atualizações periódicas são requisitadas por meio da função *requestLocationUpdates*, e executadas por meio do método *onLocationChanged()*. Para que, após a abertura do aplicativo e a manipulação do usuário no mapa, o zoom seja alterado, há, nesta página mapa, um botão em tons azuis que realiza a função.

Há, na parte superior da página Mapa, uma *toolbar* com algumas funcionalidades, dentre elas: um campo para digitação de pesquisa, um ícone de pesquisa, e um menu com o item ‘configurações’. Ao se digitar algo no campo de pesquisa, uma lista com algumas opções que correspondem à pesquisa é mostrada, e o usuário deve escolher uma das opções. Após a escolha, o aplicativo deve prosseguir com o funcionamento.

Para que as funcionalidades de pesquisa sejam possíveis, alguns métodos são aplicados, são eles: *handleIntent()*, *onNewIntent()*, *doSearch()*, *getPlace()*, *onQueryTextSubmit()*, *onQueryTextChange()*, *onCreateLoader()*, *onLoadFinish()*, *onLoaderReset()* e *showLocations()*, tendo cada um destes uma função diferente. O método *onCreateLoader()* é que o chama a classe *PredictionProvider*, classe responsável por realizar de fato a pesquisa que corresponde ao texto digitado pelo usuário. Este procedimento é feito por meio de uma requisição HTTPS à *Google Places API*, fazendo uso da funcionalidade *Place Autocomplete*. Esse é um serviço de preenchimento automático da *Google Places API* para Android que retorna previsões de locais em resposta a uma consulta de pesquisa de local em texto. Neste caso, a requisição HTTPS deve ter o seguinte formato:

*https://maps.googleapis.com/maps/api/place/autocomplete/**output**?**parameters***

Além da acima mencionada, outra requisição é feita à *Google Places API*, agora com o objetivo de se obter mais detalhes acerca dos locais previstos. A requisição é feita utilizando o valor de referência do local desejado, e deve possuir o seguinte formato:

*https://maps.googleapis.com/maps/api/place/details/**output**?**parameters***

Para essas requisições são retornados objetos JSON de resposta, e estes devem ser analisados para que apenas as informações necessárias sejam extraídas. Essa função é realizada pelas classes *PredictionJSONParser* e *PredictionDetailsJSONParser*, sendo a primeira responsável pelas respostas das requisições *autocomplete* e a segunda pelas respostas das requisições de mais detalhes.

A pesquisa representa o endereço de destino ao qual o usuário deseja se deslocar. Ao escolher alguma das opções da lista, o método *createRoute()* é executado, passando como parâmetros para a criação da rota as geocoordenadas de origem – que definem a localização atual do usuário, e as geocoordenadas de destino – que definem a localização do destino escolhido.

Este método, por sua vez, chama o método *getDirectionsUrl()*, que é responsável por formar a URL de requisição que será enviada à *Google Maps Directions API*. Os parâmetros utilizados na URL foram devidamente explicados no tópico 2.3.

Após a formação da URL, um objeto da classe *DownloadTask* é criado e executado, passando como parâmetro a URL recém criada. Essa classe é responsável por chamar o método *downloadUrl()* – que de fato fará a requisição HTTPS e, após a resposta da API, criar o objeto da classe *ParserRouteTask* – responsável por tratar o JSON de resposta, e executá-lo, passando como parâmetro este JSON.

A classe *ParserRouteTask*, por sua vez, cria um objeto da classe *DirectionsJSONParser* – responsável diretamente por analisar o JSON de resposta e extrair as informações necessárias. Após essa resposta apenas com as informações necessárias ser retornada, a classe analisa essa resposta, altera alguns arrays importantes e, por fim, chama outra atividade, representada pela classe *PopUp*. Além disso, ela executa o método *BusesInfo()*, que será detalhado posteriormente.

A classe *DirectionsJSONParser* analisa o JSON de resposta da Directions API, adiciona algumas informações a alguns *arrays* estáticos da classe MainActivity, e retorna um *array* principal como resposta. Um método importante dessa classe é o *decodePoly()*, método responsável por decodificar os pontos contidos na rota, para que seja possível o desenho correto do trajeto no mapa.

O método *BusesInfo()*, executado na classe *ParserRouteTask*, é responsável por criar um objeto da classe *BusesJSONParser()*, cuja função é extrair do JSON de resposta exclusivamente as informações referentes aos ônibus e pontos de ônibus contidos no trajeto. Estas informações serão utilizadas para adição de marcadores importantes ao mapa, como os que representam as localizações exatas dos ônibus e das paradas dos ônibus.

A atividade representada pela classe *PopUp* (chamada pela classe *ParserRouteTask*) é responsável por organizar as informações referentes às rotas em uma nova janela aberta ao usuário. Estas informações são extraídas dos diversos *arrays* que foram criados a partir do JSON de resposta, e são organizadas nesta janela por meio de uma lista de opções de rotas diferentes. Esta lista exibe ao usuário a quantidade de baldeações que estão contidas na rota, o preço total do trajeto, as distâncias a serem percorridas por caminhada e a distância e duração estimadas do trajeto. Os itens da lista são clicáveis, de forma que o usuário possa escolher a opção de rota desejada. Os métodos *prepareRoutesData()* e *treatRouteInfo()* são utilizados com a finalidade de extrair

somente as informações necessárias à nova janela. A classe interna *SimpleDividerItemDecoration* é responsável apenas por definir e adicionar linhas divisórias entre cada item da lista contida na janela. Após a abertura da janela, caso o usuário clique fora da área representada por esta, o aplicativo volta à atividade *MainActivity*.

Após o clique do usuário para a escolha da rota desejada entre as opções disponíveis na lista, a janela PopUp se fecha e o aplicativo retorna à página Mapa. No entanto, à medida que esta página é novamente acessada, são adicionadas várias informações ao mapa, sendo estas adicionadas por alguns métodos diferentes. A chamada destes métodos é feita pelo método *onResume()*. Este é executado quando a atividade começará a interagir com o usuário, ou seja, quando esta estiver no topo da pilha de atividades. Esse fato acontece ao se fechar a janela *PopUp* e se retornar à atividade *MainActivity* e, por isso, neste momento, *onResume()* é executado.

Caso a variável *routechosen* tenha o valor de '*true*', vários métodos são chamados. O primeiro deles é o *stoprepeatingtask()*, cujo objetivo é, a partir da segunda pesquisa de rota em uma mesma abertura do app, encerrar a thread de atualização – denominada *mLocationUpdater* – referente à pesquisa imediatamente anterior.

Em seguida, o método *ResetValuesChoose()* é chamado. Este tem por objetivo limpar os valores de diversas variáveis, *arrays*, e ainda remover alguns marcadores do mapa, possibilitando uma nova consulta de rota completamente independente da anterior.

Posteriormente, o método executado é o *addBusMarker()*, cujo objetivo é preparar e adicionar valores as *arrays* que serão utilizados futuramente, principalmente para adicionar marcadores ao mapa.

Em seguida, há a chamada do método *startRepeatingTask()*. Este método é responsável por executar uma thread de atualização denominada *mLocationUpdater*, a qual, a partir desse momento, passa a ser executada periodicamente a cada três segundos. A thread em questão, por sua vez, tem como função executar o método *RetrieveBD()*. Este método é responsável por buscar, no banco de dados externo, as localizações atuais dos ônibus que fazem parte da rota. Atualizações a cada três segundos devem ser feitas de modo a garantir que o aplicativo tenha sempre a localização em tempo real destes ônibus. A partir das informações coletadas no banco de dados, o método adiciona ou reposiciona

os marcadores dos ônibus nas localizações correspondentes no mapa, tornando possível o acompanhamento em tempo real do deslocamento do ônibus. Além disso, nesse contexto, o método *Estimatetrigger()* é executado. Este será explicado posteriormente.

Após *startRepeatingTask()*, o método *addBusStopsMarker()* é executado. Este tem o objetivo de, por meio de informações incluídas em *arrays* da classe *MainActivity*, adicionar ao mapa marcadores indicando exatamente as localizações dos pontos de ônibus que o usuário utilizará para embarcar em algum ônibus. O último ponto de ônibus, o qual o usuário utiliza apenas para desembarcar da última baldeação, não é assinalado no mapa.

Em seguida, o método executado é o *DrawRoute()*. Esse é um dos principais métodos a serem executados no app, e tem como objetivo o desenho da rota escolhida no mapa. Os trechos a serem percorridos de diferentes formas recebem cores diferentes, sendo a cor vermelha destinada aos trechos por caminhada, e a cor azul aos trechos por ônibus. Um marcador vermelho sinaliza o final da rota. Após o desenho da rota, o método *ZoomCorrection()* é executado, este com a finalidade de redimensionar o zoom do mapa a fim de incluir na tela todo o trajeto recém desenhado.

Neste novo cenário, com a rota e os marcadores adicionados, surge uma nova estrutura na página Mapa: um painel de deslize na parte inferior da tela com o resumo das informações acerca da rota escolhida, e com as previsões em tempo real para a chegada dos ônibus às paradas. Antes da escolha da rota, o painel estava desabilitado, porém, após a escolha desta, este passa a fazer parte da página. O painel surge minimizado, porém, pode ser expandido ou minimizado novamente a partir de um clique ou um toque na tela. As informações são inseridas no painel por meio do método *InsertInfoPanel()*, executado após o método *DrawRoute()*.

