Universidade Federal Fluminense
Bacharelado em Ciência da Computação

Felipe Lugão Eccard

Chega+: um sistema para o acompanhamento de transportes públicos

Rio das Ostras-RJ
2017

FELIPE LUGÃO ECCARD

CHEGA+: UM SISTEMA PARA O ACOMPANHAMENTO DE TRANSPORTES PÚBLICOS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação do Instituto de Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Orientador: Prof. MSc. Eduardo Marques

Rios das Ostras-RJ

2017

FELIPE LUGÃO ECCARD

CHEGA+: UM SISTEMA PARA O ACOMPANHAMENTO DE TRANSPORTES PÚBLICOS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação do Instituto de Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Aprovada em 27 de julho de 2017.

BANCA EXAMINADORA

Prof. MSc. Eduardo Marques - Orientador
Universidade Federal Fluminense

Prof. DSc. Carlos Bazilio - Avaliador
Universidade Federal Fluminense

Prof. DSc. Dalessandro Soares Vianna - Avaliador
Universidade Federal Fluminense

Rio das Ostras-RJ
2017

Aos meus pais.

Agradecimentos

À minha família, por sempre estimular o estudo.

À minha namorada (Pérola Lannes) por todo apoio, companherismo e compreensão.

Ao meu amigo Lucas Tito pela motivação e apoio.

Ao professor orientador Eduardo Marques pela sua transparência e agilidade.

Resumo

A proposta deste projeto é o desenvolvimento de um sistema capaz de acompanhar a localização de transportes públicos, criando um facilitador para a locomoção urbana. Os dados necessários para o cumprimento deste objetivo são basicamente a localização e a velocidade dos veículos. Este projeto busca utilizar o *smartphone* como fonte de captação e visualização desses dados. Esta alternativa implica em um baixo custo de implantação da proposta. Testes do sistema evidenciaram que a arquitetura e tecnologias utilizadas foram traduzidas em um sistema fluido e eficiente. Dessa forma tornando o sistema uma alternativa para a melhoria do serviço público de acesso à informação.

Palavras-chave: celular; serviço baseado em localização; Android; *smartphone*; tecnologia móvel.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

GRASP - General Responsibility Assignment Software Patterns (or Principles) UC - Use Cases

API - Application Program Interface

REST - Representational State Transfer

HTTP - Hypertext Transfer Protocol

SDK - Software development kitIDE - Integrated Development Environment

MVP - Model-View-Presenter

CGI - Common Gateway Interface

UI - User interface

GPS - Global Position System

JPEG - Joint Photographics Experts Group

XML - Extensible Markup Language

JSON - JavaScript Object Notation

SGBD - Sistema de gerenciamento de banco de dados

PC - Personal Computer

Sumário

Introdução

Um dos maiores desafios da vida contemporânea nas cidades é a mobilidade urbana. Geralmente, a maioria da população depende do transporte público, que, com o crescimento acentuado da malha urbana, tende a se tornar cada vez mais complexo e diversificado. Desta forma, é cada vez mais difícil que um indivíduo tenha pleno conhecimento das linhas de ônibus, vans, barcas, metrôs e trens disponíveis para seu deslocamento, seja ele cotidiano ou esporádico.

Dentre as soluções para este relativo desconhecimento da oferta de transporte, está a própria comunicação interpessoal. Mais recentemente, tem-se o aporte do Google Maps [24], e de alguns aplicativos que se debruçam sobre esta questão. Entretanto, a maior parte dos usuários percebe certa falta de confiança nas informações apresentadas (que nem sempre exploram todas as opções possíveis de logística) e a ausência de mecanismos facilitadores como a geolocalização dos transportes.

Com o objetivo de fornecer uma infra estrutura para a implementação de sistemas de localização em tempo real de transportes, prefeituras vêm colocando exigências de equipamento GPS no edital de licitação de transportes coletivos, e criando centros de controle para distribuição de dados [25]. Essa infraestrutura é utilizada por aplicações como Vá de Ônibus [26].

Este trabalho tem como foco o estudo e o desenvolvimento de uma aplicação que se mostre como uma boa alternativa para solucionar esta demanda por informação no segmento dos transportes. A solução é baseada em sistemas de localização para auxiliar a rotina dos usuários do transporte público. Desta forma, um usuário, através da aplicação em seu dispositivo móvel, pode obter informação sobre uma linha que passa em determinado ponto de ônibus; buscar por lugares para verificar a localização do ponto mais próximo e as linhas que passam por ele, e ver o trajeto das linhas e os pontos pelos quais passam cada linha.

Uma das justificativas para a adoção deste tipo de solução seriam os potenciais benefícios para entidades públicas, por exemplo a melhoria na qualidade dos serviços de transportes oferecidos pelo governo à população, auxiliando o cidadão no entendimento das informações do serviço prestado através de uma interface inteligível. O segundo motivo para escolha é a aplicabilidade desse tipo de serviço a dispositivos mobiles, onde estão disponíveis recursos como: grandes telas (fazendo o mapa ser facilmente visível) e tecnologia de posicionamento (GPS) [27].

O presente trabalho está dividido, além desta introdução, nos seguintes capítulos: referencial teórico, definição e modelagem da solução, arquitetura do sistema, implementação do sistema e testes e conclusão.

No capítulo de referencial teórico são apresentadas conceitos relacionados com o sistema proposto. Também as principais características, funcionalidades e problemas de alguns aplicativos que possuem um grande volume de utilização e abordam o mesmo tema que o trabalho proposto. Estas aplicações voltam a ter nossa atenção no Capítulo 5, no qual são efetivamente comparados com o Chega+.

No segundo capítulo, sobre a definição e a modelagem da solução, descrevendo a terminologia utilizada ao longo do trabalho, as condições requisitadas de funcionamento da aplicação e especificamos as funcionalidades desenvolvidas no trabalho, incluindo a elaboração de casos de uso.

No terceiro capítulo, é abordado a arquitetura do sistema. Nele, é descrita a maneira em que o sistema é dividido e são formadas suas conexões entre componentes do software, de forma a garantir sua manutenibilidade e escalabilidade.

No quarto capítulo, é abordado a implementação do sistema: os padrões de arquitetura utilizados, *frameworks* adotados e os componentes de software desenvolvidos.

Já no último capítulo é descrito testes, críticas e méritos aplicados no nosso aplicativo em comparação com Vá de Ônibus e Moovit. Concluindo com um diagnóstico da relevância do Chega+ frente aos aplicativos com propostas semelhantes.

Capítulo 1

Referencial teórico

Neste capítulo serão explorados tecnologias, trabalhos e conceitos realacionados ao sistema proposto.

1.1 Android

A seguir serão apresentados alguns conceitos da plataforma necessários para o desenvolvimento de uma aplicação Android.

Os pilares de uma aplicação Android são: o objeto *Application* e os componentes *Activity, Service, BroadcastReceiver e ContentProvider*. Componentes são iniciados através de gatilhos, realizados por meio de *Intent*(em português intenção). Uma *Intent* especifica uma ação sobre a qual um receptor deve agir. A seguir os componentes são listados e explicados.

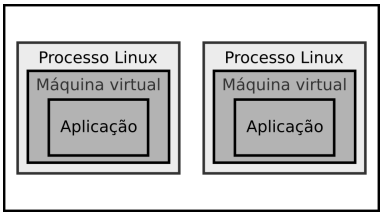
- ***Application*** **:** a representação da execução de uma aplicação no Java é o objeto android.app.Application, este tem o inicio do ciclo de vida no momento em que a aplicação iniciada, e o final no momento em que a mesma é finalizada.
- ***Activity*** **:** representa a interface com o usuário, mostra informações, controla os *inputs* do usuário. Contém componentes de interface: botões, texto, imagens e assim por diante.
- ***Service*** **:** representa um serviço que pode ser executado em segundo plano sem a interação direta com o usuário.
- ***ContentProvider*** **:** uma forma de compartilhar dados entre aplicações é utilizando o componente *ContentProvider*. este pode prover acesso a qualquer dado, mas é comumente utilizado para prover acesso a banco de dados local.
- ***BroadcastReceiver*** **:** este componente desempenha uma função bem restrita: escutar intenções. Essas podem ser disparadas pelo sistema operacional, também pela própria ou outras aplicações. Um *BroadcastReceiver* deve ser registrado dinamicamente no momento em que deve-se iniciar a escuta das intenções, e removido no momento em que deve-se parar as mesmas. É possível escutar intenções específicas através de associação de um filtro no momento em que o componente é registrado.

1.1.1 Execução

O Android é um sistema multiusuário e multitarefa. Nele podem ser executados múltiplas aplicações ao mesmo tempo, podendo ser alternadas rapidamente. O *kernel* do Linux que gerencia as múltiplas tarefas e a execução da aplicação é baseada no processo Linux. Umas das grandes alterações no sistema Android em relação ao Linux foi trazer o conceito de isolamento do usuário para a aplicação.

O isolamento de usuário no Linux é feito da seguinte forma: cada usuário recebe um identificador único que é gerenciado pelo sistema operacional. Esse garante que cada usuário tenha acesso a recursos privados protegidos por permissão, e nenhum usuário (exceto o root) pode acessar recursos privados de outro usuário. Essa configuração é criada para isolar os usuários. No sistema operacional Android, cada pacote de aplicação possui um identificador único. Desta forma, uma aplicação no Android corresponde a um usuário único no Linux e não pode acessar recursos de outras aplicações. A maneira com que o Android isola é criando para cada processo uma máquina virtual (Dalvik ou ART) [28]. A Figura 1.1 ilustra o isolamento de aplicações.

Figura 1.1: Aplicações sendo executadas em diferentes processos e máquinas virtuais.



1.2 JSON

Para compatibilidade *cross-platform*, a troca de mensagens entre componentes clientes e servidores é realizada em texto. Esse pode ter vários formatos diferentes, sendo o XML e JSON os mais comuns. Nesse trabalho é utilizado o formato JSON.

JSON é uma forma de escrever objetos em linguagem JavaScript. Apesar de parecer uma notação específica da linguagem JavaSript, é suportado pela maioria das linguagens de programação dos dias de hoje. Os pontos fortes do JSON é ser um formato simples para representação de dados e não utilizar muito espaço de armazenamento comparado com o formato XML. Desta forma a utilização do JSON traz duas grandes vantagens: é pequeno para ser transmitido através da rede e leve para ser decodificado, exigindo baixo processamento e conexão não necessariamente rápida [4]. A seguir segue um exemplo de um dado, e sua representação em JSON:

- mensagem
  - conteúdo
    - Olá galera
  - autor

        * lugão

Em JSON, esse dado deve ser representado da seguinte forma:

```
{"mensagem": {
        "conteúdo : "Olá galera",
        "autor" : "lugão"
        }
}
```

Listing 1.1: Dado representado em notação JSON

Para representar dados estruturados como propriedades de objetos (como no exemplo anterior), são utilizados chaves para indicar um novo nível de profundidade na estrutura do arquivo. Os pares de chave e valor são separados por dois pontos, e cada registro de um mesmo nível é separado por vírgula.

Se parte do dado é representado por um valor escalar, o mesmo é representado de forma limpa. Exemplo:

- peça
    - nome
        * chave de fenda
    - quantidade
        * 5

O JSON que representa o dado deve ser o seguinte:

```
{"peça":{
        "nome": "chave de fenda",
        "quantidade" : 5
        }
}
```

Uma lista de itens é representada de maneira simples. Um exemplo de uma lista de linguagens de programação:

- Pascal
- C
- C++
- Java
- Kotlin

Em JSON a lista é representado de maneira simples:

```
1 ["Pacal", "C", "C++", "Java, "Kotlin"]
```

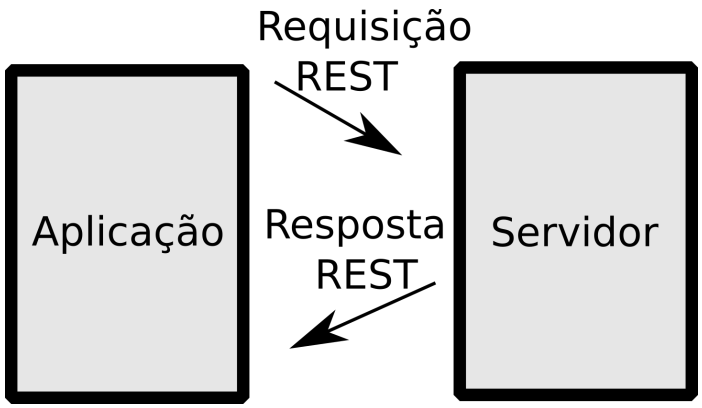
Caso a lista acima for o valor de uma propriedade, deve ser utilizado simultaneamente chaves e colchetes para representar o dado em formato JSON.

```
1 {"linguagens": ["Pacal", "C", "C++", "Java, "Kotlin"]}
```

1.3 REST

REST é um estilo de arquitetura de software para hipermídia distribuída como a World Wide Web. Como a arquitetura define, este estilo consiste em clientes e servidores. A seguir, há uma figura detalhando a comunicação.

Figura 1.2: Aplicação e servidor trocando dados usando REST.



Como se pode observar na Figura 1.2, a aplicação inicia a requisição ao servidor; o servidor processa a requisição e retorna a resposta apropriada. A seguir são apresentados os princípios do *REST* que envolvem o protocolo HTTP e URI [12]:

- **Tudo é um recurso :** todo tipo de dado disponível em um formato na internet, pode ser descrito por um um tipo de conteúdo. Exe: uma imagem JPEG e documento texto pode ser descrito respectivamente pelos tipos de conteúdo: image/jpeg e text/xml .
- **Cada recurso é identificado por um identificador único :** como a internet contém diversos recursos, cada um deve poder ser acessado por um identificador único. O identificador pode ser auto descritivo, sendo facilmente lido por um ser humano. Exe:
  - http://www.uff.com.br/arquivos/monografia/2016/
  - http://www.uff.com.br/imagens/formatura/2016
  - http://www.uff.com.br/documentos/formandos/2016?format=xml
- **Utilize os métodos HTTP padrões:** o protocolo HTTTP nativo define oito ações que também são conhecidos como verbos e métodos.

1. *GET*
2. *POST*
3. *PUT*
4. *DELETE*
5. *HEAD*
6. *OPTIONS*
7. *TRACE*
8. *CONNECT*

Os quatro primeiros métodos provêm um conjunto básico de funcionalidade: criação, leitura, atualização e deleção [13]. Estes devem ser utilizados como na tabela a seguir:

Tabela 1.1: Métodos *HTTP* [3].

Método *HTTP*	**Ação**	**Código de status de resposta**
GET	Requisita um recurso existente	”200 *OK*”se o recurso existir, ”404 *Not Found*”se o recurso não existir, e ”500 *Internal Server Error*” para outros erros.
PUT	Cria ou atualiza um recurso	”201 *CREATED*”se um novo recurso for criado, ”200 *OK*”se atualizado, e ”500 *Internal Server Error*” para outros erros.
POST	Atualiza um recurso existente	”200 *OK*”se o recurso foi atualizado com sucesso, ”404 *Not Found*”se a ser atualizado não existir, e,”500 *Internal Server Error*” para outros erros.
DELETE	Deleta um recurso	”200 *OK*”se o recurso foi deletado com sucesso, ”404 *Not Found*”se o recurso a ser deletado não existe, e,”500 *Internal Server Error*” para outros erros.

- **Recursos podem ter múltiplas representações :** um recurso deve ser representado em um formato diferente do qual é armazenado. Logo, ele pode ser requisitado e postado em diferentes representações. A seguir segue exemplos da criação de um recurso utilizando representação em formatos XML e JSON:

  – **XML :**

```
POST /estoque/produto HTTP/1.1
Content-Type: text/xml
Host: www.venda.com

<?xml version="1.0" encoding="utf-8"?>
```

```
<produto data="22082014">
<Item>Gelatina</Item>
<preço moeda="R$">100</preço>
</produto>

HTTP/1.1 201 Created
Content-Type: text/xml
Location: /estoque/produto
```

– **JSON :**

```
POST /estoque/produto HTTP/1.1
Content-Type: application/json
Host: www.venda.com

{
"produto": {
  "data": ""22082014"",
  "Item": "Gelatina",
  "preço": {
    "moeda": "R$",
    "valor": "100"
  }
 }
}

HTTP/1.1 201 Created
Content-Type: application/json
Location: /estoque/produto
```

- **Comunicação sem estado:** todas as modificações de recursos devem ser realizadas por requisições isoladas utilizando métodos HTTP. Após a execução de uma requisição, o recurso é deixado em um estado final. Isso implica que uma atualização de parte de recurso não é permitida, sempre que é necessário enviar o estado completo do recurso.

A aplicação desses princípios torna uma aplicação HTTP um serviço do tipo *RESTful* [12].

1.4 Docker

Para garantir que o software sempre será executado da mesma maneira, independentemente do seu ambiente, foi utilizado o software Docker. Este produz imagem que contém um sistema de arquivos completo com tudo necessário para executar uma aplicação: código, tempo de execução, ferramentas de sistema, bibliotecas de sistema - qualquer coisa que possa ser instalado em um servidor.

Quando a imagem gerada é colocada em execução um contêiner é criado. Os contêineres executados em uma única máquina compartilham o mesmo kernel do sistema operacional; eles começam instantaneamente e usam menos RAM. As imagens são construídas a partir de sistemas de arquivos em camadas e compartilham arquivos comuns, tornando o uso do disco e downloads de imagens muito mais eficientes. Essas são algumas das grandes vantagens da utilização do Docker. Para mais informações consultar [8].

1.5 Trabalhos Relacionados

Nesta seção, são apresentados sistemas já existentes que possuem funcionalidades semelhantes às do sistema proposto. Atualmente existem muitas aplicações de gerenciamento de frotas públicas que oferecem as funcionalidades: apresentar informações de itinerário; mostrar a localização de pontos e informações sobre de linhas de ponto e mostrar a localização em tempo real da frota.

Foram selecionadas duas aplicações que possuem uma grande popularidade na loja de aplicativos Google Play [16], e que tenham informações de frotas das cidades do Rio de Janeiro (RJ) e Niterói (RJ) para que seja tangível uma comparação com o sistema proposto, visto que testes serão realizados nessas cidades. As aplicações selecionadas foram Vá de Ônibus e Moovit. A seguir detalhes serão levantados sobre as respectivas aplicações.

1.5.1 Vá de Ônibus

Durante a copa do mundo realizada no Rio de Janeiro em 2014, a Fetranspor (Federação das Empresas de Transportes de Passageiros do Estado do Rio de Janeiro) disponibilizou para download o aplicativo Vá de Ônibus [17]. Com ele a empresa expandiu os serviços disponibilizados para o usuário final através de seu website [18]. O aplicativo pode ser utilizado em português, espanhol e inglês.

A tela inicial do aplicativo é ilustrada pela Figura 1.3. Ela contém duas abas : “LINHAS” e “PONTOS”. Quando iniciado é selecionado a aba “LINHAS”. Ela possui um mapa na parte superior com marcadores nas localizações de veículos próximos a localização do usuário. Ao tocar um marcador, é exibido detalhes sobre o veículo que representa (Figura 1.4). Na parte inferior possui uma lista onde os itens da mesma são as linhas que passam no entorno da localização do usuário. Cada item possui um menu que traz a opção de favoritar a linha. Ao clicar em um item da lista o usuário é direcionado para a tela de itinerário (Figura 1.5), onde é disponibilizado a rota da linha e as localizações dos veículos (ônibus).

O divisor entre a parte superior e inferior é um botão que traz a funcionalidade de buscar por linhas e rotas. Ao clicar nesse botão o usuário realiza a entrada de dados com o que deseja buscar e logo depois é direcionado para a tela ilustrada pela Figura 1.6, contendo os resultados em uma lista. Ao tocar em um item da lista, o usuário também é direcionado para a tela de itinerário (Figura 1.5).

Figura 1.3: Aba ”LINHAS“ da tela inicial do aplicativo Vá de Ônibus.

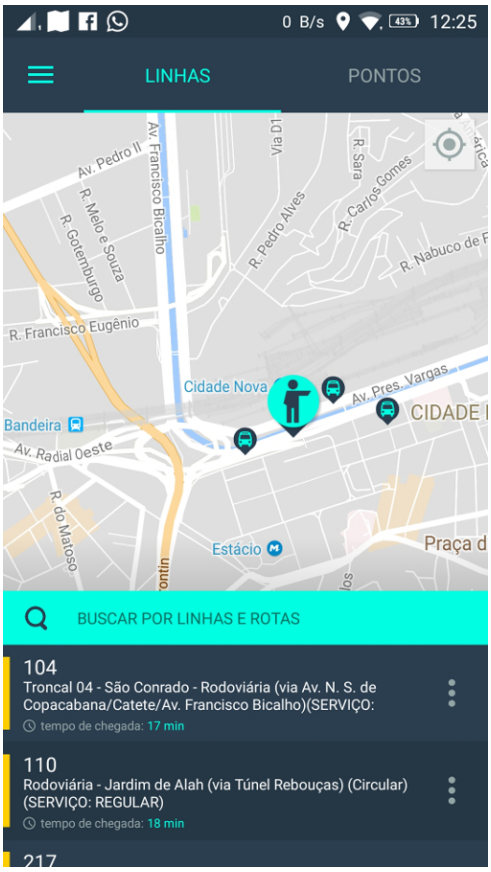


Figura 1.4: Aba ”LINHAS“ da tela inicial com detalhes do veículo do aplicativo Vá de Ônibus.

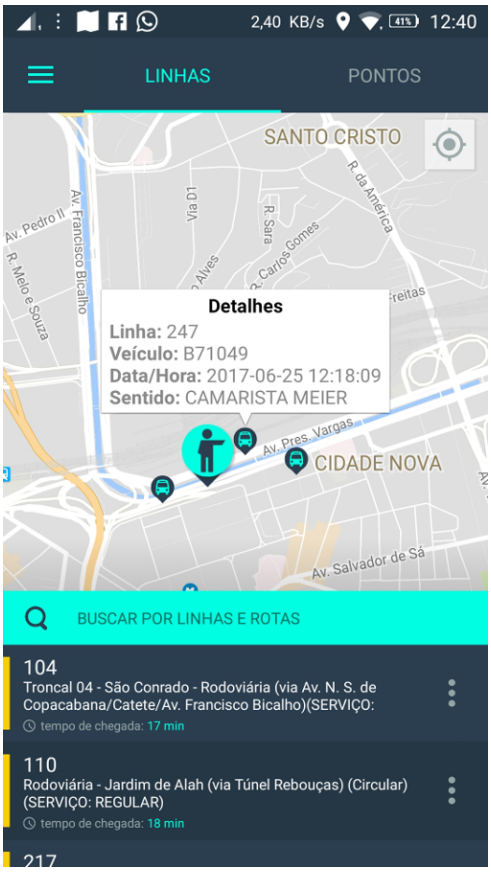


Figura 1.5: Tela de itinerário do aplicativo Vá de Ônibus.

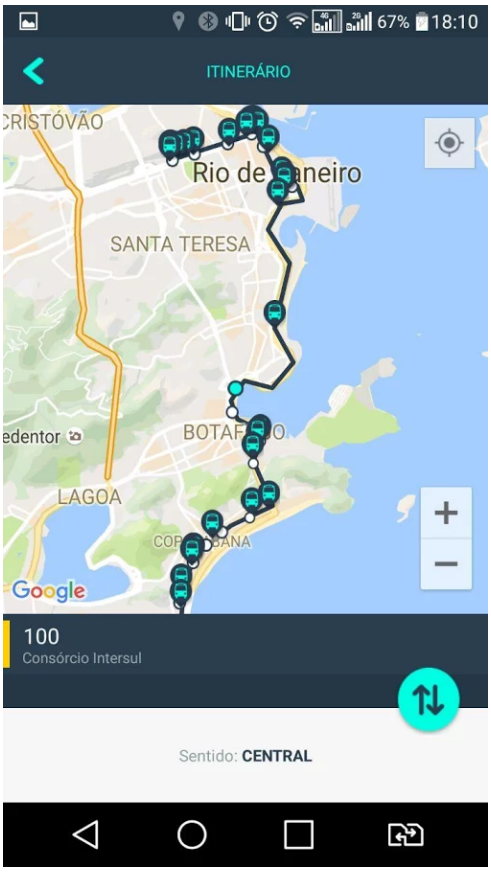


Figura 1.6: Tela de resultados da busca por linhas do aplicativo Vá de Ônibus.

Ao seleciona a aba “PONTOS” na tela inicial, o usuário é direcionado para a tela representada pela Figura 1.7. Ela possui uma interface parecida com a aba pontos. A parte superior possui um mapa com marcadores referentes as localizações de pontos. A parte inferior do mapa possui uma lista com os pontos próximos a localização do usuário. Ao tocar em um item da lista de pontos, o usuário é direcionado para a tela de linhas do ponto (representada pela Figura 1.8) onde em uma lista traz as informações das linhas que passam pelo ponto. Ao tocar em uma item da lista, o usuário é direcionado para a tela de itinerário 1.5.

O menu de navegação do aplicativo é ilustrado pela Figura 1.9. Através dele funcionalidades podem ser acessadas facilmente. Ao tocar: No item ”Onibus próximos”o usuário é direcionado para tela inicial (Figura 1.3); No item “Favoritos” o usuário é direcionado para a tela ilustrada pela Figura 1.10; No item “Avaliar app“ o usuário é direcionado para a página do aplicativo na loja do Google [17].

Figura 1.7: Aba ”PONTOS” da tela inicial do aplicativo Vá de Ônibus.

Figura 1.8: Tela de linhas que passam pelo ponto aplicativo Vá de Ônibus .

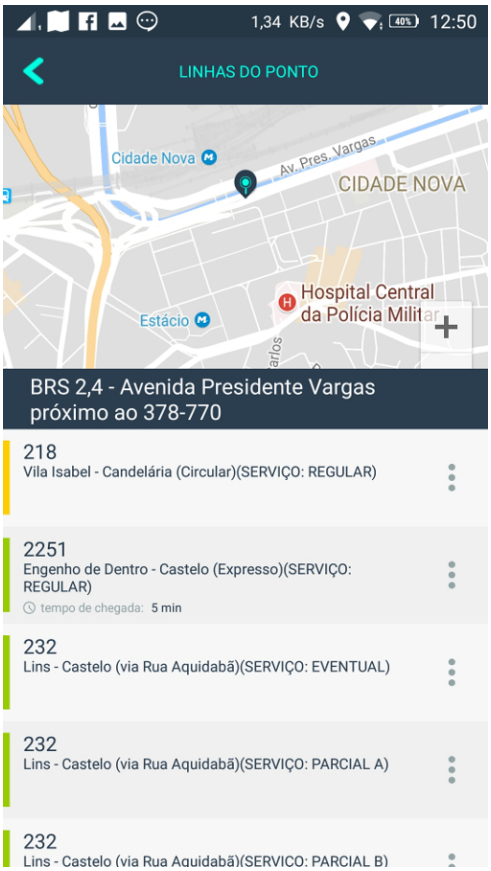


Figura 1.9: Tela de menu de navegação do aplicativo Vá de Ônibus .

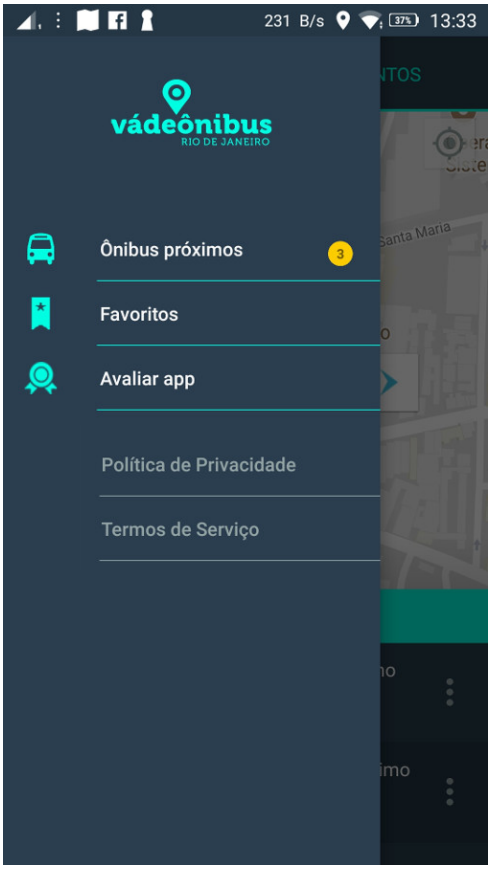
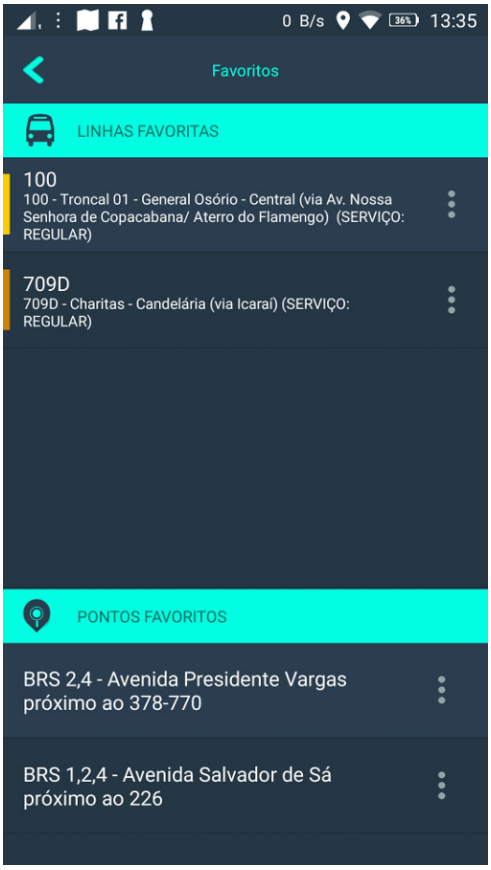


Figura 1.10: Tela de favoritos do aplicativo Vá de Ônibus.

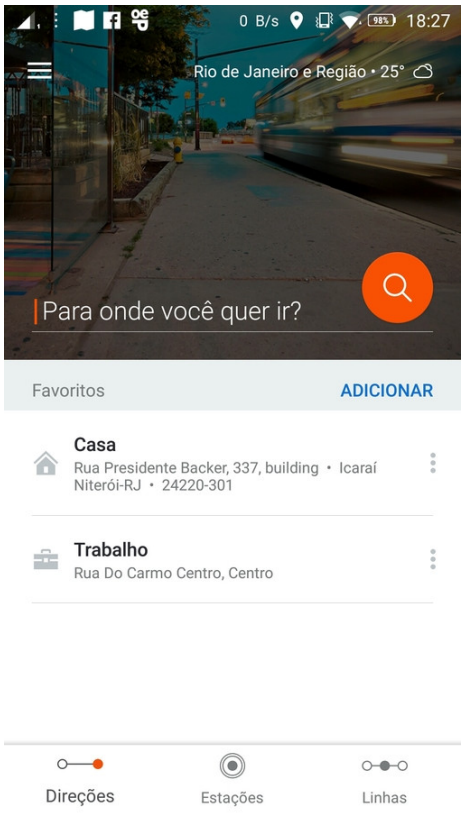


1.5.2 Moovit

Utilizado por mais de 60 milhões de usuário em mais de 1400 cidades, o aplicativo disponibiliza informações sobre rotas de ônibus, trem ou metrô, além de mostrar a tarifa, tempo de viagem e horário de cada linha . Os dados que alimentam o sistema Moovit são oriundos de grandes operadores de transporte: SPTrans (São Paulo Transporte S.A.), DFTrans (Transporte Urbano do Distrito Federal), SMTU (Superintendencia Municipal de Transportes Urbanos- RJ, Fetranspor (Federação das Empresas de Transportes de Passageiros do Estado do Rio de Janeiro), entre outros [20]. A seguir serão explorados detalhes sobre de utilização do aplicativo.

