ÁREA DEPARTAMENTAL DE ENGENHARIA ELECTRÓNICA E DE TELECOMUNICAÇÕES E DE COMPUTADORES - ADEETC

Personalized Public Transportation Information

Henrique Cabrita Marques da Silva

BSc in Electrical and Computer Engineering

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Professor João Carlos Amaro Ferreira, PhD

Jury

President: Professor Manuel Martins Barata, PhD, ADEETC - ISEL

Supervisor: Professor João Carlos Amaro Ferreira, PhD, ADEETC - ISEL

Referee: Professor Alberto Manuel Rodrigues da Silva, PhD, DEI - IST

Referee: Eng. António Pedro Simões Marcelo, Tecmic

September, 2015

Page intentionally left blank

Dedicated to my mum and my grandpa,
who always cheered me on along the way...

Page intentionally left blank

Acknowledgments

I would like to thank Instituto Superior de Engenharia de Lisboa and all of its staff for the opportunity of participating in a master's degree which surpassed all my expectations. Particularly to all the Professors who "ran the extra mile" by guiding students along better learning curves even when that was very difficult to do and without incentives other than excellence itself and for the joy of passing knowledge to their students. I would like to particularly thank my adviser for all the guidance provided in the task of writing this document, both in countless meetings and always answering questions and emails at any time during the day. I would like to thank Tecmic, the partner company in this project, for all support and for believing in this project and providing the means for its continuation into commercial production. I would also like to thank my colleagues at Tecmic who have provided tireless guidance through many difficulties along the project and participated in productive frequent debates on software development and engineering. I thank all my friends in Portugal and throughout the world who in one way or another helped get to where I am today, particularly those who during the last few years understood why I couldn't make it to many events, and still visited me either on campus or near my home for a quick coffee. Also for those thousands of kilometers away who still messaged and called for a quick catch-up. Finally I would like to thank my family, and in particular to thank my mother and grandfather for all the inspiration, teachings, life example and support you've given me right from the start. I still have a lot to learn from you.

Page intentionally left blank

Abstract

This project was developed in partnership with a technological solutions company, Tecmic. This work derives form a real world necessity expressed by Tecmic and provides it with a business opportunity. The objective was the development of a mobile application which allows the user to access real time data stored in Tecmic's infrastructure, while showing it in a filtered and personalized form more suitable to a personal application. The mobile application provides geographical context of the available data on public transportation which is a necessity for today's passengers and an incentive for the population to use the public transportation infrastructure. The project fits an intelligent city framework where integration and real time data access are a necessity. This work is one of the first steps taken by the company in this area, and due to the greenfield nature of the project all steps were taken without past work to build upon. Every development achieved during the project was started here: the market analysis of available mobile applications in the field of public transport, the software requirements specification for the prototype, the analysis of the currently available real time data which resulted in a set of functional requirements for a new API. The project continued with the architecture design following recognized architecture patterns and Google guidelines, followed by the actual development of this design with the objective of obtaining a usable prototype that can become a solid starting point for the production phase. Another side of this project included all of the planning required for the user experience and the use cases of the application, followed by the implementation and optimization of a user interface which would both suit the requirements of the company and improve the user experience. A large amount of data regarding the management of a public transportation fleet was provided by Tecmic. This large blob was mapped to a simpler model, more relevant for public transportation passengers. In the conclusion of this project, the resulting application even if still in prototype stage, allows the user to benefit from real time access to public transport infrastructure data in a simple and intuitive form.

Keywords: mobile application, personalized, geographical system, real time, intelligent public transportation

Page intentionally left blank

Resumo

O presente trabalho de projeto foi desenvolvido em parceria com uma empresa de soluções informáticas na área dos transportes, a Tecmic. Resulta de um necessidade real da empresa e de uma oportunidade de negócio. O objetivo foi o desenvolvimento de uma aplicação para dispositivos móveis que permita a personalização da informação e acesso em tempo real à informação de transportes públicos disponível nos servidores da Tecmic. Esta aplicação permite a contextualização geográfica da informação de transportes públicos disponível e é uma necessidade para o incentivo do uso dos transportes públicos pela população. De igual forma enquadra-se na problemática das cidades inteligentes onde a integração e o acesso em tempo real da informação desejada é uma necessidade. O trabalho foi um dos primeiros passos da empresa nesta área, tendo todo sido desenvolvido de raiz, desde o estudo de mercado relativo a aplicações móveis actualmente disponíveis nesta área, ao levantamento de requisitos para o protótipo, análise da informação disponível em tempo real e especificação de requisitos de uma nova API para colmatar algumas das necessidades que não eram satisfeitas pela API existente. O trabalho desenvolveu-se com o desenho da arquitectura segundo as boas regras de desenvolvimento de software seguindo as Google guidelines de desenvolvimento e implementação dessa mesma arquitectura, sendo o plano a obtenção de um protótipo experimental que servirá de ponto de partida para a fase de produção. Todo o planeamento da experiência de utilização, dos casos de utilização e desenho, implementação e optimização do interface fizeram parte também deste projecto. A enorme quantidade de informação que é disponibilizada pela Tecmic relativa à gestão de uma frota de transportes foi repensada e mapeada para entidades que representam o mundo dos transportes públicos do ponto de vista do passageiro e que lhe são familiares. No fechar deste projecto o resultado ainda que sendo um protótipo já permite ao utilizador usufruir de acesso em tempo real a toda uma infraestrutura de transportes públicos de uma forma simples e intuitiva.