Como dito anteriormente, no contexto do método *RetrieveBD()*, o método *Estimatetrigger()* é acionado. Este é o primeiro responsável pela adição ao painel-resumo das previsões das chegadas dos ônibus nas paradas. Sempre que o aplicativo faz nova consulta ao banco de dados em busca de informações atualizadas acerca da localização dos ônibus, ele adiciona ou reposiciona os marcadores referentes aos ônibus, e aciona *Estimatetrigger()* para atualizar as novas informações também no painel. Este, por sua vez, aciona o método *EstimateRoute()*, que é responsável por dar seguimento ao processo de cálculo das estimativas de chegada. Esse cálculo é feito por meio de uma nova

requisição HTTPS à *Google Maps Directions API*, agora considerando como origem da rota a localização atual do ônibus e como destino o ponto de ônibus ao qual este se desloca. Para facilitar os cálculos, o modo de transporte utilizado nessa requisição foi '*driving*', ou seja, uma rota de condução comum. O método *EstimateRoute()* aciona o método o *getEstimateUrl()*, cuja função é a de construir a URL que será utilizada na requisição e retorná-la. Após esse retorno, um objeto da classe *DownloadEstimateTask* é criado e executado, passando como parâmetro a URL retornada. Essa classe, por sua vez, é responsável por executar o método *downloadUrl()*, cuja função é de fato realizar o download do objeto JSON e retorná-lo.

Após o retorno mencionado, um objeto da classe *ParserEstimateTask* é criado e executado, tendo como parâmetro o objeto JSON de resposta. Nesta nova classe, há a criação de um objeto da classe externa *EstimateJSONParser*, a qual é responsável por realizar a análise do JSON de resposta e extração das informações necessárias, retornando estas em uma lista. Finalmente, o método *UpdateEstimate()* pode ser acionado. Esse método é responsável por, a partir da lista anteriormente retornada, de alguns *arrays* criados e algumas outras informações, adicionar e atualizar as informações referentes à previsão de chegada dos ônibus no painel-resumo. No entanto, para que o cálculo das previsões finais sejam feitas, algumas durações de trajetos devem ser somadas. Como a maioria das durações está em formato *string*, foi criado um método para analisar essas diversas *strings* e somá-las de forma correta. Esse método foi denominado *EstimateSum()*, e este é executado pelo *UpdateEstimate()*.

Com as classes e métodos acima funcionando corretamente, o usuário terá um conjunto de informações valiosas a respeito do trajeto até o destino final, das informações detalhadas dos ônibus os quais ele utilizará e da estimativa de quanto tempo esse deslocamento durará. Para que o usuário possa se planejar de forma ainda mais eficaz, o aplicativo, por meio do método *NotifyUser()*, notifica o usuário quando o próximo ônibus que deve ser utilizado estiver próximo à parada de ônibus. Essa notificação é feita por meio de um alerta vibratório no celular e por uma mensagem *popup* na tela informando a estimativa de quanto tempo o ônibus levará até chegar ao ponto. O aplicativo emite notificações ao usuário quando a estimativa de chegada do ônibus for de 5, 3 e 1 minutos. O método *NotifyUser()* é acionado pelo método *EstimateSum()*.

Além das funcionalidades descritas acima, há ainda a aba lateral com algumas opções ao usuário. Essa aba se mantém possível de ser expandida (da extrema esquerda da tela) a qualquer momento ou estágio do aplicativo.

Para que fique mais claro o funcionamento e a estrutura do app, há em seguida, representado pela Figura 2.4, um esquemático com as classes existentes e o fluxo padrão que é seguido. Em cor verde, há a classe que é acionada no início do aplicativo, antes de todas as demais. Em cinza, ao centro, há a classe principal, a qual faz o acionamento de quase todas as demais classes. Em laranja, há as classes responsáveis pelas funcionalidades de pesquisa, incluindo o *auto-complete*, o qual é responsável por prover ao usuário possibilidades de destinos que condizam com o texto digitado. Em azul, há as classes que analisam os objetos JSON retornados pela *Google Maps Directions API* e acionam as medidas necessárias posteriormente. Por fim, na cor roxa, há as classes relacionadas à janela *PopUp* de escolha das alternativas de rotas. É importante ressaltar que as flechas são de ida e volta, pois na maioria das vezes a classe recebe uma requisição de outra classe, e retorna os valores solicitados. Além disso, há no esquemático os números de 1 à 6, que representam o fluxo padrão de funcionamento do aplicativo.

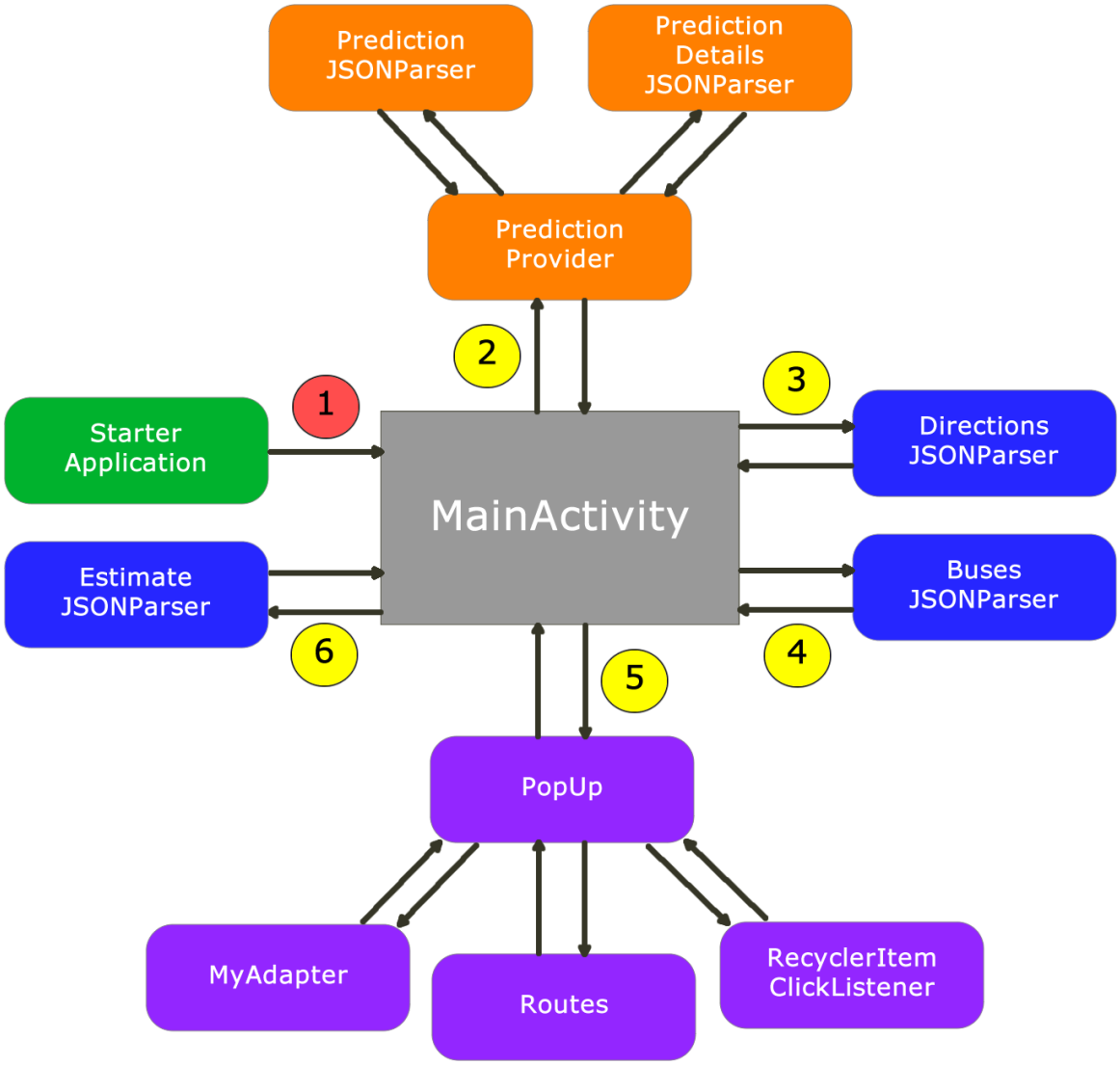


Figura 2.4: Estrutura de classes e funcionamento do aplicativo LocalizaÔnibus.

### 2.5.2. Aplicativo Rastreador

Conforme explicado anteriormente, o app deve ser usado por funcionários do sistema de transporte, e seu objetivo é enviar ao servidor e banco de dados periodicamente a localização atual do ônibus em questão, atualizando a informação antiga do banco de dados acerca deste ônibus.

De forma semelhante ao app anterior, a primeira classe a ser executada é a 'StarterApplication'. Esta classe é responsável por inicializar o serviço do servidor Parse e, para esta conexão, utiliza os mesmos parâmetros utilizados pelo outro app, que são:

- *.applicationId*: indica o ID da aplicação. ID: 'carro1dsauhdiy23h2i3huh'.
- *.clientKey*: indica a chave-cliente da aplicação, e é opcional. Não foi utilizada.

- *.server*: indica a URL da aplicação no servidor. URL: ‘https://carro1.herokuapp.com/parse/’.

Em seguida, a atividade representada pela classe *TrackActivity* é acionada. Nesta, há dois campos de digitação (*EditText*), locais onde o usuário digitará valores para identificar o ônibus em que ele se encontra, e um botão “INICIAR”. Neste botão foi configurado um *setOnClickListener*, função que permite ao app realizar alguma ação caso o botão seja clicado pelo usuário. Neste caso, o clique sinaliza ao app que as informações de identificação já foram digitadas, e o celular já está pronto para ser rastreado. Após o clique, a atividade representada pela classe *TrackerActivity2* é acionada.