A tela inicial do aplicativo (Figura 1.11) possui três abas: ”Direções“, ”Estações“ e ”Linhas“. Primeiramente será explicado o fluxo da aba ”Direções“, pois é a tela selecionada automaticamente ao abrir o aplicativo. Nessa aba temos algumas informações como a temperatura, locais favoritos e um buscador para informar a destino que deseja chegar. Após a entrada de dados, o usuário é direcionado para a tela de planejador de viagens (figura 1.12). Nessa tela são sugeridas algumas rotas que podem ser utilizadas para chegar ao destino buscado.

Figura 1.11: Tela de inicial do aplicativo Moovit.



Ao selecionar uma rota o usuário é direcionado para a tela de direções (Figuras 1.13, 1.14, 1.15, 1.16). Essa tela detalha os passos necessários pra chegar ao destino buscado. O final da tela possui possui um botão com texto ”Iniciar o Vamos“. Ao clicar nesse botão o usuário é direcionado para a tela Vamos (Figura 1.17). A parte inferior desta, possui um paginador para cada etapa da rota, representada pelas Figuras 1.18, 1.19, 1.20, 1.21.

Ao selecionar a aba ”Direções“ na tela inicial (Figura 1.11), o usuário é direcionado para a tela 1.22. A parte superior desta, possui um mapa com marcadores referentes as localizações de pontos. Ao selecionar um ponto, o usuário é direcionado para a tela de detalhes do ponto representada pela Figura 1.23. Ao realizar o movimento de rolamento na tela, são observados informações sobre as linhas e o tempo de chegada dos veículos (ônibus) que passam pelo ponto (Figura 1.24). Quando uma linha é selecionada, o usuário é direcionado para tela de itinerário da linha representada pela Figura 1.25.

Ao selecionar a aba ”Linhas“ da tela inicial (Figura 1.11), o usuário é direcionado para a tela de busca de linhas representada pela Figura 1.26. Após entrar com a linha buscada os resultados são disponibilizados em formato de uma lista e ao selecionar uma linha, o usuário é direcionado para a tela de itinerário representada pela Figura 1.25.

Figura 1.12: Tela de planejador de viagens do aplicativo Moovit.

Figura 1.13: Tela de direção do aplicativo Moovit.

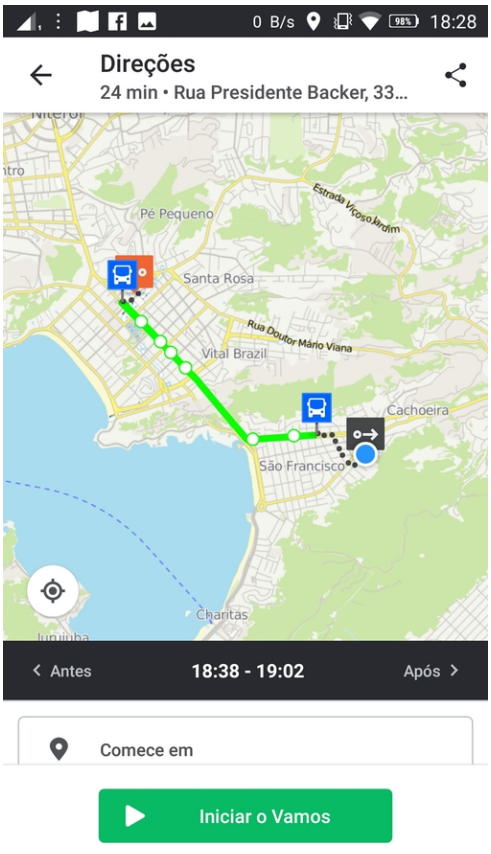


Figura 1.14: Tela de detalhes de direção do aplicativo Moovit.

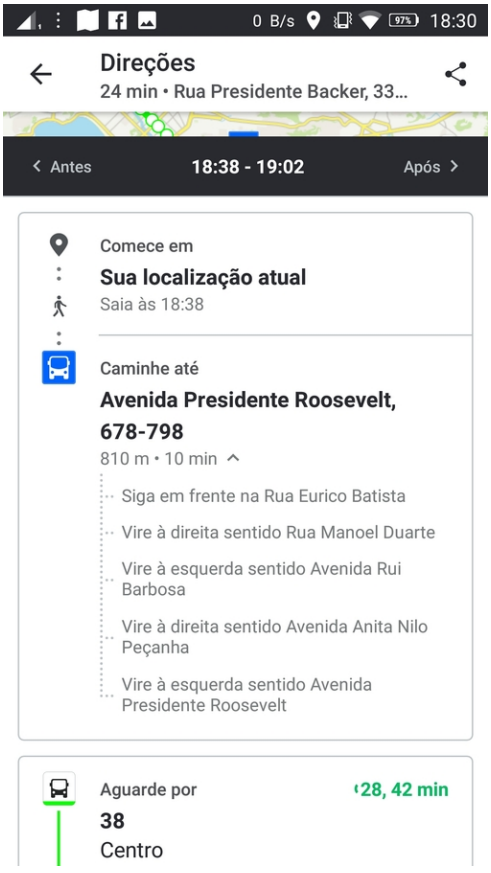


Figura 1.15: Tela de detalhes de direção do aplicativo Moovit.

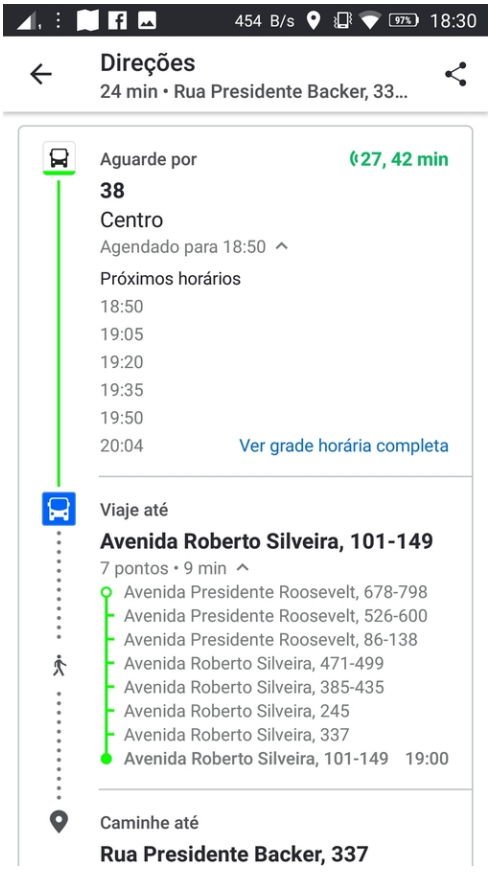


Figura 1.16: Tela de detalhes de direção do aplicativo Moovit.

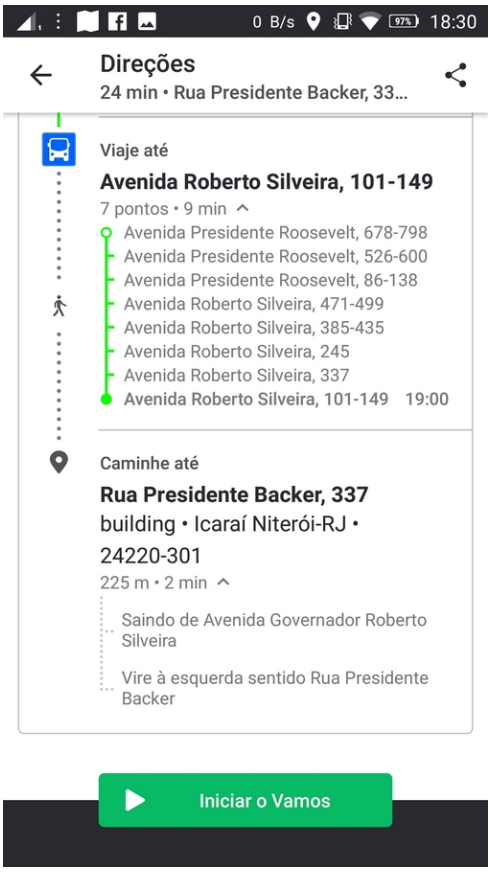


Figura 1.17: Tela vamos, paginador 1 do aplicativo Moovit.

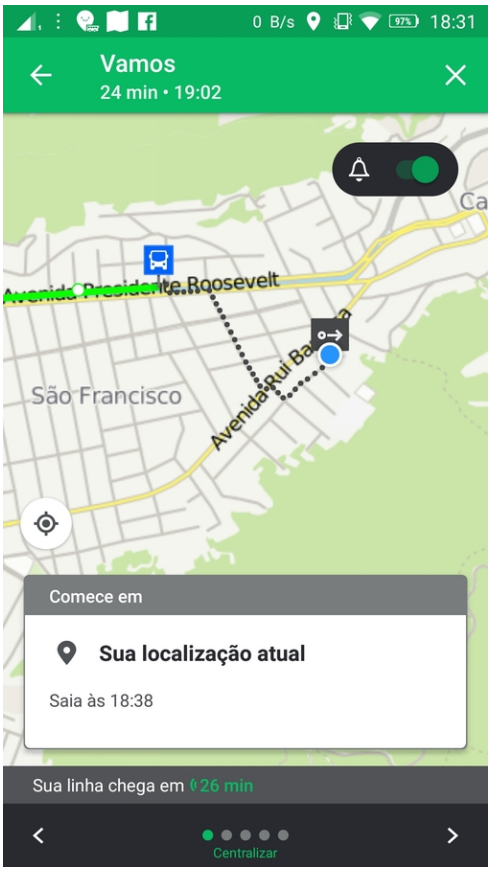


Figura 1.18: Tela vamos, paginador 2 do aplicativo Moovit.

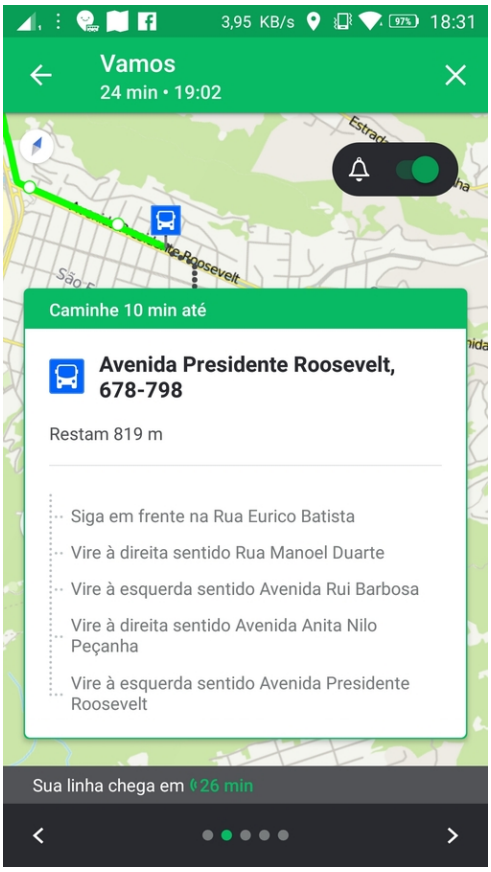


Figura 1.19: Tela vamos, paginador 3 do aplicativo Moovit.

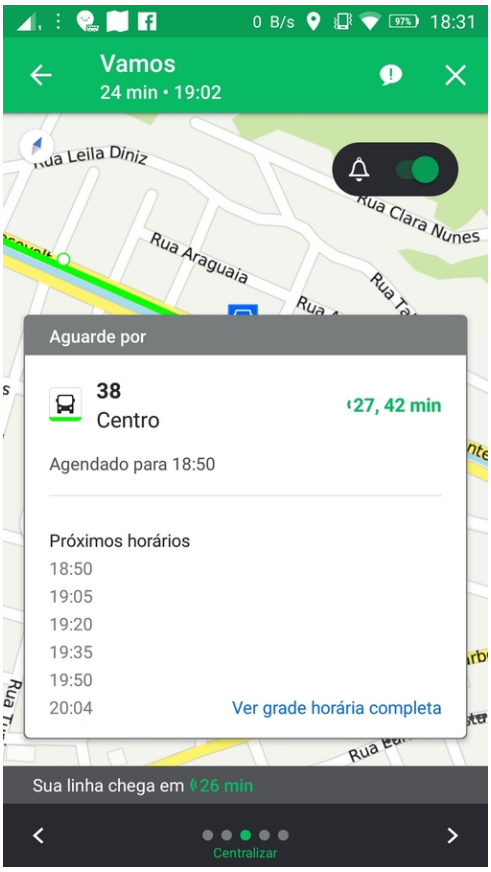


Figura 1.20: Tela vamos, paginador 4 do aplicativo Moovit.

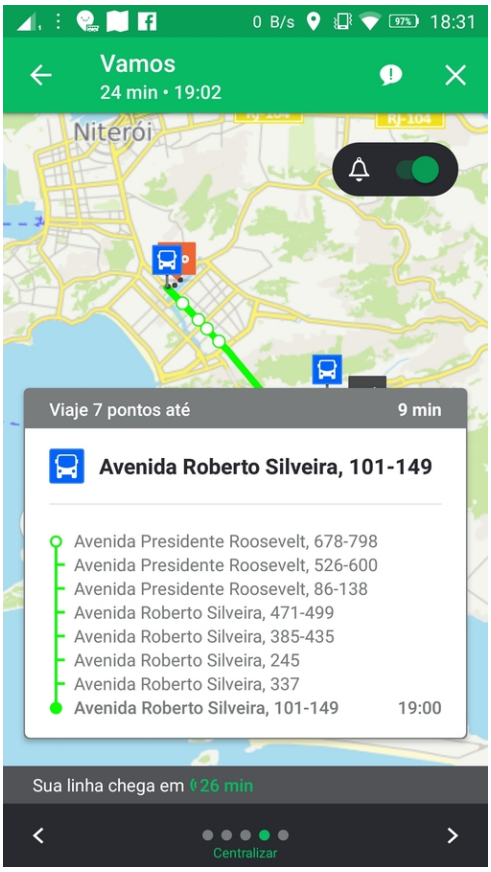


Figura 1.21: Tela vamos, paginador 5 do aplicativo Moovit.

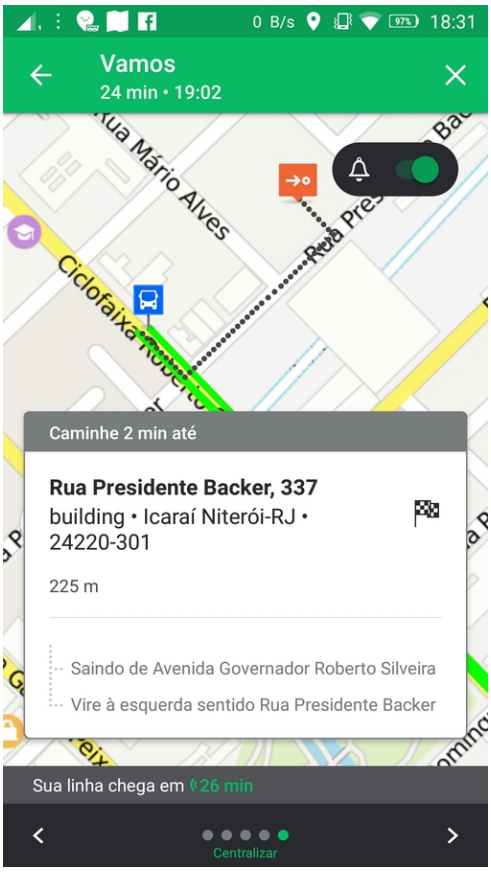


Figura 1.22: Tela de estações do aplicativo Moovit.

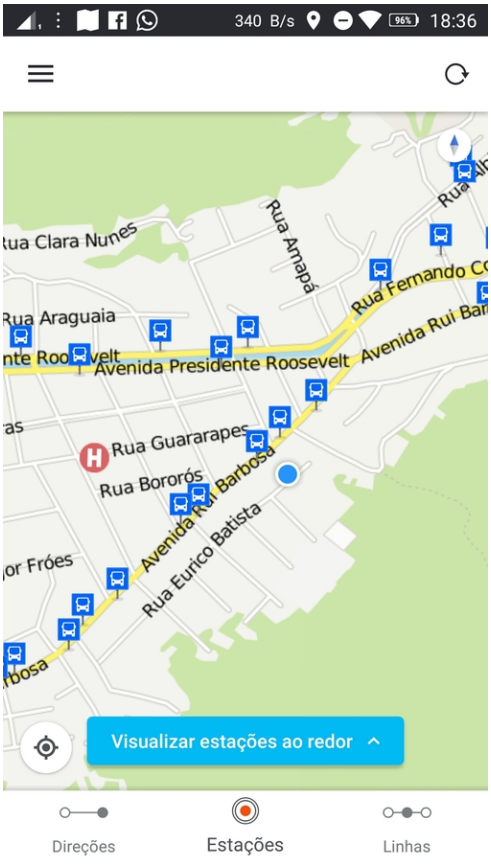


Figura 1.23: Tela ponto do aplicativo Moovit.

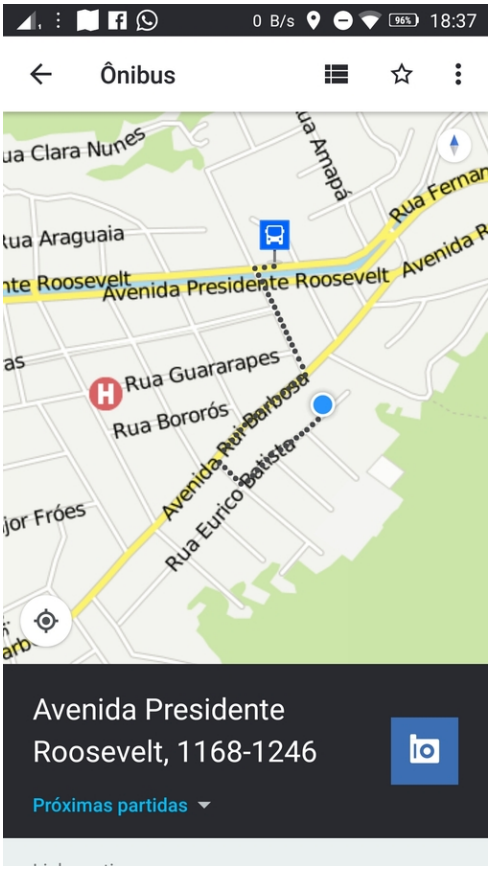


Figura 1.24: Tela de ponto do aplicativo Moovit.

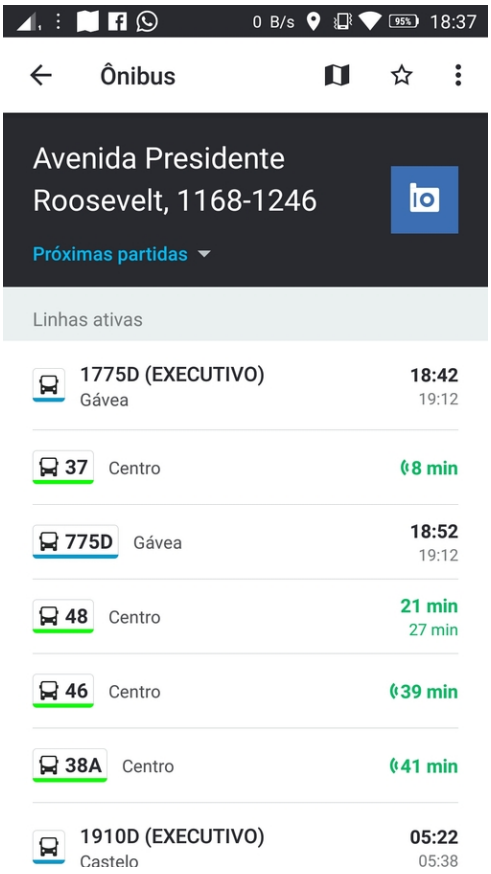


Figura 1.25: Tela itinerário do aplicativo Moovit.

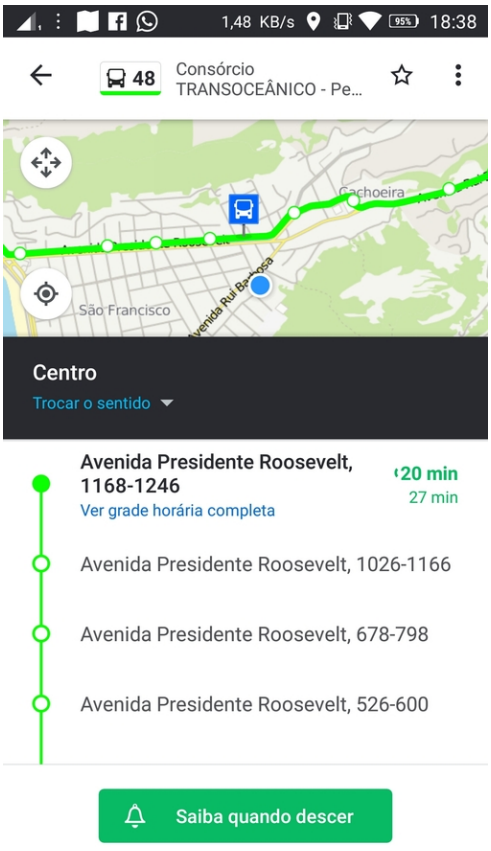
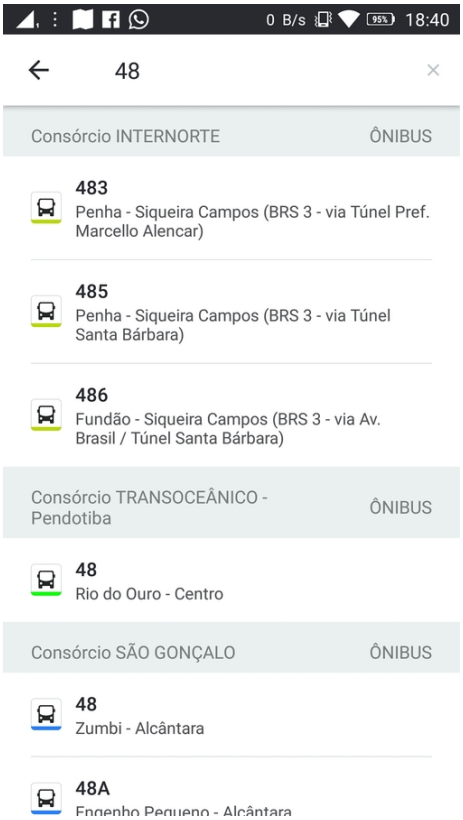


Figura 1.26: Tela de linhas do aplicativo Moovit.



1.5.3 Análise comparativa

A Tabela 1.2 apresenta uma comparação das funcionalidades dos aplicativos apresentados nas seções anteriores.

Dois problemas foram verificados em ambas as aplicações. O primeiro: Nem todas as linhas estão registradas e, mesmo nas linhas inseridas, nem todos os veículos estão catalogados. Uma hipótese para estes problemas seria que as informações que foram inseridas e ou atualizadas após a implantação do sistema não foram refletidas no mesmo. Isto causa uma perda na confiabilidade da solução do ponto de vista do usuário final. Uma forma de diminuir este tipo de problema é ter um canal de atendimento que fique a disposição do usuário, para que ele possa tirar dúvidas e fazer sugestões.

O segundo problema observado em ambos aplicativos, é a não disponibilização de funcionalidades quando o *smartphone* não possui uma conexão com a internet (modo *off-line*). Esse problema poderia ter sido evitado utilizando a persistência de dados não voláteis nas aplicações, como a informação dos itinerários das linhas.

Uma vantagem que o aplicativo Vá de Ônibus tem em relação ao Moovit, é disponibilização da funcionalidade de obter a localização em tempo real de transportes. Uma vantagem que o Moovit possui em relação ao Vá de Ônibus, é a funcionalidade de planejar a viagem. Onde o usuário informa a origem e destino, e o aplicativo fornece algumas opções de transportes.

Tabela 1.2: Comparação de funcionalidade das aplicações Vá de Ônibus e Moovit.

Funcionalidade desejada	**Vá de Ônibus**	**Moovit**
Apresentação do itinerário da linha.	O itinerário de uma linha é apresentado junto com a funcionalidade informação da localização dos veículos da mesma. Isso gera um problema que toda vez que o usuário deseja verificar o itinerário de uma linha, é realizado uma requisição a um serviço Web para obter as localizações dos veículos da linha. Essa funcionalidade é ilustrada pela Figura 1.5.	A funcionalidade é suportada pela aplicação e representada pela Figura 1.25.
Localização dos pontos próximos a localização do usuário	A funcionalidade esta presente na aplicação. É representada pela Figura 1.7.	A funcionalidade esta presente na aplicação. É representada pela Figura 1.22.
Localização dos transportes de uma linha	A funcionalidade é disponibilizada em conjunto com a funcionalidade de apresentação do itinerário de uma linha, por este motivo é impossível verificar localização de transportes de linhas distintas.	A aplicação não suporta essa funcionalidade.
Informação do tempo em que o próximo ônibus passará em um determinado ponto.	A aplicação não suporta essa funcionalidade.	A funcionalidade é suportada pela aplicação e ilustrada pelas Figuras: 1.23 e 1.24.
Planejador de viagens.	A aplicação não suporta essa funcionalidade.	A funcionalidade é suportada pela aplicação e ilustrada pelas Figuras: 1.12, 1.13, 1.14, 1.15, 1.15 e 1.16.

Capítulo 2

Definição e modelagem da solução

Este capítulo tem como objetivo descrever termos usados para a definição do sistema, as precondições para seu funcionamento e a especificação funcional do mesmo.

2.1 Glossário de Termos

Usuário comum: ator do sistema que o utiliza para encontrar a localização corrente de um veículo para saber onde se encontra um veículo de uma determinada linha e obter informações de itinerários de uma linha.
Motorista: usuário que utiliza o sistema para informar dados da linha.
***Device* ou *mobile*:** aparelho celular com sistema operacional Android, independentemente se a versão do sistema é para usuário comum ou um motorista.
Linha: representa o conjunto de pontos por onde um veículo deve passar.
Ponto ou ponto de parada: representa um local específico onde um motorista precisa parar seu veículo.
Sentido: indicativo de origem e destino, que um motorista realizará com seu veículo. Ex: sentido A para B ou sentido B para A.
Local: representa um endereço, seja ele completamente definido ou parcialmente definido.

2.2 Precondições

Para o funcionamento correto dos aplicativos do motorista e do usuário comum, é necessário que ambos os aparelhos que conterão estas aplicações estejam com uma conexão com a internet. As bases de dados que fornecerão informações para o sistema deverão estar preenchidas. Por exemplo, base de dados das empresas de ônibus, com suas respectivas linhas e pontos de parada. Neste trabalho, esta integração com estas informações foi simplificada única e exclusivamente para a prova do conceito de que é possível gerenciar a localização de pessoas, objetos e demais interesses.

2.3 Especificação Funcional

Requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir os seus objetivos [11]. Sendo assim, sua identificação e documentação é fundamental para a elaboração da arquitetura que será descrita no próximo capítulo, juntamente com a implementação do sistema.

2.3.1 Requisitos Funcionais

- **rf-001:** O sistema deve exibir em um mapa a localização atual do usuário comum.
- **rf-002:** O sistema deve permitir que um usuário comum busque por um local, digitando o nome do mesmo e exibir uma lista de locais com nomes semelhantes. Quando o usuário selecionar um dos locais da lista, ele deverá ser exibido no mapa, como também os pontos próximos a este.
- **rf-003:** O sistema deve permitir que um usuário comum visualize pontos de paradas dos veículos em um mapa.
- **rf-004:** Quando um usuário comum selecionar um ponto de parada, no mapa, ele deve ser capaz de visualizar as linhas que passam por aquele ponto de parada.
- **rf-005:** As linhas em exibição devem ser representadas no mapa, interligando seus respectivos pontos de parada.
- **rf-006:** O usuário comum deve poder selecionar uma linha e visualizar os veículos que pertencem à respectiva linha, tal como a localização atual desses veículos.
- **rf-007:** O usuário comum deve ser capaz de visualizar todas as linhas existentes e, quando uma destas for selecionada, ele deverá ser capaz de visualizar quais pontos pertencem aquela linha.
- **rf-008:** Os motoristas devem poder cadastrar seus veículos no sistema, informando: empresa para qual trabalha, linha na qual o veículo pertence (se houver) e o identificador do veículo.
- **rf-009:** O motorista deverá poder informar o sentido que irá realizar.

2.3.2 Requisitos Não Funcionais

Usabilidade

- **rnf-001:** As interações com o mapa (mover e redimensionar) devem poder ser feitas da mesma forma que na solução google maps versão mobile.
- **rnf-002:** A iconografia para usuário comum, motorista e pontos de parada deve ser diferente.

Disponibilidade

- **rnf-004:** O servidor usado para a construção do sistema deve ficar disponível na maior parte do tempo.

Portabilidade

- **rnf-005:** O sistema deve estar disponível para versões de Android 3.1 ou superior.

Interoperabilidade

- **rnf-006:** O sistema deve ser capaz de se comunicar com as APIs do google maps e google cloud.

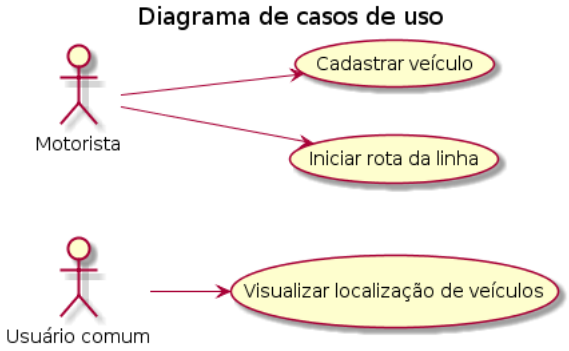
Persistência

- **rnf-007:** No aplicativo do motorista, deverão ser persistidas informações se houve ou não cadastro, bem como dados de configuração (por ex: se o veículo pertence a alguma linha).
- **rnf-008:** Na aplicação do usuário comum, deverão ser persistidas informações se já houve ou não cadastro, bem como a data e hora da última sincronização com o servidor.

2.4 Casos de Uso

No diagrama a seguir foram representados os casos de uso identificados a partir dos requisitos funcionais. Estes casos de uso estão descritos usando o modelo detalhado de descrição e estão no apêndice: Descrição de caso de uso, na sessão A .

Figura 2.1: Diagrama de casos de uso.

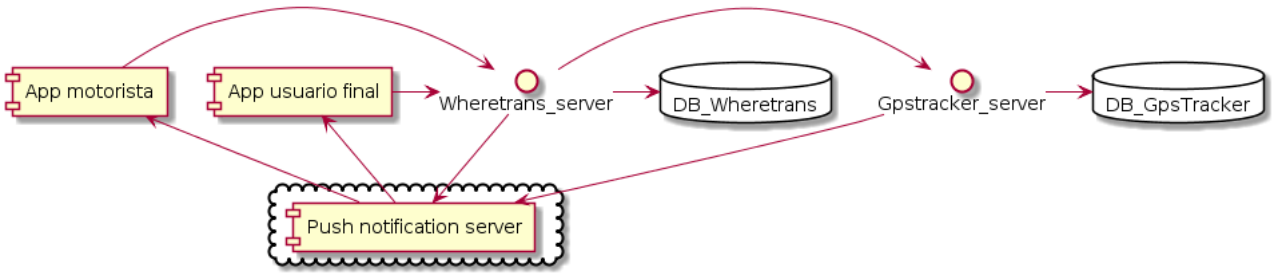


Capítulo 3

Arquitetura do Sistema

Com o intuito de promover a simplificação do desenvolvimento e da manutenção, além de aumentar a reusabilidade, foram utilizados os padrões GRASP, que consistem em um conjunto de práticas para atribuição de responsabilidades a classes e objetos em projetos orientados a objeto. Dessa forma, o sistema foi dividido em sete componentes de software, o que possibilitou, por exemplo, testar cada componente de forma isolada e substituir parte do sistema, facilitando a manutenção, visto que há baixo acoplamento e alta coesão. A seguir, observa-se o diagrama de componentes.

