UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DEBORA MENDES

APLICATIVO ANDROID PARA ACOMPANHAMENTO DAS ROTAS DE TRANSPORTE COLETIVO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2018

DEBORA MENDES

APLICATIVO ANDROID PARA ACOMPANHAMENTO DAS ROTAS DE TRANSPORTE COLETIVO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi
Coorientador: Prof. João Guilherme Brasil Pichetti

PATO BRANCO
2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

APLICATIVO ANDROID PARA ACOMPANHAMENTO DAS ROTAS DE TRANSPORTE COLETIVO

POR

DEBORA MENDES

Este trabalho de conclusão de curso foi apresentado no dia 12 de dezembro de 2018, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. A acadêmica foi arguida pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Profª Drª Beatriz Terezinha Borsoi
Orientador

Prof Esp. João Guilherme Brasil Pichetti
Coorientador

Profª MSc. Andreia Scariot Beulke

Prof. MSc. Vinicius Pegorini

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

MENDES, Debora. Aplicativo Android para acompanhamento das rotas de transporte coletivo. 2018. 50f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

O uso de meios de transporte coletivo tem sido incentivado por diversos fatores, dentre os quais podem ser destacados a redução de emissão de poluentes e a redução do tráfego. Menos veículos circulando poluem menos e deixam as ruas mais livres para pedestres e outros meios de transporte como bicicletas. Uma malha de transporte coletivo que seja efetiva em termos de cobertura da região e existência de pontos de parada estratégicos beneficia trabalhadores e estudantes, para citar duas categorias que utilizam massivamente esse meio de transporte. No Brasil, em cidades maiores há alternativas de transporte por trens e metrôs. Contudo, inclusive nessas cidades e amplamente nas cidades de médio e pequeno porte o transporte coletivo por ônibus e vans é dominante ou, ainda, pode ser o único meio de transporte público. Sistemas e aplicativos podem auxiliar as pessoas que utilizam transporte coletivo para saber qual linha apanhar para chegar ao destino, o ponto mais próximo a partir da sua localização atual ou de determinada localização, os horários e outros dados. Existem soluções disponíveis no mercado para usuários de transporte coletivo, algumas delas são dependentes de determinado serviço estar disponível. A solução proposta neste trabalho é para dispositivos móveis Android e para a cidade de Pato Branco. O aplicativo fornecerá diversas possibilidades de busca e a identificação de pontos de ônibus a partir de pontos de referência. A indicação de localização a partir de pontos de referência é uma maneira comum de as pessoas fornecerem informações sobre localização de lugares e estabelecimentos. Por exemplo: o ponto de ônibus fica próximo da praça X, do supermercado A e etc. O aplicativo funcionará no dispositivo móvel necessitando de conexão com a Internet apenas para algumas funcionalidades como a identificação automática das coordenadas geográficas de localização do usuário. Os dados das empresas de ônibus, linhas e pontos são cadastrados em um aplicativo *web* e disponibilizados para o dispositivo móvel. Esses dados são atualizados no dispositivo móvel por meio de sincronização com o servidor, que é demandada pelo usuário.

Palavras-chave: Aplicativos Android. Rotas de transporte coletivo. Aplicativo para transporte coletivo urbano.

ABSTRACT

MENDES, Debora. Android application for the monitoring of collective transport routes. 2018. 50f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

The use of means of public transport has been encouraged by several factors, among which the reduction of emission of pollutants and the reduction of traffic with less vehicles circulating. An effective service of collective transport network can benefit workers and students. These are the two categories that massively use this means of transport to go to work or to school. In Brazil, in the main cities there are alternatives of bus, such as trains and meters. However, in these cities and widely in medium and small cities, collective transport by buses and vans is dominant or even the only means of public transport. Systems and applications can help people using public transport to find out which line to pick up to get to the destination, the closest bus stop of their current location, bus schedules and other data. There are solutions available, and some of them depend on specific services or are applied to specific cities. The proposed solution is for Android mobile devices and for the city of Pato Branco. The application will provide search possibilities and the identification of bus stop from reference points. The indication of locations from reference points is a common way that people provide information about places and establishments. For example: the bus stop is close to the X square or supermarket A and so on. The application will work on the mobile device that needs to connect to the Internet only for some features such as the automatic identification of the geographical coordinates of the user's location. The data buses, lines, reference points, and bus stops are included in a web application and downloaded to the mobile device.

Keywords: Android application. Routes of collective transport. Application for urban collective transport.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

API	*Application Programming Interface*
CPTM	Companhia Paulista de Trens Metropolitanos
CRUD	*Create, Read, Update e Delete*
CSS	*Cascading Style Sheet*
DVM	*Dalvik Virtual Machine*
GPS	*Global Positioning System*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JNI	*Java Native Interface*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
JSP	*JavaServer Pages*
MVC	*Model-View-Controller*
SO	Sistema Operacional
URL	*Uniform Resource Locator*
VLT	Veículo Leve sobre Trilhos

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho que abrange as considerações inicias, os objetivos e a justificativa. O capítulo é finalizado com a apresentação do texto por meio da descrição sumária dos próximos capítulos.

1.1 CONSIDERAÇÕES INICIAIS

A utilização de meios de transporte coletivo é desejável, especialmente em grandes cidades, para reduzir os índices de poluição causada pelo dióxido de carbono que é emitido pela queima de combustíveis fósseis. Nos centros urbanos, o uso de meios de transporte coletivo também é incentivado para a redução do tráfego, com a diminuição de veículos nas ruas é reduzido o risco de acidentes, há mais espaço e menos perigo para os pedestres e para meios alternativos de transporte como as bicicletas.

Muitas pessoas utilizam o transporte coletivo para deslocar-se até o trabalho e há é grande o número de pessoas que utilizam para ir até a escola, para citar as duas formas de uso mais intenso desse tipo de transporte. Para muitas pessoas, o transporte coletivo é o único meio que elas possuem para ir e vir do trabalho e/ou da escola; para outras ele é uma alternativa viável devido à distância, custos e despesas geradas por um veículo de passeio. E para muitas dessas pessoas ele é uma opção alternativa, menos poluente e consumidor de recursos naturais de transporte de passageiros.

Aliado ao interesse de uso de transporte coletivo está a necessidade de saber os horários e o itinerário desse tipo de transporte. Para trens e metrôs, os horários e as linhas estão bem definidas e, exceto pelos problemas que causam atrasos ou interrupções de deslocamento, tais horários e itinerários são cumpridos. Para o transporte público de ônibus, saber as linhas que passam em determinados pontos e os horários pode ser mais difícil, especialmente quando um mesmo ponto é utilizado por empresas e itinerários diferentes. Ter acesso a esse tipo de informação pode ser ainda mais difícil para as pessoas que não estão acostumadas a utilizar aquele itinerário.

Existem diversos aplicativos disponíveis para esse tipo de atividade, com finalidade de fornecer informações sobre itinerários e horários de transporte coletivo, que geralmente são aplicáveis às grandes cidades. Considerando cidades como, por exemplo, Pato Branco no Estado do Paraná que são de pequeno porte, mas que possuem um número grande de usuários

de transporte coletivo, estudantes ou trabalhadores em estabelecimentos localizados nas periferias, verificou-se a oportunidade de desenvolver um aplicativo Android para acompanhamento das rotas de transporte coletivo na cidade de Pato Branco.

1.2 OBJETIVOS

A seguir estão o objetivo geral e os objetivos específicos definidos para este trabalho.

1.2.1 Objetivo Geral

Proporcionar por meio de dispositivos móveis uma forma de acompanhamento das rotas de transporte coletivo urbano na cidade de Pato Branco.

1.2.2 Objetivos Específicos

- Auxiliar na localização de qual é o melhor itinerário de ônibus para deslocar-se para um destino pretendido, considerando um ponto de partida.
- Consultar os horários das linhas de ônibus que circulam na cidade de Pato Branco.
- Buscar por meio de opções de filtros dados sobre origem e destino, horários de saída e chegada de pontos específicos, pontos de parada e de referência, entre outros.
- Definir favoritos de linhas e pontos que ficam armazenados no dispositivo móvel do usuário.

1.3 JUSTIFICATIVA

A necessidade de desenvolvimento de aplicativos que auxiliem os usuários de transporte coletivo é mais evidente em cidades de pequeno porte, como é o caso de Pato

Branco. O breve levantamento de aplicativos semelhantes apresentado no Capítulo 3 indica a existência de várias soluções para cidades maiores e grandes centros urbanos.

A possibilidade de oferecer esse tipo de aplicativo e com funcionalidades que permitam a filtragem para auxiliar nas buscas de pontos de parada de ônibus, por exemplo, pode ser bastante útil para cidades, como é o caso de Pato Branco, que recebem muitos estudantes. Esses estudantes, em sua grande maioria proveniente de outras regiões do país, podem ter dificuldades para deslocar-se até os pontos que não conhecem ou até familiarizar-se com a cidade. Além disso, um aplicativo que forneça pontos de referência para a identificação de pontos de parada de ônibus auxilia na localização desses pontos. Isso porque é comum as pessoas fornecerem informações de localização baseadas em pontos de referência ao invés de nome de ruas e respectivos números. Por exemplo: ao invés de Rua Abc, número x, próximo à praça central, em frente ao supermercado X ou ao lado do Banco ABC são formas recorrentes que as pessoas utilizam para prestar informações quando questionadas sobre a localização de estabelecimentos, pontos de parada de ônibus e outros.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos. Este primeiro capítulo apresenta as considerações iniciais com o contexto do sistema a ser desenvolvido, os seus objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico centrado em aplicações *mobile* Android. No Capítulo 3 são apresentados aplicativos com funcionalidades semelhantes à proposta deste trabalho. No Capítulo 4 estão as ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema. No Capítulo 5 é apresentado o resultado da realização do trabalho, a modelagem e a implementação do aplicativo. No Capítulo 6 estão as considerações finais. Por fim, estão as referências utilizadas na composição do texto.

2 APLICAÇÕES MOBILE ANDROID

A história do Android iniciou em 2005, quando a Google adquiriu a empresa Android Inc. Dois anos mais tarde em colaboração com o grupo Open Handset Alliance[1] foi apresentado o Sistema Operacional (SO) Android (PRIMORAC; RUSSO, 2012). Em 2012 havia cerca de 675.000 aplicações disponíveis na Google Play Store (DISIMO, 2012) ano em que a taxa de crescimento de novas aplicações para Android foi de 12.000 ao mês (APPBRAIN, 2012). Em 2014 havia mais de um bilhão desses dispositivos em uso no mundo (LUNDEN, 2014). Android é considerada a plataforma mais popular para *smartphone* (XU; ZHANG; ZHU, 2013).

O sistema operacional Android pode ser definido como um conjunto de serviços de software, especialmente portados para dispositivos móveis (GUANA et al., 2012). O sistema Android faz uso de uma pilha de software que inclui sistema operacional, *middleware*, interface com o usuário e aplicação (BRAY, 2014). Os aplicativos Android atendem ao paradigma hierárquico no desenvolvimento de aplicações com a linguagem Java: a camada de mais baixo nível utiliza C e C++ e sobre ela está o código Java que faz chamadas às bibliotecas C e C++ por meio de *Java Native Interface* (JNI) *(*LI; WANG, 2014).

O Android oferece uma *Application Programming Interface* (API) que provê a capacidade de acessar informações de *hardware* (como localização *Global Positioning System* (GPS), câmera, microfone e alto-falante), leitura do estado do telefone, leitura/escrita de dados do usuário, modificar configurações do telefone, entre outros.

A Figura 1 apresenta a estrutura do sistema operacional Android.

[1] A Open Handset Alliance é um grupo de empresas de hardware, software e telecomunicação dedicadas ao projeto de plataforma Android (MOTOROLA, 2018).