Keywords: aplicação móvel, personalização, sistema informação geografica, tempo real, transportes públicos inteligentes

Contents

Page intentionally left blank

List of Figures

Chapter 1

Introduction

1.1 Motivation

Nowadays it is the high degree of complexity of today's public transportation network that enables the daily commuting of tens of millions of people living in the metropolises of the world. The growing complexity of public transportation networks together with the worldwide shrinking of the cost of mobile data has opened an opportunity for mobile technology to contribute with a very significant improvement in everyday life for commuters all around the world. In a world where millions of people are using Android, whose time is ever more valued, the immediate availability of real time information on the status of the network can be considered a top priority issue in cities like London, Tokyo and São Paulo where the disruption of a single transport line would lead to chaos. It is beyond any kind of practicality to know each one of the 19'000 stops that London Buses makes available to people in London, or to plan a trip that takes into consideration the commuting times between so many different buses, let alone know if there is a problem with the network branch you are planning to use since there are 673 different bus routes in London alone. This topic has stirred a high level of interest from local governments and the public as well as investment from companies worldwide, such is the impact that this technology can have. The solution proposed in this thesis makes use of the widespread availability of android smartphones and focuses on the Brazilian market. The final prototype will give the common user real time information on the status of the transport that he/she is looking for, taking into account the personal preferences of this person such as their preferred commute, home location, office loca-

tion and other favourite spots. Making use of high speed mobile data this prototype will use the person's current context using the XTran Passenger API, developed by Tecmic, SA. Instead of putting all the information in front of the user, it is this application's main objective to filter the overwhelming complexity of the modern transport network information drawn both from the XTran system and all of the context information sources in modern Android mobile phones, showing only what is most relevant to the particular user and to make sure that this information is in the user's reach through the app in real time, whenever it is needed.

1.2 Business Framework

1.2.1 Existing product (B2B)

Tecmic is a company on the forefront of fleet management solutions with established business partnerships in the industries of emergency services and public security, public transport management and service planning between others. The products it has developed and the hardware and software it has designed serve more than a dozen different industries and are recognized as some of the best tools available for real-time data collection on vehicle and operational meta data. The solution this work will use as a source for most of the real time data is Tecmic's operating support system for passenger transport - the XTraN Passenger. The XTraN Passenger system currently presents the following features:

- Integration hub for information related to Buses and journeys of the company, allowing for real-time data and feedback on operational decisions.
- Real-time access to meta data on individual vehicles, including route and direction, schedule for journeys, speed, present location, last stop and other data.
- It features integration of operational data with GIS for graphical overview of status on map.
- Passenger counting for calibration of fleet optimization and resource allocation.
- Infrastructure planning (new stops and journeys).

- Meta data necessary for fuel optimization and energy efficiency.
- Security oriented features such as video surveillance, panic alarm with connection to police forces.
- Integration with ticketing systems and seasonal planning.

1.2.2 Public Information (B2C)

As of now most of the company's data output and products are consumed by fleet and business owners. Most of the existing solutions were designed and engineered towards other businesses following a B2B model. This work will be built upon the existing XTraN Passenger infrastructure but mostly on the side of public information. Tecmic already offers channels such as text message information services, bus stop panels and a public website. But those channels do not take advantage of the proliferation of smartphones, in particular the growing Android market has surpassed all expectations and is the most commonly used operating system for smartphones. Thus the opportunity has arised for a public mobile application to be created such that the huge amount of meta-data available from the existing infrastructure may be filtered, organized and presented to the user in a way that personal context and daily routines define what will be shown to the user. As a result of this project, Tecmic will open a new branch of business inside the company's public transport systems division creating a new product for the general public using their existing technology in an entirely different way (B2C) without the setbacks of any major changes to the existing infrastructure.

1.3 Objective

Mobile devices are becoming an inseparable part of our lives and personalized location-based mobile services are gaining more and more popularity. It is the objective of this work to design and implement a prototype for a public transports mobile application in partnership with the company Tecmic. The emphasis of this work will be on the personalization and contextualization of information, getting the user through most of what could be considered 'noise' so that the only information shown is exclusively what the user needs or wants.

This will involve making an analysis of the present offer in the field of public transport related mobile applications, followed by an infrastructure analysis of the information available from the on-board device, designing and producing the webservice layer for the needed information, filtering and organizing the large amounts of information available into a small pieces that are relevant to the user in a layout designed to be as intuitive as it is relevant. Finally the result of this project will be a prototype that will be as close as possible to a usable production model in the short period of time available. It is the objective of this prototype to be taken to production stage and accordingly, the partner company has decided not to publish the source code.

1.3.1 Expected Challenges

As the objective of this work became clear over the course of several meetings with Tecmic, the main challenges to overcome appear as follows:

1. Start the XTran Passenger application as a greenfield project, being responsible for most decisions made along the way.

2. Produce reusable modules following good practices of software development.

3. Design a layered architecture usable and expandable even after the dissertation is concluded.

4. Integrate the wide functionality of the XTran Passenger on-board device in a usable Android application.

5. Make a prototype that not only fulfills the software requirements specification but is also easy enough to use so that the partner company can deploy it for production.

6. Deploy prototype on location for tests.

1.4 Dissertation's Structure

In order to thoroughly describe the work required to achieve the completion of this project, the structure of the dissertation is organized as follows:

- **Chapter 1 - Introduction** : introduction, motivation and basic concepts related to the area of study. Presentation of the objectives and expected challenges.
- **Chapter 2 - Previous Work**: state of the art in the field of personally contextualized public transport mobile applications.
- **Chapter 3 - Software Requirement Specification (SRS) and Architecture Design**: discussion of the SRS and architecture design.
- **Chapter 4 - XTraN App**: presentation of the technology in use and technical details of the implementation.
- **Chapter 5 - Evaluation**: evaluation of the work done during the course of the project, objectives and progress compared to the SRS. Definition for a testing methodology and presentation of the tests performed.
- **Chapter 6 - Conclusions**: presentation of the conclusions and discussion of possible future work.