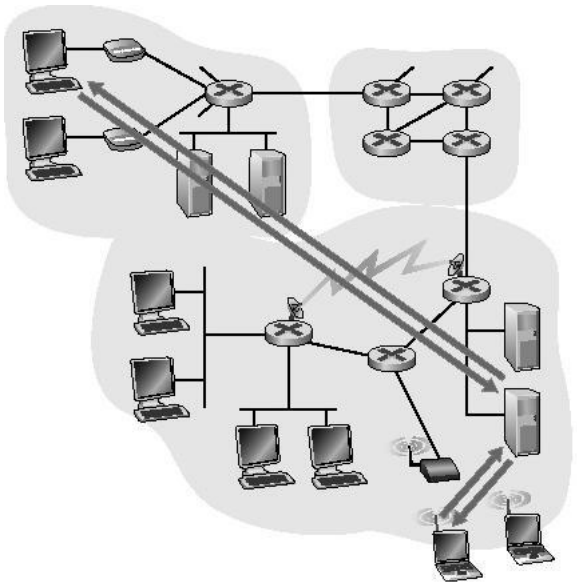
Figura 3.1: Diagrama de componentes.



A arquitetura do sistema foi definida de forma a tornar o *software* funcional independentemente da localização no qual é utilizado. O mesmo oferece um serviço baseado na localização; recurso altamente difundido em aplicações mobile, nas quais temos uma grande limitação de processamento e memória.

O padrão de arquitetura adotado é o clássico cliente-servidor. Nesta arquitetura são estabelecidas as responsabilidades para as entidades que desempenham os papéis de cliente e servidor. O servidor é um processo contínuo responsável por fornecer um recurso ou serviço consumido pelo cliente. O cliente tem a característica de não se comunicar com outros clientes. Ele inicia o processo enviando uma mensagem através da rede requisitando o serviço. O servidor processa a mensagem e responde para o cliente que aguarda a resposta [23]. A figura a seguir mostra um exemplo da arquitetura cliente-servidor.

Figura 3.2: Arquitetura cliente-servidor [23].



Levando em conta a arquitetura exemplificada na Figura 3.2, os componentes **App motorista** e **App usuário final** do sistema proposto devem se comportar como clientes. **Wheretrans_server** e **Gpstracker_server** devem se comportar como servidores, utilizando recursos de bancos de dados disponibilizados pelos componentes **Db-GpsTracker** e **Db-Wheretrans**; E o serviço de *push message* disponibilizado pelo componente **Push notification server**. Feita essa separação, a explicação sobre os componentes foi dividida em dois conjuntos:

- Componentes servidores
  - GpsTracker_server
  - Wheretrans_server
  - Push notification server
- Componentes de banco de dados
  - Db_GpsTracker
  - Db_Wheretrans
- Componentes clientes
  - App motorista
  - App usuário final

3.1 Componentes servidores

Para o bom funcionamento de uma aplicação móvel, é muito importante ter componente(s) servidor(es) para realizar qualquer tipo de operação que demande um alto processamento. Isto porque em geral os dispositivos móveis possuem uma baixa capacidade energética e pouco poder de processamento.

É necessário escolher bem as tecnologias envolvidas para que o aplicativo móvel que consome os recursos oferecidos pelo servidor não tenha um desempenho lento e/ou apresente travamentos, fazendo com que o usuário troque para outra aplicação rapidamente.

Alguns conceitos e tecnologias serão apresentados a seguir, para detalhar com maior clareza os componentes do servidor, de forma que seja possível fazer a designação da solução por meio da relação entre componentes isolados.

A comunicação entre componentes do sistema utiliza o protocolo HTTP e os princípios do *REST*.

3.1.1 Componente Gpstracker-server

O componente é um serviço do tipo RESTfull. Esse serviço expõe URLs auto-explicativas fazendo com que as respostas do sistema sejam previsíveis e contenham informações sobre um dado ou uma coleção.

O componente possui a função de armazenar, disponibilizar e requisitar dados de geolocalização dos dispositivos móveis. Ele recebe requisição do componente **Wheretrans-server** para armazenar dados de geolocalização utilizando o componente **DB-GpsTracker**. Quando requisitado, este utiliza o componente **Push notification server** para sensibilizar o dispositivo móvel do motorista que sua localização está sendo requisitada através do mecanismo de *push notification*.

3.1.2 Componente Wheretrans-server

De forma semelhante ao componente Gpstracker_server, este também é um serviço do tipo RESTfull; ou seja, expõe URLs auto-explicativas prevendo as respostas do sistema sobre um dado ou uma coleção.

O serviço exposto permite realizar inserção, consulta, atualização e deleção dos recursos presentes no componente Db-wheretrans. Quando o componente recebe um dado de geolocalização do componente App motorista que foi requisitado pelo componente App usuário final, este, se comunica com o componente *Push notification server* para requisitar a entrega da localização para componente App usuário final. Nesse caso, a localização é enviada através do mecanismo de *push notification*.

3.1.3 Componente Push notification server

Uma dificuldade dos dispositivos móveis é a verificação de um novo recurso disponível, para, então, realizar uma ação sobre o mesmo. Uma forma de implementar essa verificação é chamada de *polling*. Nela, um cliente realiza repetitivamente o consumo de um serviço. O impacto da utilização do *polling* é expressivo quando os recursos como bateria e pacote de dados são consumidos de forma demasiada. Outra consequência que se pode ter são os congelamentos, que causam perda de fluidez e instabilidade com a internet durante a utilização do dispositivo. Por causa desses motivos deve ser evitado a implementação do *polling* em dispositivos móveis. Uma forma de realizar a verificação de novos recursos disponíveis é por meio do mecanismo de *push notification*, descrito a seguir.

Para receber uma mensagem do tipo *push notification*, o cliente não requisita uma informação a um servidor. É o servidor quem envia a mensagem, potencialmente a qualquer momento, através de um servidor de *push notification*.

O componente *Push notification server* recebe mensagens do tipo *POST* vinda dos componentes **Wheretrans_server** e **Gpstracer_server** e as encaminha por meio de *push notification* para os componentes **App motorista** e **App usuário final**.

3.2 Componentes de banco de dados

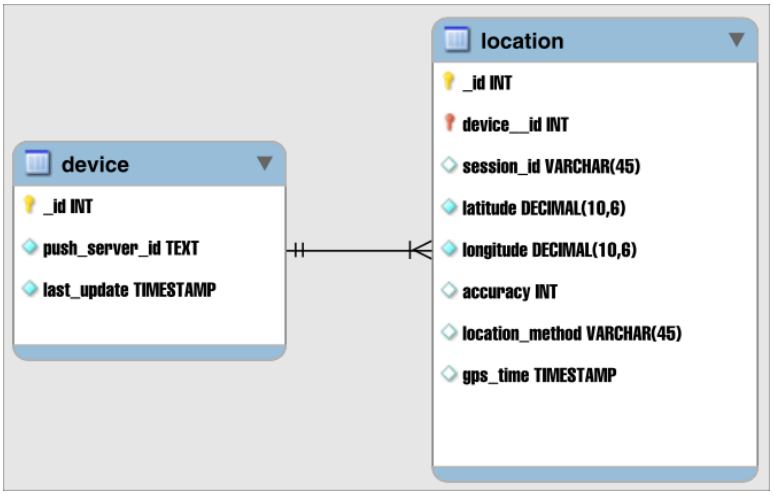
Os componentes de software apresentados a seguir são serviços de banco de dados. Esses são acessados exclusivamente por um componente de servidor.

3.2.1 Componente Db-GpsTracker

Este componente é utilizado exclusivamente por Gpstracker-server. Seu objetivo é armazenar informações referentes à geolocalização e aos dispositivos móveis. Foi criado de forma a ser minimalista, ou seja, conter a menor quantidade de dados necessários e suficientes para possibilitar a utilização de forma genérica por outras soluções, cujos dados principais são os de geolocalização.

A seguir, o diagrama ER (Entidade Relacionamento) utilizando a notação James Martin.

Figura 3.3: Diagrama entidade relacionamento Db-Gpstracker.



As entidades apresentadas no diagrama 3.3 são explicadas a seguir:

- **Entidade *device* :** a entidade *device* (em português dispositivo) é responsável por manter o token do servidor de *push message* de cada dispositivo cadastrado no sistema. Esse *token* é um identificador do dispositivo no servidor de push. Devido ao baixo acoplamento, nem todas as informações dos dispositivos são armazenadas neste componente e, para que não haja informações duplicadas, o que poderia causar inconsistência, essas informações restantes são armazenadas no componente Db-Wheretrans.

  A Figura 3.3 demonstra que há um relacionamento dessa entidade com a entidade *location*.

- **Entidade *location* :** a entidade *location* (em português localização) mantém os dados que representa a localização de um dispositivo. Seus atributos são: o identificador único do dispositivo

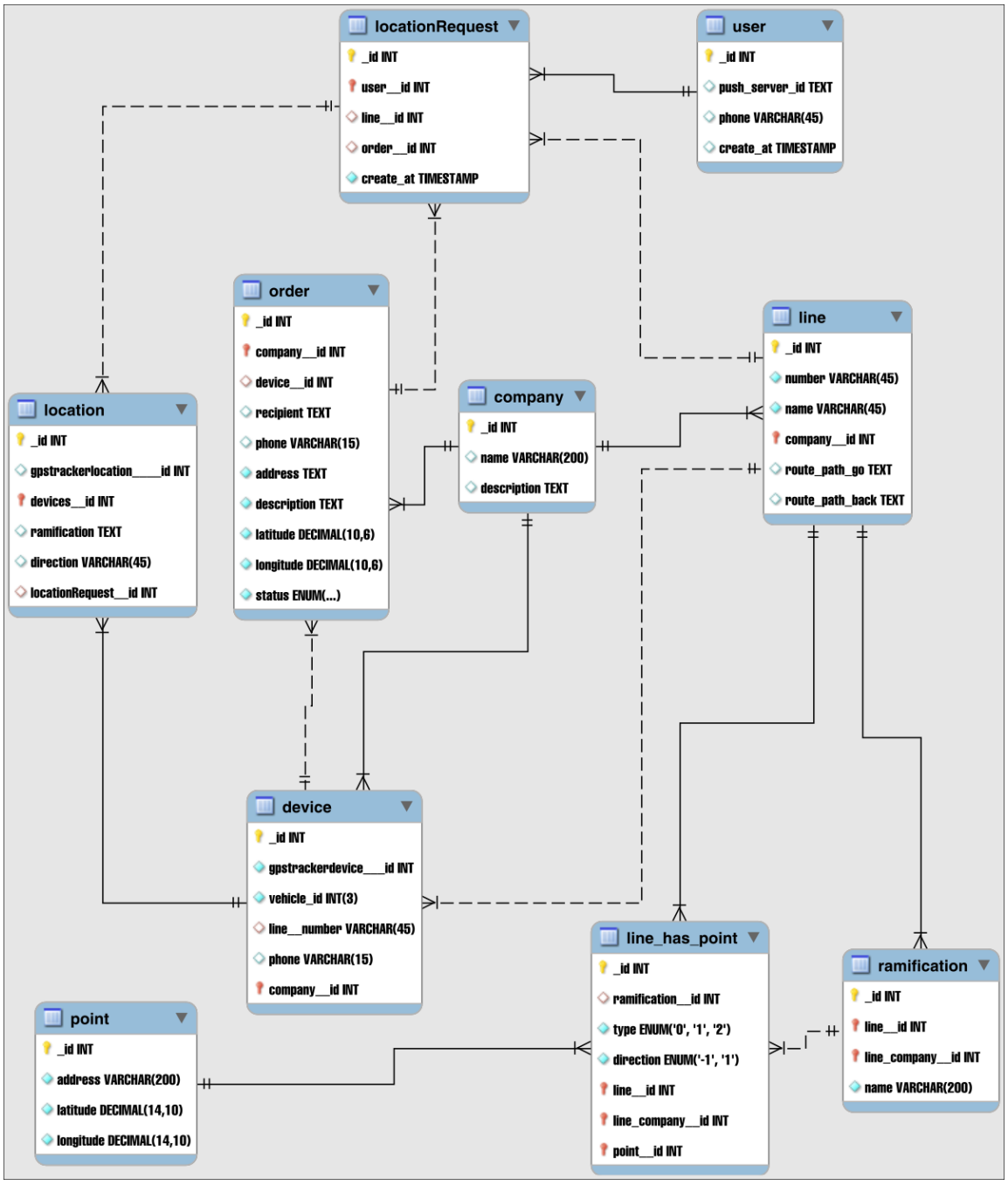
responsável pela sua criação, um identificador de seção do dispositivo, os atributos que mantém a latitude e longitude da localização do dispositivo, a acurácia obtida na captura da localização, método utilizado para a captura da localização, e o momento em que foi obtida a localização.

3.2.2 Componente Db-Wheretrans

Este componente é utilizado exclusivamente pelo componente Wheretrans_server. Tem o objetivo de armazenar informações referentes a: empresas, linhas (forma de organização de uma frota) e suas ramificações, pontos de parada dos veículos e informações de pedidos. Nesse componente também é persistida a informação de requisição da localização do dispositivo do motorista.

Abaixo segue o diagrama ER (Entidade Relacionamento) utilizando a notação James Martin.

Figura 3.4: Diagrama entidade relacionamento Db-Wheretrans.



As entidades apresentadas no diagrama 3.3 são explicadas a seguir:

- **Entidade *user* :** a entidade *user* (em português usuário) mantém os dados do usuário do componente App usuário final. Seus atributos são: um identificador do componente *Push notification server* e número do telefone. A Figura 3.4 demonstra que há um relacionamento dessa entidade com a entidade *locationRequests*.

- **Entidade *locationRequest* :** a entidade *locationRequest* (em português solicitação de localização) representa a intenção do usuário do componente App usuário final de receber a localização do componente App motorista. O dado necessário para o registro da solicitação de localização é o iden-

tificador do usuário do componente App usuário final que realizou a solicitação, e um identificador da linha ou pedido referente a localização que deseja receber.

- **Entidade *company* :** a entidade *company* (em português empresa), contém um nome e uma descrição. O motivador de criação dessa entidade foi possibilitar que uma empresa específica possa fazer alterações em um grupo restrito de dados do sistema. Dados esses providos pelos relacionamentos entre a empresa e as entidades *line*, *order* e *device*. Esses relacionamentos podem ser observados na Figura 3.4.

- **Entidade *order* :** a entidade *order* (em português pedido), mantém os dados relacionados ao pedido de entrega. Os atributos são: identificador da empresa criadora do pedido, o status da entrega (NÃO REALIZADA, REALIZANDO, POSTERGADA ou REALIZADA) e dados do destinatário: endereço, dados de geolocalização (latitude e longitude) e o telefone.

  Os atributos são inseridos através de uma API disponibilizada pelo sistema. Desta forma uma empresa pode implementar seu próprio web-site para criar, alterar ou deletar pedidos, além de obter dados de status e localização do pedido correntes.

  Após a criação de um pedido, o mesmo é associado a um veículo (dispositivo com a aplicação do motorista instalada) pertencente a empresa que criou o pedido.

- **Entidade *device* :** a entidade *device* (em português dispositivo) mantém os dados relacionados ao dispositivo com a aplicação do componente App motorista instalada. Ao cadastrar um usuário utilizando a API do componente Wheretrans_server o sistema realiza um redirecionamento para a API do componente Gpstracker_server para também realizar o cadastramento nessa API. O único dado repassado para a API Gpstracker_server é o token do servidor de push (identificador da aplicação registrada no componente Push notification server), por motivo de isolamento do componente responsável por requisitar e persistir a geolocalização do dispositivo com a aplicação do motorista instalada. Os atributos da entidade são: o identificador do dispositivo no componente Gpstracker_server, a linha que o dispositivo esta associado, o telefone de contato do motorista e a empresa na qual o dispositivo pertence. Um dispositivo com a aplicação do motorista instalada pode ou não ter uma linha associada. Como se pode ver na Figura 3.4 essa entidade possui relacionamento com as entidades: *order*, *company* e *line*.

- **Entidade *location* :** a entidade *location* (em português localização) mantém os dados referentes a localização do dispositivo com aplicação do motorista instalada. No momento em que o componente Wheretrans_server recebe dados de geolocalização e informações sobre a locomoção do dispositivo, o mesmo realiza uma requisição de armazenamento de dados de geolocalização ao componente Gpstracer_server. Após a confirmação de persistência, o componente Wheretrnas_server persiste os dados apresentados a seguir no componente DB-wheretrans. Os dados persistidos na entidade são: o identificador de localização recebido da API Gpstracker_server, o identificador do dispositivo referente a localização e informação de sobre a locomoção do dispositivo (se está realizando o trajeto de alguma ramificação e o sentido desta).

- **Entidade *line* :** a entidade *line* (em português linha), mantém dados relacionados à organização da frota dos dispositivos com a aplicação do motorista instalada. Os dados mantidos pela entidade são: um identificador da linha, o nome e a empresa à qual a linha se refere.

  Na Figura 3.4 se pode observar que a entidade linha possui relacionamentos com as entidades *ramification* e *line_has_point*.

- **Entidade *ramification* :** a entidade *ramification* (em português ramificacao), é responsável por manter dados referentes às ramificações que cada linha possui. Seus atributos são o identificador da linha e da empresa a qual a ramificação pertence, e o nome da ramificação. Como se pode observar, na Figura 3.4 essa entidade também possui relacionamento com a entidade *line_has_point*, essa relação será explicada a seguir.

- **Entidade *point* :** a entidade *point* (em português ponto), é responsável por manter dados relacionados a localização de um ponto. Seus atributos são: endereço, latitude e longitude do ponto. A Figura 3.4 demonstra que esta entidade possui relacionamento com a entidade *line_has_point*.

- **Entidade *line_has_point* :** essa entidade é responsável pelo relacionamento da entidade linha com a entidade *point*, ou seja, quando uma linha possui um ou mais pontos. Para tal, é armazenado o identificador de uma linha e o identificador de um ponto, permitindo que uma linha tenha diferentes pontos e que diferentes pontos estejam associados a diferentes linhas. Cada um dos identificadores é único para sua respectiva entidade.

3.3 Componentes clientes

Os componentes **App motorista** e **App usuário final** são aplicações desenvolvidas para dispositivos móveis que possuem conexão internet e GPS (sigla de Global Position System,em português, sistema de positionamento global).

Para as aplicações terem a capacidade de serem facilmente expandidas e mantidas, no desenvolvimento foi utilizado três padrões: injeção de dependência, repositório e MVP.

O padrão injeção de dependência é utilizado para manter um baixo acoplamento entre as classes do sistema. Nele, as dependências de uma classe são passadas por meio de uma injeção via construtor ou propriedade.

O padrão repositório é utilizado para desacoplar o código de acesso a dados do código de domínio. Para isto, é criado uma camada de abstração com uma interface, que é utilizada pela camada de domínio e implementada pela classe repositório, desta forma encapsulando as operações realizadas sobre um conjunto de dados.

O padrão de projeto MVP (*Model-View-Presenter*, em português, Modelo-Visão-Apresentação). O MVP tem a finalidade de separar a camada de apresentação das camadas de dados e regra de negócio; criando a possibilidade de realização de testes em camadas específicas. Na implementação do MVP temos a separação em três camadas, estas são explicadas a seguir:

- **View :** responsável por manipular eventos de interface com o usuário. Essa camada deve possuir uma referência para a camada de *Presenter*, sua responsabilidade é construir elementos de interface com usuários e chamar os métodos do *Presenter* ao receber uma interação do usuário.

- **Presenter :** faz o intermédio entre a camada de *View* e *Model*. Ao receber um evento de usuário, a camada de *View* aciona esta, que possui acesso ao camada *Model* e retorna o resultado para a camada de *View*.

- **Model :** essa camada contém os dados apresentados na camada de *View* e lógica de manipulação aplicada à estes.

3.3.1 Componente App motorista

O primeiro aplicativo móvel se trata da aplicação instalada no *smartphone* do motorista. Essa, foi denominada **GpsTracker** e tem a função de fornecer a localização da frota para o sistema. O componente recebe notificações através do componente **Push notification server** e utiliza as informações disponibilizadas pelo componente **Wheretrans_server**.

A modelagem de classes do aplicativo do motorista foi organizada em pacote por funcionalidade. Foram criados cinco pacotes: *data, push, register* e *sendloc* . A seguir serão expostos explicações referente a cada pacote de forma que facilite a visualização dos diagramas e entendimento desta aplicação. Os diagramas de classes estão no apêndice.

- O pacote *data* (encontrado no apêndice: Diagrama de projeto componente App motorista, na sessão B.1) é responsável pela manipulação de dados do aplicativo .

- O pacote *push* (encontrado no apêndice: Diagrama de projeto componente App motorista, na sessão B.2) é responsável pelo tratamento das mensagens recebidas do componente *Push notification server*.

- O pacote *register* (encontrado no apêndice: Diagrama de projeto componente App motorista, na sessão B.3) é responsável por receber o input de informações do usuário e realizar o registro do usuário no componente Wheretrans_server.

- O pacote *sendloc* (encontrado no apêndice: Diagrama de projeto componente App motorista, na sessão B.4) é responsável por coletar informações de geolocalização e trajeto para enviar para o componente wheretrans_server.

3.3.2 Componente App usuário final

Este componente foi batizado com o nome de **Chega+**, se trata do aplicativo disponibilizado para o usuário comum. Ele também recebe notificação do tipo *push notification* do componente **Push notification server** e consome o serviço disponibilizado pelo componente **Wheretrans_server**.

Assim como o componente App motorista, a modelagem de classes desse aplicativo foi organizada em pacote por funcionalidade. O modelo foi dividido nos pacotes: *data, push, lines, points, search.provider*

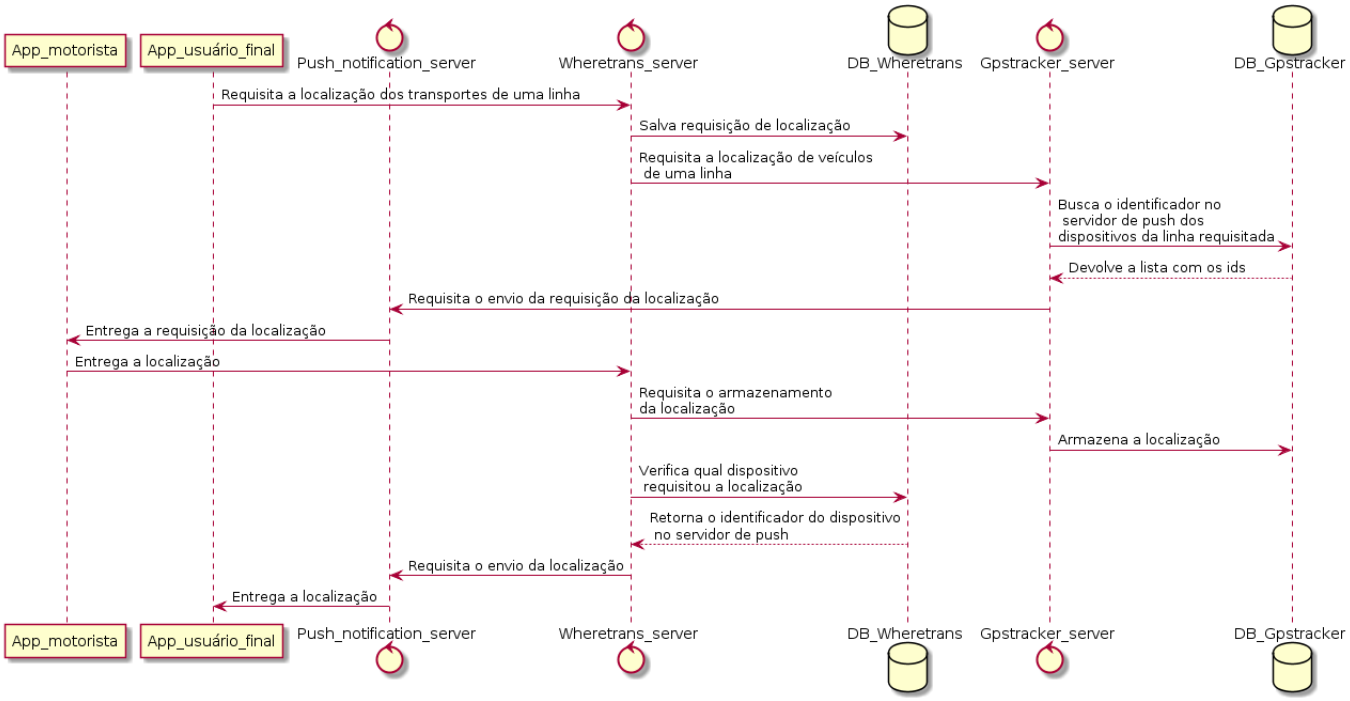
e *sync*. A seguir serão expostos as explicações referente a cada pacote de forma que facilite a visualização dos diagramas e entendimento desta aplicação. Os diagramas de classes estão no apêndice.

- O pacote *data* (encontrado no apêndice: Diagrama de projeto componente App usuário final, na sessão C.1) é responsável pela manipulação de dados do aplicativo.
- O pacote *push* (encontrado no apêndice: Diagrama de projeto componente App usuário final, na sessão C.2) é responsável pelo tratamento das mensagens recebidas do componente *Push notification server*.
- O pacote *lines* (encontrado no apêndice: Diagrama de projeto componente App usuário final, na sessão C.3) é responsável pela apresentação das informações referentes as linhas.
- O pacote *points* (encontrado no apêndice: Diagrama de projeto componente App usuário final, na sessão C.4) é responsável pela apresentação das informações referentes aos pontos e a localização dos veículos.
- O pacote *search.provider* (encontrado no apêndice: Diagrama de projeto componente App usuário final, na sessão C.5) é responsável por fornecer um mecanismo para busca de locais utilizando texto.
- O pacote *sync* (encontrado no apêndice: Diagrama de projeto componente App usuário final, na sessão C.6) é responsável por realizar a sincronização dos dados do aplicativo com os dados do servidor.

3.4 Fluxo de execução

A Figura 3.5 ilustra a sequência em que os componentes do sistema são acionados durante a principal funcionalidade que é a obtenção da localização dos transportes de uma linha.

Figura 3.5: Diagrama de sequência para funcionalidade de obter a localização dos transportes de uma linha.



Capítulo 4

Implementação do sistema

Com base na arquitetura descrita no capítulo anterior, segue o detalhamento de implementação, tal como as dificuldades encontradas e as decisões de implementação.

A Figura 4.1 ilustra as tecnologias utilizadas na implementação do sistema. Estas serão abordadas durante o desenvolvimento deste capítulo.

Figura 4.1: Tecnologias utilizadas na implementação do sistema.

4.1 Componentes servidores

Os módulos **Gpstracker_server** e **Wheretrans_server** foram construídos utilizando o microframewordk Slim [7]. O *framework* abstrai a implementação do protocolo *HTTP* de maneira que ao receber uma requisição *HTTP*, invoca uma rotina de *callback* apropriada e retorna uma resposta *HTTP*.

A funcionalidade do servidor é acessada através de chamada de API *HTTP*. API tem como objetivo simplificar e padronizar a comunicação entre os pares cliente-servidor. Tendo uma interface bem definida, a complexidade das regras de negócio são encapsuladas pelo serviço oferecido através da API, desta forma facilitando a comunicação entre os módulos. Sendo a API do servidor um tipo de serviço utilizando a arquitetura REST, a maneira com que ela é estruturada é chamada de *RESTful* [2].

O código a seguir exemplifica a implementação uma *API REST* utilizando o *framework Slim*, ao acessar a URL http://localhost/hello/World, o resultado é a mensagem ”Hello World!”.

```php
<?php
// Create Slim app
$app = new \Slim\App();
// Define app routes
$app->get('/hello/{name}', function ($request, $response, $args) {
    return $response->write("Hello " . $args['name']);
});
// Run app
$app->run();
```

A fim de simplificar a instalação deste framework, foi utilizado o gerenciador de dependência Composer [10].

Com o crescimento do software, gerenciar muitos objetos com várias dependências se torna uma tarefa árdua. Para evitar o alto acoplamento das classes do sistema foi utilizado o padrão *DIC (Dependency Injection Container*, em português injeção de dependência de contêiner). O grande ganho é que as classes que utilizam as dependências não precisam conhecer como instancia-las ou configurá-las [9] Esse padrão se encaixa bem no software proposto, onde existem várias classes controladoras que realizam modificações no banco de dados, escrevem em um arquivo de log e formatam a resposta para saída em *JSON*.

O servidor responsável por disponibilizar os componentes é o NGINX. Este é conhecido pela sua alta performance, estabilidade, por consumir poucos recursos e pela simplicidade de configuração [14]. Para possibilitar a geração de páginas dinâmicas no servidor web, nesse caso, conteúdos gerados pelos *scripts* PHP, foi configurado o CGI(Common Gateway Interface, em português interface comum de porta de entrada) para interpretação dos scripts por um programa externo, nesse caso o **php**. Para cada requisição é gerado uma instância em execução do *script* interpretado pelo programa **php**, passando os parâmetros de entrada. Ao finalizar a interpretação, a saída é enviada para o servidor.

A seguir segue o esqueleto de arquivos usados na implementação dos componentes Gpstracker_server e Wheretrans_server.

Figura 4.2: Esqueleto de arquivos usados na implementação dos componentes Gpstracker_server e Wheretrans_server.

```
server
├── public
│   └── index.php ..... Arquivo exposto para a WEB
├── src
│   ├── routes.php ..... Configuração de rotas e calbacks
│   ├── settings.php. ..... Configurações de variáveis utilizadas em execução
│   ├── dependencies.php. ..... Configuração de injeção de dependência
│   ├── Formatter.php. ..... Configuração de formatação do recurso requisitado
│   └── controler....  Esta pasta contém os arqui-
│                      vos fontes .
│       ├── Controler.php
│       ├── ...
│       ├── ...
│       └── ...
├── docker
│   ├── nginx
│   │   ├── Dockerfile..... Configuração usada na criação da imagem NGINX
│   │   └── default.conf. ..... Arquivo de configuração do servidor NGINX
│   └── php
│       ├── Dockerfile. ..... Configuração usada na criação da imagem do PHP
│       └── php.ini..... Arquivo de configuração do PHP
├── docker-compose.yml. ..... Arquivo de configuração de serviços e volumes do docker
└── composer.json. ..... Configuração de dependências
```

A seguir serão detalhados os arquivos mais importantes (presentes no apêndice Implementação componentes servidor:Configurações comuns D) :

- O arquivo **server/composer.json** (encontrado no apêndice: Implementação componentes servidor - Configurações comuns, na sessão D.1) contém as configurações de dependências instaladas pelo programa Composer.
- O arquivo **server/src/settings.php** (encontrado no apêndice: Implementação componentes servidor - Configurações comuns, na sessão D.2) contém variáveis de ambiente utilizadas pelo software em tempo de execução.
- O arquivo **server/src/dependencies.php** (encontrado no apêndice: Implementação componentes servidor - Configurações comuns, na sessão D.3) contém a configuração dos contêineres de dependência.
- O arquivo **server/docker/nginx/default.conf** (encontrado no apêndice: Implementação componentes servidor - Configurações comuns, na sessão D.4) contém configurações do servidor NGINX [14], este responsável pela disponibilização do serviço.
- O arquivo **server/docker-compose.yml** (encontrado no apêndice: Implementação componentes servidor - Configurações comuns, na sessão D.5) contém configurações utilizadas pelo *docker-compose* (programa para interpretação de arquivos de configuração do *docker*).

- O arquivo **server/src/controler/Controler.php** (encontrado no apêndice: Implementação componentes servidor - Configurações comuns, na sessão D.6) define a superclasse utilizada pelos demais arquivos presentes na pasta *controler*.

As imagens referentes aos serviços definidos no arquivo "docker-compose.yml", são geradas com o comando apresentado a seguir:

```
1 docker-compose build
```

Listing 4.1: Comando em terminal *bash* utilizado para gerar imagem(ns) referentes aos serviços definidos no arquivo de configuração docker-compose.yml

Após a geração das imagens, os serviços são iniciados através da criação de contêineres com o comando apresentado a seguir:

```
1 docker-compose up
```

Listing 4.2: Comando em terminal *bash* utilizado para iniciar os serviços definidos no arquivo de configuração docker-compose.yml

4.1.1 Gpstracker_server

Com o esqueleto de arquivos apresentado anteriormente, os demais arquivos fontes utilizados na implementação desse componente são as classes controladoras das entidades apresentadas na secção 3.2.1 do capitulo 3 e o arquivo "index.php" exposto para a WEB. A seguir segue o diretório de arquivos adicionados na implementação do componente.