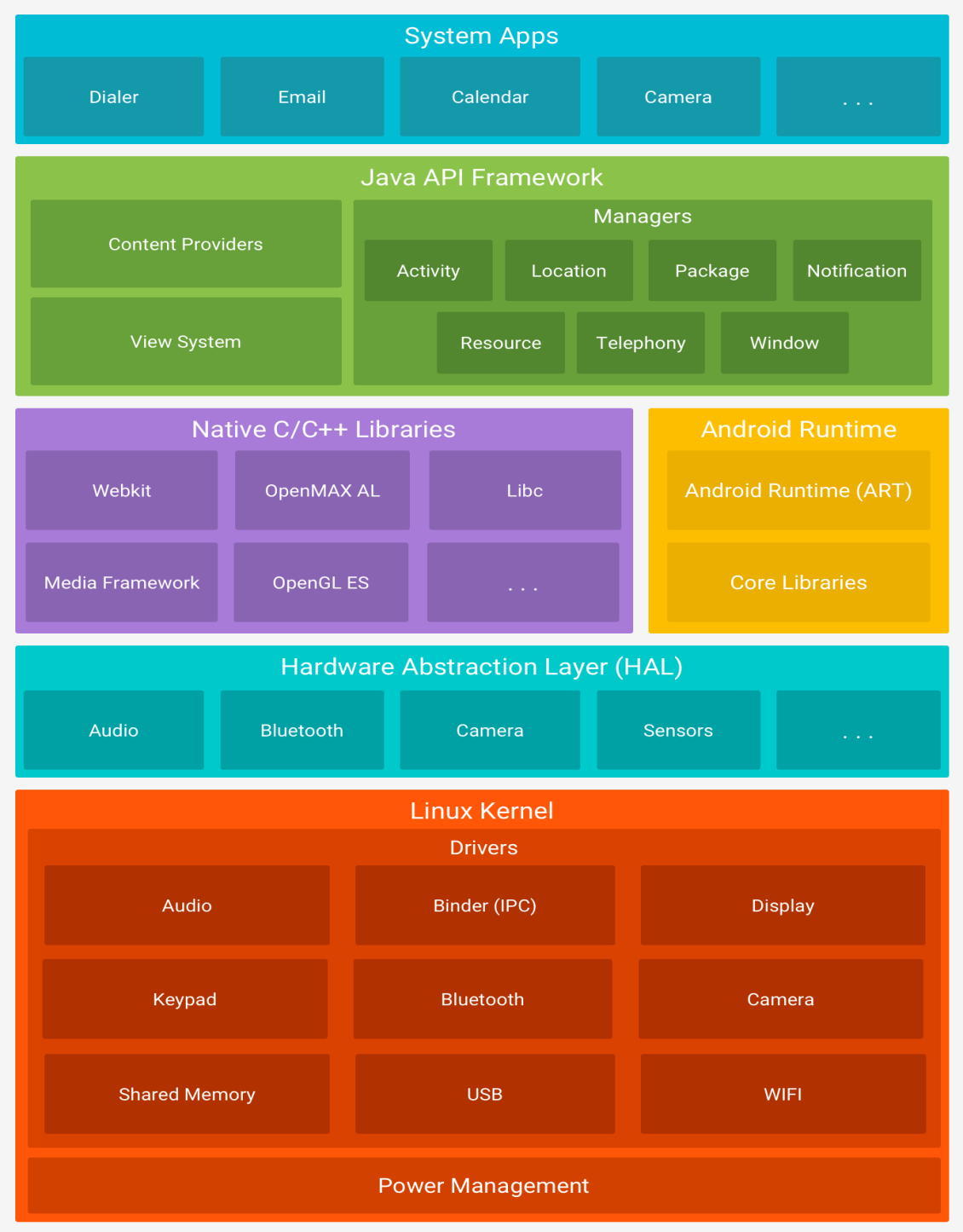


Figura 1 – Estrutura do sistema operacional Android
Fonte: Android Developers (2018,p.s.n.).

A arquitetura do SO Android, como apresentada na Figura 1, é dividida em cinco camadas que possuem funcionalidades e comportamentos específicos: aplicações,

frameworks, bibliotecas, *runtime* e *kernel* Linux. Essas camadas são descritas a seguir (GUANA et al., 2012, PRIMORAC; RUSSO, 2012):

a) Aplicações - camada de aplicação do usuário contém as aplicações visíveis para o usuário. São as aplicações utilizadas sobre o sistema operacional e que estendem as funcionalidades desse sistema como clientes de *email*, navegadores, jogos e aplicações desenvolvidas. As aplicações são escritas com a linguagem Java.

b) Quadro de aplicações - camada de *framework* da aplicação é um conjunto extensível de componentes de software (APIs) usados por todas as aplicações no sistema operacional. Essas APIs contêm ferramentas para a criação de interfaces e ferramentas de sistema, como as *intents* utilizadas para iniciar outros aplicativos ou realizar atividades como a abertura de arquivos.

c) Bibliotecas – é um conjunto de bibliotecas C/C++ utilizadas por várias componentes Android. Essas bibliotecas são usadas pelo *framework* da aplicação para gerenciar a renderização da tela, prover a segurança de dispositivos e a persistência de aplicações, entre outras.

d) Android tempo/execução – cada aplicação no Android é executada em um processo diferente no sistema operacional e para cada processo é criada uma instância da máquina virtual Dalvik. A camada de *runtime* é composta pela máquina virtual Dalvik e as bibliotecas básicas (*core libraries*) que especificam o ambiente de execução da aplicação dentro do sistema operacional. A *Dalvik Virtual Machine* (DVM) é a parte principal do ambiente de execução usada para iniciar as bibliotecas básicas escritas em Java.

e) Núcleo do Linux – é a camada responsável pelo controle de processos, gerência de memória, *threads*, protocolos de rede, agendamento de tarefas, controles de segurança e gerenciamento de arquivos.

Os principais componentes do Android de acordo com Oliveira (2018) são: *activities, services, intents* e *broadcast receivers*, descritos a seguir.

a) *activities* – são componentes que representam a tela de interação do usuário com a aplicação. Para obter dados de sensores existentes no dispositivo móvel é necessário apenas fazer com que a *activity* implemente a classe SensorEventListner e indicar o sensor do qual os dados devem ser obtidos. A Figura 2 apresenta o ciclo de vida dos componentes *activities*.

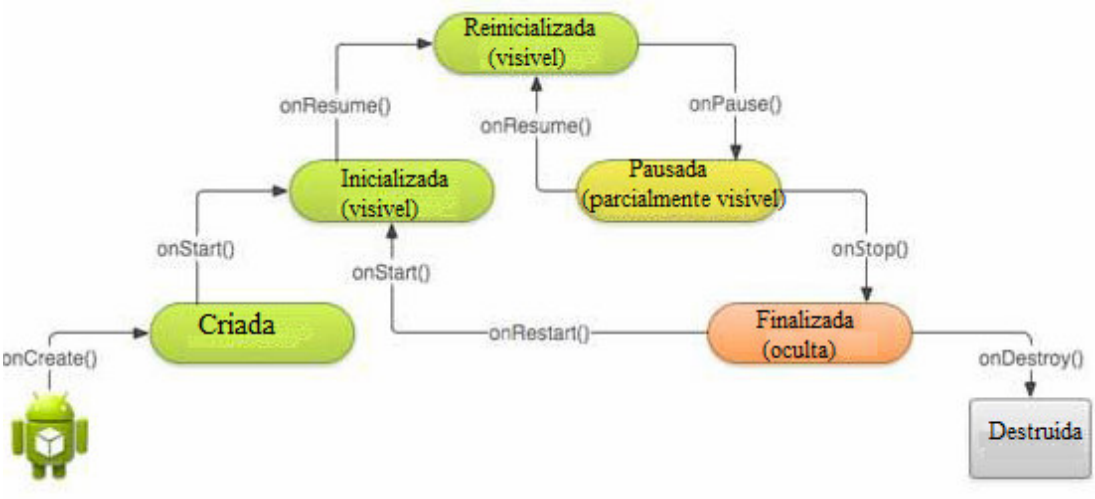


Figura 2 – Ciclo de vida das *activities*
Fonte: traduzido de Oliveira (2018, p.s.n.).

Os métodos que constam na Figura 2 são (OLIVEIRA, 2018):

1) onCreate() - criado quando uma atividade é iniciada;
2) onStart() - chamado quando a aplicação está visível para o usuário;
3) onRestart() - necessário quando uma aplicação estiver prestes a ser chamada novamente;
4) onResume() - chamado quando da interação do usuário com a aplicação;
5) onPause() - chamado quando a aplicação está prestes a retornar outra *activity* (acessar outra tela da aplicação, por exemplo);
6) onStop() - usado quando a aplicação não estiver mais sendo executada;
7) onDestroy() - chamado quando a aplicação já terminou ou quando o sistema necessita finalizar uma *activity*;

b) *services* – são os componentes responsáveis por executar as tarefas em *background*. A Figura 3 apresenta o ciclo de vida dos *services*. É importante observar que como os serviços executam em *background* eles não possuem componentes de interface.

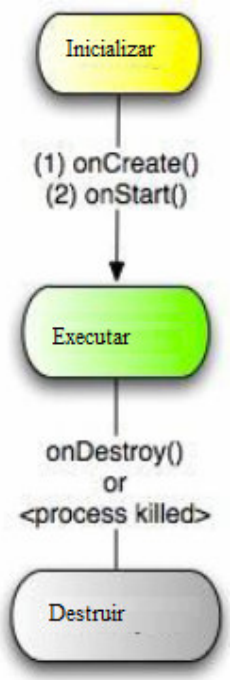


Figura 3 – Ciclo de vida dos *services*
Fonte: traduzido de Oliveira (2018, p.s.n.).

c) *intents* – são componentes que inicializam os demais componentes. Eles são utilizados para criar atividades e serviços. A Figura 4 apresenta esquematicamente a forma de atuação dos *intents*.

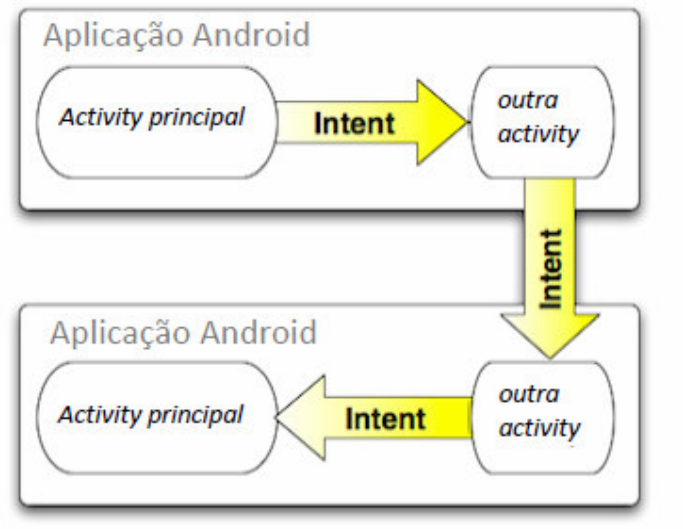


Figura 4 – Atuação dos intents
Fonte: traduzido de Oliveira (2018, p.s.n.).

d) *broadcast receivers* – são os componentes responsáveis por responder a eventos do sistema. A Figura 5 representa de maneira esquemática a atuação dos *broadcast receivers*.

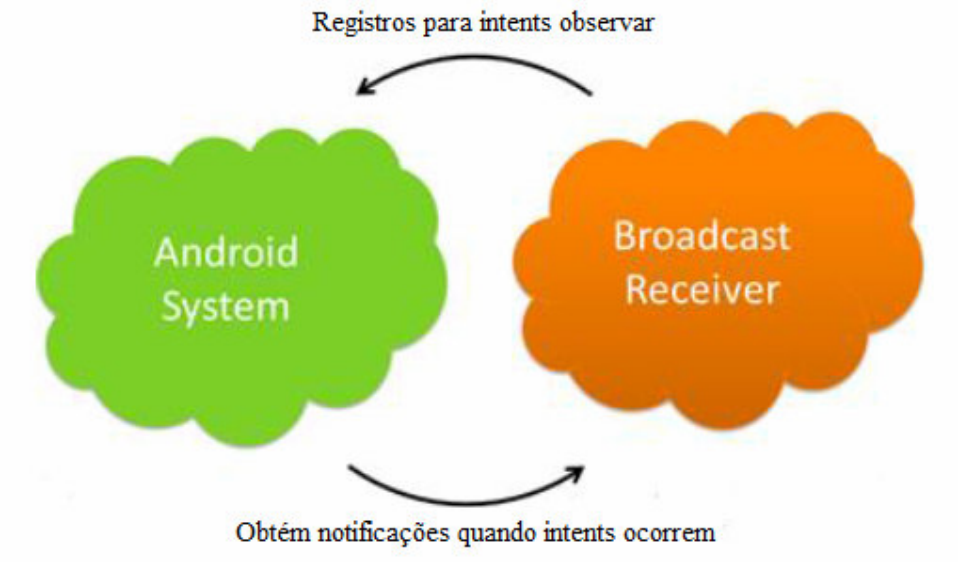


Figura 5 – Atuação dos *boradcast receivers*
Fonte: traduzido de Oliveira (2018, p.s.n.).

3 ESTADO DA ARTE

Existem diversos aplicativos para dispositivos móveis que realizam funcionalidades semelhantes às definidas para o aplicativo desenvolvido como resultado deste trabalho. Alguns deles foram desenvolvidos para cidades específicas e outras para cidades que disponibilizam o serviço utilizado pelo aplicativo. Apesar dessas soluções existentes, ressalta-se que este é um trabalho acadêmico que visa o aprendizado de tecnologias e fornecer uma solução para o auxílio no uso de transporte coletivo. Nesse caso, para experimentação, será utilizada a cidade de Pato Branco.

A seguir são apresentados alguns desses aplicativos:

a) +Ônibus Brasília - aplicativo que permite ao passageiro consultar os horários dos ônibus em tempo real e traçar o itinerário para o destino, sendo, o usuário informando de previsões de horário. A posição dos veículos é monitorada por GPS. O aplicativo é para Android e até 31/01/2018, não apresentava a funcionalidade para acompanhar os veículos em tempo real ou para o usuário ser informado de todas as paradas da referida linha. A Figura 6 apresenta uma imagem da tela do aplicativo +Ônibus Brasília.