Page intentionally left blank

Chapter 2

Previous Work

2.1 Improvements to Intelligent Transportation Systems based on GIS Web Service Technology

2.1.1 Symbiotic use of Geographical Information Systems and Intelligent Transportation Systems

The widespread use of Geographical Information Systems (GIS) for the purpose of Intelligent Transportation Systems (ITS) during the last decade, and an increased demand for business intelligence systems which optimize a city's public transportation network created a necessity for advances in the mechanisms through which geographical information systems are used. Continuous research and the worldwide use of mobile devices for the purpose day to day commute together with the mature state of current Web Service technology paved the path to the creation of GIS-T (geographical information systems for transportation). The symbiotic use of both GIS and ITS is of extreme importance for city councils or public transport operators, such is the need for an effective translation of geographical coordinates into usable data and the business intelligence derived from that data. The last decade also brought huge advances on the Internet and the proliferation of service-oriented architectures promoted the adaptation of GIS systems so that they could easily become accessible through other media [10]. In [13] the author explores the use of GIS Web Services and service-oriented architectures with the objective of achieving a symbiotic architecture and building a service

enabled Web GIS intelligent transportation application system. [13, 25, 18]

2.1.2 Example implementations and first link to mobile applications

After the necessity was established as explained in the previous subsection, several research groups and companies developed their own projects such as: The Environmental Systems Research Institute (ESRI) who presented the ArcWeb Services, and Microsoft with their own approach to this challenge presented MapPoint.NET web service for the purpose of GIS location services and mapping. Besides providing high quality mapping MapPoint offers features such as routing and driving directions[13].[1] An example WebGIS-T platform can be see in Figure 2.1.

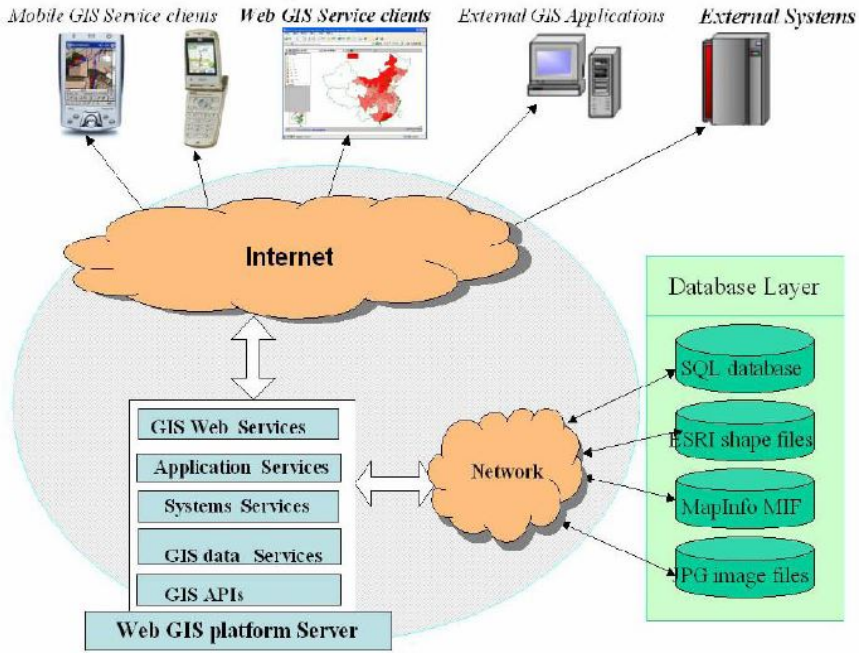


Figure 2.1: Example WebGis-T platform server environment [13]

[1] MapPoint is being discontinued by Microsoft but will be supported at least until July 2015[2, 3, 26]

2.2 XTraN Passenger System

2.2.1 Overview

The purpose of the XTraN Passenger system goes much farther beyond the scope of public information which is the focus of this work. The system was engineered to produce detailed real-time statistics on the fleet's activity, giving the operator the chance to improve on his strategy based on the feedback gained from the system. The basic functionality of the system includes real-time data on individual vehicles equipped with the XTraN Passenger on-board device allowing for precise estimations of arrival times (ETAs) and varied channels to share information with the public (such as bus stop panels and SMS information services)[23], integration with ticketing systems[21], continuous voice and text communication with the driver, fuel consumption statistics and analysis[22] as well as an on-board security feed[19] and an engine kill-switch. An overall schematics of a bus with the XTraN Passenger system installed can be seen in Figure 2.2 [24].

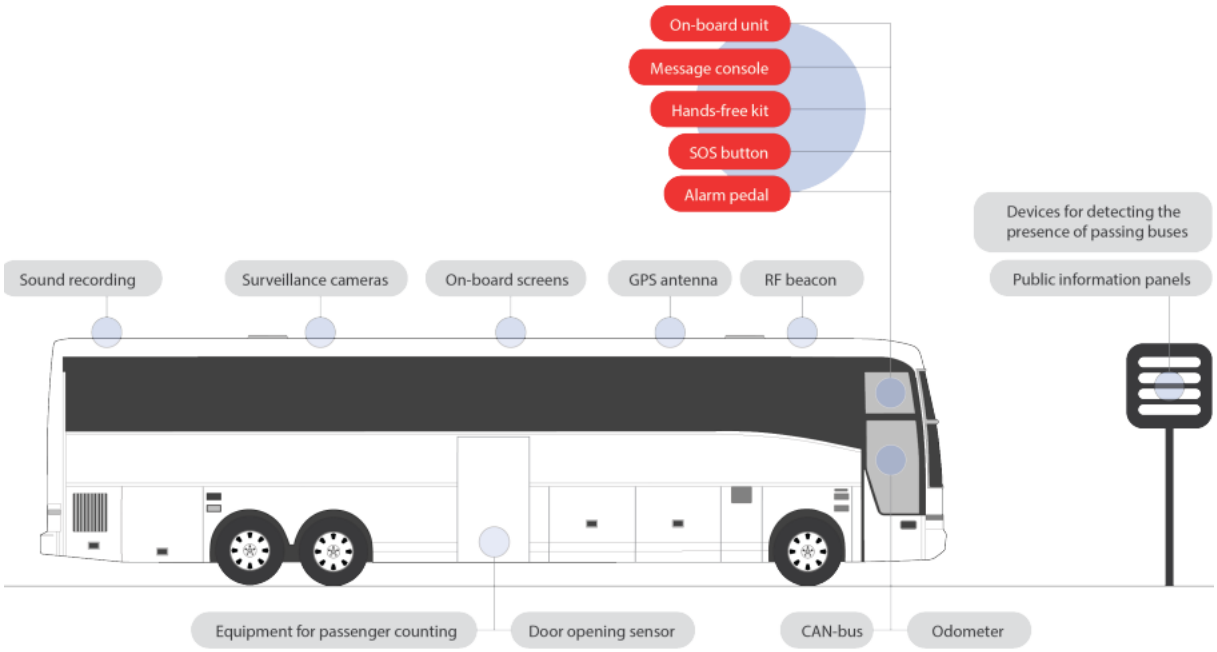


Figure 2.2: Bus Schematics[24]