*TrackerActivity2*, por sua vez, é responsável por capturar as coordenadas atuais do usuário – e por consequência, do ônibus – por meio de alguns provedores, como GPS, Internet, etc. Essa função é realizada por meio do método *getMyLocation()*, acionado no início da atividade pelo método *onCreate()*. Após a captura das coordenadas, uma requisição é feita ao banco de dados para que as coordenadas referentes ao ônibus em questão – identificado pelas informações digitadas pelo usuário – sejam atualizadas com essas novas informações capturadas. O método *getMyLocation()* também torna esse processo periódico, fazendo com que o app esteja frequentemente atualizando no banco de dados a informação acerca da localização atual do usuário. As atualizações periódicas são requisitadas por meio da função *requestLocationUpdates*, e executadas por meio do método *onLocationChanged()*. Há também nessa atividade um botão “PARAR”, o qual também tem a funcionalidade *setOnClickListener* configurada. Neste caso, após o clique o usuário, o rastreamento é bloqueado, as atualizações ao banco de dados interrompidas e o app retorna à página inicial.

Em seguida, representado pela Figura 2.5, há o esquemático que representa a estrutura de classes e o funcionamento do aplicativo em questão. A primeira classe acionada é a *StarterApplication*, de forma semelhante ao app LocalizaÔnibus. Em seguida, a classe principal, a *TrackerActivity*, é executada. Por fim, há o acionamento da classe *TrackerActivity2*. É importante ressaltar que as flechas são de ida e volta, pois há a possibilidade de abortar o funcionamento da atividade *TrackerActivity2* e se retornar à *TrackerActivity*. Além disso, há no esquemático os números 1 e 2, que representam o fluxo padrão de funcionamento do aplicativo.

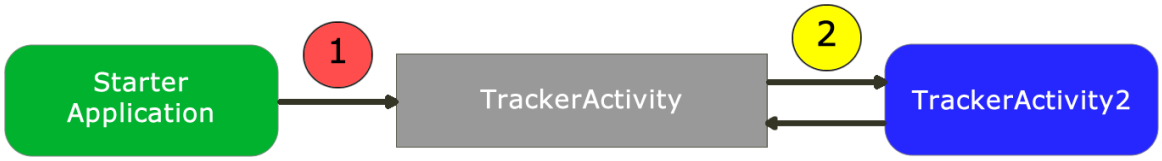


Figura 2.5: Estrutura de classes e funcionamento do Aplicativo Rastreador.

## 2.6. Banco de Dados

Conforme explicado no tópico 2.4., o servidor usado foi o Parse Server – utilizando como estrutura a plataforma do Heroku, e o banco de dados usado foi o MongoDB – utilizando a plataforma do mLab. Para um gerenciamento melhor e mais amigável ao usuário da estrutura criada, foi o utilizado o Parse *Dashboard*. Este permite um gerenciamento eficaz dos aplicativos e dos bancos de dados criados para cada um deles, possibilitando a adição ou alteração de informações no banco de dados de forma simples e rápida.

Em seguida, a Figura 2.6 exibe a página inicial do Parse *Dashboard*. Nesta página, há a lista de aplicativos que estão hospedados no Parse Server. No caso deste projeto, há somente um aplicativo criado, no qual é armazenado o banco de dados com o qual os dois aplicativos desenvolvidos interagem, seja inserindo ou atualizando informações, ou consultando informações já armazenadas.

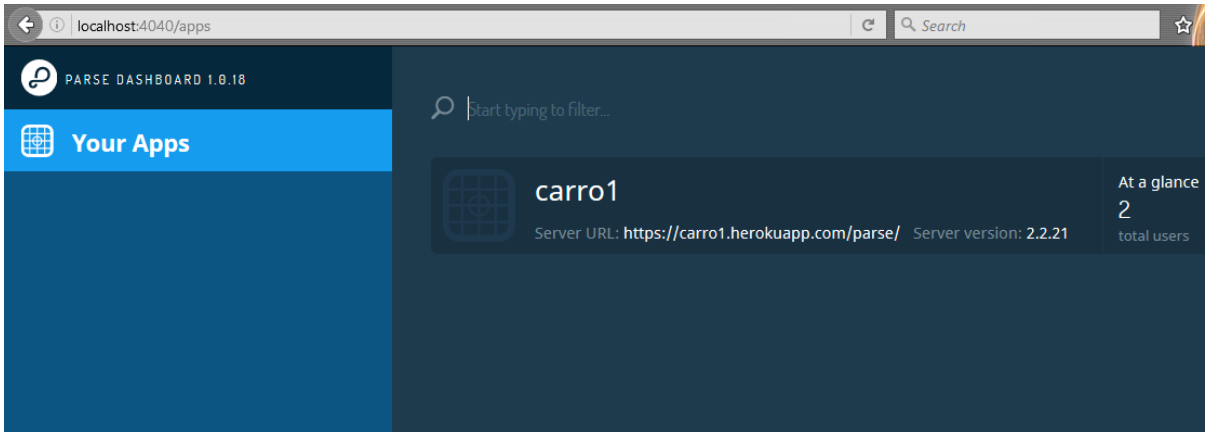


Figura 2.6: Parse Dashboard Apps.

Ao se clicar na opção do aplicativo, o *Dashboard* exibe algumas opções de infomações armazenadas no banco de dados. Para o projeto em questão, foi criado um banco de dados específico, este denominado “Onibus”. Este *id* é utilizado nas solicitações enviadas ao *Parse* pelo aplicativo. Neste banco de dados, cada linha representa um ônibus diferente, e há três colunas bastante importantes, são elas: “Longitude” e “Latitude” e

“LineID”, representando, respectivamente, a longitude do ônibus, a latitude do ônibus e a linha para qual o ônibus presta serviços. A coluna “OnibusID” pode ser usada no caso de diferenciação entre dois ônibus diferentes mas que atendem à mesma linha. A Figura 2.7 exibe a página em questão com maiores detalhes.

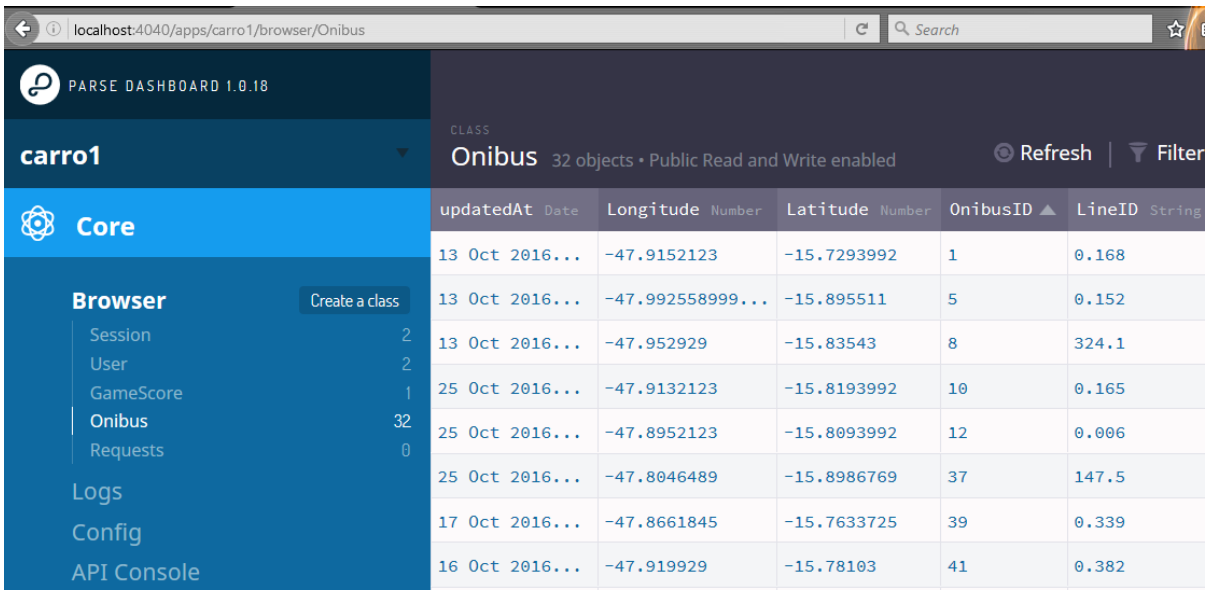


Figura 2.7: Parse Dashboard banco de dados.

Após a explicação com detalhes do ambiente de programação – plataforma escolhida, principais funções utilizadas, estrutura de classes, métodos e do banco de dados – é possível prosseguir à explicação e à demonstração do funcionamento dos aplicativos. Esta demostração será feita por meio de figuras que retratam o passo a passo do funcionamento do aplicativo.

# Capítulo 3

# Funcionamento dos Aplicativos

## 3.1. Aplicativo LocalizaÔnibus

O aplicativo em questão será utilizado pelo usuário final, e tem por objetivo principal a consulta das melhores rotas para se chegar a um destino final utilizando o transporte público por ônibus. O aplicativo deve ser de fácil e rápido manuseio, e ter uma interface gráfica atrativa ao usuário final. As cores e os ícones utilizados no app foram escolhidos com o objetivo de prover esse design agradável.

A página inicial do aplicativo é a página Mapa, é a que primeiramente é mostrada ao usuário quando este abre o aplicativo. Nesta página, exibida na Figura 3.1, é disponibilizado um mapa centralizado na localização atual do usuário, e que mostra os arredores do local. É possível alterar a área visualizada de acordo com a necessidade ou intenção, sendo necessário apenas toques na tela direcionando o mapa à localização desejada. No canto inferior direito do mapa, há um botão, em tons de azul, para atualizar a localização atual do usuário e centralizar o mapa neste local. No topo da tela, localiza-se a *toolbar*. A primeira opção para o usuário contida na *toolbar* é a abertura da aba lateral, por meio de clique no ícone localizado na extrema esquerda. Esta aba, exibida na Figura 3.2, possibilita algumas opções extras de funcionalidades, que serão detalhadas logo em seguida. Ao centro da *toolbar*, encontra-se o campo de pesquisa, área onde deve ser digitado o endereço do destino ao qual se deseja ir. Por fim, na extrema direita da *toolbar*, há um botão que fornece algumas opções de configurações do aplicativo.