Figura 6 – Interface do aplicativo +Ônibus Brasília

Fonte: Moreira (2018).

b) Moovit – esse aplicativo é um assistente pessoal de mais de 120 milhões de usuários do transporte público em todo o mundo. Ele fornece informações sobre ônibus, metrô, trem, Veículo Leve sobre Trilhos (VLT), bicicleta ou a pé. O aplicativo é disponibilizado pela Google Play e pela App Store. A Figura 7 apresenta uma imagem da tela do aplicativo disponível no próprio site da Google Play (2018).

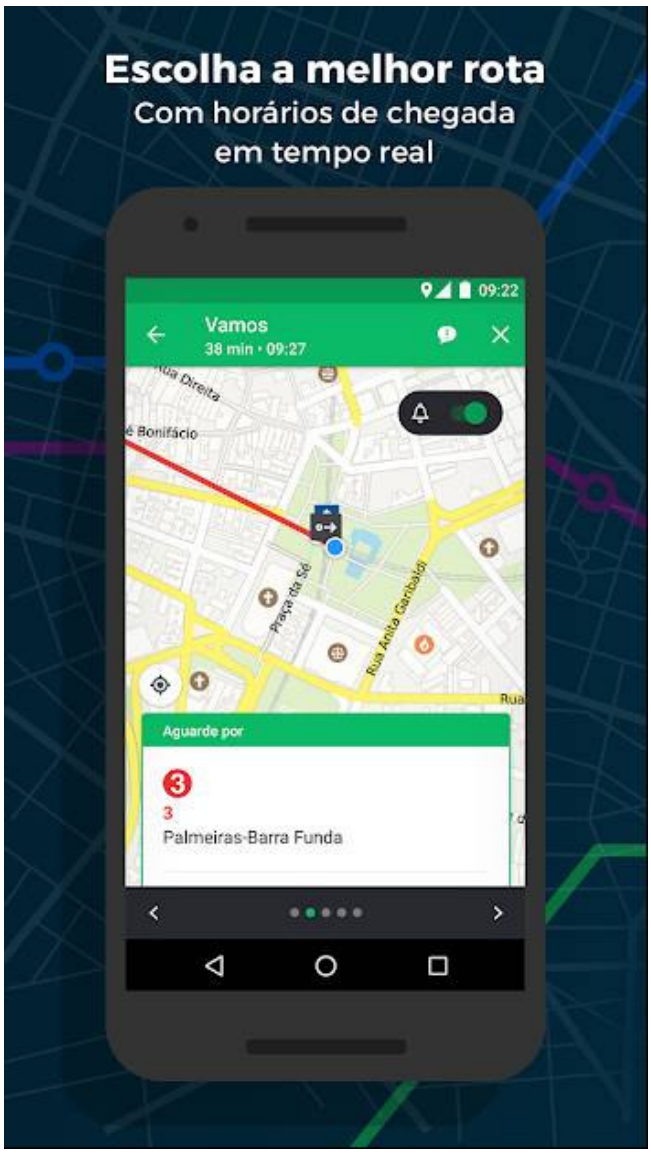


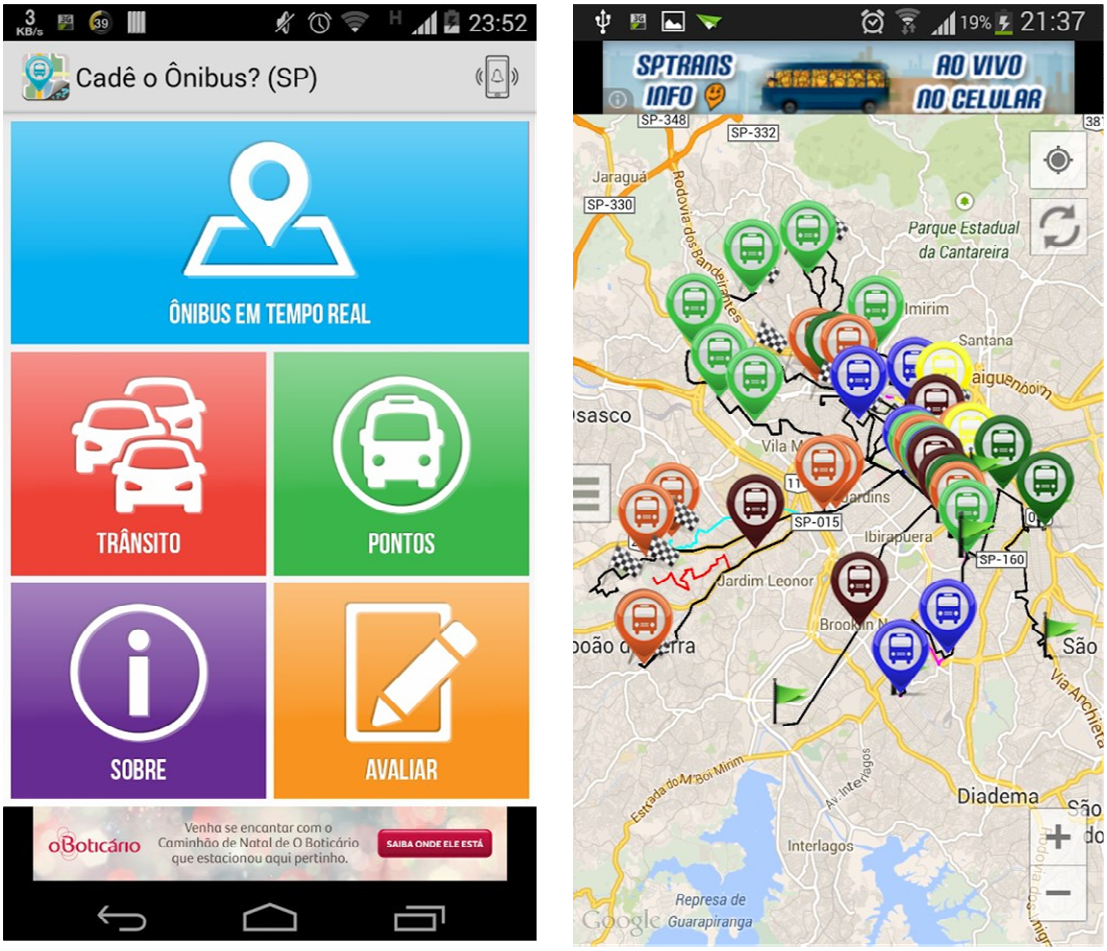
Figura 7 – Interface do aplicativo Moovit
Fonte: Google Play (2018).

c) CittaMobi - aplicativo que apresenta os pontos mais próximos do usuário e a previsão de chegada dos ônibus em tempo real e permite consultar itinerários e buscar por veículos adaptados para cadeirantes. A ferramenta está disponível em 12 cidades brasileiras (CITTAMOBI, 2018, CANALTECH, 2018): Colatina (ES), Diadema (SP), Juiz de Fora

(MG), Maceió (AL), Recife (PE), Rio Branco (AC), Rio Grande (RS), São Caetano do Sul (SP), Santo André (SP), Santa Rita (PB) e Volta Redonda (RJ). A Figura 8 apresenta a imagem de tela do aplicativo.

Figura 8 – Interface do aplicativo CittaMobi
Fonte: Google Play (2018).

d) Cadê o ônibus? – Esse aplicativo apresenta informações apenas das linhas de ônibus da cidade de São Paulo (CANALTECH, 2018). São apresentados dados sobre a posição geográfica de ônibus municipais e intermunicipais em tempo real e o itinerário completo das linhas, com horários de partida dos ônibus e os pontos mais próximos. O aplicativo também informa sobre o trânsito e emite alertas com notícias da SPTrans, do Metrô e da Companhia Paulista de Trens Metropolitanos (CPTM) e avisa sobre greves e manifestações na cidade. A Figura 9 (a) e (b) apresenta telas do aplicativo.



(a) **(b)**

Figura 9 – Interface do aplicativo Cadê o Ônibus
Fonte: Google Play (2018).

Além desses aplicativos apresentados, há outros, como o Urbanoide que é um aplicativo que mostra detalhes em tempo real sobre as linhas de ônibus de São Paulo. E Vá de Ônibus que é um aplicativo que oferece informações sobre as linhas de ônibus da cidade do Rio de Janeiro. Esse aplicativo permite calcular rotas, localizar os pontos mais próximos, consultar itinerários e acompanhar a localização dos ônibus em tempo real.

4 MATERIAIS E MÉTODO

A seguir estão os materiais e o método utilizado para a modelagem e a implementação do sistema obtido como resultado deste trabalho.

4.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas no desenvolvimento do sistema proposto neste trabalho.

Nome	**Versão**	**Aplicação no projeto**
Android Studio	3.0.1	*Integrated Development Environment* (IDE) de desenvolvimento do aplicativo móvel.
Astah Community	37	Modelagem do sistema: diagrama de classes e diagrama de caso de uso.
Bootstrap	3	Framework web para desenvolvimento de componentes de interface e *front-end*.
Google Maps Platform		Plataforma para utilizar as APIs do Google Maps. Por exemplo, adicionar um mapa a aplicação e manipula-lo.
Hibernate	4	Framework para o mapeamento objeto-relacional escrito na linguagem Java.
HTML	5	Estruturação da interface.
Java	8	Linguagem de programação para desenvolvimento.
jQuery	3	Biblioteca de funções JavaScript para interagir com o *HyperText Markup Language* (HTML)
Lombok	1	Biblioteca Java voltada para produtividade.
Netbeans	8.2	IDE para desenvolvimento da aplicação servidor que possibilitará atualizar dados do aplicativo, quando houver inclusão de paradas de ônibus em um determinado itinerário de ônibus, por exemplo.
PgAdmin	4 v 3.5	Servidor de banco de dados.
PostgreSQL	1.18.1	Banco de dados no servidor.
Retrofit	2	API para integração entre aplicações.
SQLite		Banco de dados do aplicativo armazenado no dispositivo móvel.
Vraptor	4	Framework *Model-View-Controller* (MVC) *web* para desenvolvimento ágil com Java.

Quadro 1 - Ferramentas e tecnologias

4.2 MÉTODO

Os procedimentos para definir o sistema estão baseados nas fases propostas por Pressman (2002) para o modelo sequencial linear que são análise, projeto, codificação e testes. A seguir a descrição do que foi realizado em cada uma dessas fases para o desenvolvimento deste trabalho:

a) Levantamento de requisitos – para definir o problema a autora desse trabalho identificou a necessidade no cenário atual por meio do processo de experimentação, como usuário de transporte coletivo; além disso, foi realizada a análise de aplicativos com funcionalidades semelhantes, apresentados no Capítulo 3.

O levantamento dos requisitos iniciou com a listagem dos requisitos considerados essenciais para a aplicação, enfatizando funcionalidades e aspectos de qualidade considerados relevantes para os objetivos da aplicação. A partir dessa listagem foram definidos os requisitos funcionais e a eles foram agregados requisitos não funcionais. Posteriormente desenvolveu-se um esboço inicial do padrão das telas para o sistema. O desenvolvimento das telas foi realizado na própria IDE de implementação. Sendo possível, assim, uma visão mais real dos elementos de composição e a organização das telas.

b) Análise – para realizar a análise foram utilizados conceitos e a metodologia do paradigma da orientação a objetos. Por meio da ferramenta Astah Community, foram elaborados os diagramas de casos de uso e o diagrama de classes a partir da descrição dos requisitos. Optou-se por um diagrama de classes ao invés de um diagrama de entidades e relacionamentos do banco de dados, embora, em princípio as classes sejam todas persistentes (representam entidades do banco) pela possibilidade de definir os métodos para as classes. Já que os campos das tabelas são representados pelos atributos das classes.

c) Projeto – foram utilizados os diagramas de caso de uso e de classes para a composição do projeto do sistema. Também foi determinada a cronologia de implementação do sistema conforme elementos mais importantes e relevantes em decorrência de dependências entre tarefas.

d) Implementação (codificação) do sistema – foi utilizada a ferramenta Netbeans como IDE de desenvolvimento da aplicação servidor e PostgreSQL para armazenamento dos dados no banco de dados no servidor. A linguagem Java de programação para desenvolvimento da aplicação servidor e a implementação da lógica de negócio. O NetBeans Text para construção do *frontend* do servidor. O Android Studio utilizado como IDE de desenvolvimento do aplicativo móvel; e o SQLite como banco de dados do aplicativo armazenado no dispositivo móvel.

e) Realização dos testes - na implementação foram realizados os testes relacionados ao código. Esses testes foram informais e realizados pela autora deste trabalho visando identificar erros de codificação e verificação e validação dos requisitos definidos para o aplicativo.

5 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho.