2.3 Technical Framework

2.3.1 Basic Concepts

The following model has been followed in this project in partnership with Tecmic, PLC in the image of real world public transport buses using their popular on-board device for the Xtran Passenger system. The system is currently deployed by companies such as Carris (Lisboa, Portugal), Inga and Coesa (Rio de Janeiro, Brazil) among others worldwide [24, 20].

Every bus stop in a company's transport network is already mapped, its geographic coordinates were previously recorded either manually by the bus driver or inserted in the database through Tecmic's desktop client application. All routes are built on the graph-like nature of the network. Each *route* represents the real world equivalent of a bus route and it consists of at least one course *variant*. An example would be route 777 variant 0 bus which goes through stops $\{$A, B, C, D$\}$ on any given weekday during business hours and route 777 variant 1 bus skips stop C later in the evening following the path $\{$A, B, D$\}$. Each variant will always include two ways (e.g. Eastbound, Westbound) defined by sets of *partial paths* which are directed graph segments linking network nodes (bus stops). These segments have a fixed distance-cost and a time-cost that is adjusted according to the applicable time-variants. The start and finish points (stops) are well known for all variants and each partial path may be shared by multiple buses. Circular routes will be modeled similarly to non-closed paths. Additionally the model considers the *time variants* of each route in order to provide accurate estimates to commuters. Time variants include *peak* and *off peak* daily variants as well as weekday, weekend, holiday and seasonal changes. The result of these many calibrations is a system that clearly separates timetable expected times from estimates based on real world events, time and location of the bus and the current user context (location, day, time, expected bus and desired destination).

2.3.2 XTraN Passenger Model

The data model on the server side database is organized as described in Figure 2.3. The bus routes ("Carreira") mentioned in Subsection 2.3.1 include one or more

route variants (“CarreiraVariante”) which in turn contain two ways (“CarreiraVariante-Sentido”). This model effectively describes the workings of bus companies such as Carris and Inga (the company used for testing during this project) where the course of a certain route may vary with the time of day, day of week and with the season. The data model supporting the two directions works similarly: “CarreiraVarianteSentido” are composed by many courses (“VarPerc”) which are constituted by branches (“Connection”) and bus stops (“PTParagem”).

2.3.3 XTraN Modules

The XTraN Passenger system is the foundation upon which this project is built on. XTraN Passenger is a highly modular distributed system that will be briefly described in this chapter. Some of the components of the system can be seen in a high level representation schematics in Figure 2.4. The core modules of the system are:

1. The Communications’ Server.
2. The Passenger Server.
3. The on-board driver’s console (a.k.a. XTraN Passenger Box).

There is a well defined (proprietary) protocol transmitted over UDP messages. The messages are sent between the communications’ server and each box on-board of a different bus and this protocol is used with the message processing entity - the Passenger Server. The protocol encompasses information such as the bus current position and current speed but also more detailed operational data such as the detection of a bus stop, open doors’ event, outside of expected route event. The messages are then passed to the Passenger server which is responsible for processing each message. Among the most important messages traded are the authentication of a driver and the beginning or ending of his service.

2.3.4 XTraN Architecture

The XTraN Passenger system was developed with a highly modular architecture and interchangeable components taking into account the possibility of future improvements[24]. The services that currently constitute the XTraN Passenger infrastructure include:

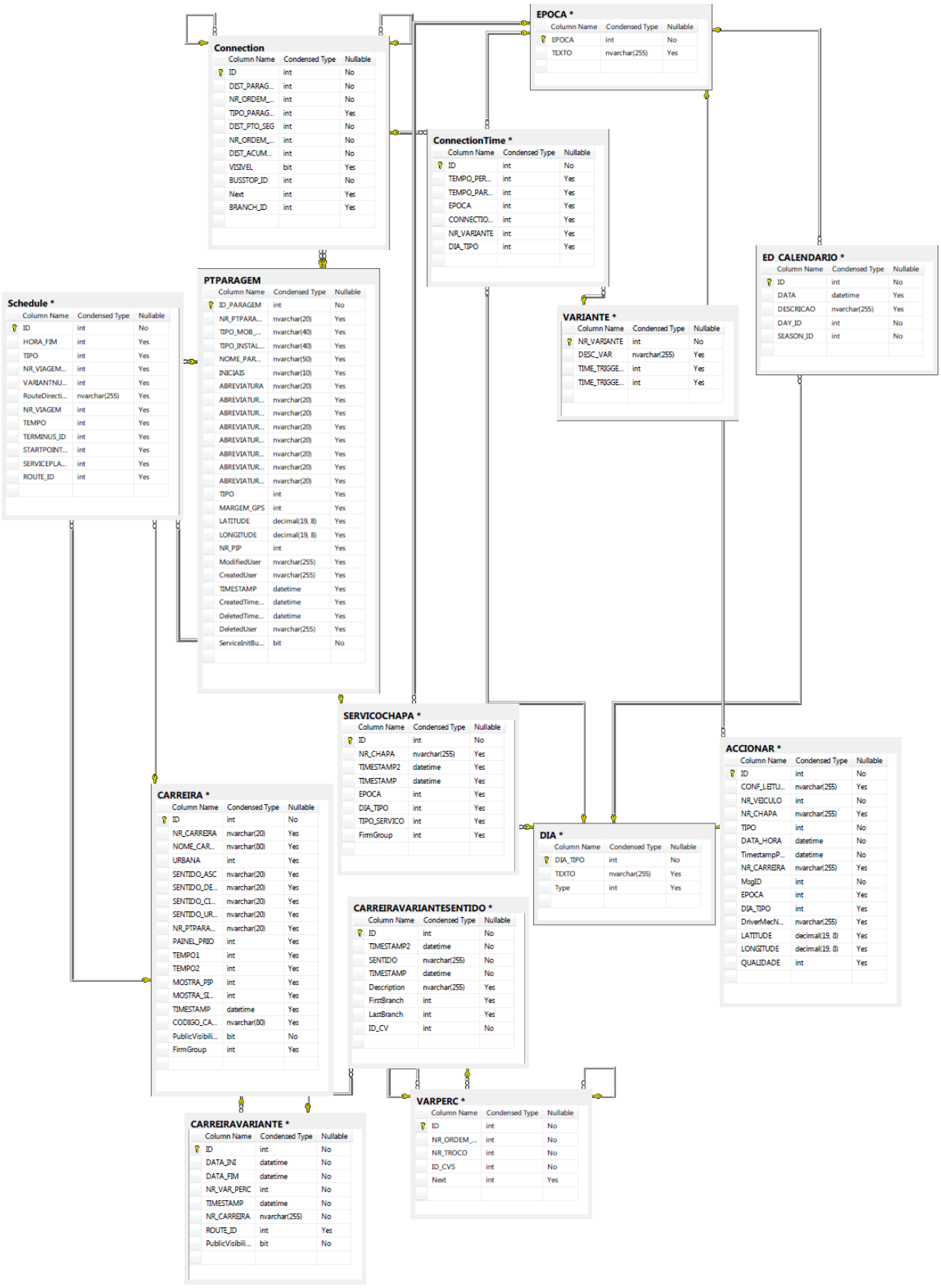


Figure 2.3: Database model

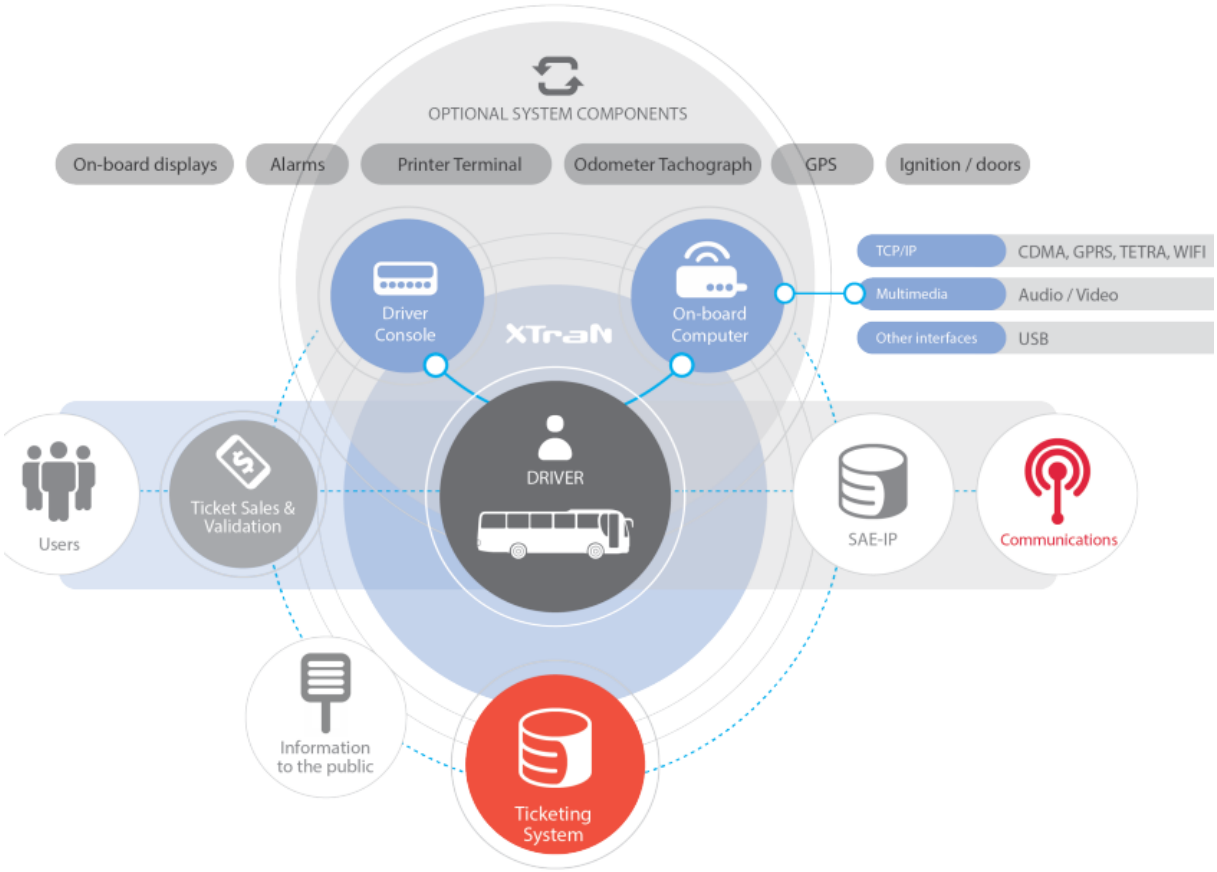


Figure 2.4: High level representation of XTraN Passenger [24]

Passenger Server is the module that processes events received in messages from the Communications' Server, it also ensures writing to the database layer for persistence.

Communications' Server responsible for UDP communication between XTraN Passenger Server and all the on-board devices.

TimeEstimator service is the service responsible for calculating the temporal cost of each bus stop connection. It distinguishes between different week days and seasons for estimation output, the more buses share that connection in their itinerary the most accurate will the estimation become.

SIMIP legacy service for the old SMSGateway. The service is implemented in Java and uses JBoss and PostgreSQL servers, it provides public information through the SMSGateway.