Figura 3.1: Tela Mapa LocalizaÔnibus.

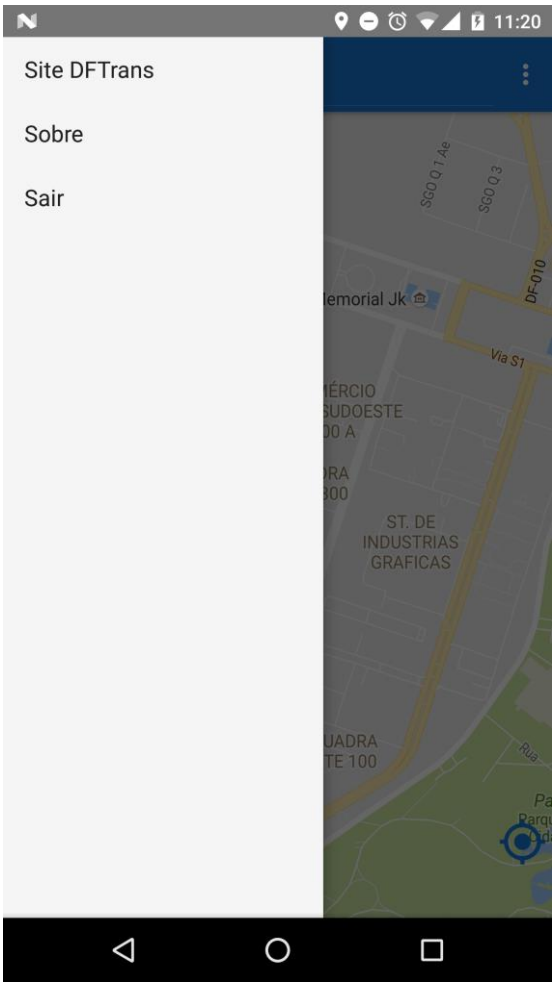


Figura 3.2: Tela Aba LocalizaÔnibus.

Detalhando melhor as funcionalidades da aba lateral, há três opções: “Site DFTrans” – possibilitando o acesso direto ao sistema de consulta no site do DFTrans; “Sobre” – fornecendo um resumo do objetivo do aplicativo e das pessoas envolvidas; “Sair” – possibilitando o fechamento imediato do aplicativo. Há a possibilidade de, futuramente, mais funcionalidades serem incluídas na aba lateral. Nas opções “DFTrans” e “Sair”, um *AlertDialog* é disponibilizado na tela para que o usuário escolha a opção “Sim” ou “Não”, indicando se realmente deseja fazer uso da opção. No caso da opção “Sobre”, há um *AlertDialog* apenas com a opção “OK”, viabilizando o retorno à página anterior e o fechamento do *AlertDialog*.

Em seguida, a Figura 3.3, representando o app após o clique na opção “Site DFTrans”, e a Figura 3.4, exibindo a página *web* externa que é aberta ao se optar pela opção “Sim” no *AlertDialog*.

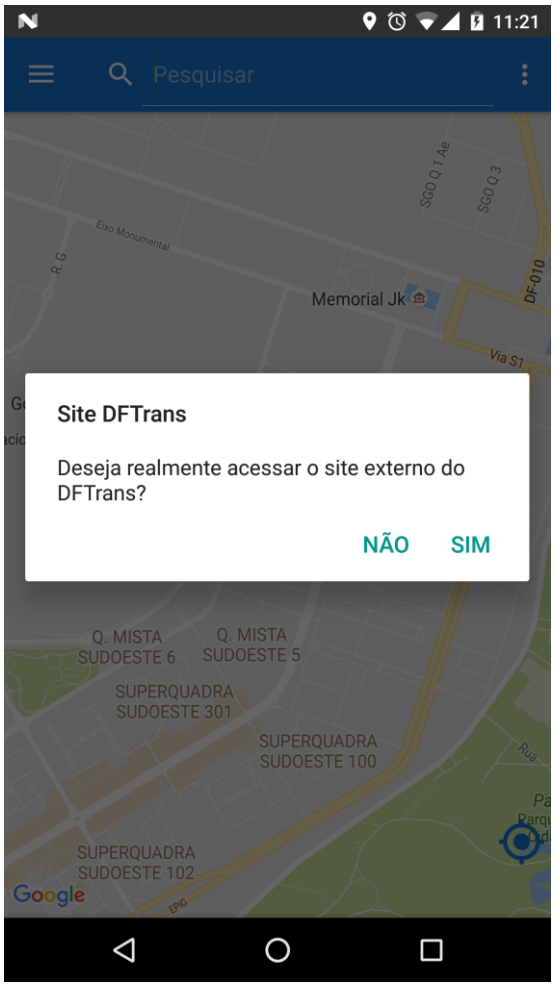


Figura 3.3: Tela Opção DFTrans LocalizaÔnibus.

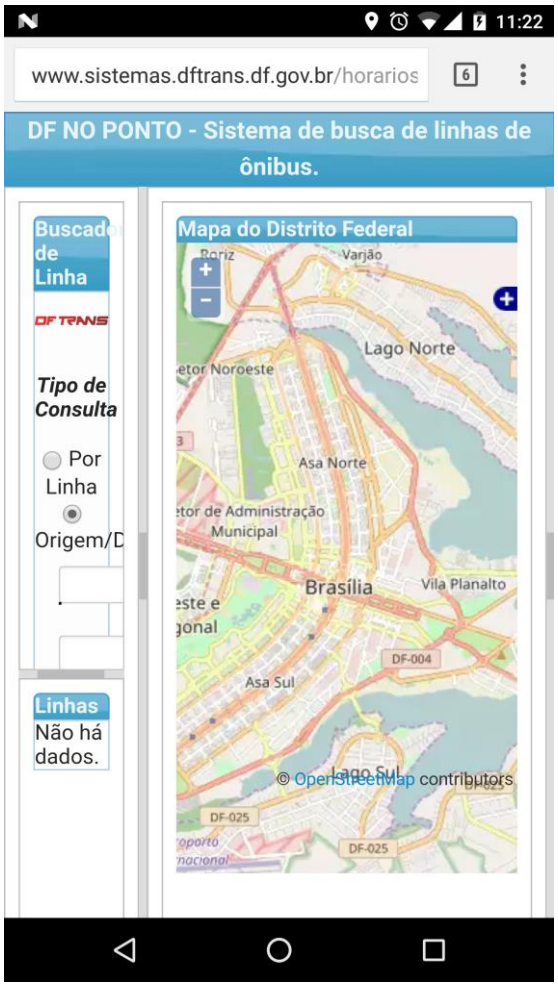


Figura 3.4: Tela Site externo DFTrans.

Em seguida, a Figura 3.5, referente à escolha da opção “Sobre” na aba lateral, e a Figura 3.6, referente à opção “Sair”. Se, na opção “Sair”, a escolha for pelo “Sim”, o aplicativo fecha imediatamente.

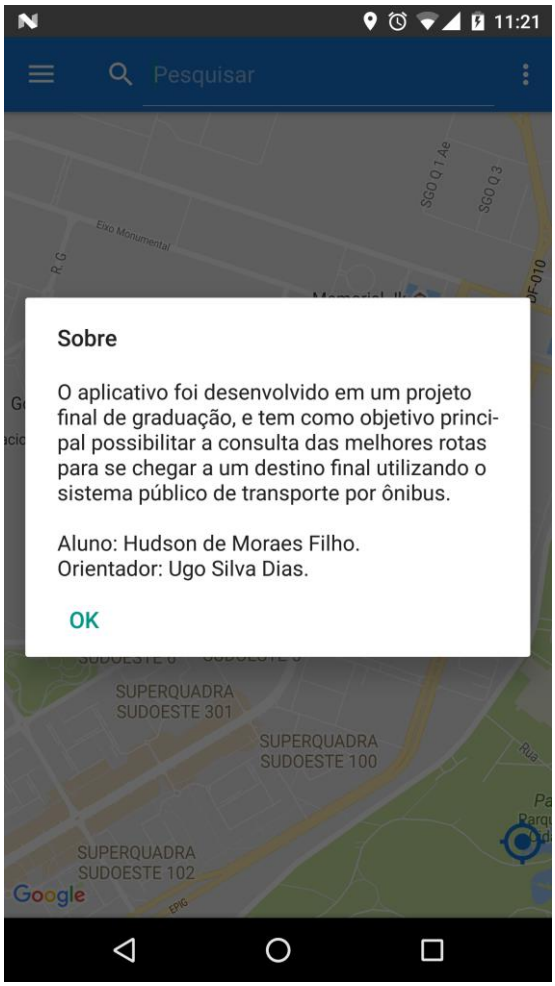


Figura 3.5: Tela opção Sobre LocalizaÔnibus.