5.1 ESCOPO DO SISTEMA

O aplicativo é composto por uma parte servidor e outra *mobile*. A aplicação servidor é desenvolvida em Java para *web* e é utilizada para a inclusão de empresas, cadastro e composição das linhas de ônibus, cadastro de pontos de parada de ônibus e de pontos de referência relacionados aos pontos de parada. A aplicação *mobile* é basicamente para consulta de dados baseadas nos cadastros realizados na aplicação servidor e para que o usuário armazene favoritos de linhas e de pontos.

O aplicativo desenvolvido é para Android, utiliza a *Application Programming Interface* (API) Google Maps e fornece informações para acompanhamento das rotas de transporte coletivo. O aplicativo permite, por exemplo, a visualização das rotas fixas, de pontos de paradas e horários correspondentes. Além disso, disponibilizará opções de filtros por origem, destino, horários, pontos de parada, pontos de referência, nome da linha, entre outros. O acesso aos filtros é disponibilizado por meio de formulário, ou ainda, o usuário poderá utilizar o mapa disponibilizado pelo aplicativo para marcar determinado ponto, como também, usar a sua localização atual como marcação de ponto, necessitando, para isso conexão com a Internet.

A partir dos dados fornecidos na filtragem são sugeridas as melhores opções de linhas, pontos de ônibus mais próximos, horários e tempo aproximado da viagem. Também será possível salvar rotas e linhas nos favoritos do dispositivo móvel que está acessando a aplicação, a fim de agilizar as buscas.

Esse aplicativo tem o intuito de substituir os dados que, geralmente, em pequenas cidades, são disponibilizados em papel ou em sites *web* normalmente na forma de cronograma das linhas, sem opções de busca.

O aplicativo desenvolvido não necessita de autenticação de usuário, nem de conexão com a Internet para uso. Os dados são armazenados no próprio aparelho, exceto quando utilizada a funcionalidade de localização atual que permite marcar as coordenadas geográficas do usuário como um ponto. Quando houver conexão com a Internet, o usuário poderá realizar atualizações disponíveis por meio da sincronização de dados com a aplicação servidor.

Futuramente, se os veículos que realizam o transporte coletivo ou os pontos de parada desses veículos forem equipados com dispositivos eletrônicos que coletem dados sobre horários que o veículo de determinada linha passou, por exemplo, esses dados poderão ser disponibilizados pelo aplicativo. Para isso será necessário complementar a implementação com as novas funcionalidades, contudo, o que já foi desenvolvido permanecerá, sendo essas novas funcionalidades agregadas ao aplicativo.

5.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta os requisitos funcionais definidos para o sistema. Os requisitos funcionais foram divididos entre a aplicação servidor, que é *web* e mantém os cadastros relacionados à funcionalidade de negócio do sistema e disponibiliza para consulta; e a aplicação cliente que é *mobile*, permite a realização de consultas a partir dos dados disponibilizados pela aplicação servidora e possibilita a inclusão de dados (definidos como favoritos de linhas e pontos de parada de ônibus) pelo usuário.

	Requisito	**Descrição**
Aplicação servidor		
01	Manter empresa	Empresas que realizam o transporte coletivo com dados como: nome fantasia, razão social e imagens de ônibus da empresa. Uma empresa pode ter imagens diferentes para os seus veículos, como ocorre com veículos comemorativos, por exemplo.
02	Manter linhas	Linhas, trajetos, realizadas pelos veículos das empresas com nome e descrição. Uma linha é composta por um conjunto de pontos de parada de ônibus.
03	Manter pontos de parada	Pontos de parada dos ônibus. Os pontos são utilizados na composição das linhas e possuem informações como latitude, longitude, horário estimado para o ônibus passar no referido ponto.
04	Manter pontos de referência	Estabelecimentos comerciais, construções e outros que podem ser utilizados como pontos de referência para melhor identificação e localização de pontos de ônibus. Exemplo: próximo da Praça Abc, em frente ao Supermercado Xyz. Os dados dos pontos de referência são representados por: nome, descrição, foto, latitude e longitude. Esses pontos serão associados com pontos de parada para facilitar a identificação dos pontos nos quais os ônibus param para embarque e desembarque de passageiros.
Aplicação cliente		
05	Manter favoritos	O usuário pode cadastrar pontos de parada como favoritos, facilitando no momento da busca de informações como horário de passagem de determinada empresa por esses pontos. O usuário pode cadastrar linhas como favoritos, facilitando no momento da busca de informações.
06	Listar linhas	Listar linhas existentes que passam por determinado ponto.
07	Listar pontos	Listar os pontos de determinada linha.

08	Buscar dados	Realizar buscas por: empresa: nome fantasia, razão social; pontos de referência: nome, descrição, latitude e longitude; pontos de parada: latitude e longitude; linhas: nome, um ou mais pontos de parada; favoritos de pontos de parada: pontos de parada; favoritos de linhas: linhas; ponto de origem: latitude e longitude; pontos de destino: latitude e longitude; horário de saída: pontos; horário de chegada: pontos.

Quadro 2 – Requisitos funcionais

No aplicativo, os pontos de parada e os pontos de referência são armazenados na mesma tabela e tratados pela mesma classe. Haverá um campo e atributo para identificar se é um ponto de referência ou de parada.

Os requisitos não funcionais são apresentados no Quadro 3.

	Requisito não funcional	**Descrição**
Aplicação servidor		
01	Ambiente de execução da aplicação servidor	A aplicação servidor deve ser *web*.
02	Integração da aplicação com Google Maps	O aplicativo deve ser integrado ao Google Maps para indicação de pontos geográficos (latitude e longitude) dos pontos de parada utilizados para a composição de rotas, bem como para apresentar o traçado das rotas a partir dos seus respectivos pontos.
03	Cadastro de pontos	O sistema deve disponibilizar o cadastro de pontos no formato de formulário. O sistema deve disponibilizar o cadastro de pontos por meio da marcação no mapa apresentado pelo aplicativo.
04	Autenticação	O sistema deve solicitar autenticação de usuário.
Aplicação cliente		
05	Filtros de busca	O sistema deve disponibilizar filtros de busca no formato de formulário.
06	Busca pelo mapa	O sistema deve disponibilizar filtros de busca de pontos por meio da marcação no mapa.
07	Busca pela localização atual do usuário	O sistema deve disponibilizar filtros de busca de pontos a partir da localização atual do usuário.
08	Execução *off-line*	O sistema não deve requerer conexão com a Internet para utilização da aplicação Android, pois os dados devem ser armazenados no próprio aparelho.
09	Atualizações	O sistema deve possibilitar que o usuário realize atualizações do aplicativo.
10	Autenticação	O aplicativo Android não deve solicitar autenticação de usuário.

Quadro 3 – Requisitos não funcionais

Os requisitos funcionais do sistema foram organizados em casos de uso. A Figura 10 apresenta o diagrama de casos de uso.

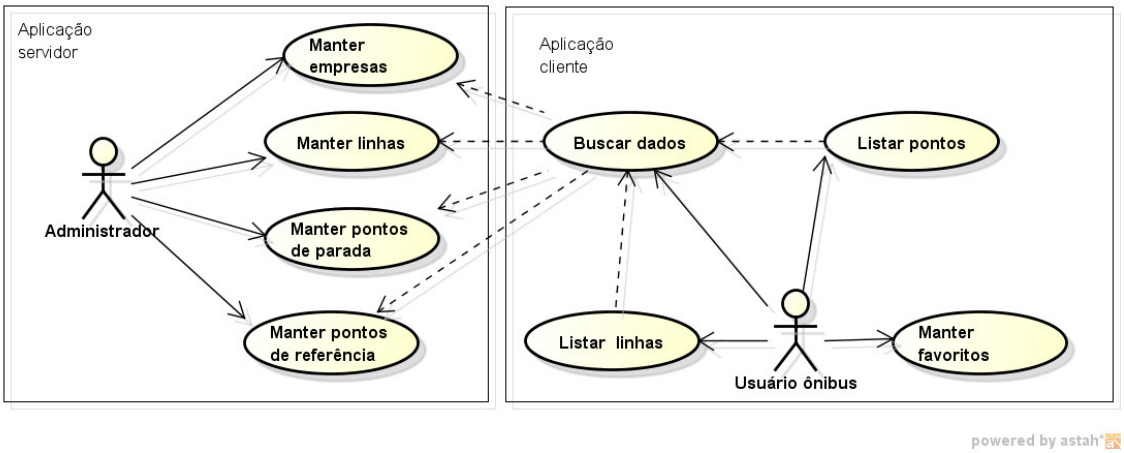


Figura 10 – Diagrama de casos de uso

Os casos de uso, apresentados na Figura 10, foram divididos entre aplicação cliente (*mobile*) e servidor (*web*). O cliente (usuário de ônibus) fará consultas e poderá armazenar seus favoritos de linhas e pontos tendo como base os dados cadastrados no servidor.

Os Quadros 4 a 7 apresentam a descrição das ações de inclusão, alteração, exclusão e consulta dos casos de uso manter que pertencem à aplicação servidor e o manter favoritos pertencente à aplicação cliente. No Quadro 4 está a expansão da operação incluir dos casos de uso manter.

<table>
<tr><td colspan="2">Caso de uso:
Incluir (refere-se à operação de inclusão de todos os casos de usos identificados como “manter”).
Descrição:
Inclusão dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita inclusão de um registro no sistema.
Atores:
Administrador na aplicação servidor e Usuário ônibus na aplicação cliente.
Pré-condição:
Não há para o aplicativo mobile (usuário ônibus), para a aplicação web (administrador) o usuário deve estar autenticado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para realizar o cadastro inserindo as informações necessárias.
2. O sistema insere as informações no banco de dados e informa ao usuário o status do procedimento.
Pós-Condição:
Registro inserido no banco de dados.
Extensões:
Campos obrigatórios faltantes e campos no formato incorreto.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Campos obrigatórios não</td><td>1.1. O usuário deixa de informar dados obrigatórios e clica em salvar.</td></tr>
</table>

informados.	1.2. O sistema valida que não foram informados todos os campos obrigatórios e exibe mensagem ao usuário sem salvar o registro. 1.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.
2. Campos informados em formato incorreto.	2.1. O usuário informa dados em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.

Quadro 4 - Operação "incluir" dos casos de uso de cadastro

A descrição da operação alterar dos casos de uso manter é apresentada no Quadro 5.

Caso de uso:
Alterar (refere-se à operação de alteração de todos os casos de usos identificados como "manter").
Descrição:
Alteração dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita alteração de um registro no sistema.
Atores:
Administrador na aplicação servidor e Usuário ônibus na aplicação cliente.
Pré-condição:
Registro estar incluso no sistema. Para a aplicação *web* (administrador) o usuário deve estar autenticado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O sistema apresenta o registro selecionado para alteração.
3. Ator altera os dados do registro.
4. O sistema altera as informações no banco de dados e informa ao usuário o *status* do procedimento.

Pós-Condição:
Registro alterado no banco de dados.
Extensões:
Campos obrigatórios faltantes e campos no formato incorreto.

Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. O usuário apaga dados obrigatórios e clica em salvar. 1.2. O sistema valida que não foram informados todos os campos obrigatórios e exibe mensagem ao usuário sem salvar o registro. 1.3. O sistema permanece na tela de edição mantendo as alterações realizadas.
2. Campos informados em formato incorreto.	2.1. O usuário altera os dados deixando em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de edição mantendo as alterações realizadas.

Quadro 5 - Operação "alterar" dos casos de uso de cadastro

O Quadro 6 apresenta a descrição da operação excluir dos casos de uso manter.

Caso de uso:
Excluir (refere-se à operação de exclusão de todos os casos de usos identificados como "manter").
Descrição:
Exclusão dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita exclusão de um registro no sistema.
Atores:
Administrador na aplicação servidor e Usuário ônibus na aplicação cliente.
Pré-condição:
Registro estar incluso no sistema. Para a aplicação *web* (administrador) o usuário deve estar autenticado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para exclusão do registro.
2. O sistema exclui as informações no banco de dados e informa ao usuário o *status* do procedimento.

Pós-Condição:
Registro excluído no banco de dados.
Extensões:
Registro possui vínculo com outros cadastros.

Nome do fluxo alternativo (extensão)	Descrição
1. Exclusão de registro com vínculos no sistema	1.1. O usuário clica em excluir um registro que possui vínculos no sistema. 1.2. O sistema verifica que o registro tem vínculos, não o exclui e exibe mensagem de alerta ao usuário.

Quadro 6 - Operação "excluir" dos casos de uso de cadastro

No Quadro 7 está a descrição da operação consultar referente aos casos de uso manter. Essa descrição também se aplica aos casos de uso Buscar linhas e Buscar pontos do ator Usuário ônibus.