SMSGateway service provides text message information upon user request. It does

not make estimations on its own but uses TimeEstimator behind the scenes.

WebInfo is the current front end for the public information services made available by Tecmic and partner transportation companies.

PassengerWeb server is the server for the front end made available by Tecmic to partner companies for fleet management purposes (non-public).

RestRemoteServices is the module of the system that is closer to this project. It serves the RESTful API that is used by the mobile Android application built during the course of this work. It was designed to be as expandable as needed and during the development phase of the project it was expanded with the addition of the crucial methods and requests needed for a public transportation app of this kind (in partnership with Tecmic). It is hosted in a IIS server from where it serves the public information panels and the mobile application.

Passenger DB Server Persistence layer provided by an SQL Server instance. The Passenger Server provides access to all its information from memory but every time there is an update on the state of the system it will update the database through N Hibernate (ORM).

A representation of the previously described XTraN modules can be seen in Figure 2.5. Each service is designed to be physically independent from the others and can ultimately be deployed each in its individual server. By default they are deployed using three machines grouped as follows: {Communications' service, Passenger service, TimeEstimator service} , {RestRemoteServices, WebInfo, PassengerWeb service}, {Passenger Database Server}. A typical situation of this system requiring the use of the main modules would be the following: The box signals an event, such as *bus stop detection*, triggering the sending of a message to the Communications' service which passes the message into a PassengerMessageQueue to be processed by the Passenger Server. To be noted that this message queue can be physically separated from the Passenger Service for system robustness, assuring that all messages are attended to. A response be the Passenger service would follow an alternate symmetrical return path, ultimately resulting in the Communication' service sending a response message to the box.

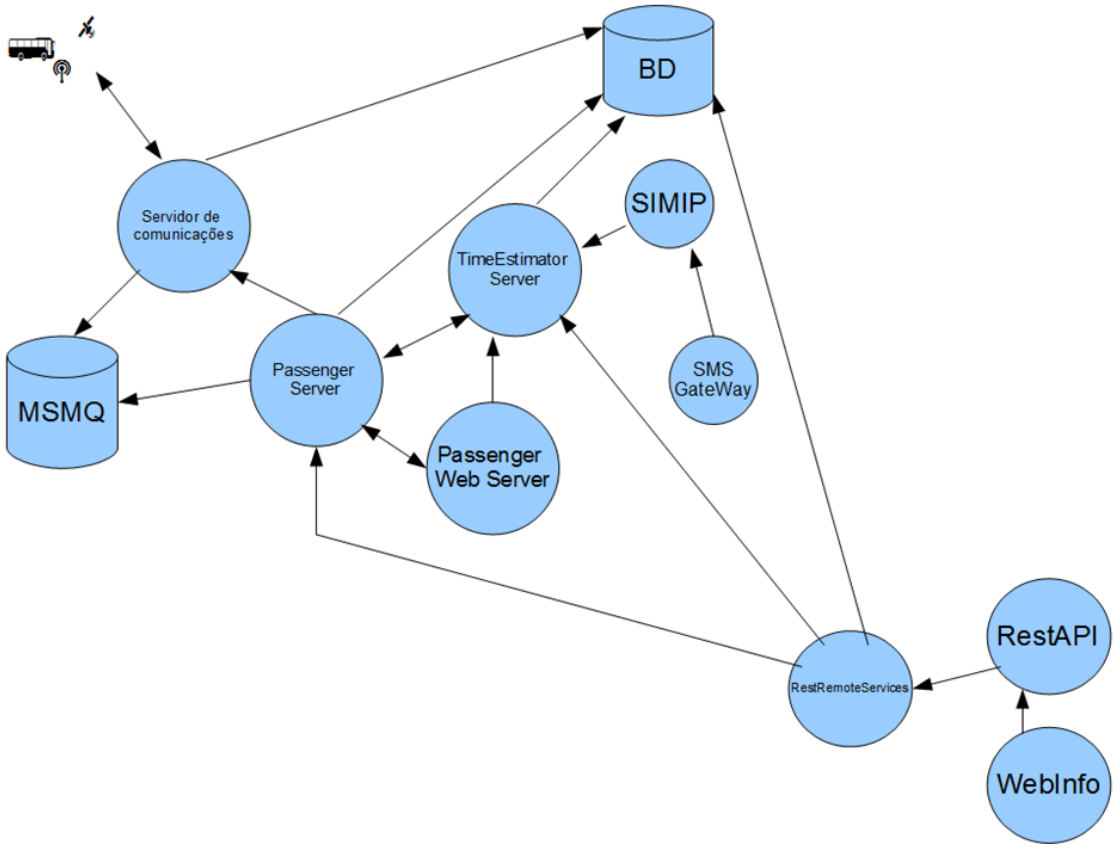


Figure 2.5: XTraN Passenger modules [24]

2.3.5 Estimating buses' arrival times

Accurate estimations of arrival times are of key importance for companies and passengers relying on the XTraN System and the TimeEstimator service is one of the most important pieces of this system, as without it there would be no real time statistics available. The TimeEstimator service relies on data provided by thousands of previous bus trips and that information is shared at the bus stop connection level, meaning that if there are 10 different bus routes which paths include the connection between stops A — B historical data from all buses traveling through that connection will be used in estimations for their respective routes. TimeEstimator also makes the distinction between different times of day, weekend, holiday and seasons so that each previous recorded trip time will be contextualized and used in a similar future trip.