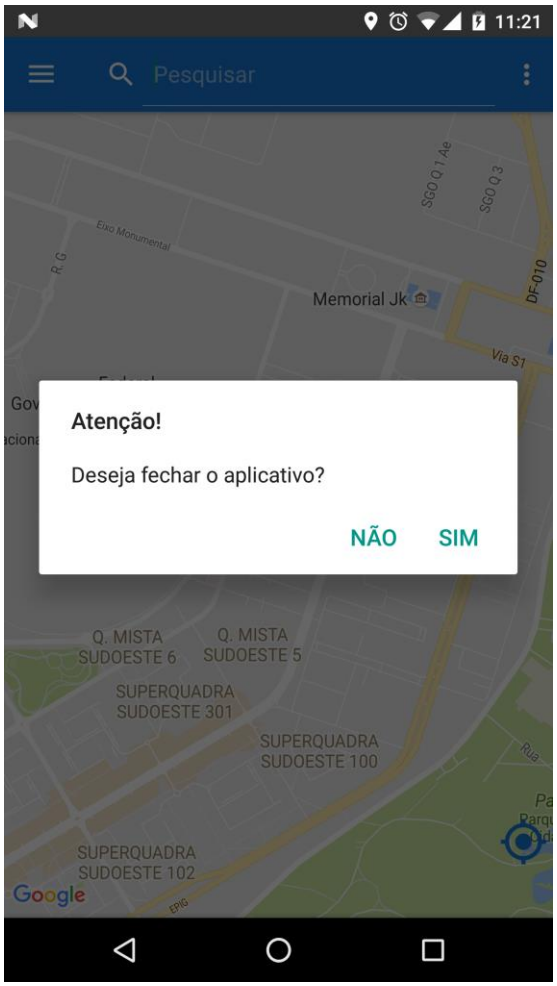


Figura 3.6: Tela opção Sair LocalizaÔnibus.

Utilizando o campo de pesquisa contido na *toolbar*, ao se digitar o endereço de destino, uma lista com opções de locais que se encaixam ao digitado na pesquisa é mostrada, e o usuário deve escolher alguma das opções. A Figura 3.7 retrara essa funcionalidade. Após essa escolha, o aplicativo envia uma requisição HTTPS à *Google Maps Directions API* solicitando rotas possíveis para se chegar ao endereço de destino. Para a criação dessas rotas, o endereço que será utilizado como o de origem será o da localização atual do usuário, e o endereço de destino será o da opção escolhida anteriormente. A resposta da *Google Maps Directions API* é em formato JSON (JavaScript Object Notation), e passa por métodos de análise no aplicativo, com o objetivo de extrair exatamente as informações necessárias para a criação das rotas e para exibição aos usuários. Na requisição HTTPS, quando se indica a necessidade de mais de uma alternativa de rota como resposta, a Google API retorna quatro alternativas no objeto JSON. Após o devido tratamento às informações contidas no JSON de resposta, abre-se no aplicativo uma janela com uma lista de alternativas de rotas que atendem aos critérios

estipulados, em conjunto com detalhes importantes destas. A Figura 3.8 retrata essa janela com as possíveis alternativas a serem escolhidas. Para a simulação, o destino escolhido foi a Universidade de Brasília – UnB, e, a partir da exibição das opções na pesquisa, a primeira opção foi escolhida.


Figura 3.7: Tela Pesquisa LocalizaÔnibus.

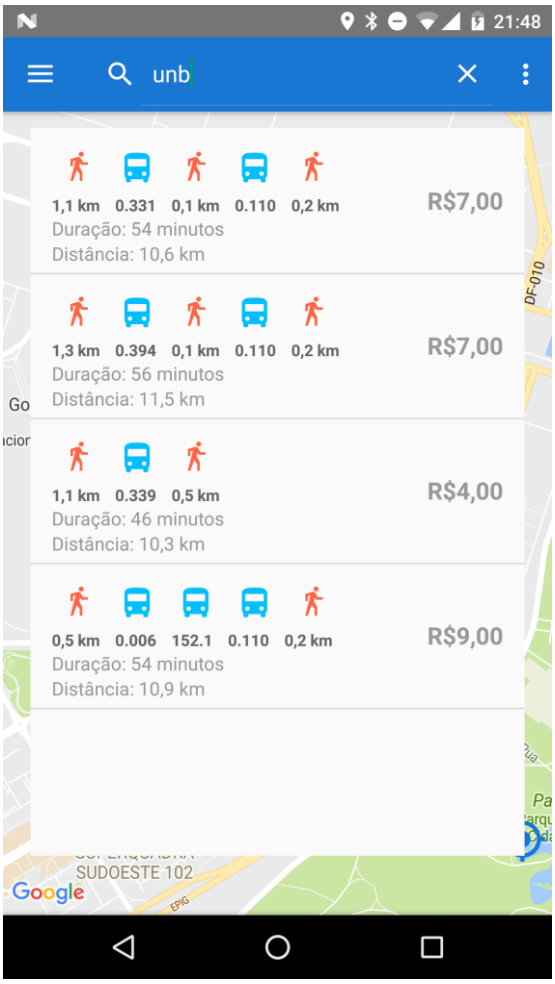


Figura 3.8: Tela Opções LocalizaÔnibus.

Conforme informado anteriormente, na Figura 3.8 com a lista de alternativas de rotas, são exibidos os detalhes que caracterizam cada uma delas. No topo, há um detalhamento, por meio de imagens, da quantidade de trechos de caminhada, e da quantidade de baldeações necessárias para a conclusão do trajeto. A imagem em vermelho representa o trecho de deslocamento do usuário por caminhada, podendo ser da localização atual até o primeiro ponto de ônibus, entre dois pontos de ônibus, ou do último ponto de ônibus até o destino final. A imagem em azul representa o trecho de deslocamento do usuário por ônibus. Neste caso, caso seja necessário o uso de mais de um ônibus até o destino final, e o trecho que inclui o ônibus posterior se inicie no mesmo

ponto em que o último trecho de ônibus terminou, ou seja, sem trechos de caminhada entre os trechos de ônibus diferentes, duas imagens em azul aparecerão de forma seguida.

Abaixo do detalhamento por imagens, há informações referentes ao número das linhas dos ônibus que fazem parte do trajeto, e informações acerca da distância necessária a ser percorrida em cada trecho de caminhada, sendo essas informadas em metros (m) ou quilômetros (km). Abaixo, há a informação da duração total estimada da viagem. Essa informação foi extraída também do objeto JSON de resposta da API, e, portanto, leva em consideração apenas a tabela de itinerários e horários teórica a qual os ônibus deveriam seguir, não havendo conexão com a situação real do trânsito, ou com as localizações reais dos ônibus, e, por consequência, não considerando possíveis atrasos. A estimativa inclui os trechos de caminhada e os trechos de ônibus. Em seguida, há a informação da distância total a ser percorrida no trajeto. Este número também considera todos os trechos da viagem. Por fim, na extrema direita, há informações referentes ao valor total a ser pago no trajeto. Este considera as tarifas necessárias para que seja feito o uso do transporte público, já incluindo as tarifas referentes a todos os ônibus contidos na rota.

Após a análise das opções por parte do usuário, este deve escolher uma delas, clicando sobre a alternativa desejada. Após o clique, a janela se fecha, o aplicativo retorna à página Mapa, e a rota escolhida é desenhada no mapa. A rota inclui os trechos de caminhada e de ônibus. O zoom do mapa é ajustado de forma a incluir todos os pontos contidos no trajeto total. Além da rota, são adicionados alguns marcadores importantes ao mapa. O primeiro deles é o marcador que sinaliza o local de destino, facilitando visualmente ao usuário. Em seguida, são adicionados novos marcadores, estes representando os pontos de ônibus que o usuário utilizará para entrar nos ônibus incluídos na rota. O último ponto de ônibus do trajeto – usado apenas para descer do último ônibus – não é sinalizado com um marcador.

Neste mesmo momento em que a rota é criada, o aplicativo envia requisições ao servidor externo, solicitando ao banco de dados as localizações atualizadas dos ônibus que fazem parte do trajeto. O tempo de resposta do servidor varia de acordo com a conectividade com a Internet, podendo ter duração inferior a 1 segundo, ou durações um pouco maiores, podendo chegar a alguns segundos. Para que o aplicativo não fique congelado na espera da resposta do servidor, antes mesmo que este responda, a rota já é

desenhada no mapa, em conjunto com os dois tipos de marcadores mencionados anteriormente (sinalizando o destino e algumas paradas de ônibus). Após a resposta do servidor, os principais marcadores são adicionados ao mapa, estes representando às localizações atualizadas – em tempo real – dos ônibus contidos no trajeto. Após a adição destes, uma nova correção de zoom do mapa é realizada, agora garantindo a inclusão também desses novos marcadores na tela vista pelo usuário.

No retorno à página Mapa, após a escolha da alternativa de rota, é incluída uma nova ferramenta na página: um painel de deslize localizado na parte inferior da tela, sinalizado por uma seta azul apontando para cima. Este será melhor detalhado posteriormente. Na Figura 3.9 encontra-se o mapa com a rota escolhida desenhada, porém, antes do recebimento da resposta do servidor, e, portanto, ainda sem a adição dos marcadores representando as localizações atualizadas dos ônibus. Na Figura 3.10 há o mapa após a resposta do servidor com essas localizações e após a adição dos marcadores.