Caso de uso:
Consultar (refere-se à operação de consulta de todos os casos de usos identificados como "manter").
Descrição:
Consulta dos dados cadastrais dos registros do sistema.
Evento Iniciador:
Ator solicita consulta de um registro no sistema.
Atores:
Administrador na aplicação servidor e Usuário ônibus na aplicação cliente.
Pré-condição:
Registro estar incluso no sistema. Para a aplicação *web* (administrador) o usuário deve estar autenticado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O ator indica os filtros desejados para consulta.
3. O sistema apresenta os dados da consulta ao usuário.

Pós-Condição:
Dados da consulta apresentados ao usuário.

Quadro 7 - Operação "consultar" dos casos de uso de cadastro

No Quadro 8 está a descrição do caso de uso buscar dados. O cliente (usuário *mobile*) conecta-se ao servidor para atualizar a sua base de dados. Essa atualização é necessária quando, por exemplo, novos pontos são incluídos em uma linha.

<table>
<tr><td colspan="2">Caso de uso:
Buscar dados
Descrição:
O cliente busca dados no servidor para atualizar a base de dados local. A sincronização do banco de dados da aplicação servidor e cliente é realizada.
Evento Iniciador:
Ator solicita atualização da base de dados.
Atores:
Usuário ônibus na aplicação cliente.
Pré-condição:
Haver mudanças na base de dados do servidor que justifiquem a atualização de dados no cliente.
Sequência de Eventos:
1. Ator solicita atualização de dados.
2. Servidor verifica se há atualizações a serem realizadas, se houver servidor e cliente são sincronizados.
3. Servidor atualiza dados na base do cliente e informa que o procedimento foi realizado.
Pós-Condição:
Base de dados do cliente atualizada.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Não há dados para atualizar</td><td>1.1. O servidor verifica que não há dados diferentes no banco de dados desde a última sincronização com o cliente.
1.2. Servidor informa que não há atualização para serem realizadas.</td></tr>
</table>

Quadro 8 - Caso de uso buscar dados

A Figura 11 apresenta o diagrama de classes desenvolvido para a aplicação servidor. A classe Ponto tem um auto-relacionamento porque um ponto pode ter pontos de referência associados. O atributo “referencia” define se é um ponto de parada de ônibus ou um ponto de referência para esses pontos de parada.

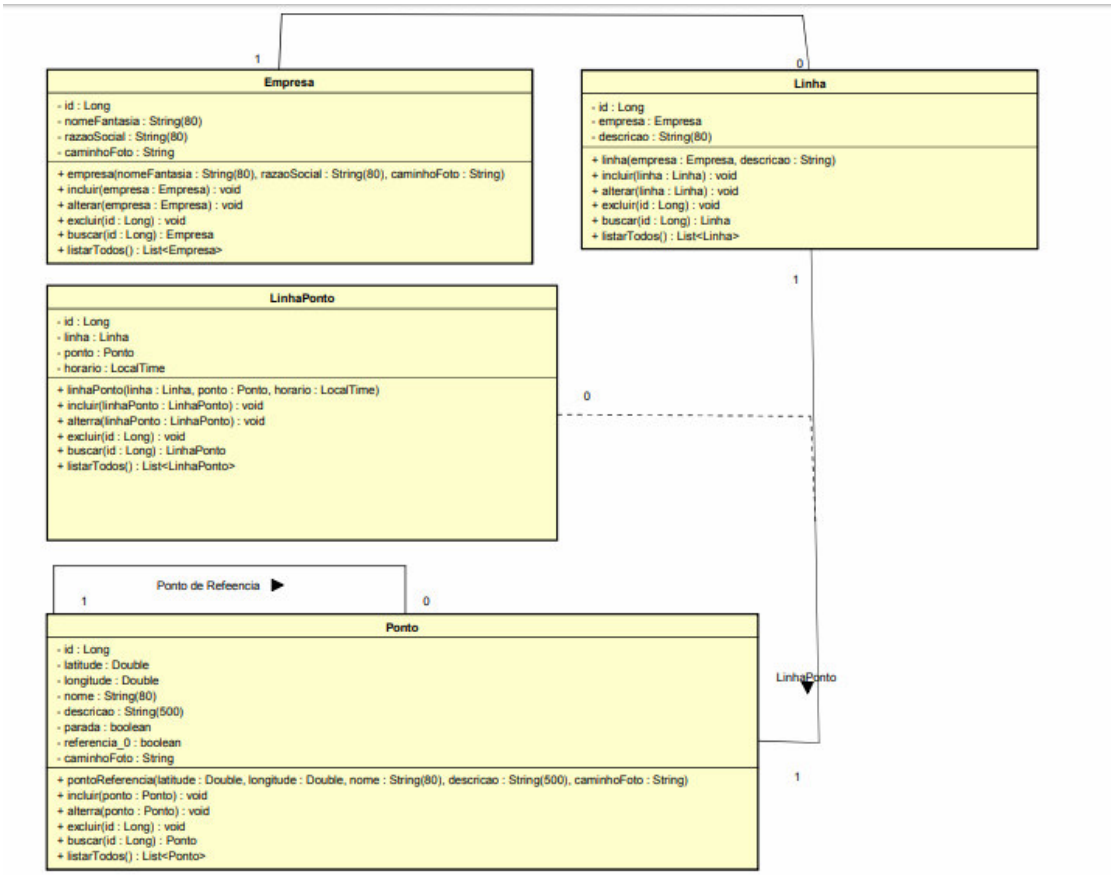


Figura 11 - Diagrama de classes

A classe LinhaPonto surge do relacionamento entre linha e ponto. Uma linha ponto é uma composição de pontos que definem uma rota que o veículo vai percorrer. Um mesmo ponto pode ser utilizado por linhas distintas.

Os Quadros 9 a 14 apresentam a descrição das classes apresentadas no diagrama da Figura 11.

O Quadro 9 apresenta a classe empresa.

Identificação:	Empresa
Descrição:	Registro individual dos dados referentes à empresa.
Atributos:	-idEmpresa (Long): número de identificação da empresa; - nomeFantasia (String): nome fantasia da empresa; - razaoSocial (String): nome da razão social da empresa; - imagem (byte): fotos dos modelos de ônibus da empresa.
Métodos:	incluir(), alterar(), excluir(), buscar(), listarTodos.
Requisitos envolvidos:	RF01.

Quadro 9 – Classe empresa

O Quadro 10 apresenta a classe linha.

Identificação:	Linha
Descrição:	Registro individual dos dados referentes à linha de cada empresa.
Atributos:	-idLinha (Long): número de identificação da linha; - empresa (Long): empresa da linha; - descricao (String): descrição da linha;
Métodos:	incluir(), alterar(), remover(), excluir(), buscar(), listarTodos.
Requisitos envolvidos:	RF01, RF02.

Quadro 10 – Classe linha

No Quadro 11 está a descrição da classe ponto.

Identificação:	Ponto
Descrição:	Registro individual dos dados referentes ao ponto.
Atributos:	-idPonto (Long): número de identificação do ponto; - latitude (Double): latitude do ponto; - longitude (Double): longitude do ponto; - nome (String): nome de identificação do ponto; - descricao (String): descrição para identificação do ponto; - imagem (byte): fotos do ponto; - referencia (boolean): para identificar se o ponto é um ponto de referencia e/ou um ponto de parada;
Métodos:	incluir(), alterar(), excluir(), buscar(), listarTodos.
Requisitos envolvidos:	RF01, RF02, RF03, RF04.

Quadro 11 – Classe ponto

O Quadro 12 apresenta a classe linhaPonto. O método buscarPonto() é utilizado para localizar pontos próximos. Por exemplo, quando não se sabe exatamente a localização de um ponto de passagem de ônibus, mas se sabe o endereço da localização atual ou que ser quer ir. Nesse caso, o aplicativo busca, no conjunto ordenado de pontos, as latitudes e longitudes que se aproximem do ponto informado, tanto para o ponto de origem (localização do usuário) com para o ponto de destino (localização que o usuário deseja ir).

Identificação:	LinhaPonto
Descrição:	Registro individual dos dados referente à composição da linha a partir de um conjunto de pontos de parada.
Atributos:	-idLinhaPonto (Long): número de identificação da relação da linha com o ponto de parada; - linha (Long): conjunto ordenado de pontos que compõem a linha; - horários (Double): horário de passagem do ônibus em cada ponto daquela linha;
Métodos:	incluir(), alterar(), excluir(), buscar(), buscarPonto, listarTodos.
Requisitos envolvidos:	RF01, RF02, RF03, RF04.

Quadro 12 – Classe linhaPonto

A classe linhaFavorita no aplicativo cliente é apresentadas no Quadro 13.

Identificação:	LinhaFavorita
Descrição:	Registro individual dos dados referente às linhas favoritas.
Atributos:	- linha (Long): identificação da linha favorita;
Métodos:	incluir(), excluir(), buscar(), listarTodos.
Requisitos envolvidos:	RF01, RF02, RF03, RF04, RF06.

Quadro 13 – Classe linhaFavorita

O Quadro14 apresenta a classe pontoFavorito no aplicativo cliente.

Identificação:	PontoFavorito
Descrição:	Registro individual dos dados referente aos pontos favoritos.
Atributos:	- pontoLinha (Long): identificação do ponto favorito;
Métodos:	incluir(), excluir(), buscar(), listarTodos.
Requisitos envolvidos:	RF01, RF02, RF03, RF04, RF06.

Quadro 14 – Classe pontoFavorito

A Figura 12 exibe o diagrama de entidade e relacionamento do banco de dados SQLite da aplicação Andriod.

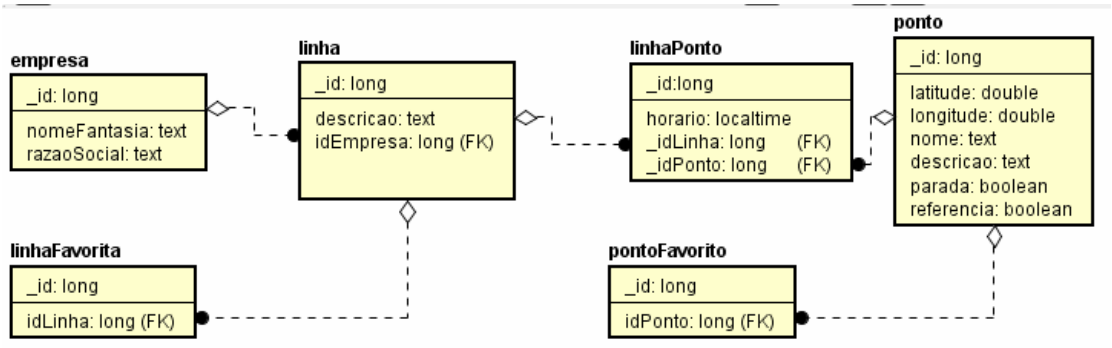


Figura 12 - Diagrama Entidade Relacionamento – SQLite

No diagrama do banco de dados a tabela linha está associada a uma empresa que realiza o referido trajeto de ônibus. Uma linha é composta por pontos. E o usuário mobile pode definir linhas e pontos como favoritos.

5.3 APRESENTAÇÃO DO SISTEMA

A seguir são apresentadas as funcionalidades e o uso de recursos tecnológicos do sistema por meio de suas telas e definição de padrões.

A Figura 13 exibe a tela de autenticação da aplicação servidor.

Figura 13 - Tela de login da aplicação servidor

Após autenticado, o usuáro tem acesso à tela inicial e ao menu do sistema. No menu são disponibilizadas as seguintes opções: Usuários, Empresas, Linhas, Pontos, Pontos das Linhas, Sobre e Sair. Os itens de menu são representados por ícones que ficam na lateral esquerda da tela, como apresentado na Figura 14. Cada item possui uma descrição associada que visa auxiliar na identificação da funcionalidade vinculada a cada ícone. Essa descrição é apresentada quando o referido ícone recebe o foco .

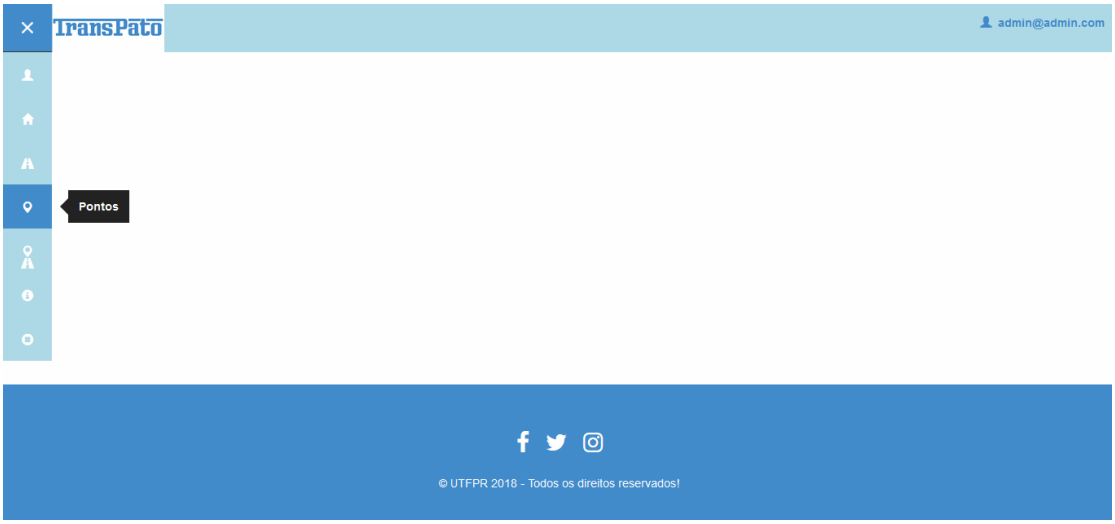


Figura 14 - Tela inicial e menu da aplicação servidor

Ao acessar uma opção de menu é apresentada ao usuário a relação correspondente dos dados de cadastro da funcionalidade associada ao respectivo ícone. Na listagem são apresentados os dados em forma de tabela e na lateral direita de cada linha estão os ícones para editar e excluir o respectivo cadastro. Essa listagem visa possibilitar a busca e/ou a

manutenção dos registros. Na Figura 16 são apresentados dados do cadastro de linhas de ônibus.