2.4 Mobile Apps

The preparation for the development phase of this dissertation required a broad market search for applications in the public transportation business. The number of analyzed comparable applications shrunk when during the software requisites specification was taken into account (detailed in chapter 3), and the final pool of analyzed apps was reduced to:

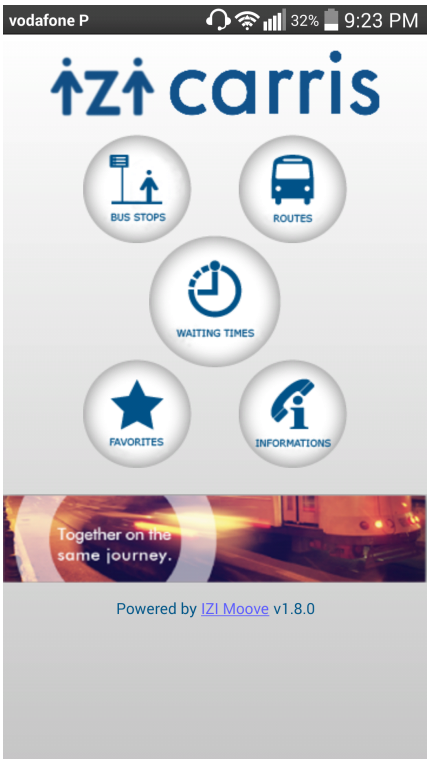
1. IZI Carris (Lisbon specific: single-operator, Carris) [11].
2. Lisboa Move-Me (Lisbon specific: Multi-Operator) [16].
3. MoovIt (International, Multi-city: Multi-Operator) [15].
4. CityMapper (International, Multi-city: Multi-Operator) [12].

All the tests performed in order to write this section of the dissertation were conducted on three different Android devices: a LG E980 (from AT&T) using Android 4.4.2, an HTC One X (from AT&T) using Android 4.2.2 and a Google Nexus 7 (v.2012, no carrier) with Android 4.4.4.

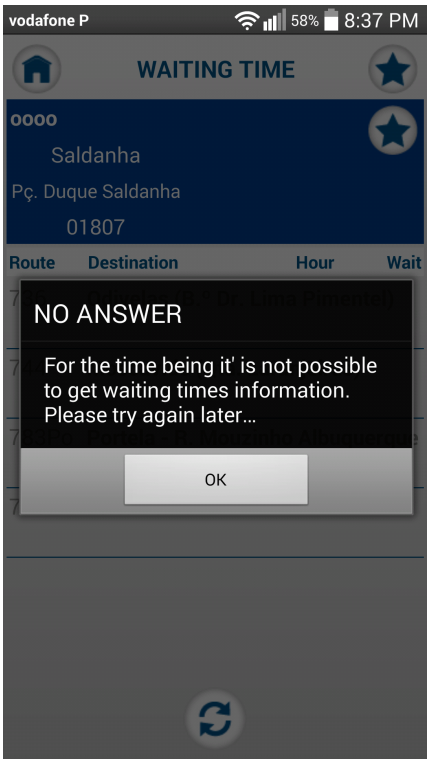
2.4.1 IZI Carris

IZI Carris is one of the “official” applications advertised in the operator website [1]. The developers advertise that the application provides static information plus real time access to bus waiting times in any bus stop operated by Carris. The company advertises personalization through the use of favourite stops and real time estimations based on real-time GPS location of the buses. The application itself does not ask for permission to access user location.

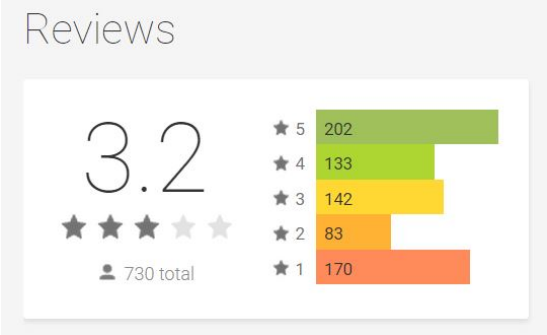
If starting at the main screen the user can consult a set of static data such as a “list of bus routes” and the corresponding list of bus stops that allows the user to use the application as a map of the public transport network. As soon as a user clicks on one of the static stops it is brought to a real time “Bus Stop Status” activity that will keep the user waiting until the default time-out. If the user wants to check a list of bus stops he can do so but cannot check the waiting times for the buses.



(a) IZI Carris home screen



(b) IZI Carris timeout when real time estimates are invoked



(c) IZI Carris Ratings [11]

Figure 2.6: IZI Carris application

The first impression upon testing was that it is intuitive although the design does not comply with current Google usability guidelines. If a user tries to use the Waiting times this option forces the user straight to an activity that is stuck on a “Getting waiting times . . . ” spinner which completely discards the real time features of the application.

Putting this application side by side with the requirements for this project IZICarris does not take advantage of the current user context, there is no map available, there is no check for current user position (a giveaway for this fact is that the app does not require Location check permission) against closest bus stops and even the favourites section that could be used as a personalization tool is hindered by the fact that every time the application tries to check for real time information it makes the user wait for timeout.

Current user review for this application in the store is divided mostly between 5 star and 1 star ratings reflecting the current status of an application that addresses some key needs of the user but still a good idea that does is not working in its current form.

2.4.2 Lisboa Move-Me

The Move-Me application was one of the public transportation apps promoted by the Lisbon city council when it launched. Lisboa Move-Me has several qualities - one of which is knowing the core functions that its app should have. Still while the information is all present it can become a challenge to use. While the home screen (Figure 2.7) features in a simplistic way what most users want, almost all use cases are multiple levels deep breaking Android guidelines and application usability and simplicity.