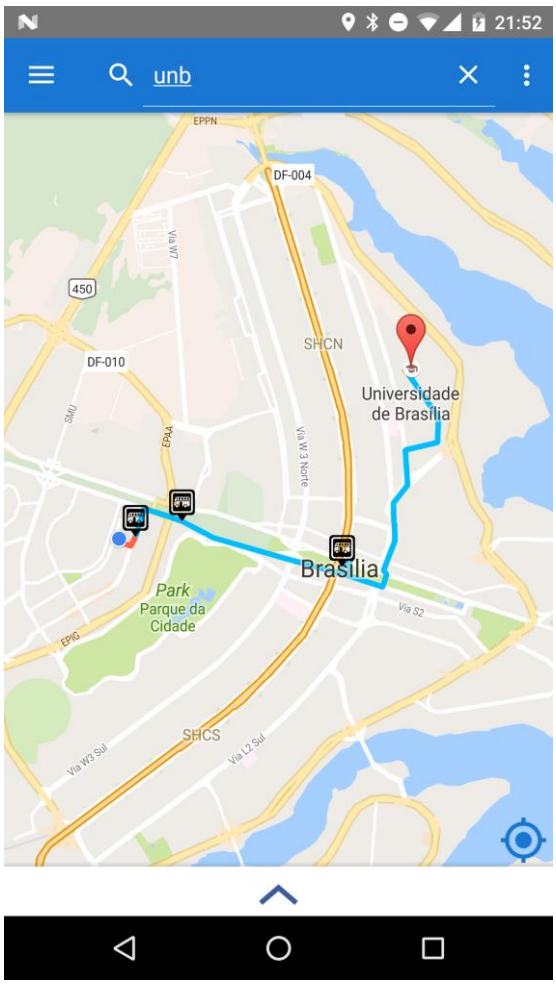


Figura 3.9: Tela Trajeto LocalizaÔnibus.

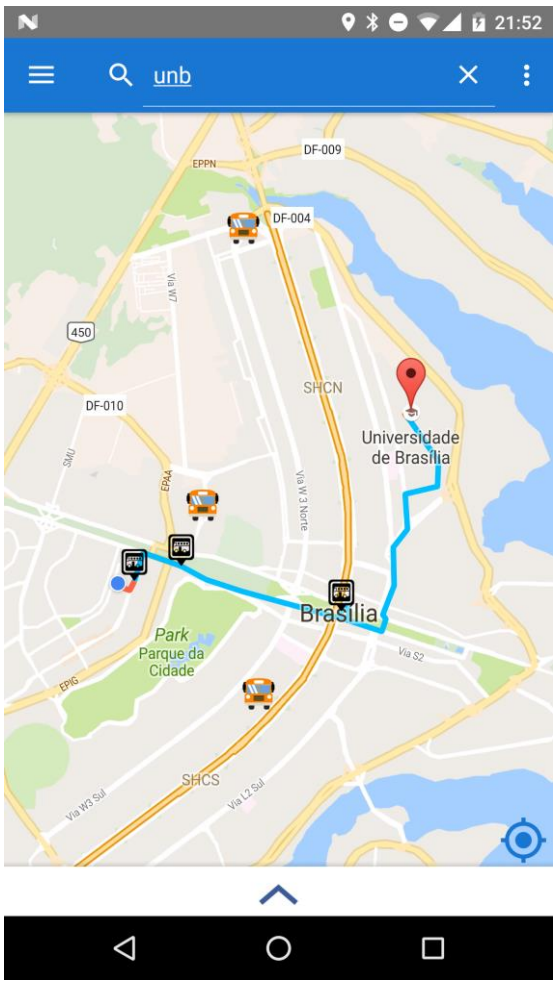


Figura 3.10: Tela Ônibus LocalizaÔnibus.

Além da requisição inicial com o objetivo de adicionar os marcadores dos ônibus ao mapa, o aplicativo envia requisições periodicamente ao servidor e banco de dados, visando sempre manter atualizadas no mapa as localizações desses marcadores que representam os ônibus. A frequência assinalada para essas atualizações foi de 3 (três) segundos. Ao se clicar nestes marcadores, um informativo de qual linha aquele ônibus representa aparece na tela. A Figura 3.11 representa essa funcionalidade. Procedimento semelhante acontece quando se clica no marcador de destino da rota, sendo exibido um informativo que detalha o endereço deste local de destino, retratado na Figura 3.12.

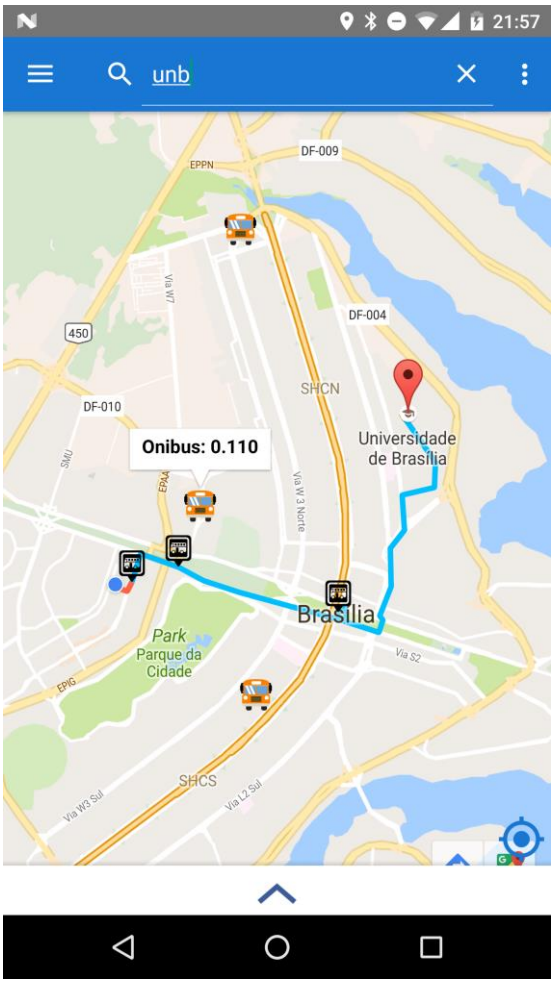


Figura 3.11: Tela Linha LocalizaÔnibus.

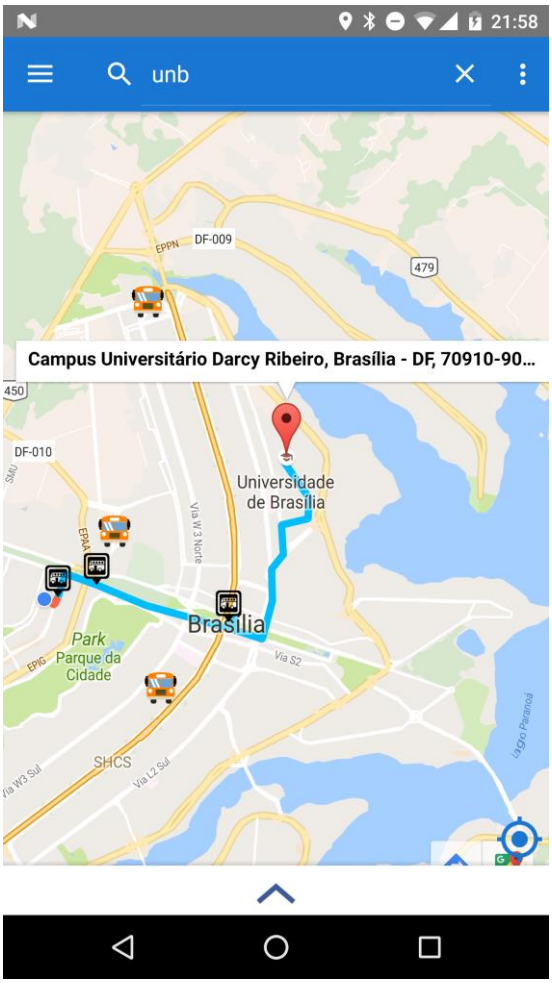


Figura 3.12: Tela Destino LocalizaÔnibus.

No caso descrito pelas figuras, o usuário é sinalizado pelo círculo azul, mais à esquerda da tela. Na rota que ilustra o trajeto, os trechos de caminhada tem cor diferente dos trechos de ônibus. Na parte de caminhada, a cor é o vermelho, e na parte dos ônibus, a cor é o azul.

Na área inferior da tela há o painel de deslize criado após a escolha da rota. O painel surge minimizado, mas pode ser facilmente ampliado, sendo necessário apenas o deslize de baixo para cima na tela. O painel tem dois objetivos principais: disponibilizar as informações acerca da rota escolhida, de forma semelhante à exibida na janela com as alternativas, e, ainda mais importante, informar ao usuário as previsões de chegada dos ônibus às paradas, considerando as localizações dos ônibus atualizadas em tempo real.

Para o cálculo das previsões em tempo real, novas requisições são enviadas à *Google Maps Directions API*, porém, para facilitação dos cálculos, foi considerada a condução normal como forma de transmissão, e não mais transporte público. Outra mudança foi a utilização da localização atualizada do ônibus como ponto de origem da rota, e o ponto de ônibus para o qual este se dirigirá como destino. Do objeto JSON de resposta, após os devidos tratamentos, é extraída a duração do trajeto, e esta é utilizada como estimativa de tempo necessário do ônibus se deslocar até o ponto. Como os ônibus não possuem transmissores instalados, as localizações dos ônibus no banco de dados são apenas simulações, ou seja, não condizem com a situação real. Para o cálculo da estimativa de chegada do primeiro ônibus do trajeto ao ponto é usada apenas a duração extraída do objeto JSON. Para os demais ônibus contidos na rota (no caso dos trajetos que possuem mais de uma baldeação), a previsão é calculada somando-se as durações – extraídas dos objetos JSON – de cada trajeto ônibus até a parada e as durações dos trechos que deverão ser percorridos pelos ônibus anteriores. Estas previsões são atualizadas periodicamente por meio de novas requisições HTTPS à API. A frequência assinalada para atualização é a mesma utilizada nas consultas às localizações dos ônibus, portanto, 3 (três) segundos.

A Figura 3.13 retrata o detalhe das diferentes cores utilizadas em trechos de natureza distinta, e a Figura 3.14 exibe o painel com o resumo das informações acerca da rota escolhida, e com as previsões em tempo real para a chegada dos ônibus às paradas.