Figura 15 - Tela padrão para listagem de registros na aplicação servidor

A filtragem dos dados é efetuada diretamente na tabela de listagem conforme a digitação ocorre. No caso das ações de inclusão e alteração de registros, a aplicação apresenta uma tela contendo o formulário padrão semelhante ao da Figura 15. A tela dessa Figura é para realizar o cadastro de pontos de parada de ônibus ou de pontos de referência a um ponto de parada de ônibus.

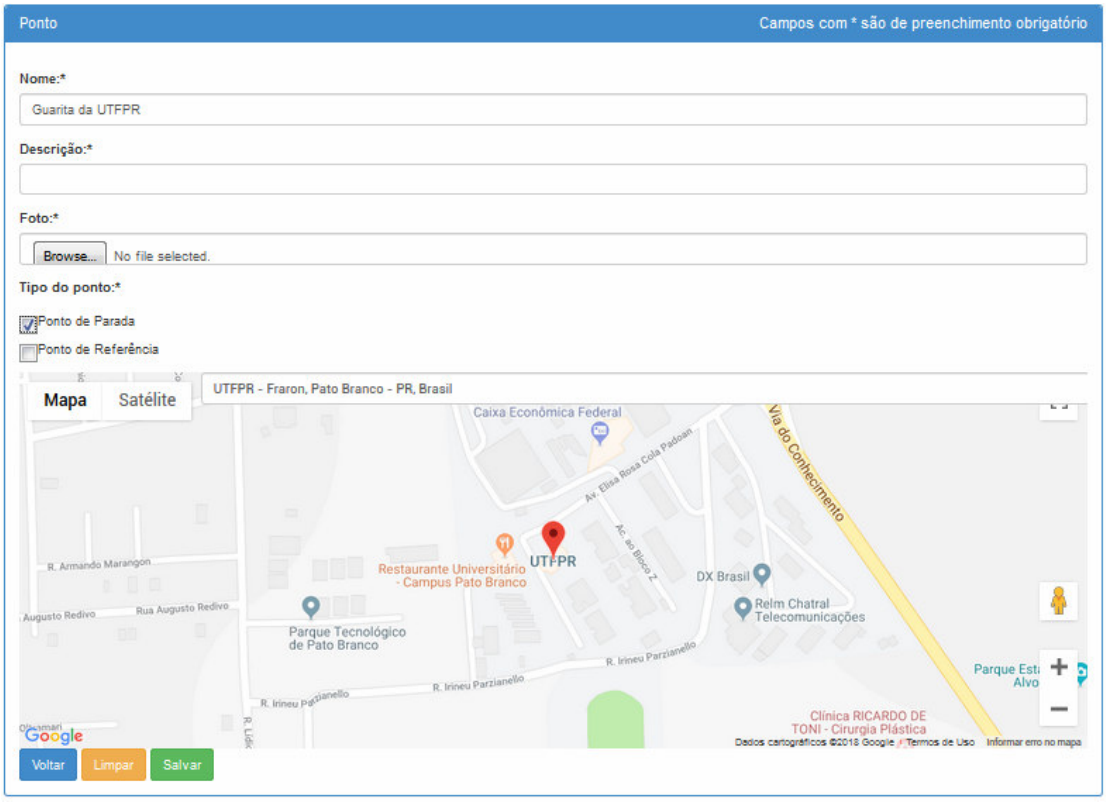


Figura 16 - Tela padrão de formulário da aplicação servidor

Na ação de exclusão, o sistema solicita confirmação ao usuário para realizar a execução dessa função. A Figura 17 é da tela que é apresentada quando a opção de exclusão de registro é selecionada em qualquer cadastro.

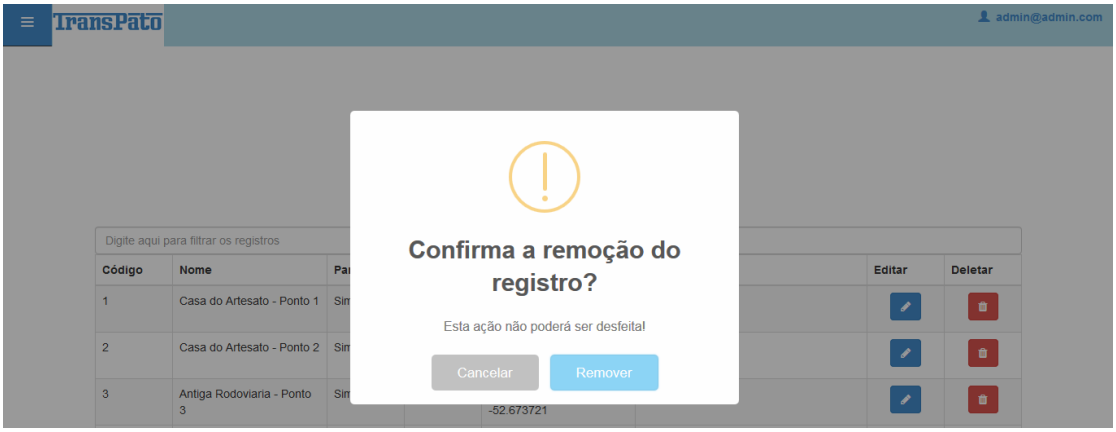


Figura 17 - Tela padrão de confirmação para remoção de registros da aplicação servidor

Considerando o contexto do aplicativo Android. A tela inicial é constituída pelo mapa e pelo menu, que por sua vez possui as seguintes opções: Definir rota, Empresas, Linhas, Meus Favoritos, Sincronizar e Sobre. A Figura 18 apresenta a listagem dessas opções. Após a instalação do aplicativo a primeira opção a ser escolhida é Sincronizar.

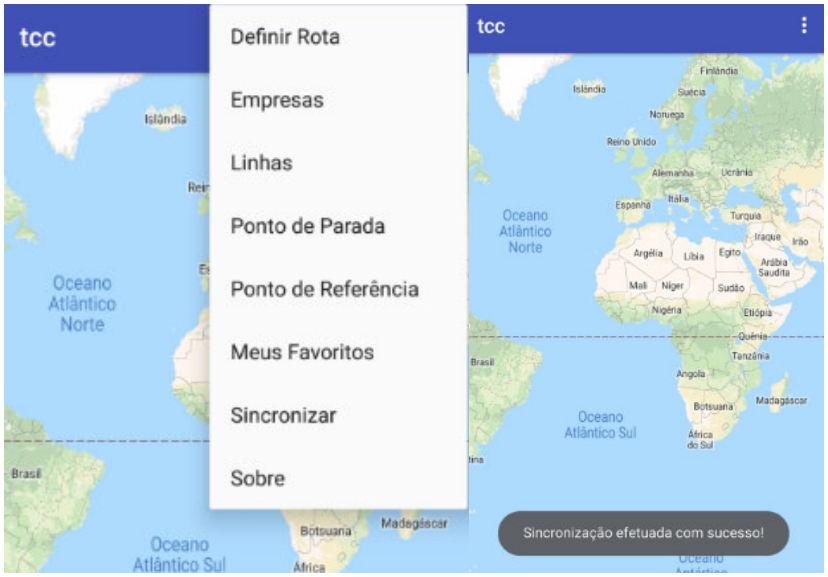


Figura 18 - Tela inicial e menu no aplicativo Android

A opção de menu Definir Rota possibilita que o usuário informe o ponto de origem e de destino, bem como o horário pretendido para realizar a pesquisar das linhas e pontos. A Figura 19 apresenta a tela de definição de rota.

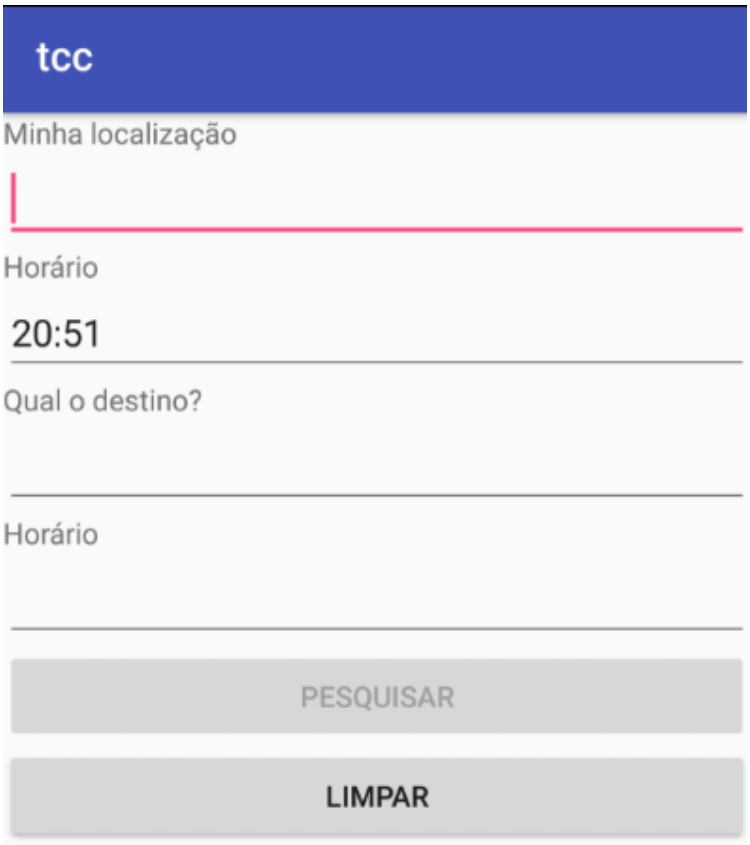


Figura 19 - Tela definir rota do aplicativo Android

Em Empresas, o aplicativo exibe uma lista das empresas cadastradas, como mostrado na Figura 20.

Figura 20 - Tela padrão de empresas no aplicativo Android

Ao clicar sobre determinada empresa, são apresentadas as linhas dessa empresa, conforme mostra a Figura 21.

Figura 21 - Tela padrão de linhas do aplicativo Android

Ao clicar sobre um registro de linha o aplicativo apresenta os pontos que compõem a referida linha, bem como o horário que foi pré-definido para passagem do ônibus nos respectivos pontos. A Figura 22 apresenta a listagem dos pontos de uma linha selecionada.

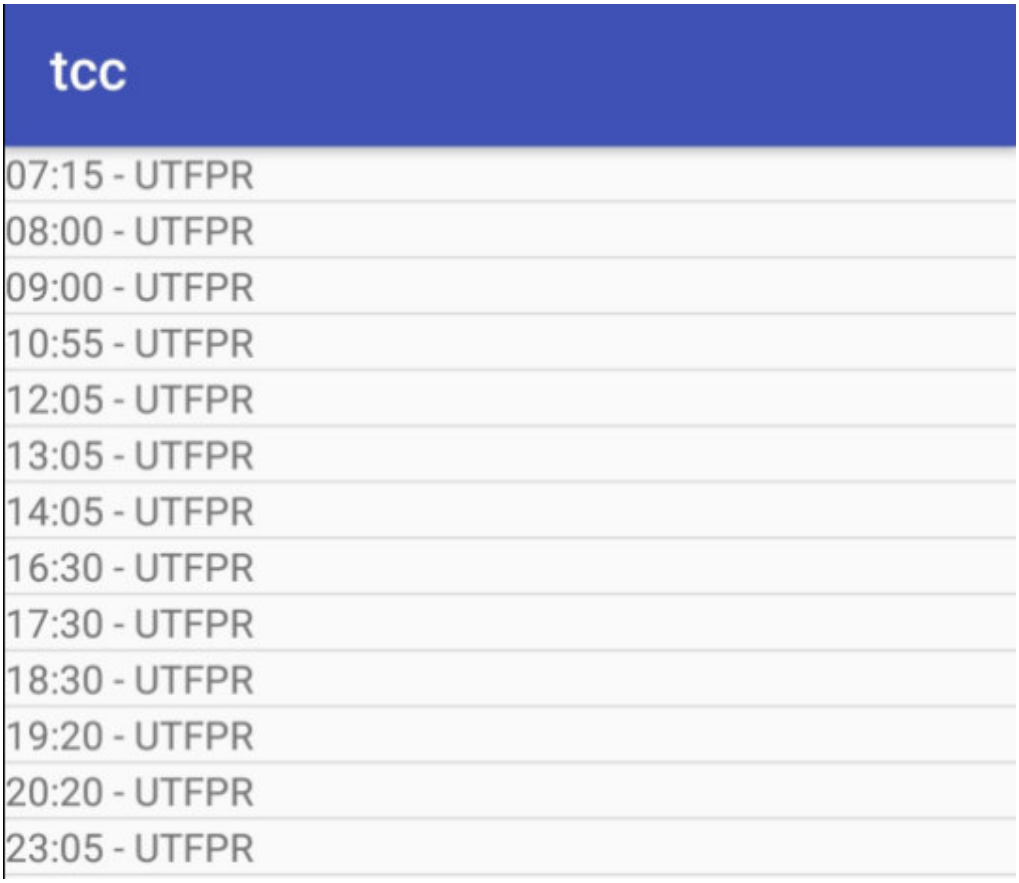


Figura 22 - Tela padrão de pontos do aplicativo Android

É possível abrir o detalhamento do ponto para visualizar outras informações cadastradas do referido ponto. A Figura 23 apresenta os dados de um ponto cadastrado.