Figura 4.3: Arquivos adicionais codificados na implementação do componente Gpstracker_server.

```
server
├── src
│   └── controler
│       ├── Location.php
│       └── Device.php
└── public
    └── index.php
```

A seguir é detalhado o esqueleto dos arquivos adicionados na implementação do componente:

- O arquivo **server/public/index.php** (encontrado no apêndice: Implementação do componente Gpstracker_server, na sessão E.1) é quem disponibiliza uma instancia da aplicação, ele contém o mapeamento de métodos que são invocados ao receber uma requisição *HTTP*.

- O arquivos presentes na pasta **server/src/controler/** contém implementação dos *callbacks* invocados ao requisitar métodos *HTTP*.

4.1.2 Wheretrans_server

Esse componente assim como o componente Gpstracker_server utiliza o esqueleto apresentado na Figura 4.2. Adicionalmente esse componente consiste da implementação de classes controladoras referentes às entidades apresentadas na seção 3.2.2 do Capítulo 3 e o arquivo "index.php" exposto para a WEB. A seguir seguem os diretórios correspondente aos arquivos codificados na implementação deste componente.

Figura 4.4: Arquivos adicionais codificados na implementação do componentes Wheretrans_server.

```
server
├── src
│   └── controler
│       ├── RequestLocation.php
│       ├── Point.php
│       ├── Location.php
│       ├── Company.php
│       ├── User.php
│       ├── Line.php
│       ├── LineHasPoint.php
│       ├── Device.php
│       ├── Ramific.php
│       └── Order.php
└── pubilc
    └── index.php
```

Seguindo o padrão de desenvolvimento, a pasta **server/src/controler/** e o arquivo "server/public/index.php" (encontrado no apêndice: Implementação do componente Wheretrans_server, na sessão F.1) possuem as mesmas responsabilidades que o componente anterior.

4.1.3 Componente Push notification server

A implementação desse componente foi realizada utilizando o Google Cloud Messaging (GCM). Esse é um serviço gratuito que permite o envio de mensagem entre servidor e aplicação mobile utilizando o mecanismo de *push notification* [15].

Para que os dispositivos móveis possam receber mensagem desse componente, é necessário realizar o registro, obter um identificador único do serviço e armazenar esse em um local que possa ser utilizado pelos componentes de software que devem enviar mensagem utilizando o serviço de *push notification*.

4.2 Componentes de banco de dados

Foi eleito para o desenvolvimento dos componentes Db-GpsTracker e DB-wheretrans o SGBD MySQL. Este é um sistema muito utilizado por possuir um excelente desempenho e estabilidade. Para manipulação e modelagem foi utilizado a ferramenta de visualização de banco de dados MySQL Wordkbench gratuitamente disponibilizada pela Oracle.

4.3 Componentes clientes

Os componentes App motorista e App usuário final são aplicações nativas da plataforma Android. A plataforma possui o SDK desenvolvido em Java, documentação disponível, uma comunidade envolvida com o desenvolvimento da plataforma, um robusto ambiente de desenvolvimento (SDK, IDE, monitor de aplicação e emulador) e o motivador para a escolha da plataforma para desenvolvimento das aplicações móveis, o fato do projeto do sistema operacional ser *open source* (em português, significa que o projeto possui código aberto).

4.3.1 App motorista

O aplicativo do motorista é a aplicação que alimenta o sistema com os dados de geolocalização referente aos veículos que constituem a frota de uma empresa. Como foi mencionado no capítulo anterior, trata-se de um aplicativo móvel nativo da plataforma Android.

A seguir serão exibidos telas do aplicativos referente as funcionalidades implementadas e outras características importantes.

Tela de registro

A Figura 4.5 ilustra a primeira tela do aplicativo. Ela é responsável por receber os dados necessários para cadastrar o motorista no sistema. Ao ser apresentada, é feita uma consulta na API do servidor para retornar as empresas cadastradas no sistema. O usuário pode selecionar a empresa a partir de uma lista. Ao selecionar, o aplicativo realiza outra consulta à API servidor para verificar se a empresa selecionada possui organizações de frotas. Caso afirmativo, as opções são disponíveis em outra lista para que o usuário selecione a organização a qual pertence. Além das opções para seleção, possuem dois campos para entrada de dados, o primeiro campo para entrada do identificador do veículo e outra para telefone do motorista.

Após clicar no botão de cadastrar, as informações preenchidas são enviadas para o servidor realizar o cadastro do usuário. Enquanto o cadastro é realizado, uma barra de progresso é exibida como na Figura 4.6.

Figura 4.5: Telas de registro

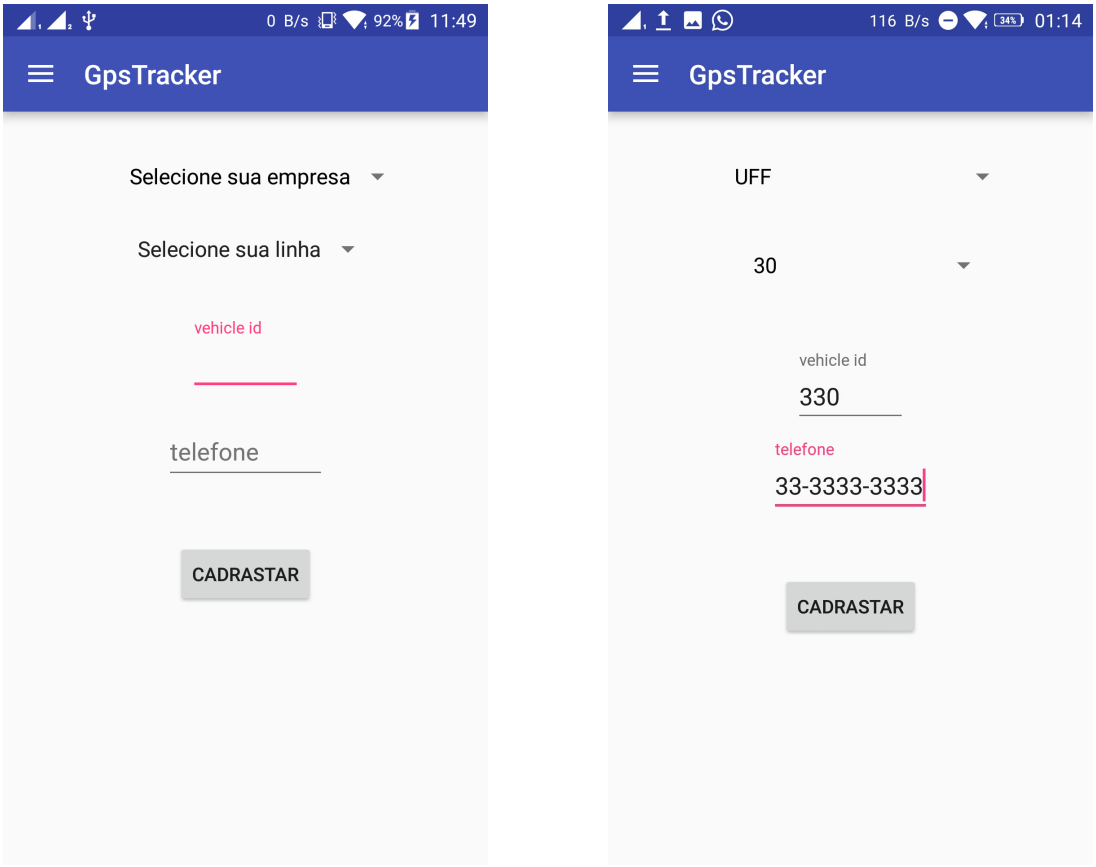
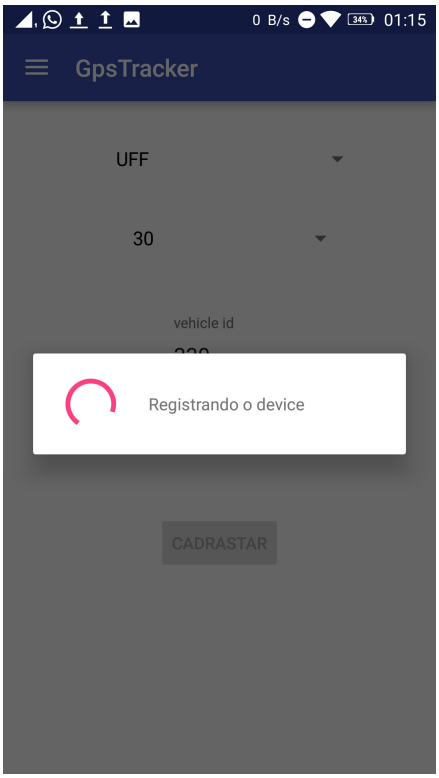


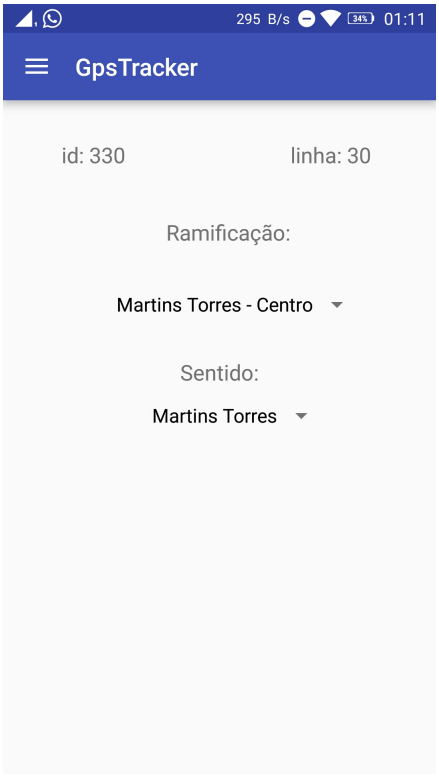
Figura 4.6: Tela de barra de progresso mostrada enquanto o cadastro é realizado no servidor.



Tela de rota

Antes de iniciar um itinerário, o motorista entra com o dado referente à rota de ramificação e ao sentido que está presta a iniciar. A Figura 4.7 ilustra essa iteração com o aplicativo.

Figura 4.7: Tela com dados de rota .



4.3.2 App usuário final

Da mesma forma que o aplicativo do motorista, este componente é um aplicativo nativo da plataforma Android. Disponibilizada para o usuário comum, o aplicativo batizado de Chega+ facilita o dia-a-dia do usuário com suas funcionalidades.

A seguir serão exibidas as telas, funcionalidades implementadas e outras características referentes à aplicação.

Tela de *download* de dados

Ao abrir o aplicativo pela primeira vez, o mesmo realiza o cadastro do dispositivo no servidor, obtém e persiste os dados referentes às linhas e pontos cadastrados no servidor.

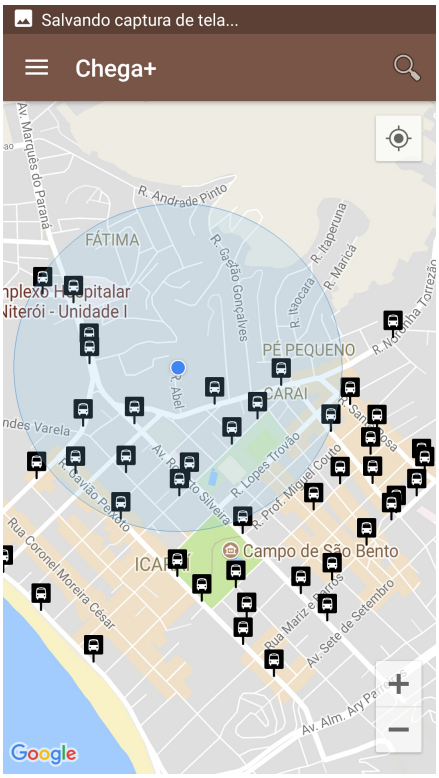
Figura 4.8: Tela de sincronização de dados.

Tela inicial

Ao abrir o aplicativo, o usuário é direcionado para a tela ilustrada pela Figura 4.9. Na tela podem ser realizadas operações sobre o mapa, como movimentação e zoom. Nela o usuário encontra os pontos cadastrados no sistema. Esses são representados pela Figura de placa de ônibus. Somente os pontos próximos são mostrados, para não poluir a interface. Ao selecionar um ponto, é mostrado uma janela com a informação do endereço, e ao clicar nessa janela, o usuário é direcionado para a tela de informações do ponto.

No canto superior direito há um botão com o desenho de uma lupa. Ao clicar nesse botão o usuário é direcionado para a tela de busca. No canto superior esquerdo, há um botão com o desenho de três linhas paralelas. Quando acionado, o usuário é direcionado para a tela de gaveta de navegação.

Figura 4.9: Tela inicial.



Tela de informações do ponto

A tela é ilustrada pela Figura 4.10. Nela o usuário encontra: a informação da distância que está do ponto selecionado; caminho para chegar no ponto; e uma lista das linhas (organizações da frota) que passam pelo ponto seguido de uma caixa de seleção. Ao selecionar uma caixa, é enviado para o componente Wheretrans_server a requisição das localizações dos dispositivos com a aplicação do motorista com a rota configurada com a linha selecionada. Após a aplicativo receber a localização, o mesmo as coloca no mapa com o número identificador do veículo dentro de um balão (tela ilustrada na Figura 4.11).

Figura 4.10: Tela de informações do ponto.

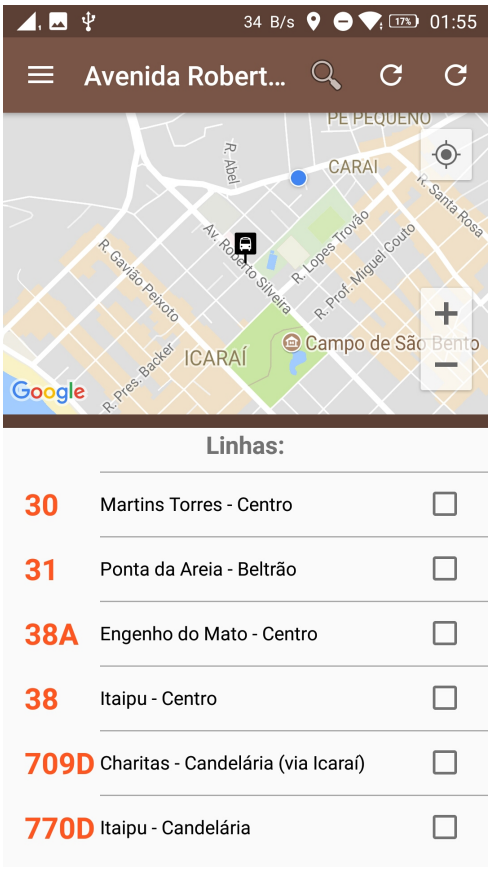
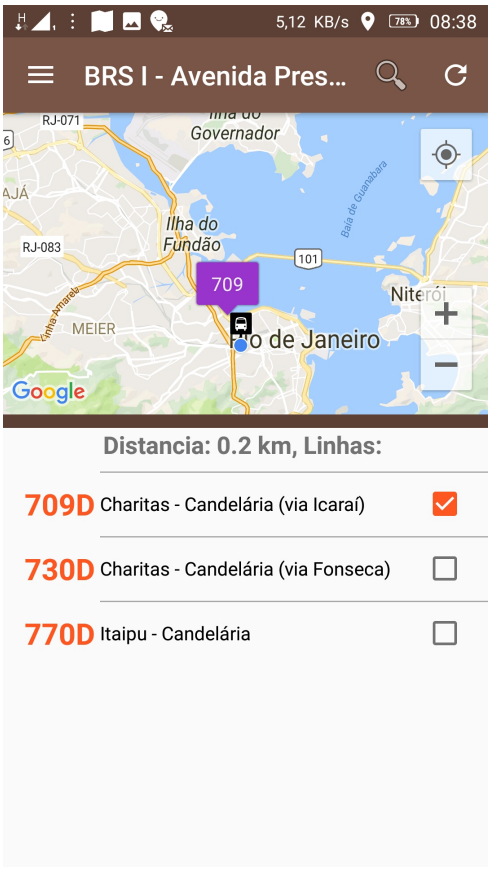


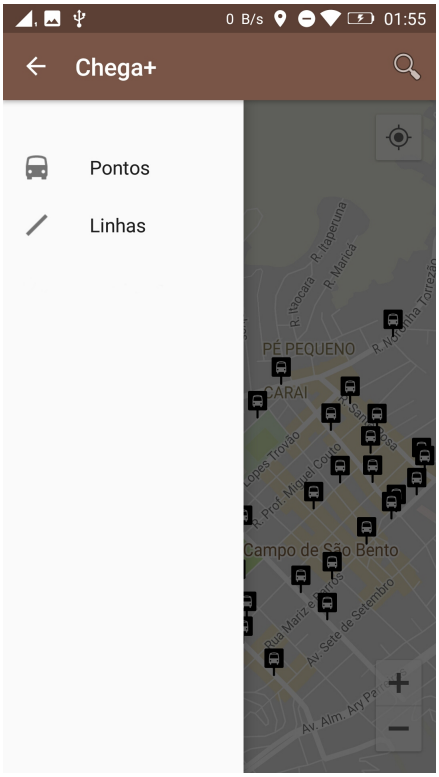
Figura 4.11: Tela de informações de pontos, com a localização do veículo.



Tela de gaveta de navegação

A tela é ilustrada pela Figura 4.12. Ela possui duas opções para seleção: Pontos e Linhas. Ao selecionar a opção ”Pontos”o usuário é direcionado pra a tela inicial. Ao selecionar a opção ”Linhas”o usuário é direcionado para a tela de linhas.

Figura 4.12: Tela de gaveta de navegação.



Tela busca de locais

A tela é ilustrada pela Figura 4.13. Após a entrada de texto, o aplicativo, utilizando uma API do Google, faz uma busca por locais contendo o texto entrado no nome do local. A resposta é listada e ao ser selecionada um marcador é colocado no mapa para a localização do local selecionado, como o ilustrado na Figura 4.14.

Figura 4.13: Tela busca.

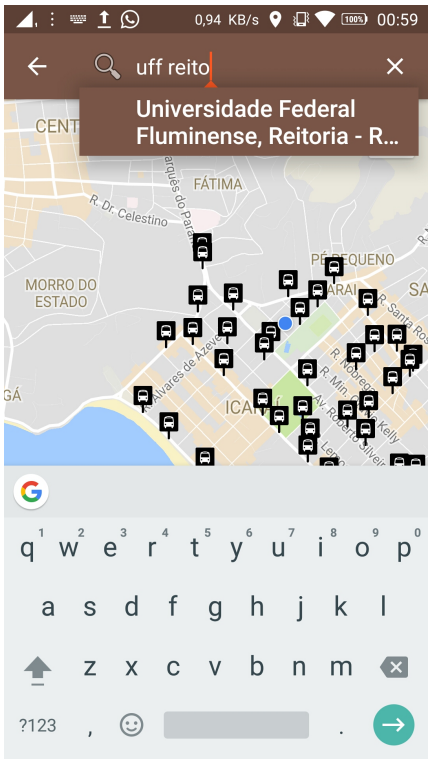
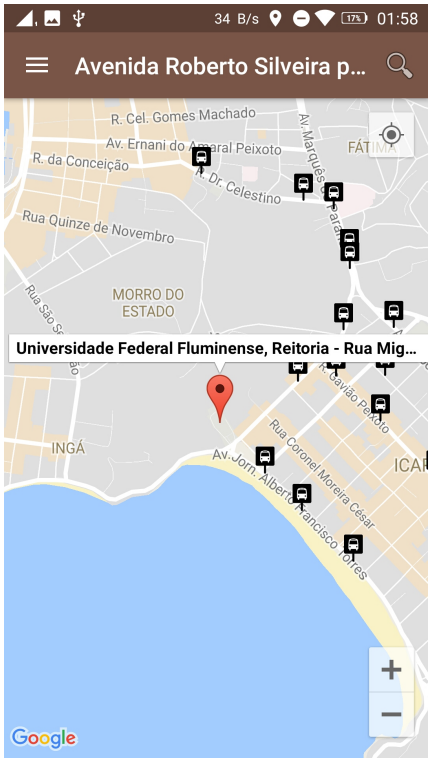


Figura 4.14: Tela de busca com marcador.



Tela de linhas

A tela é ilustrada pela Figura 4.15. Nela, as linhas cadastradas no sistema são mostradas em forma de lista. As linhas são organizadas por números seguidos das ramificações. Ao selecionar uma linha, o usuário é direcionado para a tela de itinerário de linha.

Figura 4.15: Tela de linhas.

Tela de itinerário de linha

A tela é ilustrada pela Figura 4.16. Nela um campo de texto indica que as informações pertencem a linha ”Martin Torres X Centro”, um tracejado sobre o mapa indica o itinerário da linha. Abaixo do mapa um campo de texto informa a quantidade de paradas que a linha possui e abaixo, uma lista contendo os endereços das paradas. Ao selecionar um endereço da lista, sua localização é centralizada no mapa.

Figura 4.16: Tela intinerário de linha.

Capítulo 5

Testes

Para a realização dos testes o sistema foi carregado com informações do Aplicativo Vá de Ônibus, obtidas por meio da captura de dados trafegados na internet. Os testes foram divididos em três fases. Na primeira fase foi abordado o funcionamento e utilização do sistema proposto. A segunda fase teve foco na comparação do funcionamento do sistema proposto com os trabalhos relacionados descritos no capítulo 1.5. Na última fase foram comparados a utilização de banda do aplicativo proposto com os trabalhos relacionados.

As definições a seguir são utilizadas para facilitar a explicação da dos testes realizados.

- **Passageiro:** é um voluntário que utiliza o sistema para encontrar a localização de um veículo utilizando o aplicativo do usuário comum (Chega+).
- **Motorista:** é um voluntário que alimenta o sistema com dados de localização do veículo utilizando o aplicativo do motorista (GpsTracker).

5.1 1ª Fase - Funcionamento e utilização do sistema

Para realização do testes foi aplicada a seguinte metodologia:

1. Explicação dos conceitos e funcionalidades do aplicativo para o motorista e passageiro;
2. Definição do trajeto a ser realizado pelo motorista de modo que seu caminho vá de encontro à posição do passageiro;
3. Definição de pontos de verificação ao longo do trajeto;
4. Deslocamento pelo trajeto definido por parte do motorista, que ao chegar a cada um dos pontos de verificação, deverá indicar sua localização para o passageiro através de mensagem. O passageiro deverá realizar a verificação se a localização textualmente descrita pelo motorista é a indicada no seu aplicativo;
5. Envio de um questionário aplicado com o objetivo de extrair informações e opiniões sobre a experiência.

O teste foi realizado com um motorista e três passageiros. O transporte estava inicialmente a dois quilômetros do passageiro. Durante o teste o passageiro recebeu atualizações da localização do transporte; quando o motorista chegou em cada ponto de verificação, foi realizado a troca de mensagem do motorista com o passageiro para avaliar a acurácia do sistema.

O teste piloto com o primeiro passageiro ocorreu numa terça feira (30 de maio de 2017) às nove horas da manhã em Niterói. Após o experimento foi aplicado um questionário contendo as seguintes perguntas:

Tabela 5.1: Questionário versão 1.

Número	**Perguntas**
1	A funcionalidade de mostrar o itinerário de uma linha é facilmente compreendida? Considera relevante?
2	Durante a realização do experimento, a informação de localização do veículo foi confiável?
3	Deseja relatar alguma sugestão ou melhoria ?

As respostas para o questionário aplicado ao primeiro passageiro que inclusive é um usuário do aplicativo Vá de Ônibus, estão na tabela a seguir.

Tabela 5.2: Respostas do primeiro passageiro.

Número	**Respostas**
1	Considero a pesquisa por linhas intuitiva e agradável. Julgo esta funcionalidade muito útil.
2	Considero que o nível de acurácia do sistema de geolocalização foi bom. Dentre as atualizações de percurso que recebi, apenas uma indicou que o hipotético motorista estava no interior de um prédio na mesma rua por onde passava. Acredito que um indicativo de um bom nível de acurácia foram os resultados das verificações através de mensagens, que indicaram que as localizações indicadas estavam sendo representadas no mapa do aplicativo Chega+.
3	Como sugestão para hipotéticos usos futuros do aplicativo, julgo que teria sido preferível apresentar uma identidade visual em uma outra cor. Considero que cores frias como azul, verde e cinza tem um impacto psico-cognitivo que invoca fluidez e eficiência, diferentemente do marrom utilizado, que transmite uma ideia assentamento e rigidez. Portanto, dados os objetivos do sistema, seria mais adequado.

O teste realizado com o segundo e terceiro passageiros foram realizados simultaneamente num

domingo (04 de junho de 2017) às 12:40 em Niterói. Graças à experiência anterior do teste piloto, o questionário dado para os passageiros foi atualizado, obtendo a seguinte forma:

Tabela 5.3: Questionário versão 2.

Número	**Perguntas**
1	A funcionalidade de mostrar o itinerário de uma linha é facilmente compreendida e útil?
2	Durante a realização do experimento, a informação de localização do veículo foi confiável?
3	Ao requisitar a atualização das localizações dos veículos de uma linha, foi recebida atualização?
4	Deseja relatar alguma sugestão ou melhoria ?

As respostas para o questionário aplicado ao segundo passageiro estão na tabela a seguir.

Tabela 5.4: Respostas do segundo passageiro.

Número	**Respostas**
1	Sim, interpretei as informações de forma intuitiva. A funcionalidade é uma das grandes vantagens do aplicativo. Quando me mudei de cidade e quando estive a passeio em um lugar diferente, senti falta de tal funcionalidade. O uso de um aplicativo deste tipo teria evitado experiências anteriores negativas.
2	De acordo com as verificações realizadas concluo que as informações foram confiáveis.
3	Sim, a atualização foi recebida poucos segundos após a requisição da mesma.
4	Julgo que seria interessante que esta atualização acontecesse automaticamente em tempo real, pois acredita que o sistema ficaria mais intuitivo.

As respostas para o questionário aplicado ao terceiro passageiro estão na tabela a seguir.

Tabela 5.5: Respostas do terceiro passageiro.

Número	Respostas
1	Sim, como motorista de transporte público, vejo nesta funcionalidade um grande meio de informação, visto que sou frequentemente questionado sobre itinerários.
2	Sim, sempre que foi realizado a verificação por meio de mensagens me senti confiável com relação a informação apresentada no aplicativo Chega+.
3	As atualizações foram recebidas em um intervalo curto de tempo sempre que requisitadas.
4	O aperfeiçoamento de tal aplicativo poderia ajudar a resolver o problema de logística quanto aos intervalos irregulares que na prática caracterizam esse transporte; fazendo com que algumas fiquem muito cheios e outras muito vazios.

A realização do teste de funcionamento evidenciou que o sistema atendeu aos requisitos funcionais estabelecidos no Capítulo 2. Também revelou que alguns ajustes como: a realização de filtro nas localizações com baixa acurácia; atualização automática da localização da frota e alteração da identidade visual, tornariam o sistema mais funcional. O ajuste de atualização automática foi realizado após sua identificação.

5.2 2ª Fase - Comparação do sistema proposto

Essa fase do teste foi realizada com o objetivo de comparar o sistema proposto com os trabalhos relacionados descritos no capítulo 1.5.

A metodologia utilizada para realização do teste foi a seguinte: um motorista foi embarcado em um ônibus da linha 709D próximo ao Campo São Bento, localizado no bairro de Icaraí, Niterói; enquanto uma passageira localizada no ponto situado na Rua Presidente Vargas, próximo ao número 500, Centro, Rio de Janeiro, aguardava o ônibus acompanhando as alterações de localização do mesmo através das aplicações Chega+ e Vá de Ônibus.

Logo no início do teste, foi observado que o aplicativo Moovit não possui a informação em tempo real da localização de frotas. Por esse motivo não foi possível realizar a comparação da principal funcionalidade do sistema entre os aplicativos. Durante o teste foram realizadas verificações via mensagens de texto para comparação da real localização do veículo com a apontada no mapa pelos aplicativos.

Após a realização do teste foi realizada uma conversa com a voluntária e a coleta de *screenshots* dos aplicativos. Com base nessas informações, foi feita a análise a seguir.

Na Figura 5.1 se pode verificar que além do sentido conter um erro de digitação (CANCELARIA ao invés de CANDELÁRIA), o sistema informa que este sentido não possui veículos. A Figura 5.2 informa

que no sentido Charitas, o itinerário possui um veículo. Como eu estava dentro deste, foi aferido que o sistema não realiza o identificação do sentido corretamente. Após a localização do veículo através do aplicativo Vá de Ônibus, chegou a vez de realizar a localização através do aplicativo Chega+. Para isso foi necessário abrir o aplicativo, selecionar o ponto desejado e selecionar a linha que passa pelo ponto. Como se pode observar na Figura 5.3, no mesmo minuto foi recebida a localização do veículo através do aplicativo Chega+; tendo a mesma localização nos dois sistemas.

Figura 5.1: Screenshot da tela de itinerário da linha 709D sentido Candelária do aplicativo Vá de Ônibus.

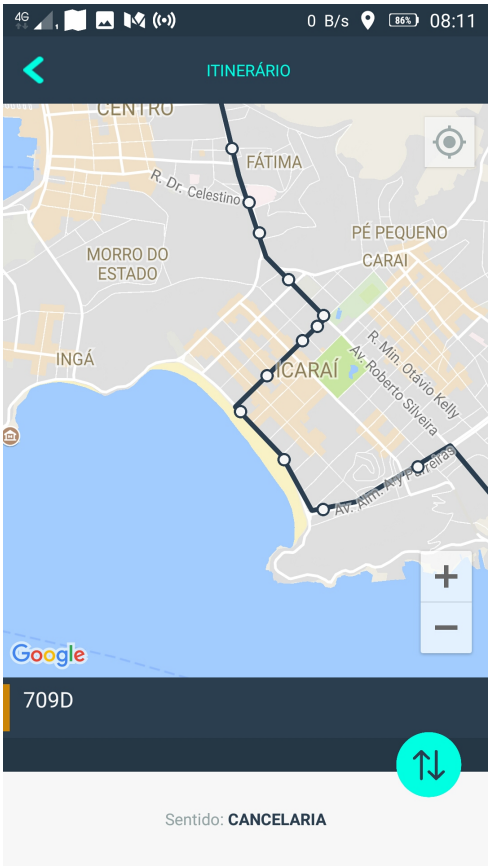


Figura 5.2: Screenshot da tela de itinerário da linha 709D sentido Charitas do aplicativo Vá de Ônibus.

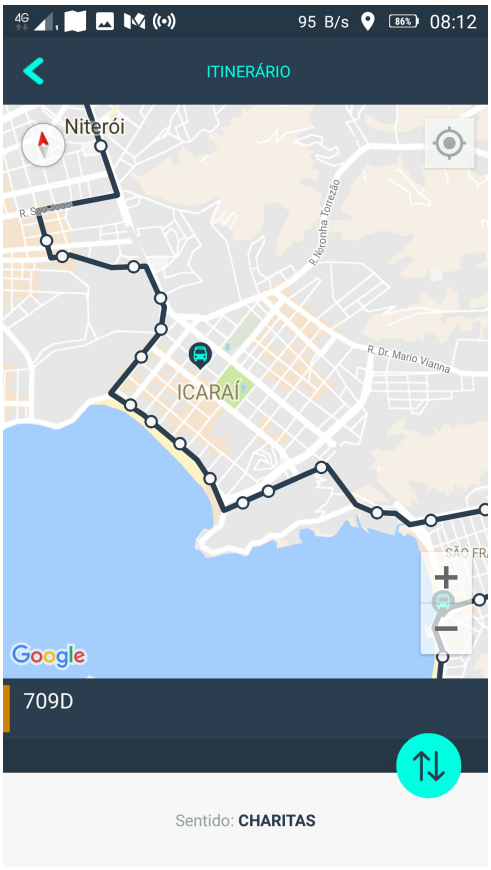
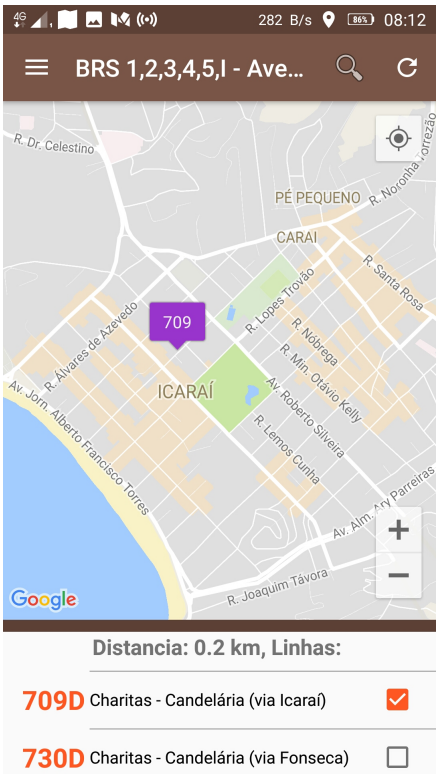


Figura 5.3: Screenshot da tela de localização das linhas que passam pelo ponto do Chega+.