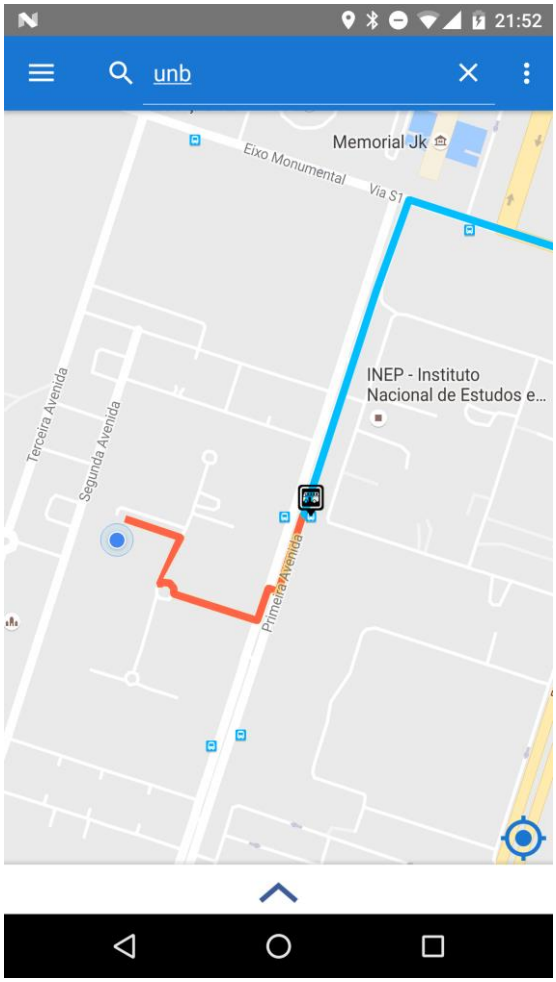


Figura 3.13: Tela Cores LocalizaÔnibus.

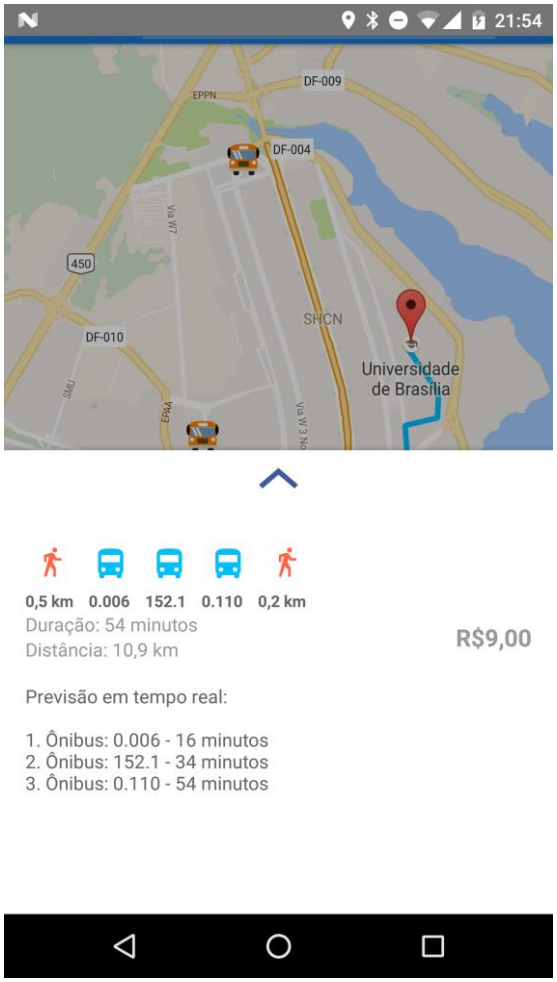


Figura 3.14: Tela Resumo LocalizaÔnibus.

## 3.2. Aplicativo Rastreador

Diferentemente do aplicativo descrito anteriormente, este não deve ser utilizado pelos usuários finais, mas sim pelos motoristas ou cobradores que estiverem em serviço nos ônibus. O objetivo do app é enviar ao servidor e banco de dados periodicamente a localização atual do ônibus em questão, atualizando a informação antiga do banco de dados acerca deste ônibus. Isso permite que o aplicativo para consultar rotas tenha sempre a informação mais atualizada e precisa da localização atual do ônibus. Ambos os aplicativos fazem uso do mesmo banco de dados, tendo um como objetivo apenas a consulta das localizações dos ônibus, e o outro a alimentação do banco de dados com as localizações em tempo real.

A página inicial do app possui dois campos para digitação do usuário, ambas são utilizadas para identificação correta do ônibus em questão. O primeiro campo é destinado à linha a qual o ônibus se refere, e o segundo campo à identificação específica e

única do ônibus. Os dois valores são importantes, pois indicarão ao aplicativo qual o ônibus exatamente que está iniciando o serviço e, portanto, que deve ser rastreado. Ao se clicar nos campos, o teclado se torna disponível. Após a digitação dos valores corretos, deve-se clicar no botão 'INICIAR'. Esse botão dá início à captura periódica da geolocalização do usuário e, consequentemente, da geolocalização do ônibus.

A Figura 3.15 exibe a tela inicial imediatamente após a abertura do aplicativo, e a Figura 3.16 exibe essa mesma tela, porém já com os campos preenchidos com as informações do ônibus.


Figura 3.15: Tela Inicial Rastreador.

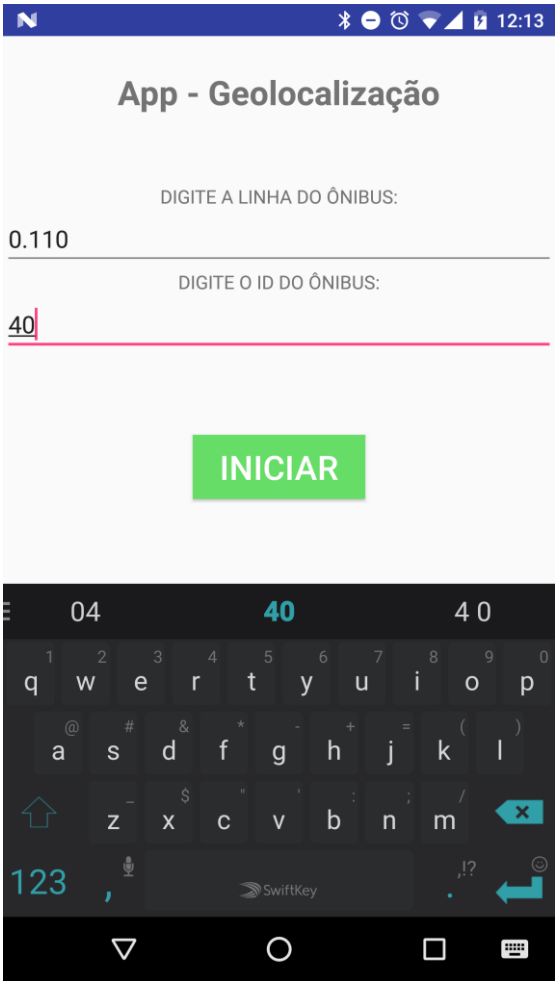


Figura 3.16: Tela Digitação Rastreador.

Após clique no botão 'INICIAR', o aplicativo abre uma nova tela, com novas funcionalidades. Nesta tela, há dois quadros que exibem a latitude e longitude momentânea do usuário, os quais são atualizados periodicamente, e há também um botão "PARAR", cuja função é interromper a captura periódica de geolocalizações e retornar à página inicial. Porém, o principal objetivo desta página é a atualização do banco de dados

externo com as coordenadas capturadas pelo aplicativo. Esta atualização é feita também periodicamente, e é vital para o correto funcionamento do outro aplicativo, o manuseado pelo usuário final. É somente por meio destas coordenadas atualizadas que se faz possível a atualização da localização dos marcadores dos ônibus na página Mapa, e, consequentemente, o acompanhamento do deslocamento do ônibus desejado em tempo real pelo usuário.

Na Figura 3.17, há a tela com as informações preenchidas, sem o teclado disponível. A Figura 3.18 retrata a tela após o clique em "INICIAR".

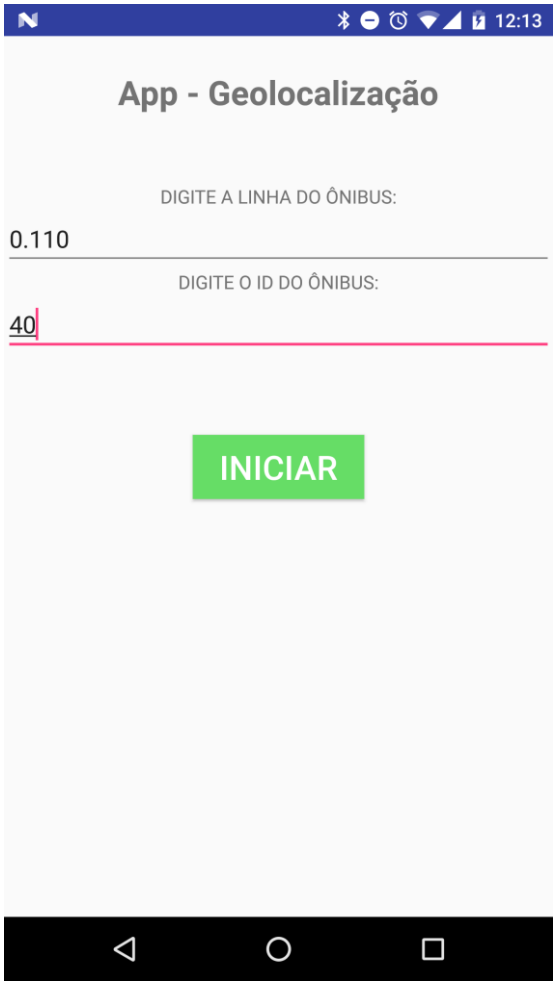


Figura 3.17: Tela Iniciar Rastreador.