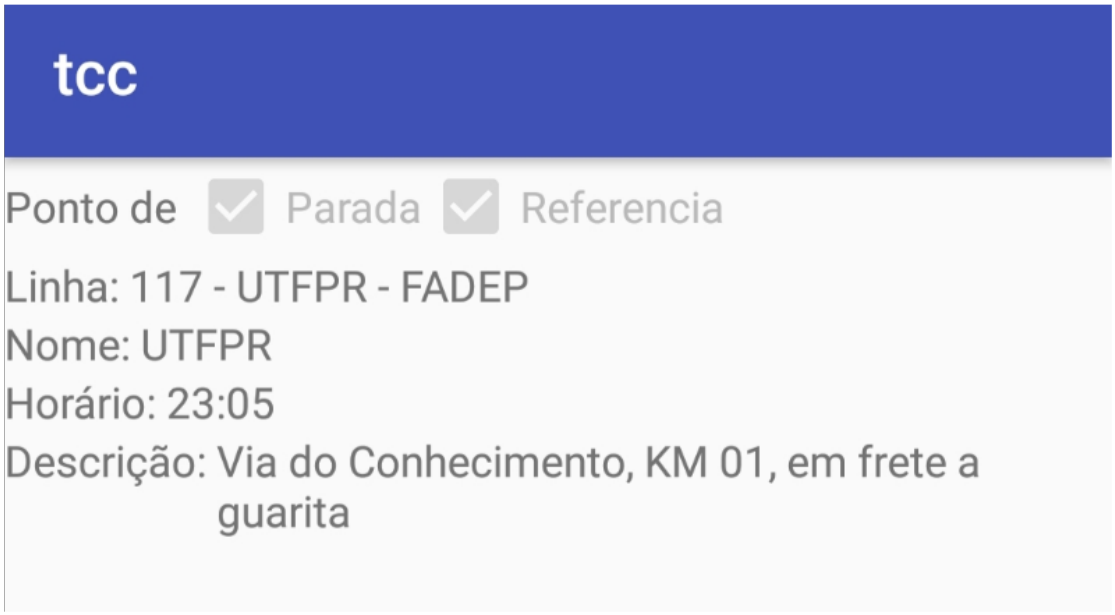


Figura 23 - Tela padrão do detalhamento dos pontos no aplicativo Android

Além disso, em Linhas é possível pesquisar as linhas por descrição como mostra a Figura 24.

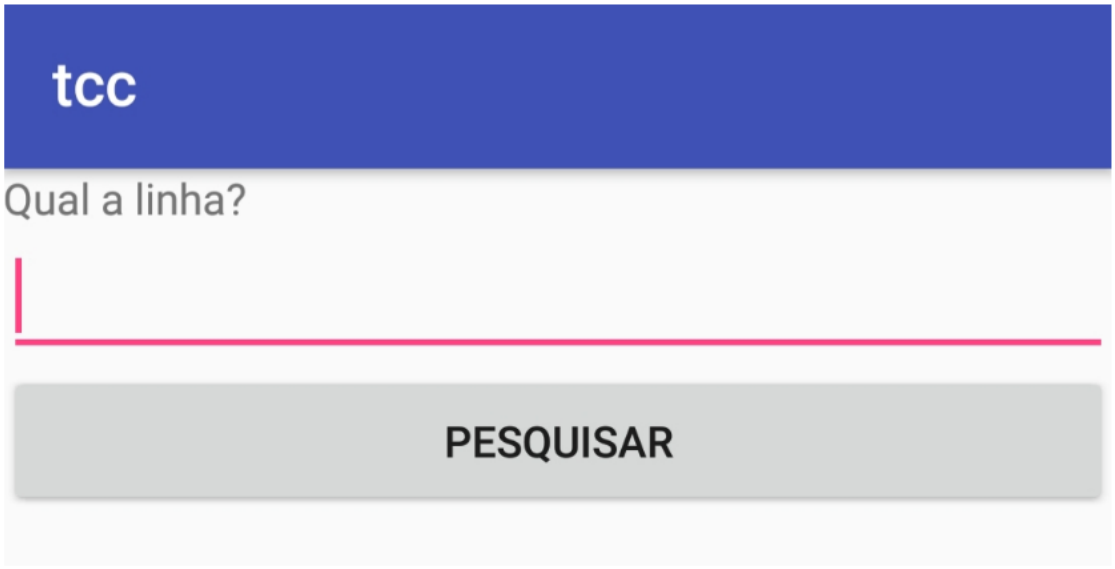


Figura 24 - Tela padrão para pesquisa de linhas no aplicativo Android

Para adicionar linhas e/ou pontos aos favoritos, basta executar um clique longo no registro em questão. A Figura 25 apresenta uma linha que foi adicionada aos favoritos do usuário.

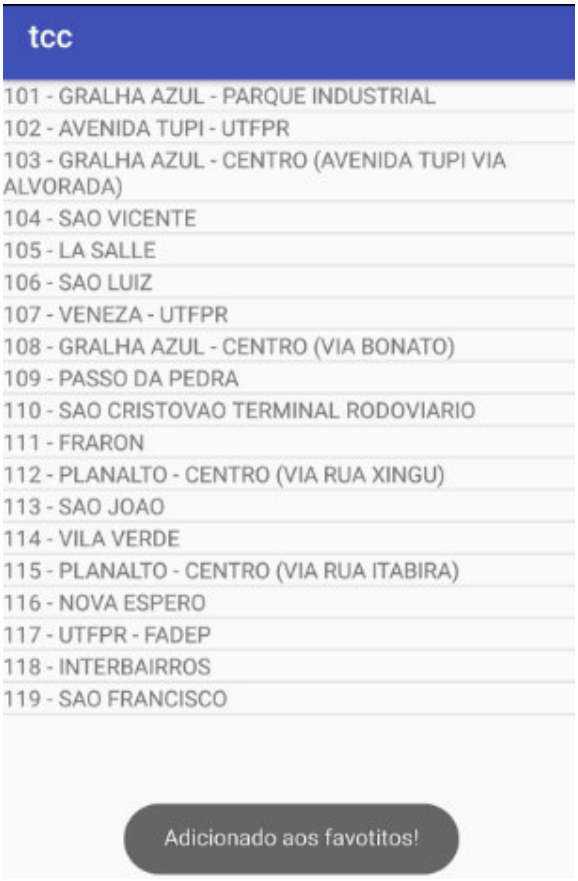


Figura 25 - Mensagem padrão para inclusão de linhas e/ou pontos favoritos no aplicativo Android

Os registros de linhas e/ou pontos favoritos serão apresentados na opção Meus Favoritos, conforme apresentado na Figura 26.

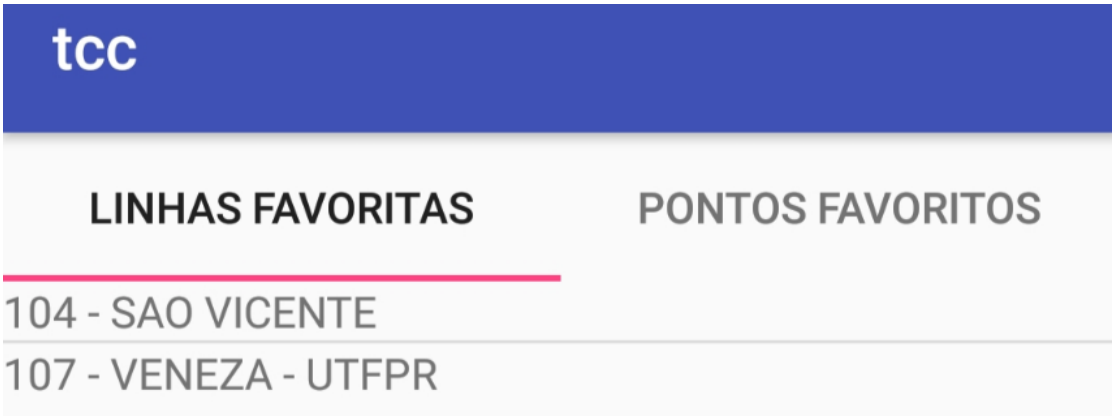


Figura 26 - Tela padrão para meus favoritos de linhas e/ou pontos no aplicativo Android

Ao executar um clique longo em um registro favorito, ele será removido de Meus Favoritos.

5.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção é documentada a implementação do sistema com exemplos de código de partes consideradas relevantes, como rotinas e funções.

Nas Listagens 1 e 2 são apresentadas partes das configurações do arquivo *persistence.xml*. Esse arquivo é utilizado pelo *Java Persistence API* (JPA) e seu conteúdo possui informações essenciais para que a aplicação possa executar a conexão com o banco de dados. Por exemplo, a *Uniform Resource Locator* (URL) de conexão com o banco de dados. Nesse caso foi utilizado o PostgreSQL.

```
<property                                name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/tcc"/>
```

Listagem 1- PgAdin - url de conexão com o banco de dados

Além disso, é necessário fornecer os dados de usuário e a senha, como mostra o código da Listagem 2.

```
<property name="javax.persistence.jdbc.user" value="postgres"/>
            <property         name="javax.persistence.jdbc.password"
value="123"/>
```

Listagem 2 - PgAdin - usuário e senha do banco de dados

Tais informações são necessárias para que o *framework Hibernate* possa realizar o mapeamento das classes para elas se tornarem as tabelas do banco de dados. Esse mapeamento é definido por meio das anotações. Conforme mostra a Listagem 3, na classe Empresa, por exemplo, a anotação *@Entity* foi utilizada para indicar que a classe será persistida no banco de dados. Na anotação *@Table* a propriedade *name* pode ser utilizada para declarar um nome para a tabela no banco de dados, caso necessário. Por padrão, a tabela é gerada com o mesmo nome da classe.

```
@Entity
@Table(name = "empresa")
@Data
public class Empresa implements Serializable {

    private static final long serialVersionUID = 1L;
    public static final String findAll = "Empresa.findAll";

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 80, nullable = false)
    private String nomeFantasia;
```

Listagem 3 - Hibernate - Lombok - Anotações

Ainda na Listagem 3, é possível observar a anotação @*Data*. Esta pertencente à biblioteca Java *Lombok*. @Data é uma anotação atalho conveniente porque ela agrupa as características de @ToString, @EqualsAndHashCode, @Getter/@Setter e @RequiredArgsConstructor em conjunto (PROJECT LOMBOK, 2018).

A Listagem 4 exibe um exemplo da utilização do VRaptor. Por intermédio da anotação @*Controller* foi definido que a classe *EmpresaController*, por exemplo, disponibilizará recursos de *Create, Read, Update e Delete* (*CRUD*) para o cliente. O @*Path* possibilita a customização da *Uniform Resource Locator* (URL) de acesso aos métodos e as anotações @*Get*, @*Post*, @*Delete* estão relacionadas com as requisições do protocolo HTTP dos tipos GET, POST e DELETE, respectivamente.

```
@Controller
@Path("/empresa")
public class EmpresaController {

    private Result result;
    private EmpresaRepository empresaRepository;

    public EmpresaController() {
    }

    @Inject
    public   EmpresaController(Result   result,   EmpresaRepository
empresaRepository) {
        this.result = result;
        this.empresaRepository = empresaRepository;
    }
```

Listagem 4 – Vraptor - Anotações

As páginas foram geradas com a utilização de *JavaServer Pages* (*JSP*), que são uma página Java para web, pois se trata da junção de *tags* HTML estáticas, *tags* JSP e código Java.