Na Figura 5.4 é observado que na aplicação Vá de Ônibus, o veículo está localizado um pouco abaixo do marcador ”Niterói”. Após essa verificação, a aplicação Vá de Ônibus foi colocada em segundo plano, e a aplicação Chega+ foi colocada em primeiro plano. A Figura 5.5 ilustra que na aplicação Chega+, o veículo está localizado um pouco acima do marcador ”Niterói”. Foi realizada a troca de mensagem para verificar se localização informada na aplicação Chega+ estava precisa, e foi constatado que o veículo estava no inicio da ponte Rio-Niterói. Logo a informação da aplicação Chega+ estava correta. Com isso chegamos a conclusão que o sistema desenvolvido consegue receber a atualização de localização de veículo rapidamente.

Figura 5.4: Screenshot da tela de itinerário da linha 709D sentido Charitas do aplicativo Vá de Ônibus, início da ponte.

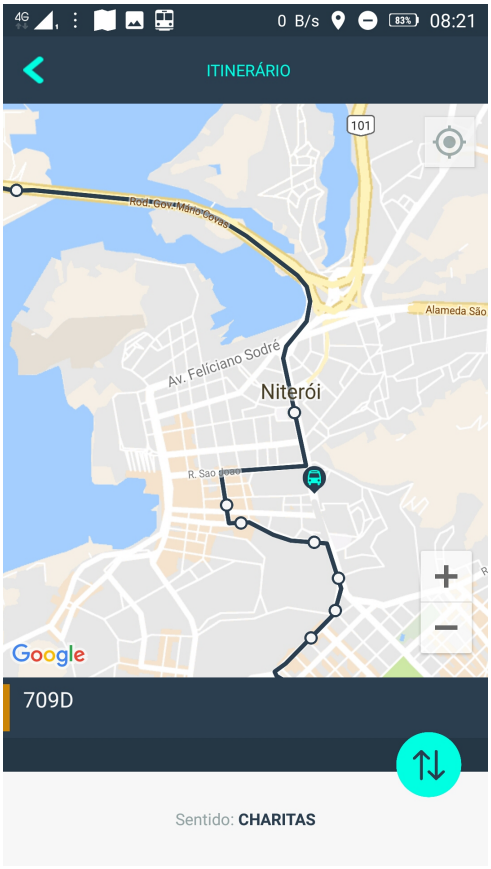
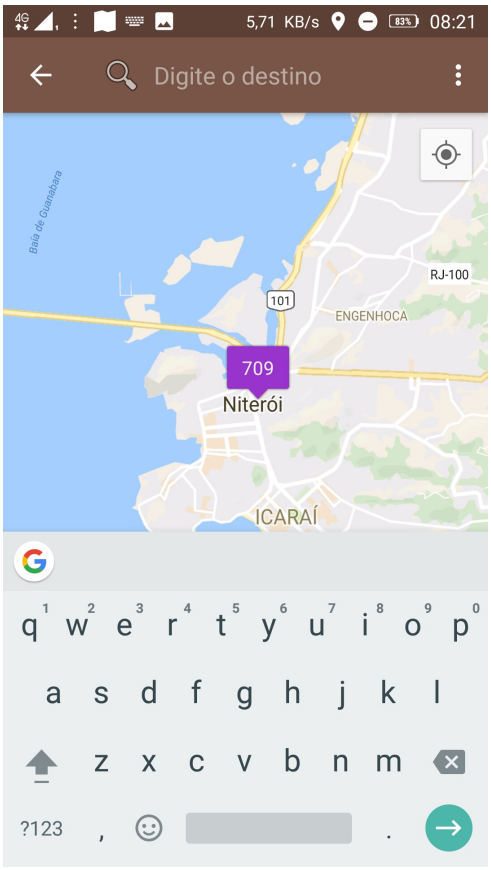


Figura 5.5: Screenshot da tela de localização das linhas que passam pelo ponto do Chega+, início da ponte.



Comparando as localizações do veículo nas Figuras 5.4, 5.6, 5.7 e 5.8, é verificado que o mesmo se deslocou ao longo do tempo, assim constatando que os sistemas tiveram um comportamento esperado. No momento em que o veículo chegou no ponto, foi requisitado sua localização no aplicativo Chega+. Rapidamente o sistema localizou o veículo (Figura 5.9).

Alguns detalhes de usabilidade foram ressaltados pela voluntária: o aplicativo Vá de Ônibus não permite localizar no mapa simultaneamente a localização de veículos de linhas distintas. Também a demora para receber a informação de localização de veículos de uma linha, quando é utilizado uma rede de internet móvel.

Esta última observação motivou a realização de testes de banda, que possivelmente esclarecessem as razões para tal demora. Esta etapa acabou compondo a terceira fase de testes.

Figura 5.6: Screenshot da tela de localização das linhas que passam pelo ponto do Chega+, próximo da ilha do Mocanguê.

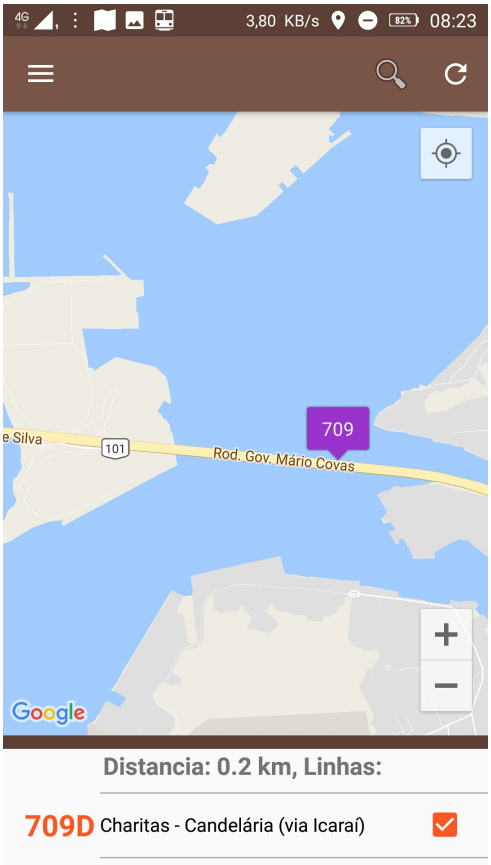


Figura 5.7: Screenshot da tela de localização das linhas que passam pelo ponto do Chega+, próximo da ilha do Mocanguê.

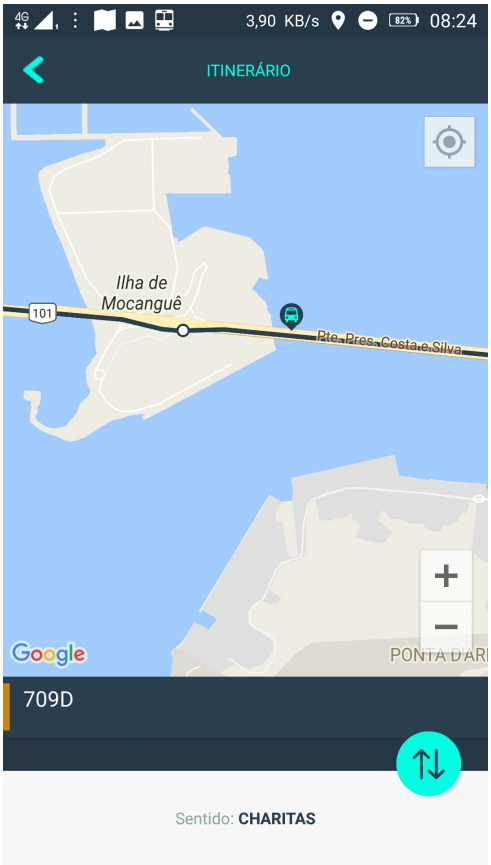


Figura 5.8: Screenshot da tela de localização das linhas que passam pelo ponto do Chega+, próximo da ilha do Mocanguê.

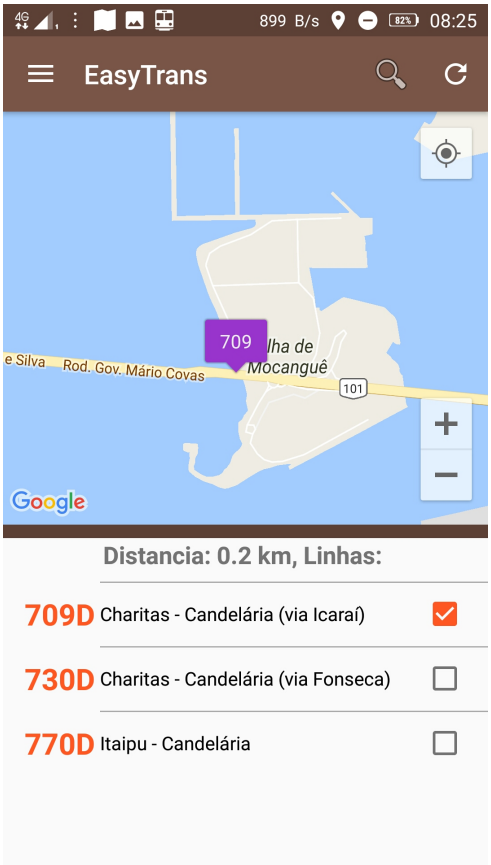
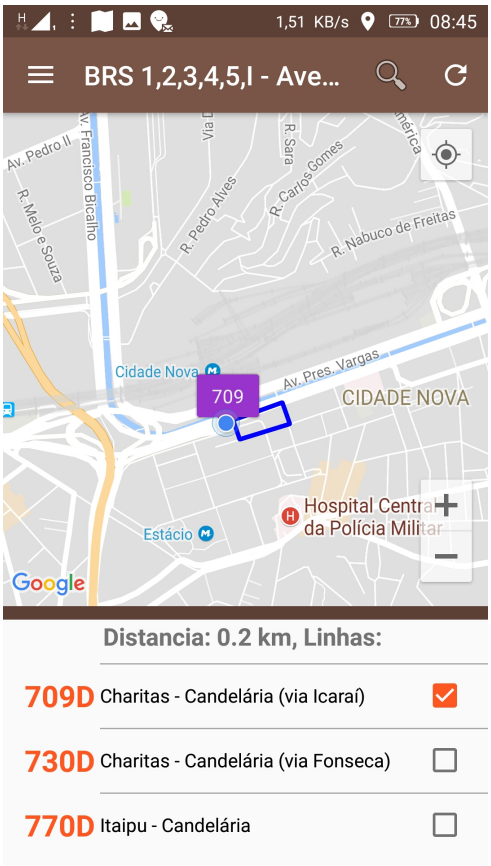


Figura 5.9: Screenshot da tela de localização das linhas que passam pelo ponto do Chega+, chegada ao ponto.



5.3 3ª Fase - Utilização de banda

Durante a fase anterior de teste foi notado que o aplicativo Vá de Ônibus apresentou alguns congelamentos. Este comportamento motivou a realização de uma análise no fluxo de dados das aplicações com a internet.

As aplicações Vá de Ônibus e Chega+ comunicam com serviços externos através da internet utilizando requisição HTTP (setas 1 e 2 da Figura 5.10). Para analisar o fluxo de dados trocados, foi configurado um servidor de proxy, e feita a configuração para o sistema Android se conectar ao proxy para então obter o acesso a internet. As setas 1 e 2 da Figura 5.11, ilustram a conexão dos aplicativos ao proxy, e a seta 3 ilustra a comunicação do proxy com a internet. Com essa configuração os aplicativos têm acesso a internet, e é possível analisar requisições que passam pelo proxy. Para realizar essa análise, foi utilizada a aplicação Charles Proxy [22].

Figura 5.10: Arquitetura padrão da comunicação das aplicações com a internet.

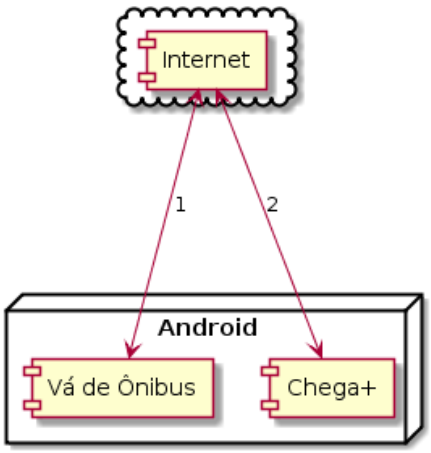
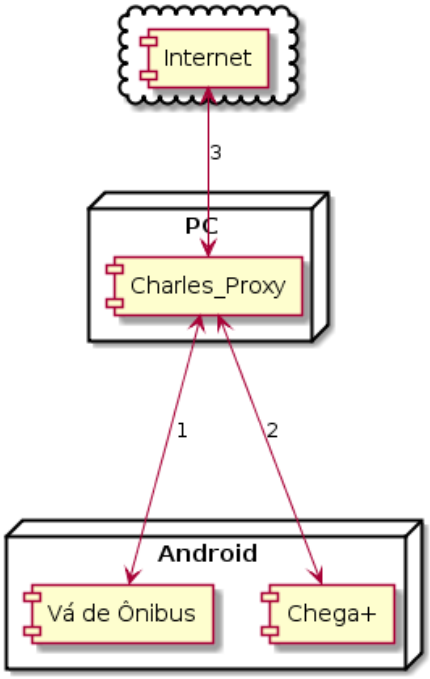


Figura 5.11: Arquitetura da comunicação das aplicações com a internet utilizando um proxy.



Primeiramente foi analisado o fluxo de mensagens durante a inicialização da aplicação Vá de Ônibus. As requisições que o aplicativo realizou para a internet são listadas na tabela 5.6 e detalhadas na tabela 5.7. É observado que algumas requisições se repetem, formando o mesmo caminho, com parâmetro diferentes; sendo, portanto, desnecessárias e redundantes. É o caso dos seguinte pares de requisições: 3 e 5 (obter pontos próximos); e 4 e 6 (buscar linhas próximas). Foi realizado o teste algumas vezes, e foi percebido que a aplicação sempre realiza as requisições 3 e 4 com os mesmos parâmetros "lat=-22.858065&lon=-43.232082", que não possuem relação com a localização demandada pelo usuário.

Com o objetivo de verificar se as requisições 3 e 4 não estariam ocorrendo por um comportamento do GPS do aparelho utilizado, foi feito o mesmo teste com um segundo aparelho. As requisições

Tabela 5.6: Requisições da aplicação Vá de Ônibus ao ser iniciada. Dispositivo Samsung.

Requisição	Caminho
1	obtermunicipiosatendidos?token=17412336481479
2	obterinfoversaoapp?token=17412336481479&plataforma=android&nomeVersao=3.0
3	obterpedproximogps?token=17412336481479&lat=-22.858065&lon=-43.232082&raio=500
4	buscarlinhasproximasgps?token=17412336481479&lat=-22.858065&lon=-43.232082
5	obterpedproximogps?token=17412336481479&lat=-22.9001546&lon=-43.107441&raio=500
6	buscarlinhasproximasgps?token=17412336481479&lat=-22.9001546&lon=-43.107441

interceptadas na aplicação instalada em um segundo dispositivo estão na tabela 5.8. É observado que de fato as requisições 3 e 4 possuem o mesmos parâmetros que no teste anterior. Já as requisições 5 e 6 possuem parâmetros diferentes do teste realizado com o outro dispositivo, desta forma constatando que não é problema de atualização de GPS. Uma hipótese possível é que esses parâmetros estejam no código fonte da aplicação.

Após a inicialização da aplicação, foi dado continuidade ao teste de requisitar a localização dos veículos. Para isso, na aplicação Vá de Ônibus foi selecionado um ponto de ônibus e uma linha que passa por ele. As requisições estão listadas na tabela 5.9 e detalhadas na tabela 5.10.

A resposta a requisição ao item 7 é quais linhas passam pelo ponto com o id especificado no parâmetro. Já ao item 8, são as informações de uma linha específica, ou seja, localização dos veículos, rota e todos os pontos da linha.

Por possuírem muitas informações, notavelmente essas são maiores que as requisições realizadas na inicialização da aplicação.

Ao realizar este tipo de teste com a aplicação proposta neste trabalho, é verificado a realização de somente uma requisição para obter a localização dos veículos de uma linha. A requisição é apresentada na tabela 5.11 e detalhada na tabela 5.12. O tamanho da resposta para essa requisição é muito pequena se comparada com aquela referente a mesma funcionalidade na aplicação Vá de Ônibus (requisição 8 da tabela 5.10).

Essa diferença se dá pela aplicação proposta não obter as localizações dos veículos por meio do consumo de um serviço do tipo RESTful. As localizações chegam na aplicação pelo mecanismo de *push message*. Outra vantagem da aplicacação proposta é a realização de persistência de dados referentes às linhas e pontos, dado que são consumidos através do serviço RESTFul. Dessa maneira não seria necessário obte-las através de requisição HTTP toda vez que o usuário desejasse visualizar o itinerário de uma linha, como acontece na aplicação Vá de Ônibus.

Outro ponto negativo da aplicação Vá de Ônibus é que para todo evento de atualização de localização que o aplicativo recebe do sistema Android, a aplicação dispara requisições para obter linhas e pontos próximos; sem ao menos realizar um filtro para verificar se a nova localização é diferente da anterior e possuem uma distância significativa. Este problema pode ser observado na Figura 5.12.

Tabela 5.7: Detalhes das requisições da aplicação Vá de Ônibus ao ser iniciada. Dispositivo Samsung.

Requisição	**Hora**	**Tamanho da requisição**	**Tamanho da resposta**
1	18:34:58	235 bytes	858 bytes
2	18:34:58	263 bytes	552 bytes
3	18:35:01	268 bytes	895 bytes
4	18:35:01	264 bytes	1.35 KB (1,383 bytes)
5	18:35:02	269 bytes	1.57 KB (1,611 bytes)
6	18:35:02	265 bytes	1.26 KB (1,295 bytes)

Tabela 5.8: Requisições da aplicação Vá de Ônibus ao ser iniciada. Dispositivo LG.

Requisição	**Caminho**
1	obtermunicipiosatendidos?token=17412336481479
2	obterinfoversaoapp?token=17412336481479&plataforma=an&nomeVersao=3.0
4	obterpedproximogps?token=17412336481479&lat=-22.858065&lon=-43.232082&raio=500
3	buscarlinhasproximasgps?token=17412336481479&lat=-22.858065&lon=-43.232082
5	obterpedproximogps?token=17412336481479&lat=-22.9001611&lon=-43.1073937&raio=500
6	buscarlinhasproximasgps?token=17412336481479&lat=-22.9001611&lon=-43.1073937

Tabela 5.9: Requisições da aplicação Vá de Ônibus ao requisitar localização de veículos.

Requisição	**Caminho**
7	obterlinhasdoponto?token=17412336481479&ponto_id=18627932
8	obteritinerarioslinha?token=17412336481479&routeName=18954731&servico=REGULAR

Tabela 5.10: Detalhes das requisições da aplicação Vá de Ônibus ao requisitar localização de veículos.

Requisição	**Hora**	**Tamanho da requisição**	**Tamanho da resposta**
7	18:36:01	247 bytes	1.31 KB (1,344 bytes)
8	18:36:02	267 bytes	6.48 KB (6,631 bytes)

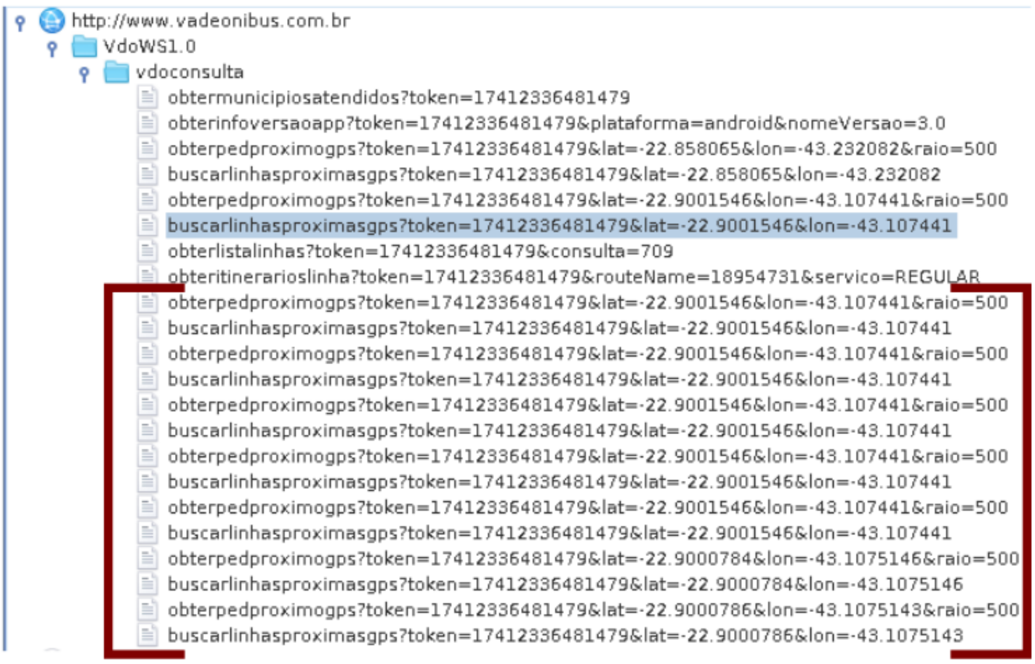
Tabela 5.11: Requisições da aplicação chega+ ao requisitar localização de veículos.

Requisição	**Caminho**
1	requetDevicesLocation?user_id=2&line_number=709D

Tabela 5.12: Detalhes das requisições da aplicação chega+ ao requisitar localização de veículos.

Requisição	**Hora**	**Tamanho da requisição**	**Tamanho da resposta**
1	18:37:01	280 bytes	244 bytes

Figura 5.12: Requisições acionadas pela altualização de localização na aplicação Vá de Ônibus.



Com os resultados do testes de banda, foram criados os gráficos representados pelas Figuras 5.13 e 5.14.

Figura 5.13: Gráfico de consumo de banda da funcionalidade de obter localização de veículos ao abrir o aplicativo.

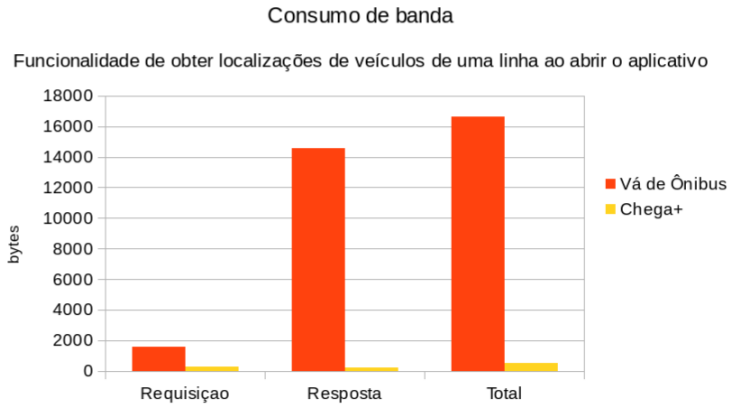
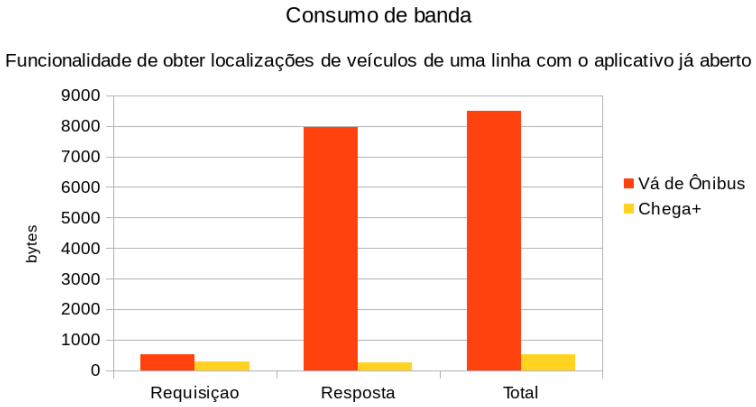


Figura 5.14: Gráfico de consumo de banda da funcionalidade de obter localização de veículos com aplicativo já aberto.



Os gráficos evidenciam que as tecnologias e arquitetura empregadas na implementação do sistema proposto, proporcionam o benefício da aplicação trafegar poucos dados através da internet.

Capítulo 6

Conclusão

Este trabalho apresentou o desenvolvimento de um sistema de acompanhamento de frota desde a fase de definição de arquitetura e modelagem da solução até o desenvolvimento.

Os testes de funcionalidade, demonstraram que o sistema proposto atende aos requisitos definidos no capítulo 2. O teste de comparação evidenciou que a aplicação possui um desempenho fluido. O teste de utilização de banda, comprovou que o sistema proposto trafega uma quantidade de dados pequena através da internet, e quando comparada a outra solução, a solução proposta chega a consumir no mínimo dezesseis vezes menos, podendo chegar até trinta vezes menos. Isso evidencia o quanto as tecnologias e arquitetura empregadas trazem benefícios. Que apesar de existirem outros aplicativos que fornecem funcionalidades semelhantes, as mesmas são eficientemente implementadas no sistema proposto.

A grande potencialidade do trabalho aqui apresentado seria facilitar a dinâmica de mobilidade urbana, tópico urgente nas cidades brasileiras.Especificamente para uma cidade onde necessita implementar uma solução para melhoria no transporte com um baixo investimento, é considerado que a implementação prática do sistema proposto poderia decorrer, em médio prazo, não só em uma melhoria das condições de vida da população residente, mas também na atração de um contingente turístico maior de classes baixa e média, que levam em consideração a disponibilidade de transporte público no momento de eleger em qual cidade passar férias. Este possível fator de atração turística é especialmente relevante em um contexto de crise econômica como a que o país tem vivenciado. Possivelmente, o Chega+ (e eventuais melhorias na organização das cooperativas facilitadas por seu uso) serviria como um diferencial regional neste sentido.

Além de tudo, a disponibilização deste sistema representaria um retorno da comunidade científica da UFF para a sociedade, contribuindo para o estreitamento dos laços de integração com a instituição e a efetivação da responsabilidade social da universidade pública.

Apêndice A

Descrição de caso de uso

Tabela 1.1: Descrição e passo a passo do caso de uso ”Visualizar localização de veículos”.

Código:	UC-01
Nome:	Visualizar localização de veículos.
Descrição:	Este caso de uso aborda o caminho percorrido para visualizar a localização de veículos.
Atores:	Usuário comum.
Pré-condições:	Ter acesso à internet.
Pós-condições:	Usuário ter informações de localização de veículos.
Fluxo principal	
1 - Usuário toca no botão de gaveta de navegação	
2 - O sistema mostra as opções de navegação.	
3 - Usuário escolhe a opção ”Pontos”.	
4 - O sistema mostra um mapa com os marcadores identificando as localizações dos pontos.	
5 - O usuário escolhe um ponto e o seleciona.	
6 - Sistema apresenta informações sobre o ponto: endereço, distância até o mesmo, uma lista com as linha que passam pelo ponto.	
7 - Usuário seleciona a linha que pertence os veículos da qual deseja informação de localização.	
8 - O sistema mostra sobre o mapa marcadore(s) posicionado(s) sobre a localização do(s) veículo(s).	

Tabela 1.2: Descrição e passo a passo do caso de uso ”Cadastrar veículo”.

Código:	UC-02
Nome:	Cadastrar Veículo.
Descrição:	Este caso de uso aborda o caminho percorrido para cadastrar um veículo.
Atores:	Usuário motorista de empresa
Pré-condições:	Ter acesso à internet.
Pós-condições:	Usuário ter veículo cadastrado.
Fluxo principal	
1 - Usuário motorista preenche dados solicitados pelo sistema.	
2 - Sistema verifica se existe algum veículo criado com as informações fornecidas pelo usuário motorista. **(E1)**	
3 - Sistema cadastra veículo.	
Fluxo de exceção	
E1	1 - Sistema informa que já existe veículo com as informações fornecidas.

Tabela 1.3: Descrição e passo a passo do caso de uso ”Iniciar rota da linha”.

Código:	UC-03
Nome:	Iniciar rota da linha.
Descrição:	Este caso de uso aborda o caminho percorrido para iniciar rota da linha.
Atores:	Usuário motorista de empresa
Pré-condições:	Ter acesso à internet.
Pós-condições:	Ter a rota configurada.
Fluxo principal	
1 - Usuário toca no botão de gaveta de navegação	
2 - O sistema mostra as opções de navegação.	
3 - Usuário escolhe a opção ”Linha”.	
4 - Usuário preenche os campos ”Ramificação”e ”Sentido”.	
5 - O sistema utiliza os dados preenchidos para configurar a rota.	

Apêndice B

Diagrama de projeto componente

App motorista

B.1 Pacote data

Esse pacote possui três subpacotes: *server.source, server.source.model e shareprefs* organizados conforme a estrutura a seguir:

```
data
├── server.source
├── server.source.model
└── shareprefs
```

A seguir segue o diagrama do pacote ***data.server.source***.

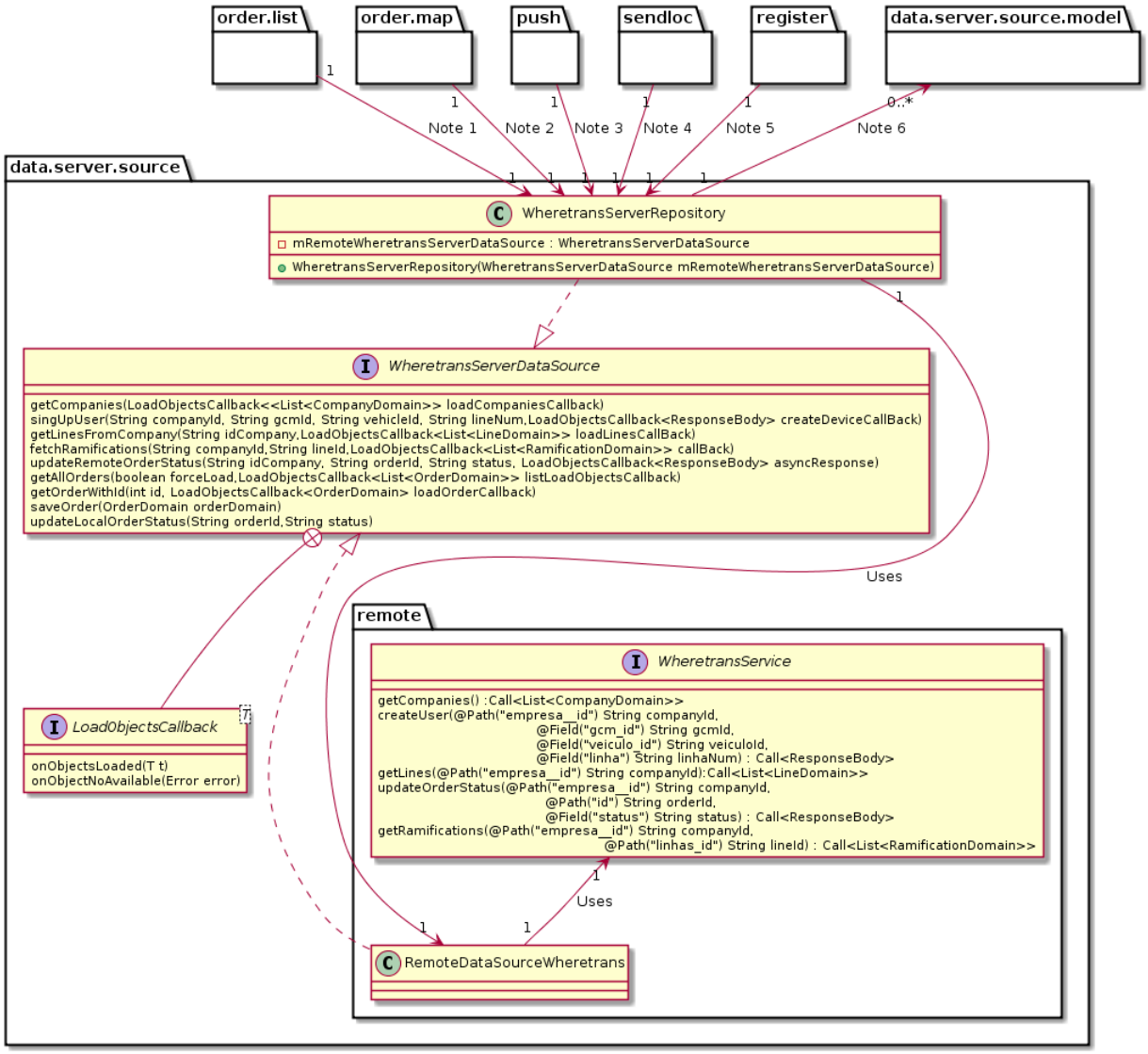


Figura B.1: Diagrama de classes do pacote data.server.source

- ***Note 1 :*** A classe OrderListPresenter do pacote order.list utiliza a classe WheretransServerRepository.
- ***Note 2 :*** A classe OrderMapPresenter do pacote order.map utiliza a classe WheretransServerRepository.
- ***Note 3 :*** A classe MyPushListenerService do pacote push utiliza a classe WheretransServerRepository.
- ***Note 4 :*** A classe SendLocPresenter e GetLocationService do pacote sendlog utiliza a classe WheretransServerRepository.
- ***Note 5 :*** A classe RegisterPresenter do pacote register utiliza a classe WheretransServerRepository.
- ***Note 6 :*** A classe WheretransServerRepository utiliza as classes do pacote model.