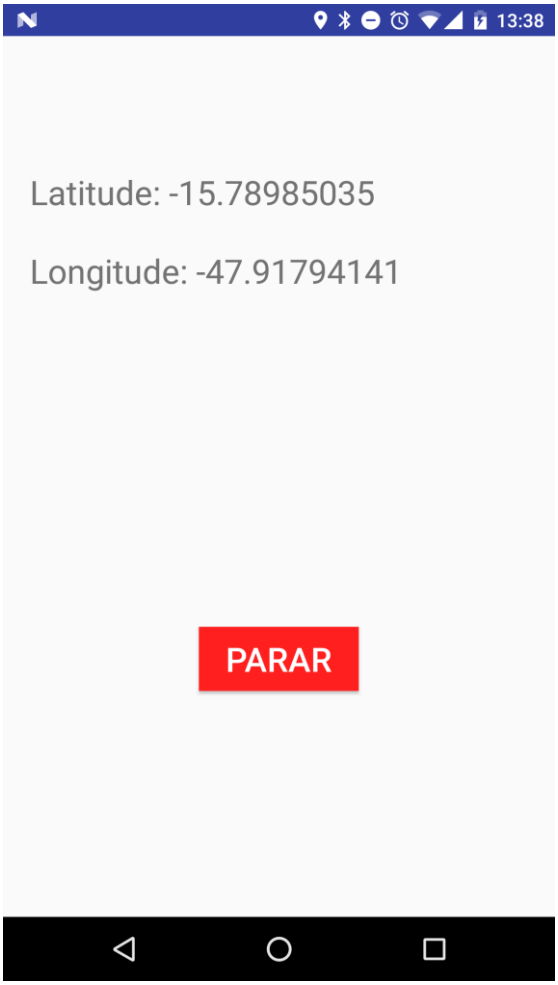


Figura 3.18: Tela Resultado Rastreador.

As ambos os aplicativos – LocalizaÔnibus e Rastreador – atingiram os resultados planejados e funcionam de forma bastante interligada. Há ainda, no entando, enormes possibilidades de melhorias e adição de funcionalidades extras. Estas possibilidades futuras estão melhores descritas no Capítulo 4.

# Capítulo 4

# Conclusões e Trabalhos Futuros

O uso da tecnologia tem muito a agregar aos sistemas de transporte coletivo no Distrito Federal. Soluções simples, já amplamente utilizadas e com vários exemplos concretos de sucesso, gerariam impactos extremamente positivos. Um dos pontos críticos é a não-confiabilidade do sistema de transporte coletivo por ônibus. Em um cenário de diversos problemas, dentre eles: greves de rodoviários, interdições em vias públicas, ônibus quebrados com uma frequência altíssima, o cidadão, elemento mais frágil desta rede, é quem mais é penalizado. Os aplicativos propostos neste projeto visam dar ao cidadão a opção de se planejar melhor no dia-a-dia, e se tornar menos vulnerável ao ainda fraco sistema de transporte coletivo por ônibus no DF.

Para trabalhos futuros, há algumas questões bastante importantes a serem levadas em consideração:

1) Otimização do uso de bateria e de dados móveis pelo aplicativo

Pelo fato de várias funções serem executadas de forma periódica, tais como as atualizações das localizações do usuário e dos ônibus e as frequentes requisições de consulta e escrita no banco de dados, o aplicativo está com uma taxa alta de consumo de bateria e de dados móveis. Por meio de estruturas e métodos mais eficientes, o consumo pode ser reduzido drasticamente, tornando o app mais utilizável.

2) Inclusão de mais informações e funcionalidades

Há algumas informações referentes aos ônibus que podem ser adicionadas futuramente ao aplicativo, tais como: se possuem *wi-fi* e ar-condicionado, se são adaptados a portadores de necessidades especiais, a identificação da placa, o nome da companhia de ônibus, dentre outras. Além disso, há algumas funcionalidades que tornariam o app mais útil e, portanto, mais atrativo, são elas: consulta de rotas se iniciando em outros horários (não apenas no momento atual), consulta dos próximos ônibus que passarão em cada ponto de ônibus, possibilidade de favoritar destinos para

consultas futuras, acompanhamento de notícias, alertas e avisos acerca do sistema público de transporte, etc. A quantidade de possíveis incrementos ao aplicativo é enorme.

3) Contato com os órgãos responsáveis pelo sistema de transporte público

Como explicado anteriormente, para o funcionamento ótimo do aplicativo, há a necessidade deste ter as informações atualizadas das localizações dos ônibus, para que, assim, as estimativas de chegada destes possam ser calculadas de forma correta. Já há, no DF, alguns ônibus com tecnologias de envio da localização atual instaladas, e alguns que ainda demandam a instalação da tecnologia. Para estes que já estão sendo rastreados, há a possibilidade de obter a autorização para acesso a essas informações de localização, sendo necessário, para isso, negociações com os órgãos responsáveis pelo sistema de transporte público. Para trabalhos e aprimoramentos futuros, esse contato visando acordos com essas entidades é essencial.

Apesar das grandes possibilidades de melhorias, os aplicativos propostos atingiram os objetivos planejados, e, com os devidos acordos com as instituições responsáveis pelo sistema de transporte público por ônibus, são capazes de prover uma boa opção de planejamento dos deslocamentos diários aos usuários desse referido sistema na região do Distrito Federal.

# Referências Bibliográficas


[1] CADEOONIBUS. Disponível em: http://www.cadeoonibus.com.br/CoO/SiteV2. Acesso em: 05/10/2016.

[2] CITTAMOBI. Disponível em: http://www.cittamobi.com.br/. Acesso em: 05/10/2016.

[3] DEVCENTER.HEROKU. Heroku Command Line. Disponível em: https://devcenter.heroku.com/articles/heroku-command-line#download-and-install. Acesso em: 11/09/2016.

[4] DEVELOPERS.GOOGLE. A Google Maps Directions API. Disponível em: https://developers.google.com/maps/documentation/directions/intro?hl=pt-br. Acesso em: 14/06/2016.

[5] DEVELOPERS.GOOGLE. Google Places API: Getting Started. Disponível em: https://developers.google.com/places/android-api/start. Acesso em: 21/06/2016.

[6] DIWAKAR, Sapan. Recycler View Item Click Listener. Disponível em: http://sapandiwakar.in/recycler-view-item-click-handler/. Acesso em: 20/09/2016.

[7] GITHUB. AndroidSlidingUpPanel. Disponível em: https://github.com/umano/AndroidSlidingUpPanel. Acesso em: 11/10/2016.

[8] GRIFFITHS, Dawn; GRIFFITHS, David. Head First Android Development: A Brain-Friendly Guide. Sebastopol: O'Reilly, 2015.

[9] KUMAR, Sandeep; QADEER, Mohammed; GUPTA, Archana. Location Based Services using Android. 2010. 9 p. Aligarh Muslim University, Aligarh, Índia.

[10] LACKER, Kevin. Moving On. Disponível em: http://blog.parse.com/announcements/moving-on/. Acesso em: 12/09/2016.

[11] MAROTTO, Fosco. What is Parse Server? Disponível em: http://blog.parse.com/announcements/what-is-parse-server/. Acesso em: 11/09/2016.

[12] MATHEW, George. Adding Google Places Autocomplete API as custom suggestions in Android Search Dialog. Disponível em: http://wptrafficanalyzer.in/blog/adding-google-places-autocomplete-api-as-custom-suggestions-in-android-search-dialog/. Acesso em: 25/08/2016.

[13] MATHEW, George. Drawing driving route directions between two locations using Google Directions in Google Map Android API V2. Disponível em: http://wptrafficanalyzer.in/blog/drawing-driving-route-directions-between-two-locations-using-google-directions-in-google-map-android-api-v2/. Acesso em: 27/06/2016.

[14] MCLAUGHLIN, Brett D. et al. Head First Object Oriented Analysis & Design: A Brain Friendly Guide to OOA&D. Sebastopol: O'Reilly, 2006.

[15] MECHLING, Gautier. SimpleDividerItemDecoration. Disponível em: https://github.com/Nilhcem/bblfr-android/blob/master/app/src/main/java/com/nilhcem/bblfr/core/ui/recyclerview/SimpleDividerItemDecoration.java. Acesso em: 29/09/2016.

[16] MOOVIT. Disponível em: http://moovitapp.com/pt-br/. Acesso em: 04/10/2016.

[17] PARSE. Migrating an Existing Parse App. Disponível em: https://parse.com/migration. Acesso em: 08/09/2016.

[18] PARSEPLATFORM. Android Guide. Disponível em: http://parseplatform.github.io/docs/android/guide/#updating-objects. Acesso em: 15/09/2016.

[19] QUEIRÓS, Ricardo. JSON on Mobile: is there an Efficient Parser? 2014. 8 p. CRACS & INESC-Porto LA & DI-ESEIG/IPP, Porto, Portugal.

[20] SINGHAL, Manav; SHUKLA, Anupam. Implementation of Location based Services in Android using GPS and Web Services. 2012. 6 p. ABV-Indian Institute of Information Technology and Management, Gwalior, India.

[21] WHITING, Timothy. Setting up your own Parse Server. Disponível em: https://medium.com/@timothy_whiting/setting-up-your-own-parse-server-568ee921333a#.jw0tuuo88. Acesso em: 11/09/2016.