Na Listagem 5 é apresentado um dos recursos do JSP, a criação de *tags* personalizadas. A *tag* desse exemplo se trata de um campo *input*, que, posteriormente, foi utilizado em todos os formulários da aplicação, como é apresentado na Listagem 6. Isso foi possível porque a *tag* recebe parâmetros como *id*, *label*, *name* e *value*, sendo, assim, ela pode ser utilizada para diferentes objetos, agilizando o tempo de desenvolvimento.

```
<%@tag description="Campo para Input" pageEncoding="utf-8"%>

<%@attribute name="id" required="true"%>
<%@attribute name="label" required="true"%>
<%@attribute name="name" required="true"%>
<%@attribute name="value" required="true"%>
```

```
<div class="form-group">
    <label for="${id}">${label}</label>
    <input    class="form-control"    type="text"    name="${name}"
id="${id}" value="${value}" required/>
</div>
```

Listagem 5 - HTML - JAVA – JSP - Bootstrap

Conforme pode ser verificado no exemplo da Listagem 5, o Bootstrap foi utilizado para a estilização das páginas. No entanto, também foi criado um arquivo *Cascading Style Sheet* (CSS) para personalizar o leiaute do sistema.

Na Listagem 6 é apresentada a forma de utilização da *tag* JSP citada na Listagem 5:

a) Exemplo 1: por meio dessa linha de código a biblioteca de *tags* personalizadas se torna acessível por meio do prefixo “utfpr”.

b) Exemplo 2: utilizando o prefixo “utfpr” a *tag* é acessada e os parâmetros requisitados são repassados.

```
<!--Exemplo 1:-->
<%@taglib tagdir="/WEB-INF/tags/componentes/" prefix="utfpr" %>
<!--Exemplo 2:-->
<utfpr:fieldInput id="nomeFantasia" name="empresa.nomeFantasia" label="Nome Fantasia:*"
value="${empresa.nomeFantasia}"/>
```

Listagem 6 - Tag JSP

De acordo com que foi proposto no item 5.2 que explicita a modelagem do sistema, a aplicação servidor possibilita a manutenção de dados (com exceção das linhas e/ou pontos favoritos) pelo usuário. A sincronização do banco de dados da aplicação servidor e cliente é realizada por meio da *Application Programming Interface (*API) Retrofit.

A Listagem 7 exibe um exemplo de como o Retrofit foi utilizado para criar uma interface responsável por acessar recursos em uma API REST. As anotações nos métodos da interface e seus parâmetros indicam como uma solicitação será tratada. Todo método deve ter uma anotação HTTP que forneça o método de solicitação e a URL relativa. Há cinco anotações disponíveis, GET, POST, PUT, DELETE e HEAD. A URL relativo do recurso é especificada na anotação (RETROFIT, 2018). Nesse projeto foi necessário utilizar apenas um método para cada tipo de objeto, anotado com *@GET*, com a finalidade de receber todos os registros do servidor.

```
public interface EmpresaService {
    @GET("empresa/json")
    Call<List<Empresa>> getAll();
}
```

Listagem 7 - Retrofit – interface

Na Listagem 8 é apresentada a classe *ServiceGenerator* na qual é responsável por fazer as chamadas HTTP ao servidor, sendo que o endereço do servidor é especificado na constante API_BASE_URL.

```
public class ServiceGenerator {

    private static final String API_BASE_URL =
"http://172.30.2.19:8084/tcc/";

    private static Retrofit retrofit ;
    private static Retrofit.Builder builder = new
Retrofit.Builder().baseUrl(API_BASE_URL).
            addConverterFactory(GsonConverterFactory.create());

    public static <S> S createService(Class<S> serviceClass){
        retrofit = builder.build();
        return retrofit.create(serviceClass);
    }
}
```

Listagem 8 - Retrofit – implementação da interface

Na Listagem 9 está o método que realizada a sincronização dos dados. A requisição efetuada retorna uma lista de objetos no formato *JavaScript Object Notation* (JSON), que por sua vez é repassado para uma lista e posteriormente persistido no banco de dados do cliente.

```
public static void synchronizeEmpresas(final Context c){

    try{
        EmpresaService empresaService =
ServiceGenerator.createService(EmpresaService.class);
        Call<List<Empresa>> call = empresaService.getAll();
        call.enqueue(new Callback<List<Empresa>>() {
            @Override
            public void onResponse(Call<List<Empresa>> call,
Response<List<Empresa>> response) {
                if (response.isSuccessful()) {
                    List<Empresa> empresas = response.body();
                    PersistEntity.persistEmpresas(c, empresas);
                } else {
                    onFailure(call, null);
                }
            }
            @Override
            public void onFailure(Call<List<Empresa>> call, Throwable t) {
                Log.e("synchronizeEmpresas", "Erro ao obter empresas", t);
            }
        });

    }catch(Exception ex){
        ex.printStackTrace();
    }
}
```

Listagem 9 - Retrofit - sincronização

A persistência dos objetos é realizada no banco de dados SQLite, conforme mostra a Listagem 10, quando invocado o método *getConnection* da classe *DatabaseConnection*, é criada a *database*, caso a mesma não exista, e retornada a conexão.

```
public class DatabaseConnection {
    private static SQLiteDatabase bd;
    public static SQLiteDatabase getConnection (Context c){
        if (bd == null){
            bd = c.openOrCreateDatabase("tcc", Context.MODE_PRIVATE,
null);
            bd.execSQL(" CREATE TABLE IF NOT EXISTS empresa (_id LONG
PRIMARY KEY, nomeFantasia TEXT, razaoSocial TEXT)");
            bd.execSQL(" CREATE TABLE IF NOT EXISTS linha (_id LONG
PRIMARY KEY, idEmpresa LONG, descricao TEXT, FOREIGN KEY(idEmpresa)
REFERENCES empresa (_id))");
            bd.execSQL(" CREATE TABLE IF NOT EXISTS ponto (_id LONG
PRIMARY KEY, latitude DOUBLE, longitude DOUBLE, nome TEXT, descricao TEXT,
parada BOOLEAN, referencia BOOLEAN)");
            bd.execSQL(" CREATE TABLE IF NOT EXISTS linhaPonto (_id LONG
PRIMARY KEY, idLinha LONG, idPonto LONG, horario String, FOREIGN
KEY(idLinha) REFERENCES linha (_id), FOREIGN KEY(idPonto) REFERENCES ponto
(_id))");
            bd.execSQL(" CREATE TABLE IF NOT EXISTS pontoFavorito (_id
LONG PRIMARY KEY, idPonto LONG, FOREIGN KEY(idPonto) REFERENCES ponto
(_id))");
            bd.execSQL(" CREATE TABLE IF NOT EXISTS linhaFavorita (_id
LONG PRIMARY KEY, idLinha LONG, FOREIGN KEY(idLinha) REFERENCES linha
(_id))");
        }
        return bd;
    }
}
```

Listagem 10 - SQLite - database

Como mostra a Listagem 11 o método de persistência recebe a lista de objetos mencionada anteriormente na Listagem 9, efetua a conexão com o banco de dados por meio do método citado na Listagem 10 e inclui o registro na database.

```
public static void persistLinhas(Context c, List<Linha> linhas) {
    bd = DatabaseConnection.getConnection(c);
    try {
        for (Linha linha : linhas) {
            ContentValues linhaBd = new ContentValues();
            linhaBd.put("_id", linha.getId());
            linhaBd.put("idEmpresa", linha.getEmpresa().getId());
            linhaBd.put("descricao", linha.getDescricao());

            bd.insert("linha", null, linhaBd);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Listagem 11 - SQLite – persist

6 CONCLUSÃO

Pato Branco é uma cidade que possui muitos usuários de transporte coletivo urbano. Por ser uma cidade com muitos estudantes e com uma Universidade e outras instituições de ensino superior, muitos alunos utilizam transporte coletivo. As zonas industriais, algumas das quais recentemente estabelecidas, também são localizadas a certa distância do centro. Assim, trabalhadores e estudantes têm feito uso bastante intensivo de transporte coletivo.

Associado ao interesse de uso desse tipo de transporte está à necessidade em saber horários e itinerários, visto que algumas pessoas podem não estar familiarizadas com tais informações. Sendo assim, ocorreu à autora deste trabalho a ideia de proporcionar aos usuários desse tipo de transporte uma maneira para realizarem o acompanhamento das rotas de transporte coletivo urbano na cidade de Pato Branco, por meio de um aplicativo para dispositivos móveis. As ferramentas e as tecnologias utilizadas na solução foram, basicamente, *Web* e *Android*, tendo em vista que o usuário do sistema provavelmente estará se locomovendo.

A intenção é contribuir com a comunidade acadêmica e profissional e para a sociedade como um todo, disponibilizando dados que até então estavam acessíveis somente por meio de cartilhas de difícil compreensão.

Além disso, o desenvolvimento deste trabalho agregou em conhecimento utilizando ferramentas, tecnologias e métodos que foram empregados. Conforme previsto, foram enfrentadas algumas dificuldades durante as implementações. No entanto, essas dificuldades foram superadas por meio de pesquisa, pelo próprio processo de experimentação, dedicação e também, contou com o auxilio de professores da Universidade. Naturalmente há desvantagens e/ou limitações no trabalho, todavia, por meio dos resultados obtidos conclui-se que os objetivos definidos para o trabalho foram alcançados.

REFERÊNCIAS

ANDROID ARCHITRECTURE. Disponível em: <http://www.tutorialspoint.com/android/android_architecture.htm>. Acesso em: 25 jun. 2018.

ANDROID DEVELOPERS. Disponível em: <http://developer.android.com>. Acesso em: 25 jun. 2018.

APPBRAIN. **Number of available android applications**. 2012. Disponível em: <http://www.appbrain.com/stats/number-of-android-apps>. Acesso em: 25 set. 2016.

BRAY, Tim. **What Android is**. 2010. Disponível em: <http://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is>. Acesso em: 20 ago. 2016.

CANALTECH. Disponível em: <https://canaltech.com.br/apps/5-aplicativos-para-acompanhar-onibus-em-tempo-real/>. Acesso em: 17 mar. 2018.

CITTAMOBI. Disponível em: <https://www.cittamobi.com.br/home/>. Acesso em: 17 mar. 2018.

DISTIMO. **Google android market tops 675,000 applications**. 2012. Disponível em: <http://officialandroid.blogspot.com/2012/09/google-play-hits-25-billion-downloads.html>. Accesso em: 25 set. 2016.

GOOGLE PLAY. Moovit: metro, ônibus e trens. 2018. Disponível em: <https://play.google.com/store/apps/details?id=com.tranzmate&hl=pt_BR>. Acesso em: 17 mar. 2018.

GUANA, Victor; ROCHA, Fabio; HINDLE, Abram; STROUL, Eleni. **Do the stars align? Multidimensional analysis of Android's layered architecture**. MSR 2012, p. 124-127.

HIBERNATE. Disponível em: <http://hibernate.org/>. Acesso em: 04 dez. 2018.

LI, Yang; WANG, Xinning. **Design of adaptive media transmission based on Android platform**. In: 2014 IEEE International Conference on Consumer Electronics, China, 2014, p. 1-4.

LUNDEN, Ingrid. **Android breaks 1B mark for 2014, 81% of all 1.3B smartphones shipped**. 2015. Disponivel em: <http://techcrunch.com/2015/01/29/android-breaks-1b-mark-for-2014-81-of-the-1-3b-smartphones-shipped-in-total>. Acesso em: 26 set. 2016.

MOREIRA, Braitner. **Aplicativo para monitorar ônibus no DF é lançado com 35% das linhas; G1 testou**. 2018. Disponível em: <https://g1.globo.com/df/distrito-federal/noticia/aplicativo-para-monitorar-onibus-no-df-e-lancado-com-35-das-linhas-g1-testou.ghtml>. Acesso em: 17 mar. 2018.

MOTOROLA. **O que é a Open Handset Alliance?** Disponível em: <https://motorola-global-portal-pt.custhelp.com/app/answers/detail/a_id/36985>. Acesso em: 25 abr. 2018.

OLIVEIRA, Victor A. de J. **Introdução ao desenvolvimento para dispositivos móveis**. Instituto de Informática UFRGS. Disponível em: <http://www.inf.ufrgs.br/~vajoliveira/images/cei/docs/IntroToMobileII.pdf>. Acesso em: 09 abr. 2018.

PRESSMAN, Roger. **Engenharia de software**. São Paulo: Makron Books, 2002.

PRIMORAC, Sanja; RUSSO, Mladen. **Android application for sending SMS messages with speech recognition interface**. In: 5th International Convention MIPRO, 2012, p. 1763-1767.

PROJECT LOMBOK. Disponível em: <https://projectlombok.org/>. Acesso em: 05 dez. 2018.

RETROFIT. Disponível em: <https://square.github.io/retrofit/>. Acesso em: 06 dez. 2018.

VRAPTOR. Disponível em: <http://www.vraptor.org/pt/>. Acesso em: 04 dez. 2018.

XU, Wei; ZHANG, Fangfang; ZHU, Sencun. **Permlyzer: analyzing permission usage in android applications**. In: 24th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2013, p. 400-410.