A seguir segue o diagrama do pacote ***data.server.source.model***.

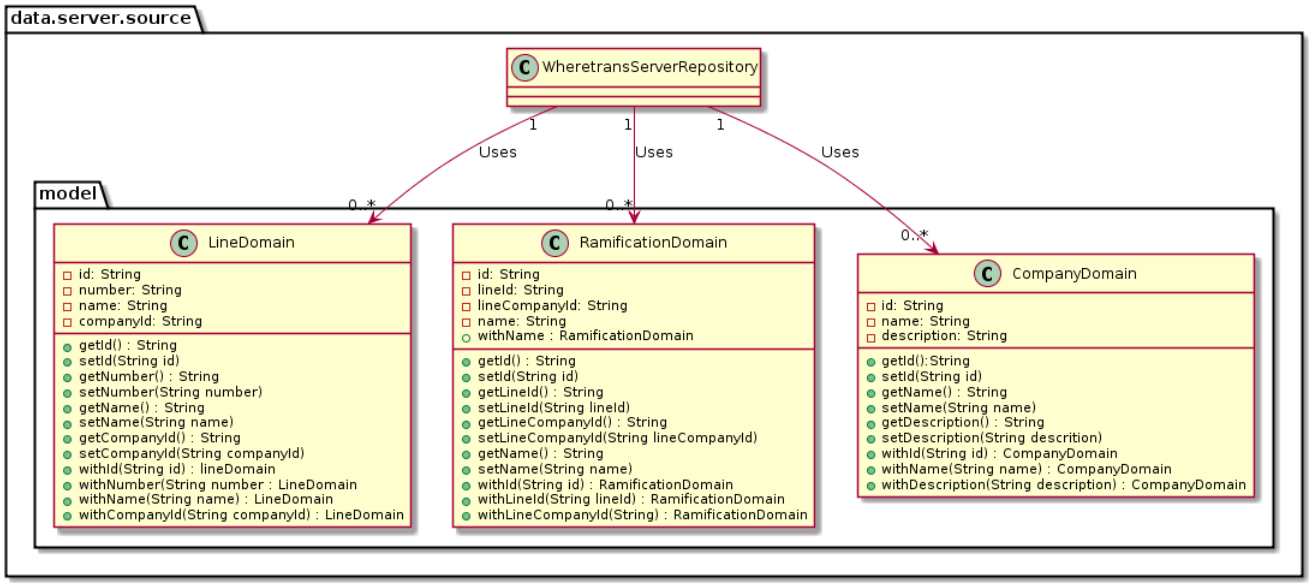


Figura B.2: Diagrama de classes do pacote data.server.source.model

A seguir segue o diagrama do pacote ***data.shareprefs***.

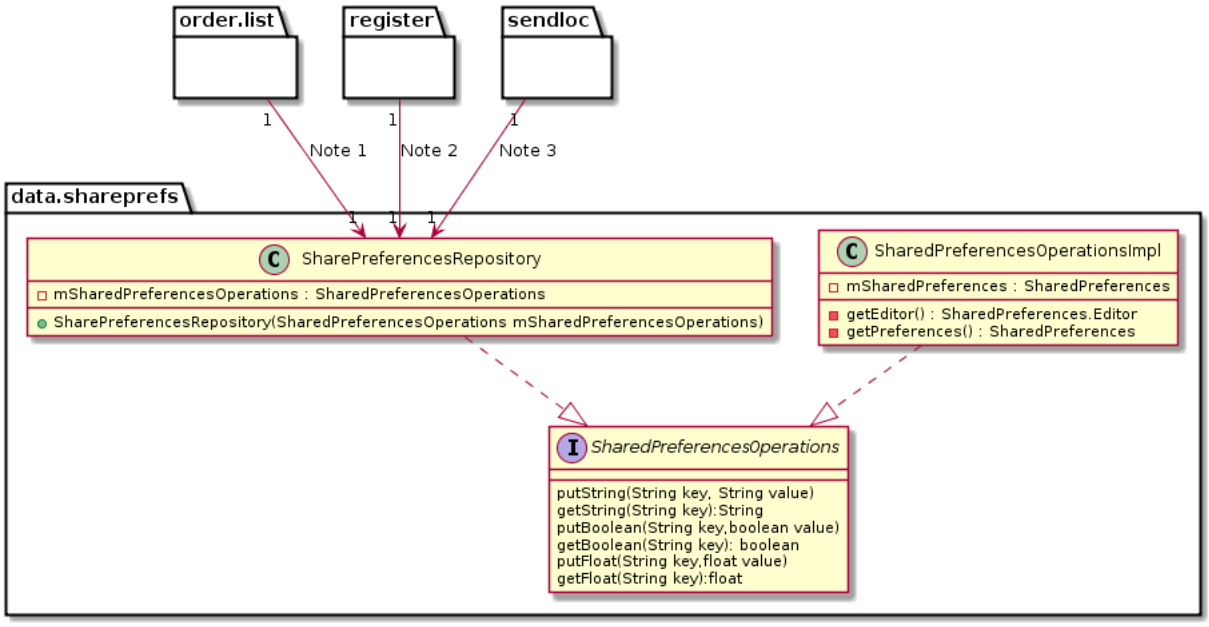


Figura B.3: Diagrama de classes do pacote data.shareprefs

- ***Note 1*** **:** A classe OrderListPresenter do pacote order.list utiliza a classe SharePreferencesRepository.
- ***Note 2*** **:** As classes RegisterFragment, RegisterPresenter e RegistrationIntentService do pacote register utilizam a classe SharePreferencesRepository.
- ***Note 3*** **:** As classes SendLocFragment, SendLocPresenter e GetLocationService do pacote sendloc utilizam a classe SharePreferencesRepository.

B.2 Pacote push

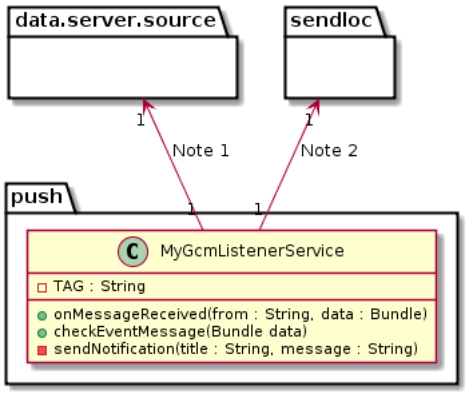


Figura B.4: Diagrama de classes do pacote push

- ***Note 1*** **:** A classe MyPushListenerService utiliza a classe WheretransServerRepository do pacote data.server.source para salvar os dados do pedido no banco de dados.
- ***Note 2*** **:** A classe MyPushListenerService utiliza a classe GetLocationService do pacote sendloc.

B.3 Pacote register

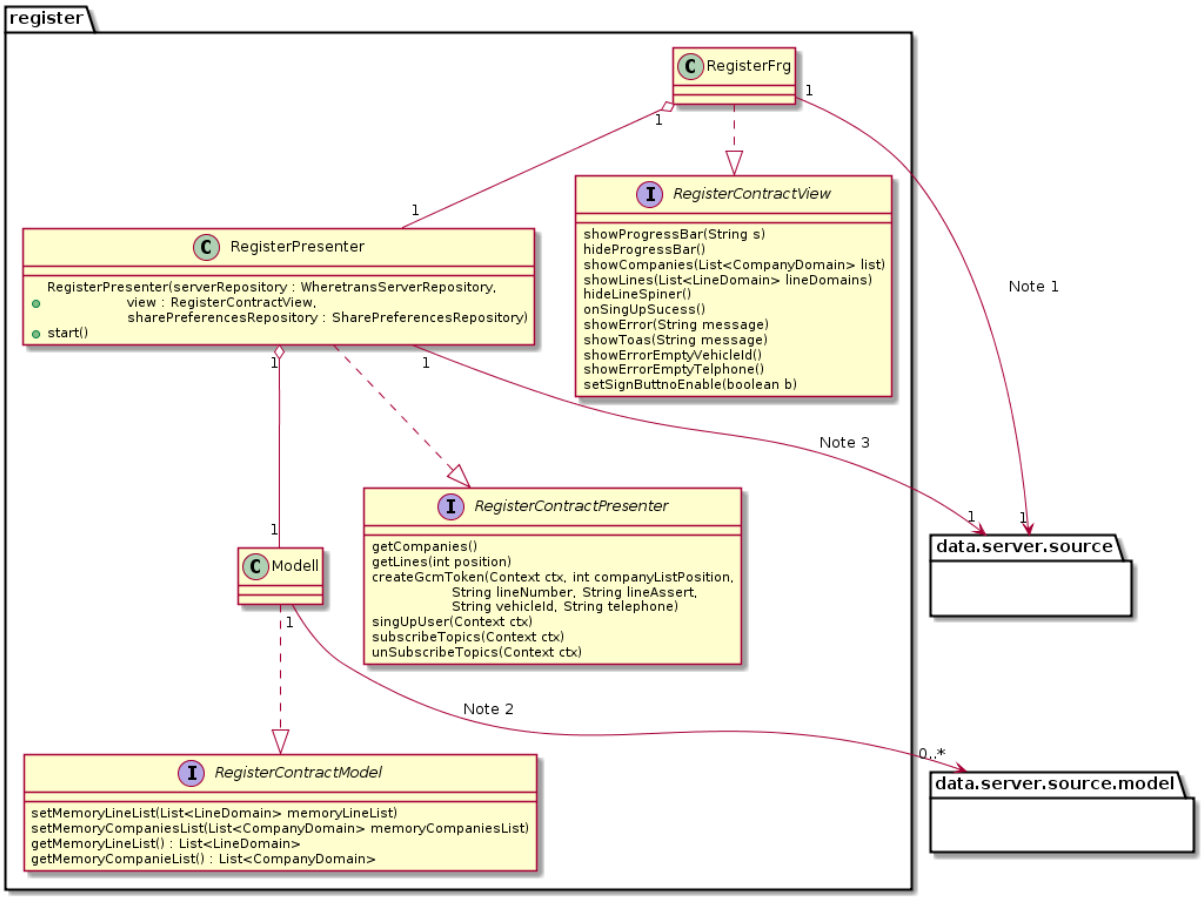


Figura B.5: Diagrama de classes do pacote register

- ***Note 1*** **:** A classe RegisterFrg utiliza a classe SharePreferencesRepository do pacote *data.server.source*.
- ***Note 2*** **:** A classe Model utiliza as classes CompanyDomain e LineDomain do pacote *data.server.source.model*.
- ***Note 3*** **:** A classe RegisterPresenter utiliza as classes WheretransServerRepository e SharePreferencesRepository do pacote *data.server.source*.

B.4 Pacore sendloc

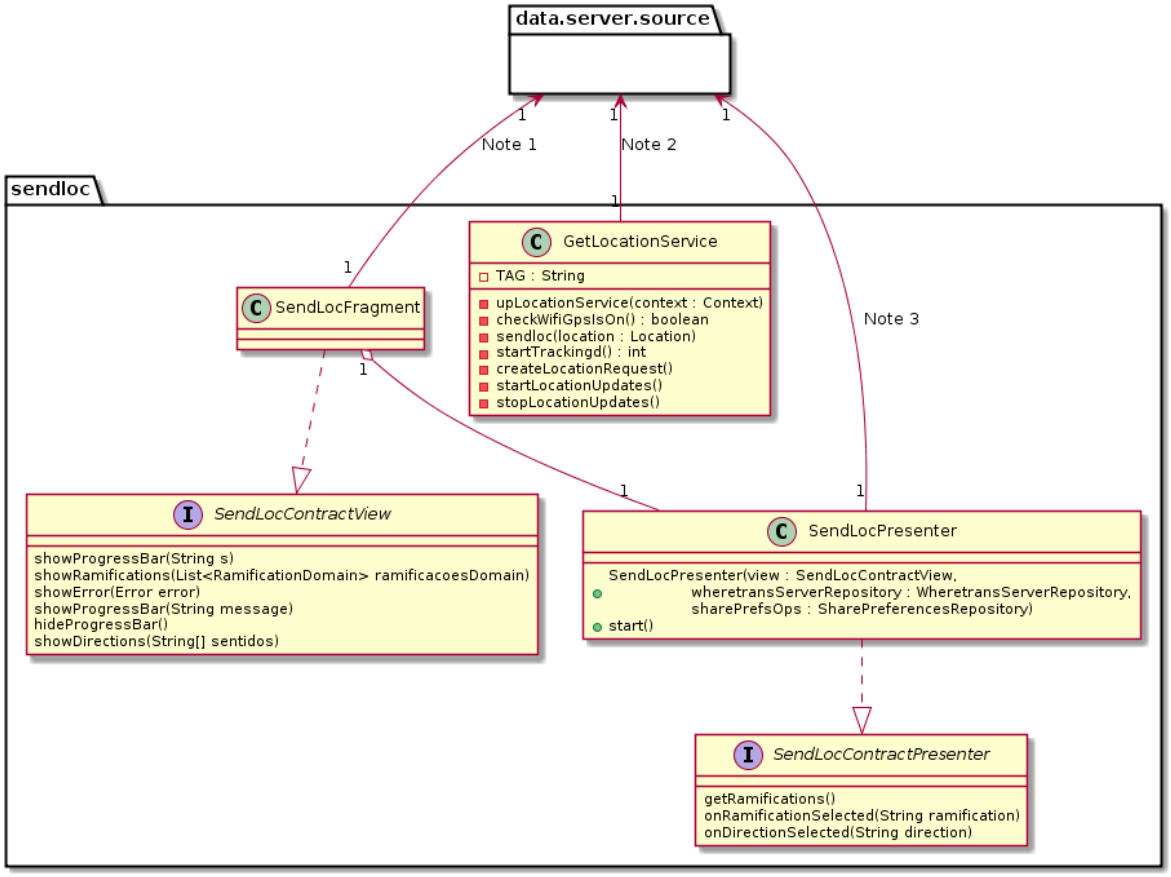


Figura B.6: Diagrama de classes do pacote sendloc

- ***Note 1*** **:** A classe GetLocationService utiliza as classes WheretransServerRepository e SharePreferencesRepository do pacote *data.server.source*.
- ***Note 2*** **:** A classe SendLocFragment utiliza a classe SharePreferencesRepository *data.server.source*.
- ***Note 3*** **:** A classe SendLocPresenter utiliza as classes WheretransServerRepository e SharePreferencesRepository do pacote *data.server.source*.

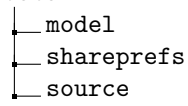
Apêndice C

Diagrama de projeto componente

App usuário final

C.1 Pacote data

Para melhor visualização vamos dividir este pacote em três diagramas.



A seguir segue o diagrama do pacote ***data.model***.

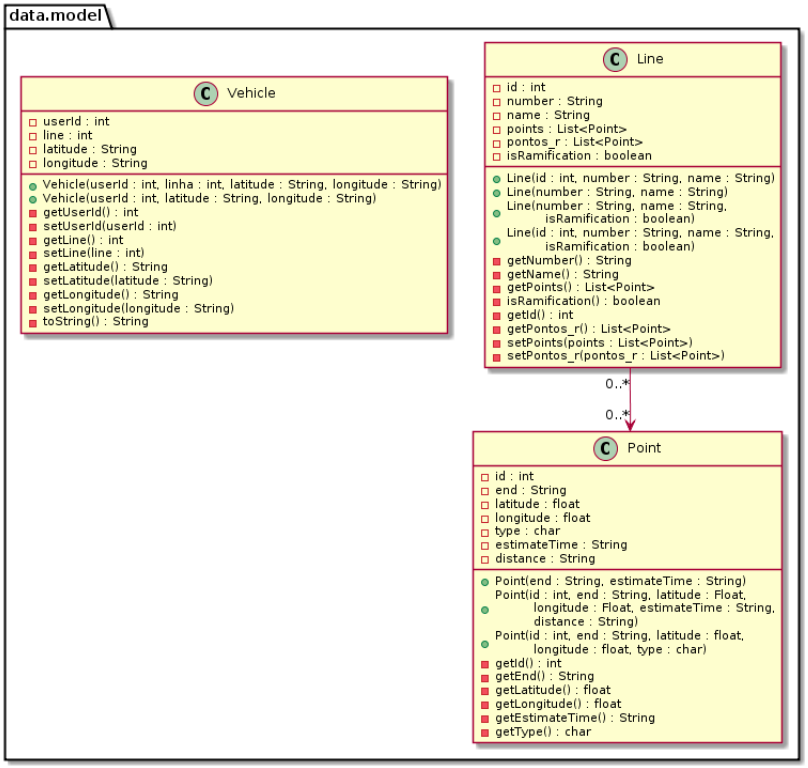


Figura C.1: Diagrama de classes do *data.model*

A seguir segue o diagrama do pacote ***data.shareprefs***.

Como podemos observar na figure C.2 temos algumas notas, estas serão apresentadas abaixo:

- ***Note 1:*** A classe SearchSuggestionProvider do pacote *search.provider* utiliza a classe SharePreferencesRepository.
- ***Note 2:*** A classe PointsMapFrg do pacote *points.map* utiliza a classe SharePreferencesRepository.
- ***Note 3:*** A classe SyncDataPresenter do pacote *sync* utilizam a classe SharePreferencesRepository.
- ***Note 4:*** A classe RegistrationIntentService do pacote *push* utilizan a classe SharePreferencesRepository.

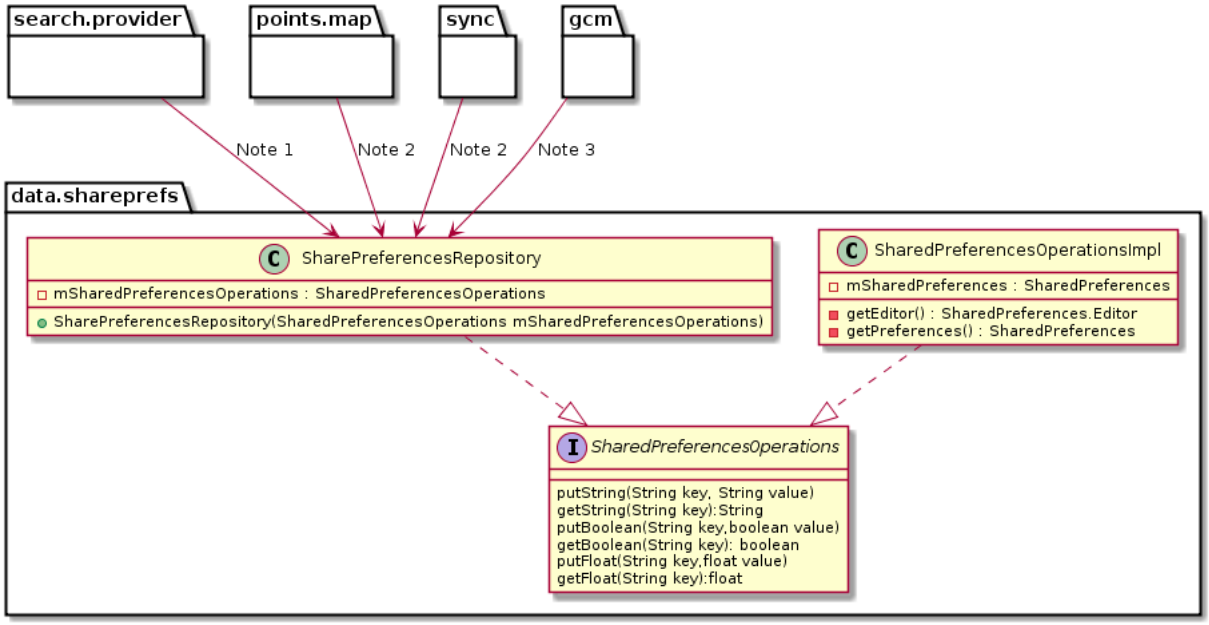


Figura C.2: Diagrama de classes do pacote *data.shareprefs*

A seguir segue os detalhes do pacote *data.source*. Para melhor visualização vamos dividir este pacote em oito sub-pacotes com seus respectivos diagramas apresentados a seguir.

```
data.source
├── companies
├── gautocomplete
├── linehaspoints
├── lines
├── orders
├── points
├── ramifications
└── user
```

Pacote *data.source.companies*:

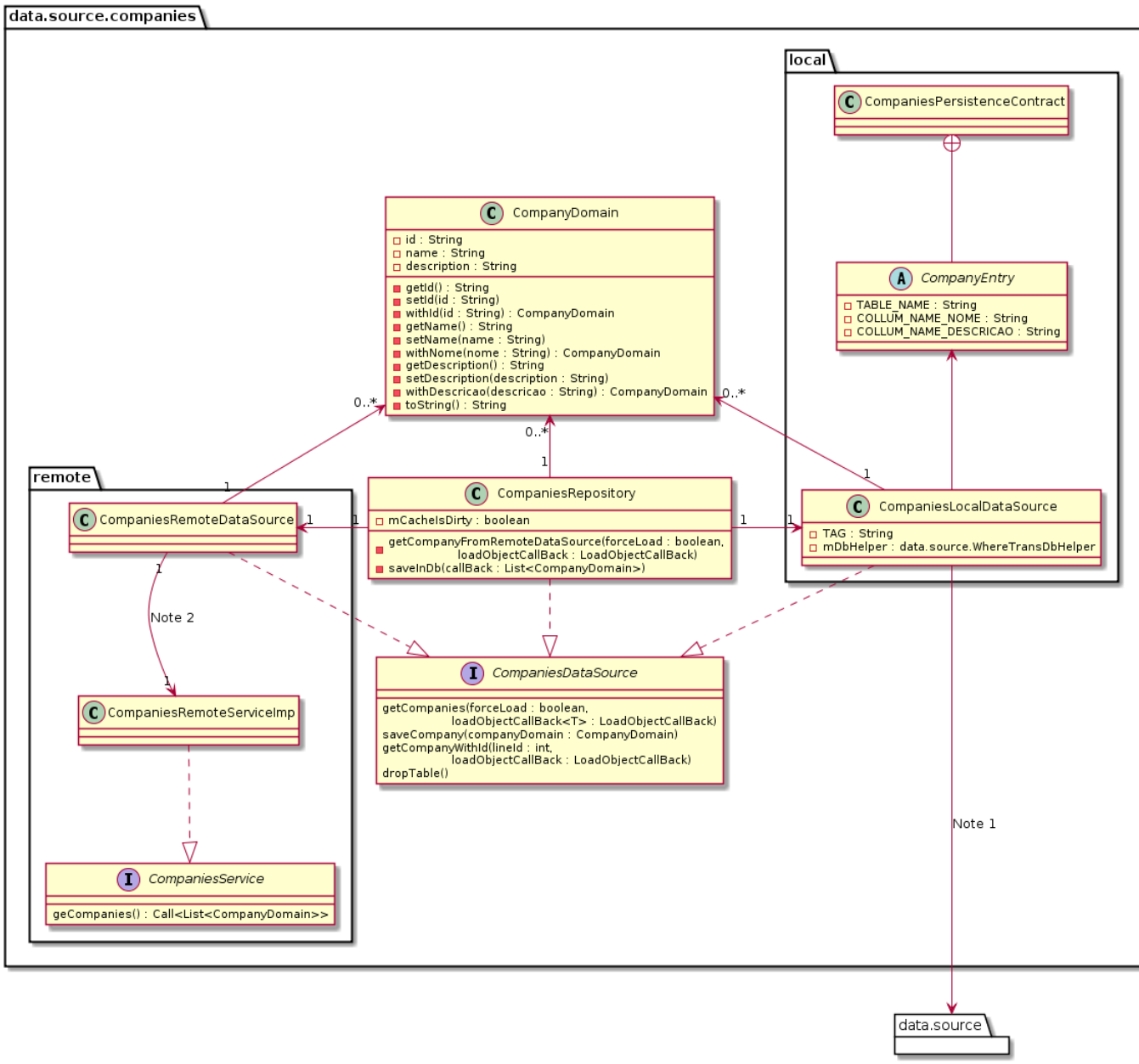


Figura C.3: Diagrama de classes do pacote *data.source.companies*

- ***Note 1:*** A classe CompaniesLocalDataSource do pacote *data.source.companies.local* utiliza a classe WhereTransDbHelper do pacote *data.source* .

Pacote *data.source.gautocomplete*:

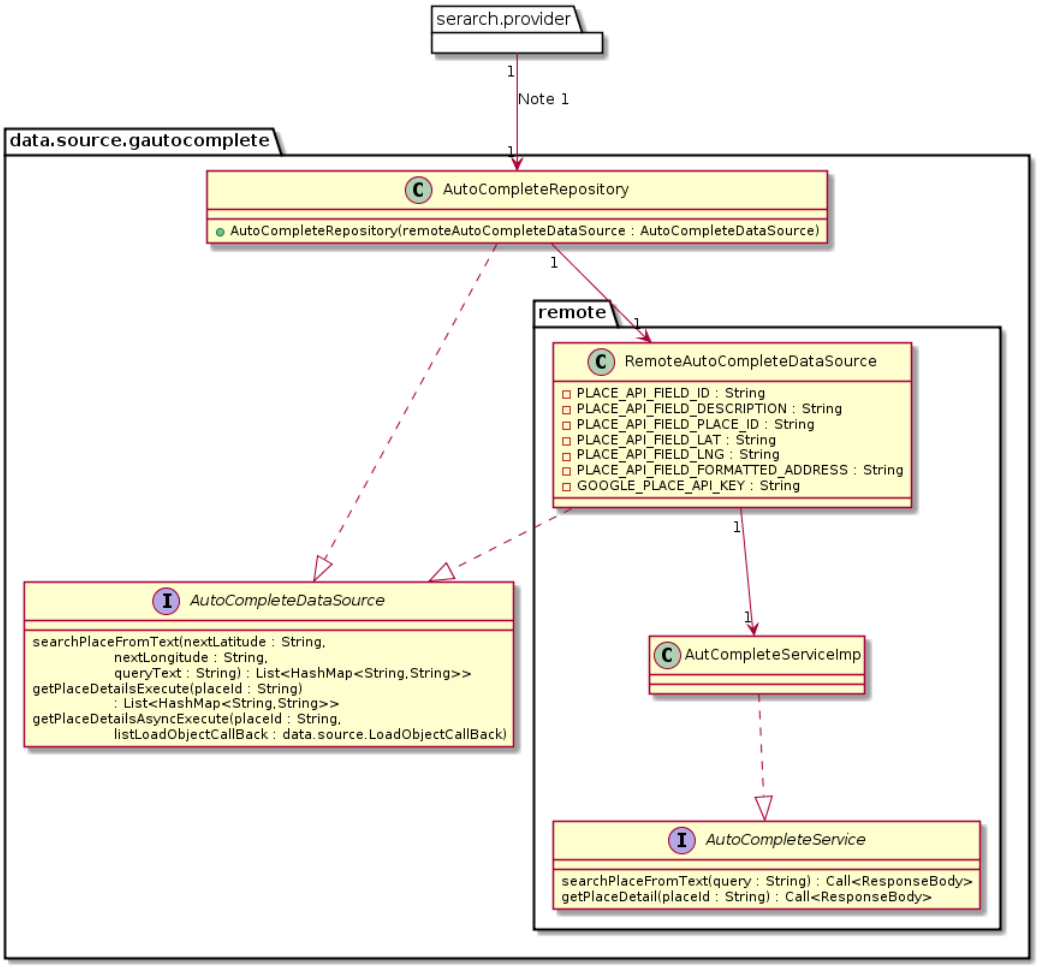


Figura C.4: Diagrama de classes do pacote *data.source.gautocomplete*

- ***Note 1:***A classe SerarchSuggestionProvider do pacote *serarch.provider* utiliza a classe AutoCompleteRepository do pacote *data.source.gautocomplete* .

Pacote *data.source.linehaspoints*:

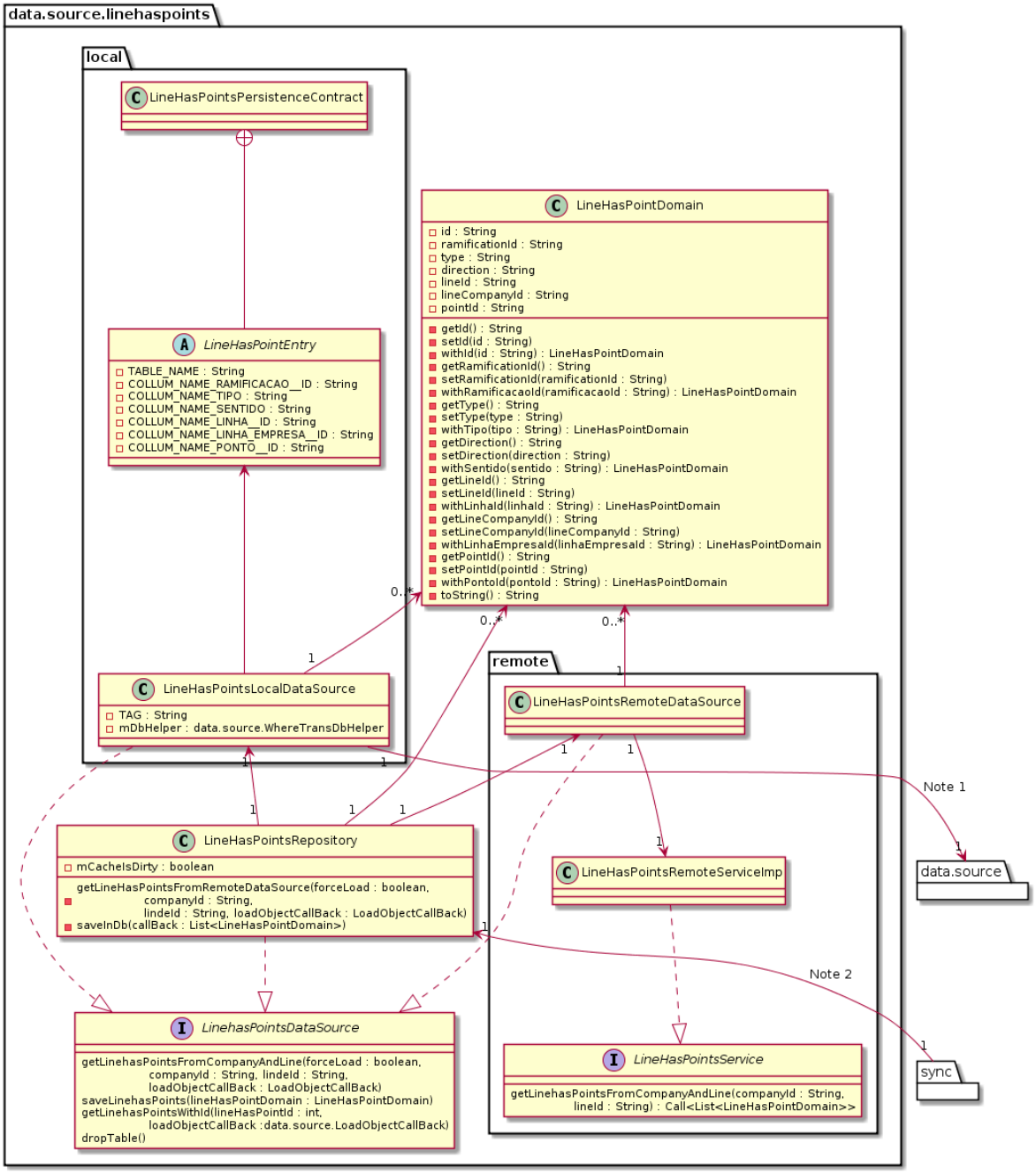


Figura C.5: Diagrama de classes do pacote *data.source.linehaspoints*

- ***Note 1:*** A classe LineHasPointsLocalDataSource do pacote *data.source.linehaspoints.local* utiliza a classe WhereTransDbHelper do pacote *data.source* .
- ***Note 2:*** A classe SyncDataPresenter do pacote *sync* utiliza a classe LineHasPointsRepository .

Pacote *data.source.lines*:

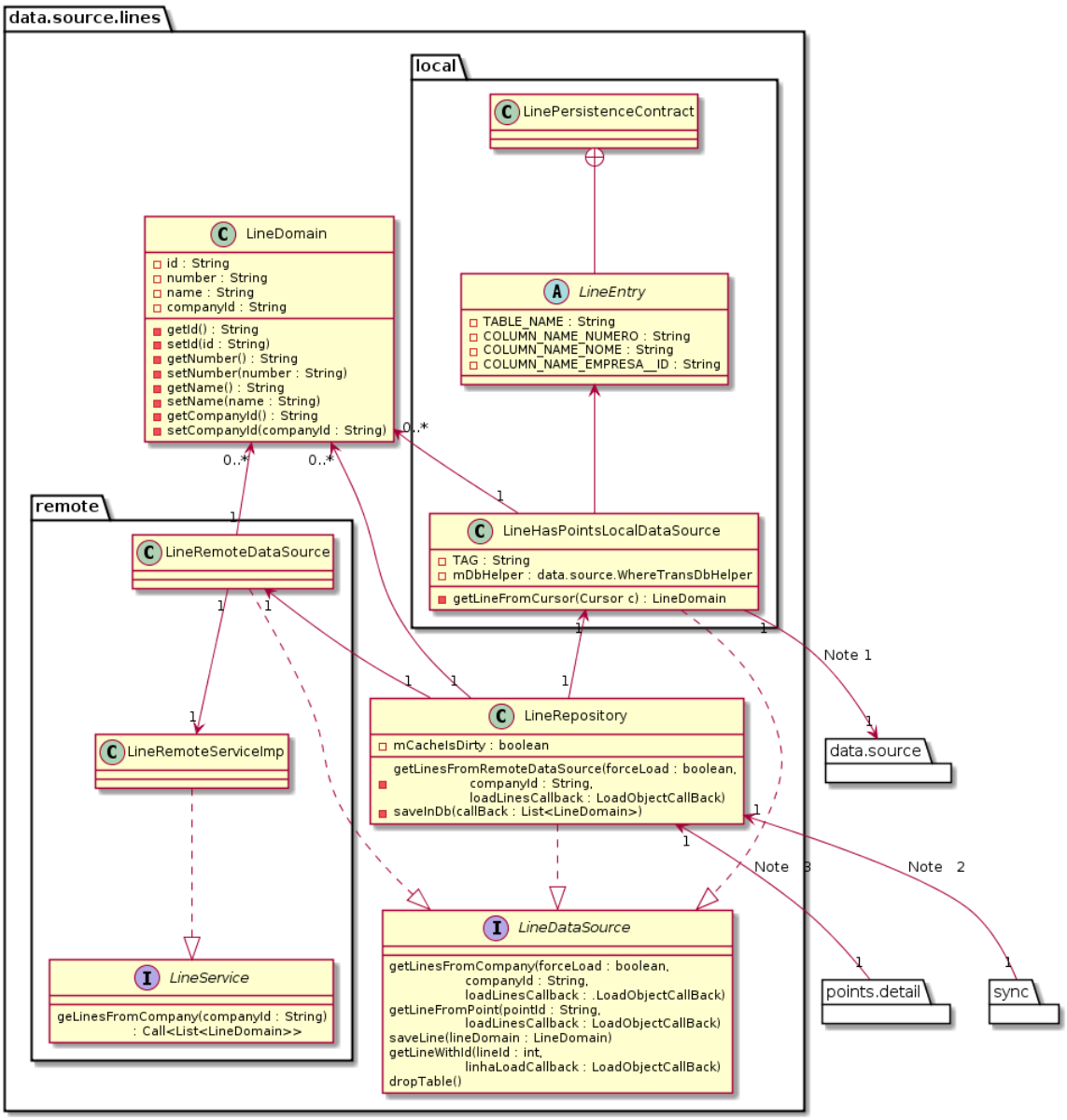


Figura C.6: Diagrama de classes do pacote *data.source.lines*

- ***Note 1:*** A classe LineHasPointsLocalDataSource utiliza a classe WhereTransDbHelper do pacote *data.source*
- ***Note 2:***A classe SyncDataPresenter do pacote *sync* utiliza a classe LineRepository.
- ***Note 3:***A classe PointDetailPresenter do pacote *points.detail* utiliza a classe LineRepository.

Pacote *data.source.points*:

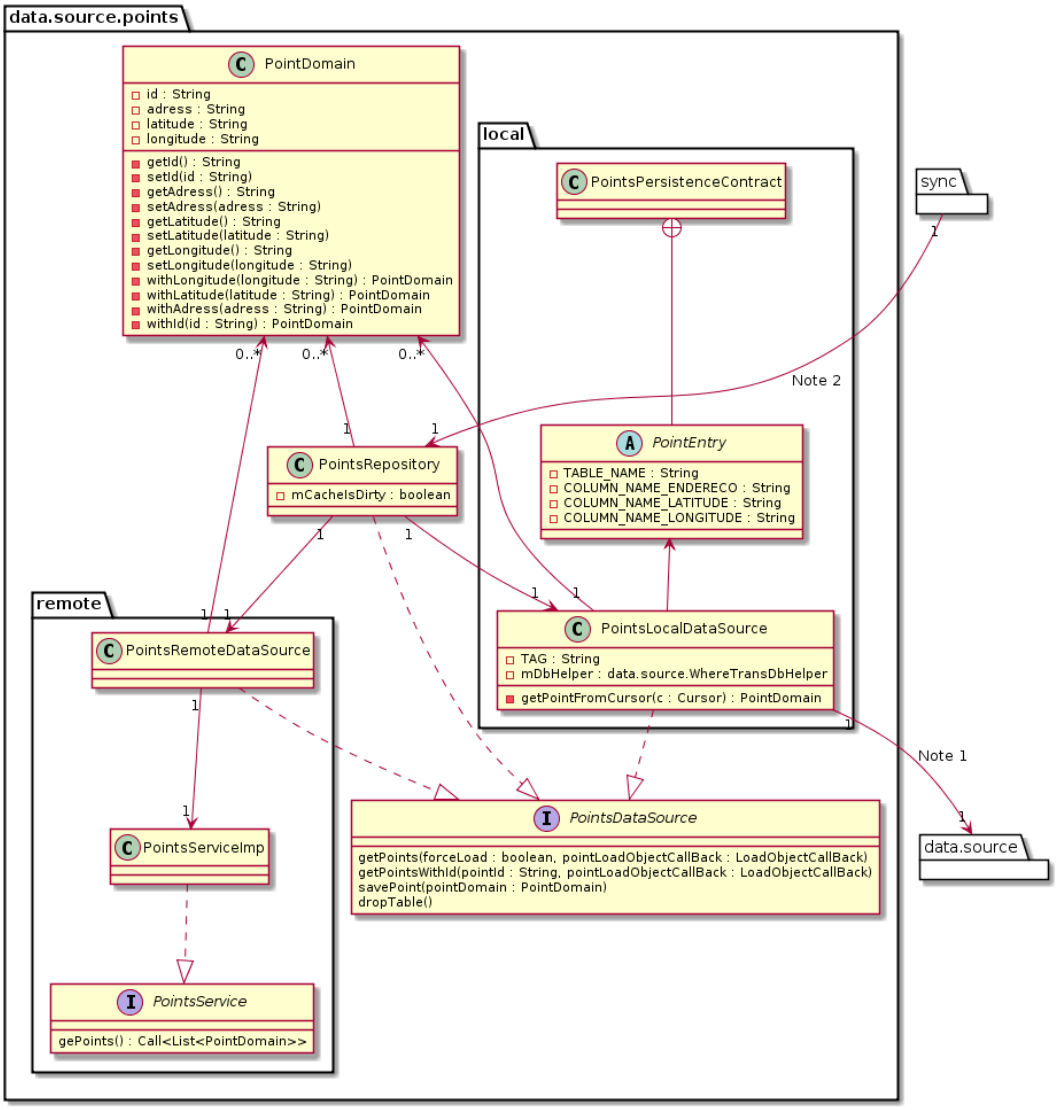


Figura C.7: Diagrama de classes do pacote *data.source.points*

- ***Note 1:*** A classe PointsLocalDataSource utiliza a classe WhereTransDbHelper do pacote *data.source* .

- ***Note 2:*** A classe SyncDataPresenter do pacote *sync* utiliza a classe PointsRepository .

Pacote *data.source.ramifications*:

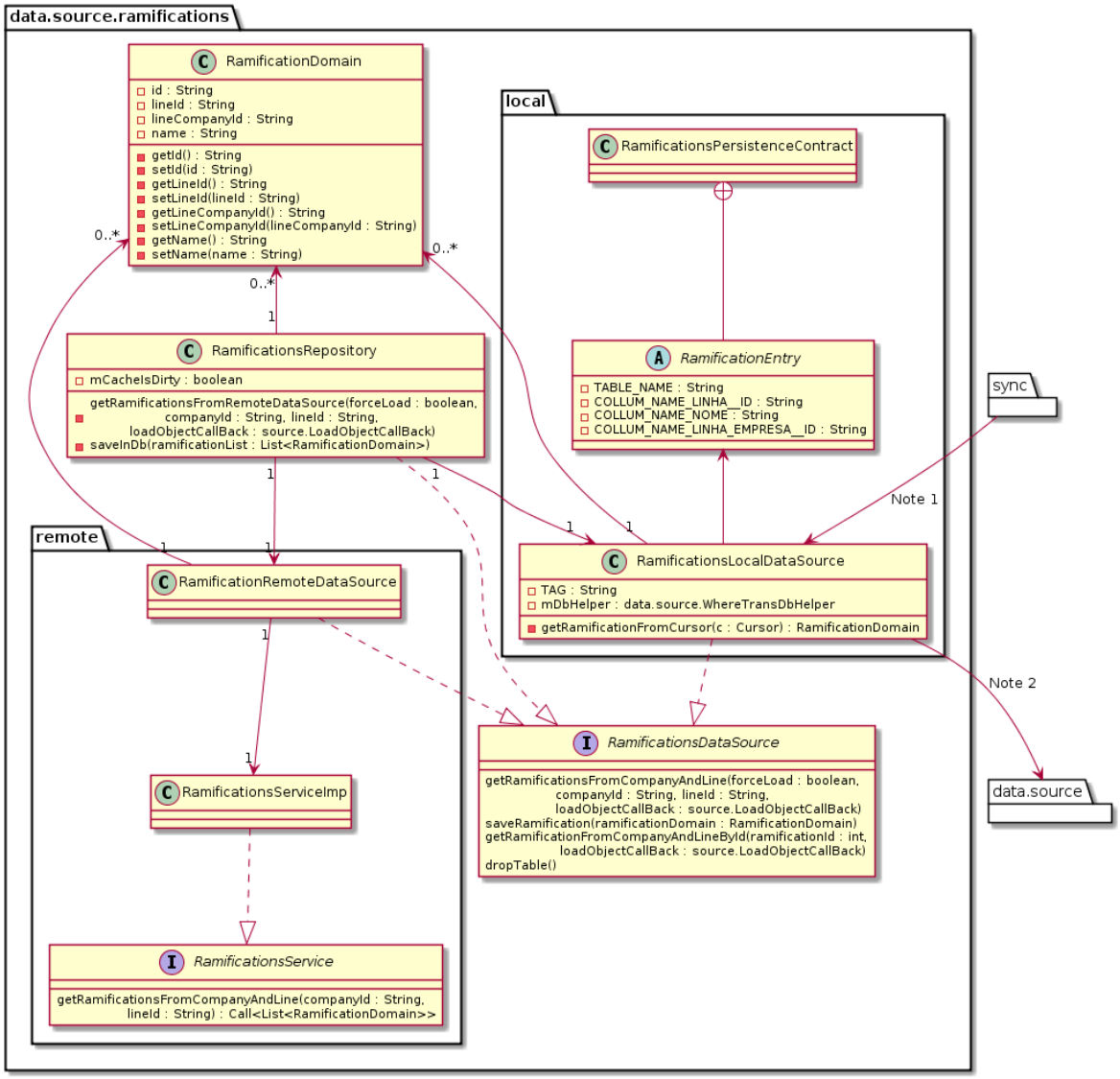


Figura C.8: Diagrama de classes do pacote *data.source.ramifications*

- ***Note 1:*** A classe RamificationsLocalDataSource utiliza a classe WhereTransDbHelper do pacote *data.source* .
- ***Note 2:*** A classe SyncDataPresenter do pacote *sync* utiliza a classe RamificationsRepository .

Pacote *data.source.users*:

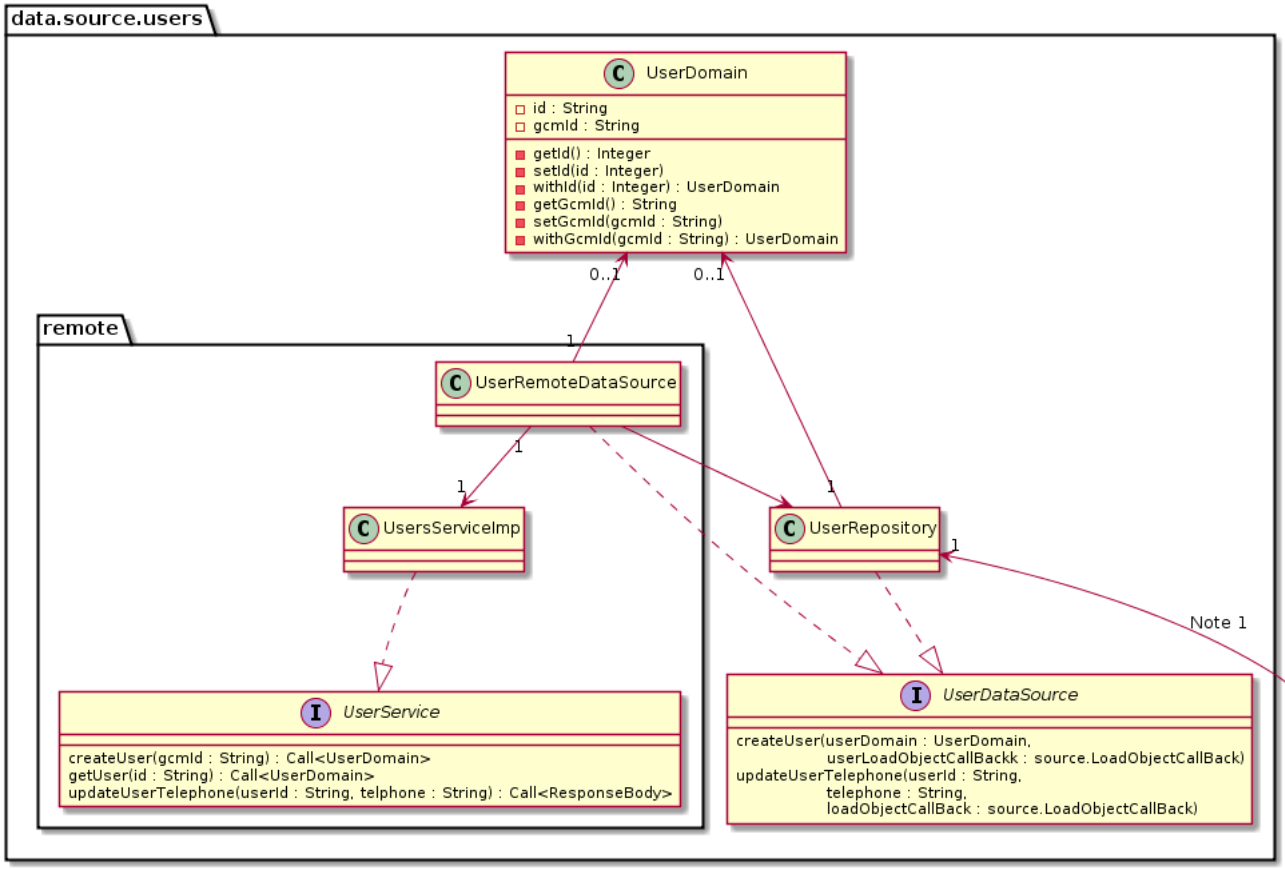


Figura C.9: Diagrama de classes do pacote *data.source.users*

- ***Note 1:*** A classe RegistrationService do pacote *sync* utiliza a classe UserRepository.

C.2 Pacote *push*

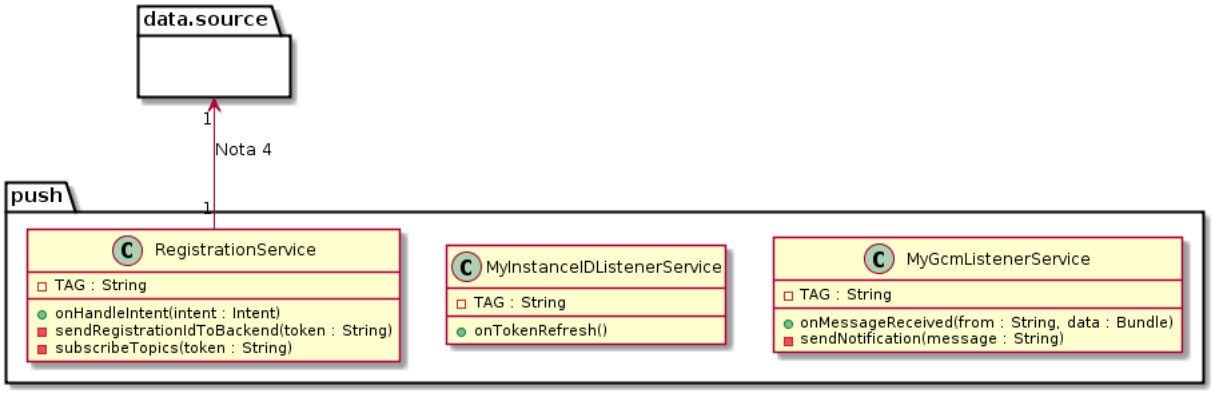


Figura C.10: Diagrama de classes do pacote *push*

- ***Note 1:*** A classe RegistrationService utiliza a classe WhereTransDbHelper do pacote *data.source* .

C.3 Pacote *lines*

Para melhor visualização vamos dividir este pacote em dois diagramas.

```
lines
├── detail
└── list
```

A seguir segue o diagrama do pacote ***lines.detail***.

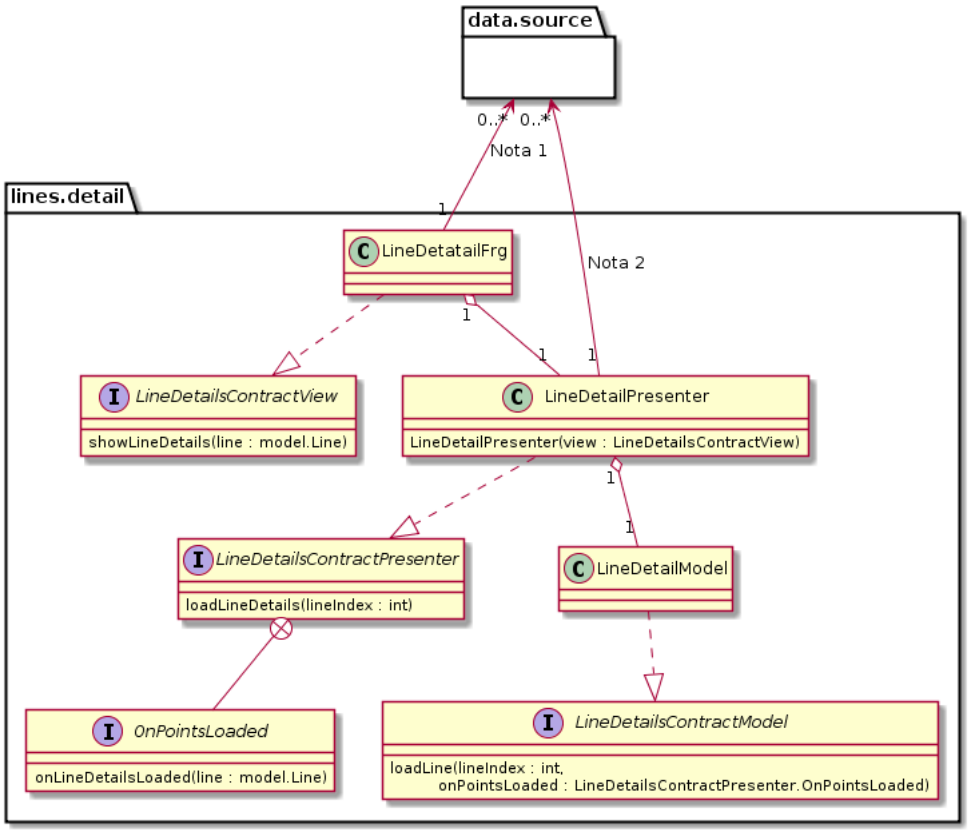


Figura C.11: Diagrama de classes do *lines.detail*

- ***Note 1:*** A classe LineDetailFrg utiliza a classe Line do pacote *data.model* .
- ***Note 2:*** A classe LineDetailPresenter utiliza a classe Line do pacote *data.model* .

A seguir segue o diagrama do pacote ***lines.list***.

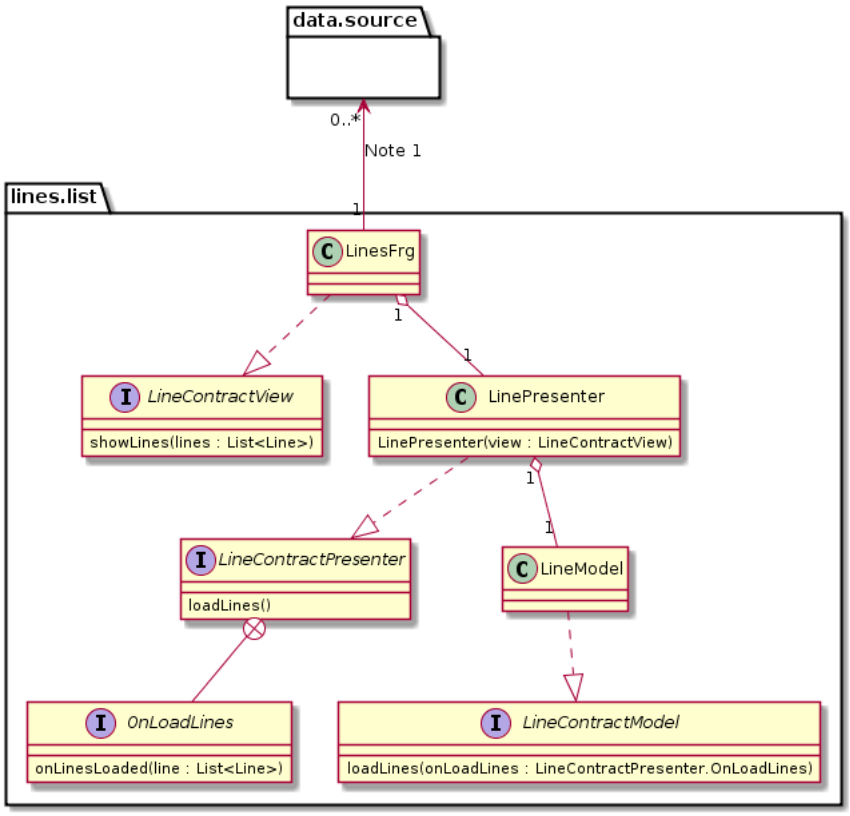


Figura C.12: Diagrama de classes do *lines.list*

- ***Note 1:*** A classe LinesFrg utiliza a classe Line do pacote *data.model* .

C.4 Pacote *points*

Para melhor visualização vamos dividir este pacote em dois diagramas.

```
points
├── details
└── map
```

A seguir segue o diagrama do pacote ***points.detail***.

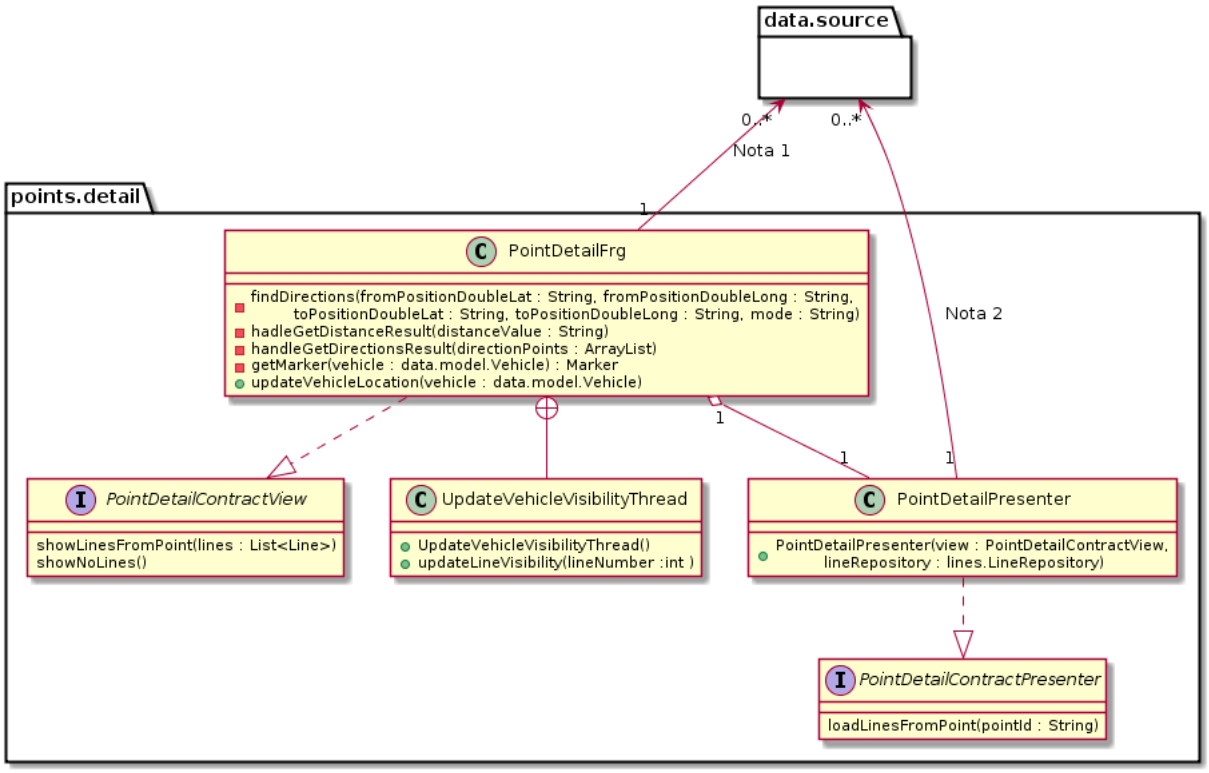


Figura C.13: Diagrama de classes do *points.detail*

- ***Note 1:*** A classe PointDetailFrg utiliza as classes Line e Vehicle do pacote *data.model* .
- ***Note 2:*** A classe PointDetailPresenter utiliza a classe Line do pacote *data.model* .

A seguir segue o diagrama do pacote ***points.map***.

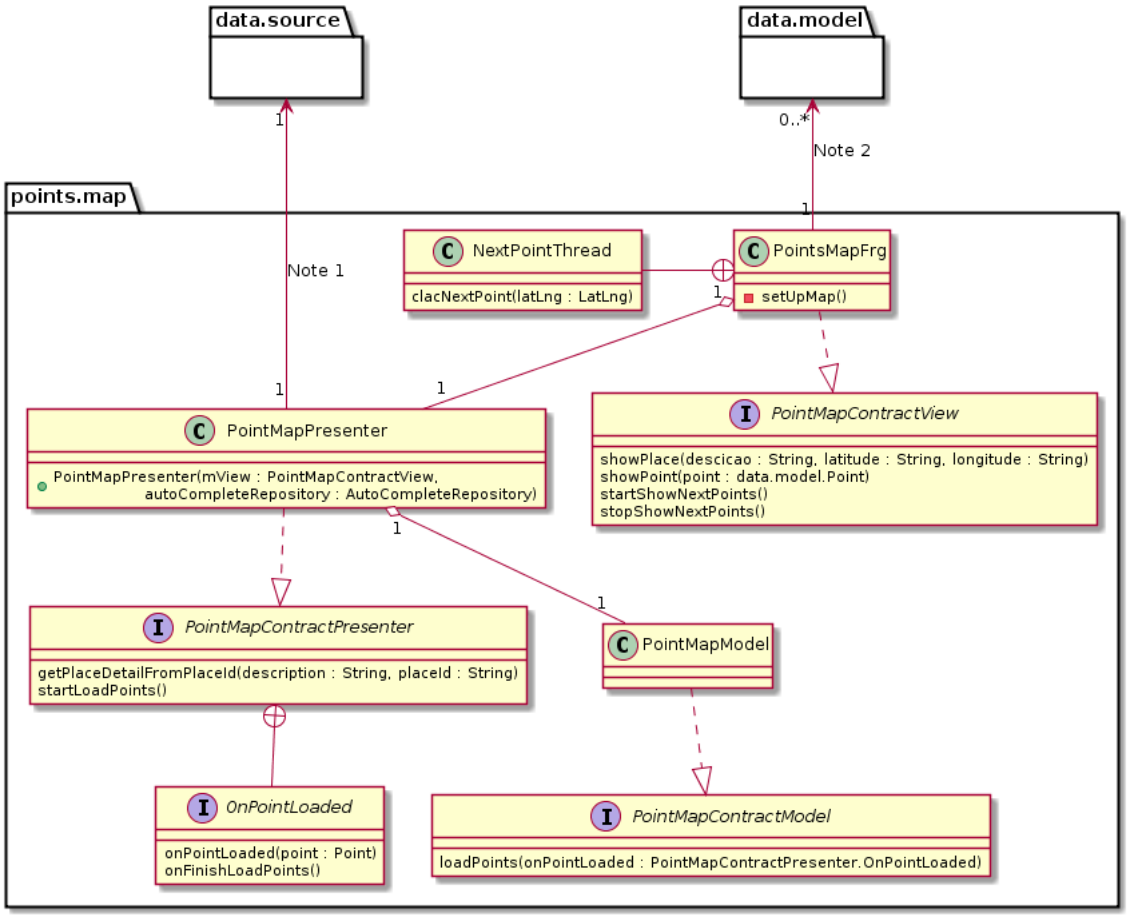


Figura C.14: Diagrama de classes do *points.map*

- ***Note 1:*** A classe PointMapPresenter utiliza a classe AutoCompleteRepository do pacote *data.source.gautocomplete* .
- ***Note 2:*** A classe PointsMapFrg utiliza as classes Line e PointDetailFrgdo do pacote *data.model* .

C.5 Pacote *search.provider*

A seguir segue o diagrama do pacote ***search.provider***.

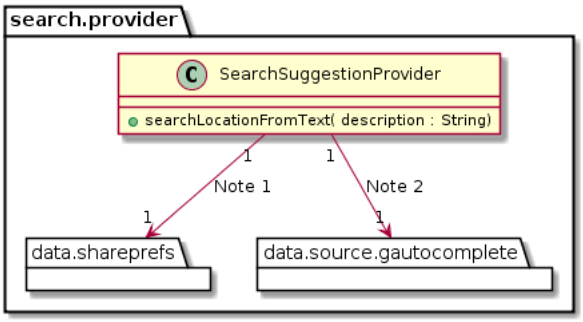


Figura C.15: Diagrama de classes do *search.provider*

- ***Note 1:*** A classe SearchSuggestionProvider utiliza a classe SharePreferencesRepository do pacote *data.shareprefs* .
- ***Note 2:*** A classe SearchSuggestionProvider utiliza a classe AutoCompleteRepository do pacote *data.source.gautocomplete* .

C.6 Pacote *sync*

A seguir segue o diagrama do pacote ***sync***.

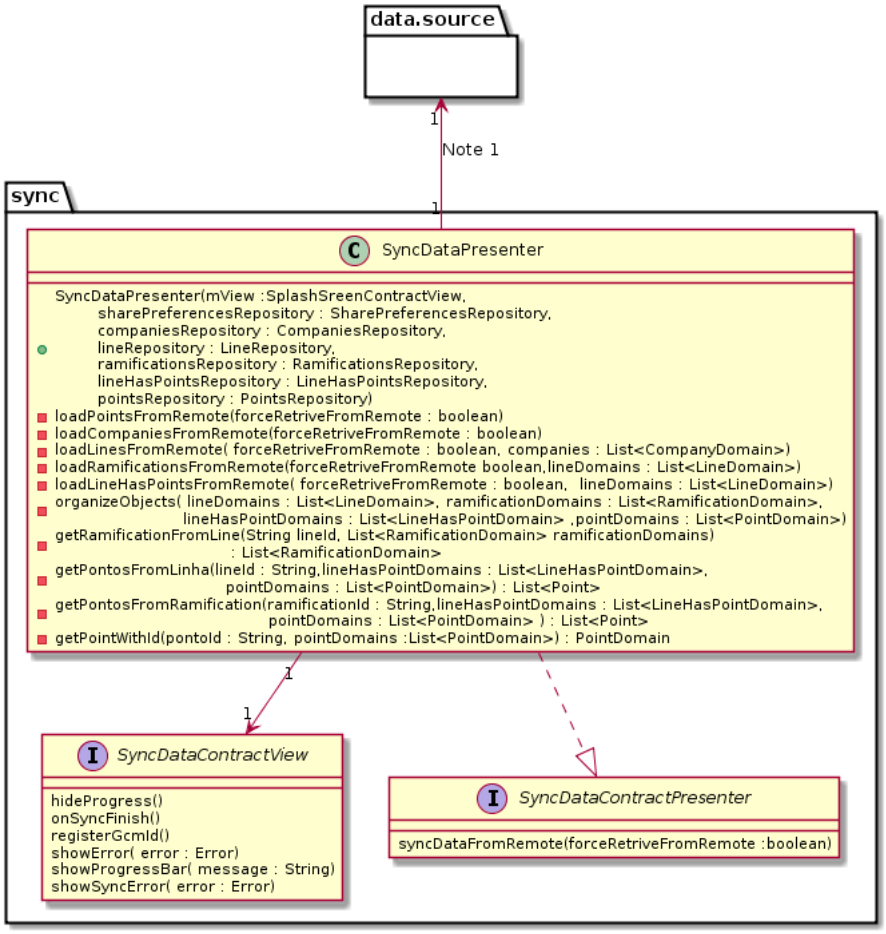


Figura C.16: Diagrama de classes do *sync*

- ***Note 1:*** Essa conexão representa que a classe SyncDataPresenter utiliza as classes SharePreferencesRepository, CompaniesRepository, LineRepository, RamificationsRepository, LineHasPointsRepository e PointsRepository do pacote *data.shareprefs* .

Apêndice D

Implementação componentes servidor - Configurações comuns

D.1 Configuração de dependências

```
{
    "name": "slim/slim-skeleton",
    "description": "A Slim Framework skeleton application for rapid
       development",
    "keywords": ["microframework", "rest", "router", "psr7"],
    "homepage": "http://github.com/slimphp/Slim-Skeleton",
    "license": "MIT",
    "authors": [
        {
            "name": "Josh Lockhart",
            "email": "info@joshlockhart.com",
            "homepage": "http://www.joshlockhart.com/"
        }
    ],
    "require": {
        "php": ">=5.5.0",
        "slim/slim": "^3.1",
        "slim/php-view": "^2.0",
        "monolog/monolog": "^1.17",
    },
    "require-dev": {
        "phpunit/phpunit": ">=4.8 < 6.0"
    },
```

```
23      "autoload-dev": {
24          "psr-4": {
25              "Tests\\": "tests/"
26          }
27      }
28  }
```

Listing D.1: Arquivo composer.json utilizado para configuração de dependências

Podemos observar que esse componente possui uma série de dependências. A utilização de destas se deram pelos seguintes motivos:

- **PHP :** É um liguagem interpretada amplamente utilizada para desenvolvimento web, sua sintaxe lembra Java e C.
- **Slim :** Utilizado para simplificar o roteamento e das requisições HTTPs.
- **Monolog :** Responsável por simplificar a escrita de *logs* do sistema.
- **PHPunit :** Framework utilizado para realização de testes unitários.

D.2 Configuração de variáveis

```
1  <?php
2  return [
3      'settings' => [
4          // Monolog settings
5          'logger' => [
6              'name' => 'slim-app',
7              'path' => __DIR__ . '/../logs/app.log',
8              'level' => \Monolog\Logger::DEBUG,
9          ],
10
11         // Database connection settings
12         "db" => [
13             "host"   => "172.17.0.1:2345",
14             "dbname" => "gpstracker",
15             "user"   => "root",
16             "pass"   => "root"
17         ],
18         //GCM configurations
19         'gcm' =>[
20             'api_key' => 'AIzaSyAqnkI9K1O2JKRxgdsoXWoQDRZJCYhA9PQ',
21             'url'     => 'https://android.googleapis.com/gcm/send'
```

```
22          ],
23      ],
24  ];
```

Listing D.2: Arquivo settings.php utilizado para configuração de variáveis de ambiente

Explicações referentes ao arquivo D.2 são expostas a seguir:

- **Linhas 7 à 8 :** Local onde se encontra o arquivo de logs e o nível de output do log.
- **Linhas 12 à 17 :** Conexão com o componente de banco de dados DB_GpsTracker.
- **Linhas 19 à 21 :** Chave de acesso à *API* do Google Cloud Message e o endpoint de utilização do da *API*.

D.3 Contêineres de dependência

```
1  <?php
2  // DIC configuration
3  $container = $app->getContainer();
4
5  $container['formatter'] = function ($c) {
6      return new Formatter($c->get('request'));
7  };
8  // monolog
9  $container['logger'] = function ($c) {
10     $settings = $c->get('settings')['logger'];
11     $logger = new Monolog\Logger($settings['name']);
12     $logger->pushProcessor(new Monolog\Processor\UidProcessor());
13     $logger->pushHandler(new Monolog\Handler\StreamHandler($settings['path'],
           $settings['level']));
14     return $logger;
15 };
16 //Database
17 $container['db'] = function ($c) {
18     $settings = $c->get('settings')['db'];
19     $mysqli = new mysqli($settings['host'],$settings['user'],$settings['pass'],
           $settings['dbname']);
20     return $mysqli;
21 };
```

Listing D.3: Arquivo dependencies.php utilizado para configuração de contêineres de dependência.

Explicações referentes ao arquivo D.3 são expostas a seguir:

- **Linhas 5 à 7 :** Configuração do contêiner ”formatter”, responsável por fornecer a implementação de formatação de output.
- **Linhas 9 à 15 :** Configuração do contêiner ”logger”, responsável pela instanciação e configuração do da dependência Monolog.
- **Linhas 17 à 21 :** Configuração do contêiner ”logger”, responsável por fornecer a conexão com o banco de dados.

D.4 Configuração do servidor NGINX

```
1  server {
2      listen       80 default_server;
3
4      root   /public;
5      index index.php;
6
7      location / {
8          try_files $uri $uri/ /index.php$is_args$args;
9      }
10
11     location ~ \.php {
12         try_files       $uri =404;
13         fastcgi_split_path_info ^(.+\.php)(/.+)$;
14         include                 fastcgi_params;
15         fastcgi_param   SCRIPT_FILENAME $document_root$fastcgi_script_name;
16         fastcgi_param   SCRIPT_NAME $fastcgi_script_name;
17         fastcgi_index   index.php;
18         fastcgi_pass    php:9000;
19     }
20 }
```

Listing D.4: Arquivo default.conf utilizado para configuração do servidor NGINX

Explicações referentes ao arquivo D.4 são expostas a seguir:

- **Linha 1 :** Inicia configuração de um servidor virtual.
- **Linha 2 :** Configura a porta na qual o servidor irá escutar.
- **Linha 4 :** Configura a a pasta raiz para as tratamento de requisições.
- **Linha 5 :** Configura o arquivo que será usado como default para requisições.
- **Linha 7 à 9 :** Configura que todas as requisições assim como seus argumentos serão tratados pelo arquivo ”index.php”.

- **Linha 11 à 19 :** Configurações para interpretação das requisições que possuam URI terminadas em ".php". O ponto que vale a pena ressaltar, que, essas serão tratados pelo domínio "php" na porta "9000".

D.5 Configuração do *docker*

O *docker* desempenha o importante papel de empacotar a aplicação, dependências e variáveis de ambiente necessárias para funcionamento do componente em imagem(ns). Para o disponibilização de serviços de servidor é utilizado três dependências: um servidor *nginx*, um programa *php* para execução dos *scripts* e os arquivos fontes. No *docker* cada dependência é um serviço disponibilizado por um *container*. A seguir é apresentado um arquivo de configuração interpretado pelo *docker-compose* (programa para interpretação de arquivos de configuração do *docker*).

```
1  version: '2'
2  services:
3    nginx:
4      build: docker/nginx
5      container_name: nginx
6      ports:
7        - 8080:80
8      volumes_from:
9        - app
10   app:
11     command: "true"
12     image: alpine:3.4
13     volumes:
14       - .:/var/www/app
15   php:
16     build: docker/php
17     container_name: php
18     expose:
19       - 9000
20     volumes:
21       - composer-cache:/composer/cache
22     volumes_from:
23       - app
24 volumes:
25   composer-cache:
26     driver: local
```

Listing D.5: Arquivo docker-compose.yml utilizado para configuração dos contêineres do docker

Explicações referentes ao arquivo D.5 são expostas a seguir:

- **Linha 1 :** Configura a versão do programa docker-compose utilizada na estruturação do arquivo.
- **Linha 3 :** Configura um serviço chamado nginx.
- **Linha 6 à 7 :** Configura que o serviço é acessado através da porta 8080 e direcionado para a porta 80, a qual é utilizada pelo programa *NGINX*.
- **Linha 8 à 9 :** Configura que o contêiner possui acesso a arquivos do serviço *app*.
- **Linha 10 :** Configura um serviço chamado *app*, este que possui os arquivos fontes. O motivador desses arquivos serem oferecidos em por um serviço específico, é que este será utilizado pelo serviço *nginx* e *php*.
- **Linha 13 à 14 :** Configura que os arquivos serão oferecidos na pasta "/var/wwww/app".
- **Linha 15 :** Configura um serviço chamado *php*, este será o responsável por interpretar os *scripts* repassador pelo servidor *NGINX*.
- **Linha 18 à 19 :** Configura a porta que de entrada para interpretação dos *scripts*.
- **Linha 20 à 21 :** Configura a utilização de cache, para instalação de pacotes utilizados pelo programa *php*.
- **Linha 22 à 23 :** Configura que o contêiner possui acesso a arquivos do serviço *app*.

D.6 Arquivo fonte *Controler.php*

```
1  <?php
2
3  class Controler
4  {
5      public $db;
6      public $logger;
7      public $formatter;
8      public $settings;
9
10     public function __construct($c) {
11         $this->db        = $c->get('db');
12         $this->logger    = $c->get('logger');
13         $this->formatter = $c->get('formatter');
14         $this->settings  = $c->get('settings');
15     }
16
17     public function getAll($response,$sql){
18         $this->logger->addInfo('executing query '. $sql);
19         if ($result = $this->db->query($sql)) {
```

```
20              /* fetch object array */
21              $objs = array();
22              while ($obj = $result->fetch_object()) {
23                  array_push($objs,$obj);
24              }
25              $result->close();
26              $total_elements = count($objs);
27              if($total_elements > 0){
28                  $response = $this->formatter->render($response, $objs);
29              }else{
30                  $code = 555;
31                  $response = Utils::setErrorStatus($response,$code);
32              }
33
34          }else{
35                 $this->logger->addInfo("could not query -"
36                             . json_encode(mysqli_error($this->db)));
37                  $code = 552;
38                  $response = Utils::setErrorStatus($response,$code);
39
40          }
41          $this->db->close();
42          return $response;
43
44      }
45
46      public function getById($response,$sql,$id,$id2=null){
47
48          if ($stmt = $this->db->prepare($sql)) {
49              if($id2 == null)
50                  $stmt->bind_param("i", $id);
51              else
52                  $stmt->bind_param("ii", $id,$id2);
53              $stmt->execute();
54              $res = $stmt->get_result();
55              $ojb = $res->fetch_object();
56
57              $affectd_rows = $this->db->affected_rows;
58              $stmt->close();
59              $this->db->close();
60
61                          if($affectd_rows == 0){
```

```
62                  $response = $response->withStatus(500);
63                  $this->logger->addInfo("erro in query ".$sql
64                  .' with parameter ' . $id );
65              }else{
66
67                                  $response = $this->formatter->render($response,
                                        $ojb);
68                                  $this->logger->addInfo("executing query ".$sql.'
                                        with parameter'
69                                  . $id . ' result ' . json_encode($ojb));
70                          }
71
72          }
73          return $response;
74      }
75
76      public function deleteById($response,$sql,$id,$id2=null){
77          if ($stmt = $this->db->prepare($sql)) {
78              if($id2 == null)
79                  $stmt->bind_param("i", $id);
80              else
81                      $stmt->bind_param("ii", $id,$id2);
82              $stmt->execute();
83              $affectd_rows = $this->db->affected_rows;
84              $this->logger->addInfo("query ".$sql
85                  .' with parameter ' . $id
86                  .' stm error ['.json_encode($stmt->error));
87              $stmt->close();
88
89
90              if($affectd_rows == 0){
91                  $code = 557;
92                  $response = Utils::setErrorStatus($response,$code);
93                  $this->logger->addInfo("erro in query ".$sql
94                      .' with parameter ' . $id
95                      .' error ['.json_encode(mysqli_error($this->db)));
96              }
97
98          }else{
99              $this->logger->addInfo("error in prepere query ".$sql
100                 .' with parameter '. $id
101                 .'error [ '.json_encode(mysqli_error($this->db)).']');
```

```
102             $code = 552;
103             $response = Utils::setErrorStatus($response,$code);
104         }
105         $this->db->close();
106         return $response;
107     }
108 
109     protected function updateFielById($response,$sql,$param,$id,$id2=null){
110         if ($stmt = $this->db->prepare($sql)) {
111                 if($id2 == null)
112                     $stmt->bind_param("si", $param,$id);
113                 else
114                     $stmt->bind_param("sii", $param,$id,$id2);
115                 $stmt->execute();
116 
117                 $affectd_rows = $this->db->affected_rows;
118                 if($affectd_rows > 0) {
119 
120                     $stmt->close();
121 
122 
123                     $input['_id']    = $id;
124                     $input['update'] = $param;
125 
126                     $this->logger->addInfo("patch no PedidoControler add - ".
                            json_encode($input)
127                                         .' query: '.$sql);
128                     $response = $this->formatter->render($response, $input);
129                 }else{
130                     $code = 556;
131                     $response = Utils::setErrorStatus($response,$code);
132                     $this->logger->addInfo("patch no PedidoControler add - "
133                                         .' query: '.$sql. ' error '
134                                         .json_encode(mysqli_error($this->db)
                                                ));
135                 }
136             }else{
137                 $code = 552;
138                 $response = Utils::setErrorStatus($response,$code);
139 
140             }
141         $this->db->close();
```

```
142         return $response;
143     }
144 
145     public function executQuery($response,$sql){
146         $this->logger->addInfo('executing query '. $sql);
147         if ($result = $this->db->query($sql)) {
148             /* fetch object array */
149             $objs = array();
150             while ($obj = $result->fetch_object()) {
151                 array_push($objs,$obj);
152             }
153             $result->close();
154             $total_elements = count($objs);
155             if($total_elements == 1){
156                 $response = $this->formatter->render($response, $objs[0]);
157             }
158             else if($total_elements > 1){
159                 $response = $this->formatter->render($response, $objs);
160             }else{
161                 $code = 555;
162                 $response = Utils::setErrorStatus($response,$code);
163             }
164 
165         }else{
166                $this->logger->addInfo("could not query -"
167                             . json_encode(mysqli_error($this->db)));
168                 $code = 552;
169                 $response = Utils::setErrorStatus($response,$code);
170 
171         }
172         $this->db->close();
173         return $response;
174 
175     }
176 }
```

Listing D.6: Arquivo fonte Controler.php

Apêndice E

Implementação do componente Gpstracker_server

E.1 Arquivo fonte *index.php*

O arquivo ”index.php” é o quem disponibiliza uma instancia da aplicação. A seguir segue os principais detalhes de sua estrutura:

```
<?php
require __DIR__ . '/../vendor/autoload.php';
require __DIR__ . '/../src/Formatter.php';
session_start();

// Instantiate the app
$settings = require __DIR__ . '/../src/settings.php';
$app = new \Slim\App($settings);

require __DIR__ . '/../src/dependencies.php';// Set up dependencies
require __DIR__ . '/../src/controler/Device.php';
require __DIR__ . '/../src/controler/Location.php';


$app->group('/api/gpstracker', function () use ($app) {
    $app->post('/devices/{id:\d+}/sendDevicePushMessage',
                'Device:sendPushMessageForDeviceId');
    $app->get('/device','Device:getAllDevices');
    $app->post("/device",'Device:insertNewDevice');
    $app->get('/device/{id:\d+}','Device:getDeviceById');
    $app->delete('/device/{id:\d+}','Device:deleteDeviceById');

```

```
23      $app->get('/location','Location:getAllDevices');
24      $app->post('/location','Location:insertNewGpsLocation');
25      $app->get('/location/{id:\d+}','Location:getLocationById');
26      $app->delete('/location/{id:\d+}','Location:deleteLocationById');
27
28      $app->get('/requesDeviceLocale/{id:\d+}',
29                'Device:requestDeviceLocationById');
30
31      $app->post('/sendDevicePushMessage',
32                  'Device:sendDevicesPushMessage');
33
34  });
```

Listing E.1: Arquivo fonte index.php

Explicações referentes ao arquivo E.1 são expostas a seguir:

- **Linha 2 :** Carregamento de dependências do *framework slim*.
- **Linha 3 :** Importação do arquivo que faz trata a resposta do componente (realiza a formatação dos dados para apresentação).
- **Linhas 7 à 8 :** Carrega as configuração e inicializa a execução do *framework slim*.
- **Linha 10 :** Carrega as configuração de dependências.
- **Linhas 11 à 12 :** Carrega classes controladoras.
- **Linha 15 :** Define um grupo de rota para acesso à *API*.
- **Linha 16 :** Define que a rota "/api/gpstracker/devices/{id}/sendDevicePushMessage" deve tratar solicitações ao método *HTTP POST* invocando o método *sendPushMessageForDeviceId* da classe *Device*, onde o parâmetro "{id}" deve ser um número inteiro positivo.

Apêndice F

Implementação do componente Wheretrans_server

F.1 Arquivo fonte *index.php*

```
1  <?php
2  require __DIR__ . '/../vendor/autoload.php';
3  require __DIR__ . '/../src/Formatter.php';
4
5  session_start();
6
7  // Instantiate the app
8  $settings = require __DIR__ . '/../src/settings.php';
9  $app = new \Slim\App($settings);
10
11 require __DIR__ . '/../src/dependencies.php';// Set up dependencies
12
13 require_once __DIR__ . '/utils/Utils.php';
14 require_once __DIR__ . '/controler/Controler.php';
15 require_once __DIR__ . '/controler/Device.php';
16 require_once __DIR__ . '/controler/Location.php';
17 require_once __DIR__ . '/controler/RequestLocation.php';
18 require_once __DIR__ . '/controler/User.php';
19 require_once __DIR__ . '/controler/Point.php';
20 require_once __DIR__ . '/controler/Ramific.php';
21 require_once __DIR__ . '/controler/Empresa.php';
22 require_once __DIR__ . '/controler/Pedido.php';
23 require_once __DIR__ . '/controler/Linha.php';
24 require_once __DIR__ . '/controler/LineHasPoint.php';
25
```

```
26
27 $app->group('/api/wheretrans', function () use ($app) {
28
29
30     $app->get('/requetDevicesLocation','RequestLocation:getAllRequestLocations')
           ;
31     $app->post('/requetDevicesLocation',
32                'RequestLocation:requetDeviceLocation');
33     $app->get('/requetDevicesLocation/{id:\d+}','RequestLocation:
           getRequestLocById');
34     $app->delete('/requetDevicesLocation/{id:\d+}',
35                  'RequestLocation:deleteRequestLocById');
36
37     $app->get('/users','User:getAllUsers');
38     $app->post('/users','User:insert');
39     $app->get('/users/{id:\d+}','User:getUserById');
40     $app->patch('/users/{id:\d+}','User:updateUserById');
41     $app->delete('/users/{id:\d+}','User:deleteUserById');
42
43
44     $app->get('/pedidos','Pedido:getPedidoWithTelephoneAndStates');
45
46     /*
47     $app->get('/lineFromPoint/{ponto_id:\d+}',
48               'Point:getLineFromPoint');
49
50     $app->get('/ramificacao/{line:\d+}',
51                 'Ramific:getAllFromLine');
52     */
53
54     //devices
55     $app->get('/empresas/{empresa__id:\d+}/devices/{veiculo_id:\d+}',
56             'Device:getDeviceFromCompanyByVehicleId');
57
58     $app->delete('/empresas/{empresa__id:\d+}/devices/{veiculo_id:\d+}}',
59                 'Device:deleteDeviceFromCompanyByVehicleId');
60
61     $app->post('/empresas/{empresa__id:\d+}/devices',
62                 'Device:insertNewDevice');
63
64     $app->get('/empresas/{empresa__id:\d+}/devices',
65             'Device:getAllDevicesFromCompany');
```

```
66
67      //locations
68      $app->get('/empresas/{empresa__id:\d+}/devices/{veiculo_id:\d+}/locations/{
            id:\d+}',
69              'Location:getLocationById');
70
71      $app->delete('/empresas/{empresa__id:\d+}/devices/{veiculo_id:\d+}/locations
            /{id:\d+}',
72                'Location:deleteLocationById');
73
74      $app->get('/empresas/{empresa__id:\d+}/devices/{veiculo_id:\d+}/locations',
75              'Location:getAllLocationFromDeviceAndCompany');
76
77      $app->get('/empresas/{empresa__id:\d+}/locations',
78              'Location:getAllLocationFromCompany');
79
80      $app->post('/empresas/{empresa__id:\d+}/devices/{veiculo_id:\d+}/locations',
81               'Location:insertNewLocation');
82
83      //points
84      $app->get('/points/{point_id:\d+}',
85              'Point:getPointById');
86      $app->get('/points',
87          'Point:getAllPoint');
88      $app->post('/points','Point:insertNewPoint');
89
90      $app->delete('/points/{id:\d+}','Point:deletePointById');
91
92      /*$app->delete('/empresas/{empresa__id:\d+}/linhas/{linha_id:\d+}/points/{
            point_id:\d+}',
93              'Point:deletePointFromLineAndCompany');*/
94
95      $app->get('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/points',
96              'Point:getAllPointsFromLineAndCompany');
97
98      //lineHasPoint  POST do ponto
99      // POST empresa/1/line/5/hasPoint todo: falta fazer
100     //       empresa/1/line/4/point // ja crio o ponto e a ligação da linha com o
            ponto
101     $app->post('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/points',
102                               'Point:insertNewPointAndLineHasPoint');
103
```

```
104     //LineHasPoint
105     $app->get('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/lineHasPoint/{
            id:\d+}',
106                             'LineHasPoint:getPointHasLineFromEmpresaAndLinhaById
                                ');
107     $app->delete('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/
            lineHasPoint/{id:\d+}',
108                     'LineHasPoint:deleteLineHasPointFromCompanyAndLineById');
109
110     $app->get('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/lineHasPoint',
111                                 'LineHasPoint:getPointHasLineFromEmpresaAndLinha
                                    ');
112     $app->post('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/lineHasPoint'
            ,
113                                 'LineHasPoint:insertNewLineHasPoint');
114
115
116
117     //ramificação
118     $app->get('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/ramificacoes/{
            id:\d+}',
119                     'Ramific:getRamificacaoFromCompanyAndLineById');
120     $app->delete('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/
            ramificacoes/{id:\d+}',
121             'Ramific:deleteRamificacaoFromCompanyAndLineById');
122     $app->get('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/ramificacoes',
123                                 'Ramific:getAllRamificacaoFromCompanyAndLine');
124
125     $app->post('/empresas/{empresa__id:\d+}/linhas/{linha__id:\d+}/ramificacoes'
            ,
126                             'Ramific:insertNewRamificacao');
127
128     //line
129     $app->get('/empresas/{empresa__id:\d+}/linhas/{linha_id:\d+}',
130                                 'Linha:getLinhaByIdFromEmpresa');
131     $app->delete('/empresas/{empresa__id:\d+}/linhas/{linha_id:\d+}',
132                                 'Linha:deleteLinhaByIdFromEmpresaId');
133     $app->get('/empresas/{empresa__id:\d+}/linhas','Linha:getAllLinhaFromEmpresa
            ');
134     $app->post('/empresas/{empresa__id:\d+}/linhas','Linha:insertNew');
135
136     //empresas
```

```
137     $app->get('/empresas','Empresa:getAllEmpresa');
138     $app->post('/empresas','Empresa:insertNew');
139     $app->get('/empresas/{id:\d+}','Empresa:getEmpresaById');
140     $app->delete('/empresas/{id:\d+}','Empresa:deleteEmpresaById');
141
142     //pedidos
143     $app->get('/empresas/{empresa__id:\d+}/pedidos','Pedido:
            getAllPedidoFromCompany');
144     $app->post('/empresas/{empresa__id:\d+}/pedidos','Pedido:
            insertNewPedidoFromCompany');
145     $app->get('/empresas/{empresa__id:\d+}/pedidos/{id:\d+}',
146                                 'Pedido:getPedidoByIdFromCompany');
147     $app->delete('/empresas/{empresa__id:\d+}/pedidos/{id:\d+}',
148                             'Pedido:deletePedidoByIdFromCompany');
149     $app->patch('/empresas/{empresa__id:\d+}/pedidos/{id:\d+}',
150                             'Pedido:updatePedidoByIdFromCompany');
```

Listing F.1: Arquivo fonte index.php

Referências Bibliográficas

[1] W.Frank Ableson, Robi Sen, Chris King, C. Enrique Ortiz , *Android in Action - Third Edition* , Nanning, 2012.

[2] Richard Ferraro, Murat AkTihanoglu, *Location-Aware Applications*, Nanning, 2011.

[3] Valentin Bojinov, *RESTful Web API Design with Node.js*, Packt Publishing, 2015.

[4] Lorna Jane Mitchell, *PHP Web Services*, O'Reilly Media Inc., 2016.

[5] Android Developers. Disponível em: *<https://developer.android.com/index.html>*. Acesso em: Nov. 2014.

[6] PHP Documentation, Disponível em: *<http://php.net/docs.php>*. Acesso em: Nov. 2014.

[7] Slim Framework, Disponível em: *<http://www.slimframework.com>*. Acesso em: Nov. 2015.

[8] What is Docker, Disponível em: *<https://www.docker.com/what-docker>*. Acesso em: Nov. 2016.

[9] Dependency Injection Container (DIC) for Lazy Loading Services, Disponível em: *<http://brady.lucidgene.com/articles/dependency-injection-container>*. Acesso em: Nov. 2016.

[10] Composer, Disponível em: *<https://getcomposer.org>*. Acesso em: Julh. 2017.

[11] Pfleeger, Shari Lawrence, *Software Engineering: Theory and Practice*, Pearson Education India, 2004.

[12] Fielding, Roy T and Taylor, Richard N, *Architectural styles and the design of network-based software architectures*, University of California, Irvine Doctoral dissertation, 2000.

[13] Webber, Jim and Parastatidis, Savas and Robinson, Ian, *REST in practice: Hypermedia and systems architecture*, O'Reilly Media, Inc., 2010.

[14] NGINX documentation, Disponível em: *<https://nginx.org/en/docs/>*. Acesso em: Dez. 2016.

[15] Google Clouc Messaging, Disponível em: *<https://developers.google.com/cloud-messaging/gcm>*. Acesso em: Jan. 2017.

[16] Google Play, Disponível em: *<https://play.google.com/store>*. Acesso em: Junho. 2017.

[17] Vá de Ônibus - Apps para Android no Google Play, Disponível em: *<https://play.google.com/store/apps/details?id=br.com.fetranspor.vadeonibus>*. Acesso em: Junho. 2017.

[18] Vá de Ônibus - Consulte como se deslocar de Ônibus no Estado do Rio de Janeiro, Disponível em: *<http://vadeonibus.com.br/Vdo/index.php>*. Acesso em: Junho. 2017.

[19] Cadê o Ônibus?® - Apps para Android no Google Play, Disponível em: *<https://play.google.com/store/apps/details?id=br.nanoit.viewbus>*. Acesso em: Junho. 2017.

[20] Moovit: Metro, Ônibus e Trens, Disponível em: *<https://play.google.com/store/apps/details?id=com.tranzmate>* Acesso em: Junho. 2017.

[21] Tecnologia embarca nos ônibus do Rio - O Dia no Coletivo - O Dia, Disponível em: *<http://odia.ig.com.br/noticia/odia-no-coletivo/2015-07-30/tecnologia-embarca-nos-onibus-do-rio.html>*. Acesso em: Junho. 2017.

[22] Charles Web Debugging Proxy - Official Site, Disponível em: *<https://www.charlesproxy.com/>*. Acesso em: Junho. 2017.

[23] Kurose, James F and Ross, Keith W, *Redes de Computadores e a Internet*, São Paulo, 2010.

[24] Google Maps, Disponível em: *<https://www.google.com.br/maps>*. Acesso em: Junho. 2017.

[25] Tecnologia embarca nos ônibus do Rio - O Dia no Coletivo - O Dia, Disponível em: *<http://odia.ig.com.br/noticia/odia-no-coletivo/2015-07-30/tecnologia-embarca-nos-onibus-do-rio.html>*. Acesso em: Junho. 2017.

[26] http://www.ofluminense.com.br/pt-br/cidades/frota-de-niterói-já-'cabe'-no-celular, Disponível em: *<http://www.ofluminense.com.br/pt-br/cidades/frota-de-niter%C3%B3i-j%C3%A1-%E2%80%98cabe%E2%80%99-no-celular>*. Acesso em: Junho. 2017.

[27] Ferraro, Richard and Aktihanoglu, Murat, *Location-aware applications*, Manning Publications Co, 2011.

[28] Goransson, Anders, *Efficient Android Threading*, O'Reilly Media, 2014.