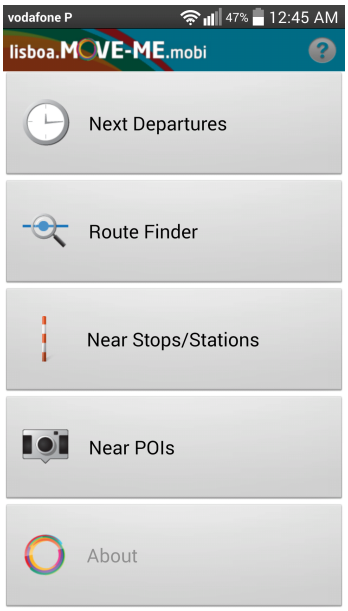
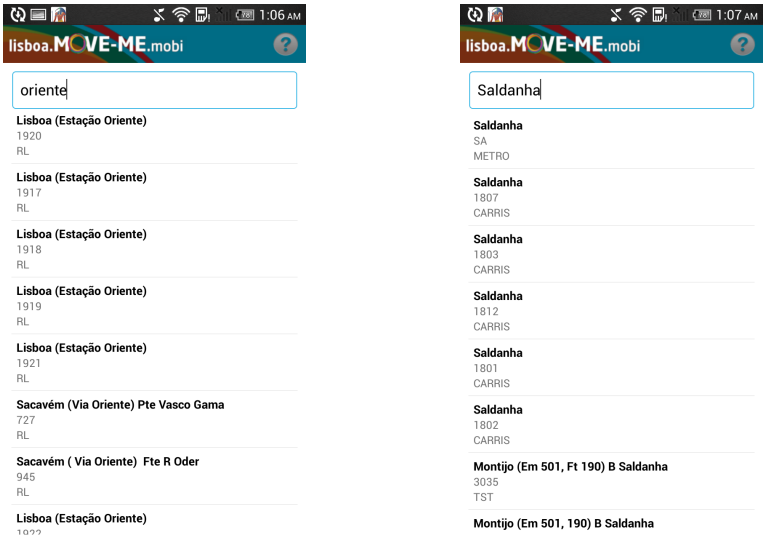


Figure 2.7: Lisboa Move-Me home screen

Such a case would be one in which a new user clicks on the first option in the home screen “Next Departures” which immediately prompts a pop-up menu consisting of four options: Favourites, History, Find and Select. The first two options would be empty for this use case, leaving as choices: *Next Departures — Find* or *Next Departures — Select*. This is not an obvious choice but trying the first option the application crashes in both the LGE980 and the Google Nexus 7 while it works in the HTC One X listing several bus stops filtered by name presenting results such as the following two searches

in Figure 2.8.



(a) A working search example

(b) Another working search example

Figure 2.8: Lisboa Move-Me unclear “Search” results on the HTC One

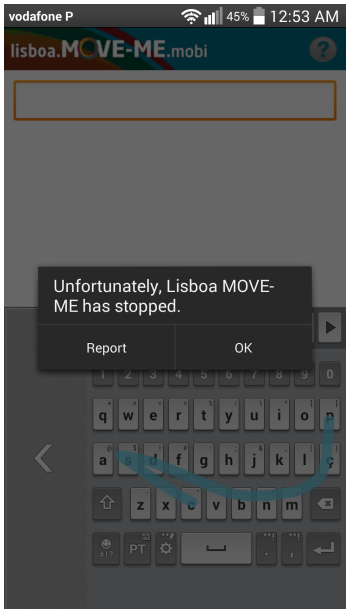
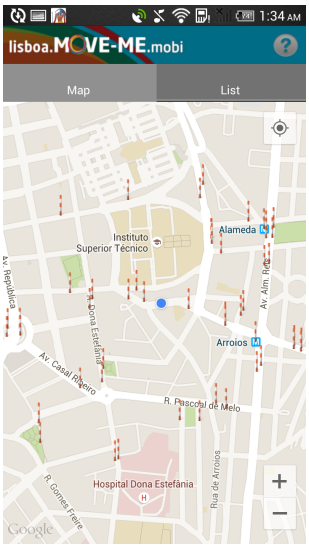


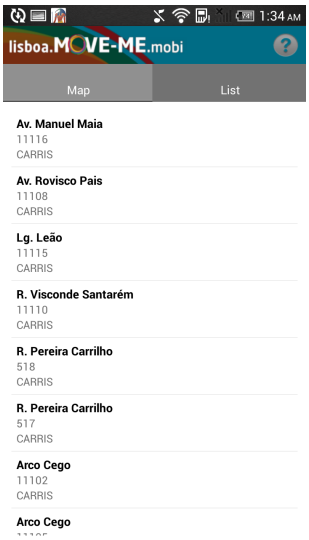
Figure 2.9: “Search” on both the LG and Google Nexus devices frequently crash

Even in the case that the search does not crash the app (avoiding the situation in Figure 2.9, after getting search results such as the ones obtained in Figure 2.8 if the user clicks the wrong station he will get the wrong buses estimations' and by pressing "back" the activity will be terminated bringing the user to his starting point (Figure 2.7), still having no favourites or any quick way to get to the correct estimations, forcing the user to go through all of the options and trying the results one by one until getting the right one by chance. In the screen represented in Figure 2.8(a) six different stops with the same name can be counted, the same number as in Figure 2.8(b). Although one of these can be easily distinguished from the other 5, it is still a large number to multiply by the number of steps necessary for the user to perform a single search.

One option which worked flawlessly was the "Nearby Stops//Stations". The results can be seen in the two forms available in the next figure.



(a) Nearby Stops//Stations - Map view



(b) Nearby Stops//Stations - Bus stop list view

Figure 2.10: Lisboa Move-Me "Nearby Stops/Stations"

The user still does not have any way other than going one by one looking for a “needle in a haystack” bus stop which includes data on the bus desired by the user. This being said and despite some minor malfunctions the application provides data estimates on bus stops as advertised - providing the user can navigate the application and add his most used bus stops to his favourites. This fact is recognized by the users in the Google Play Store who provided the ratings seen in Figure 2.11.

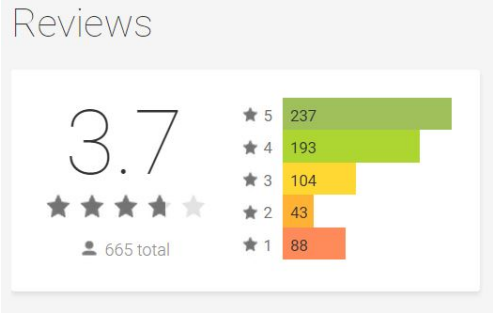


Figure 2.11: Lisboa Move-Me Ratings [16]

One noticeable fact is that the use of “Favourites” is of extreme relevance for anyone using the Move-Me app. The users give to their favourites’ importance in the usage of this application in the Figure 2.12.

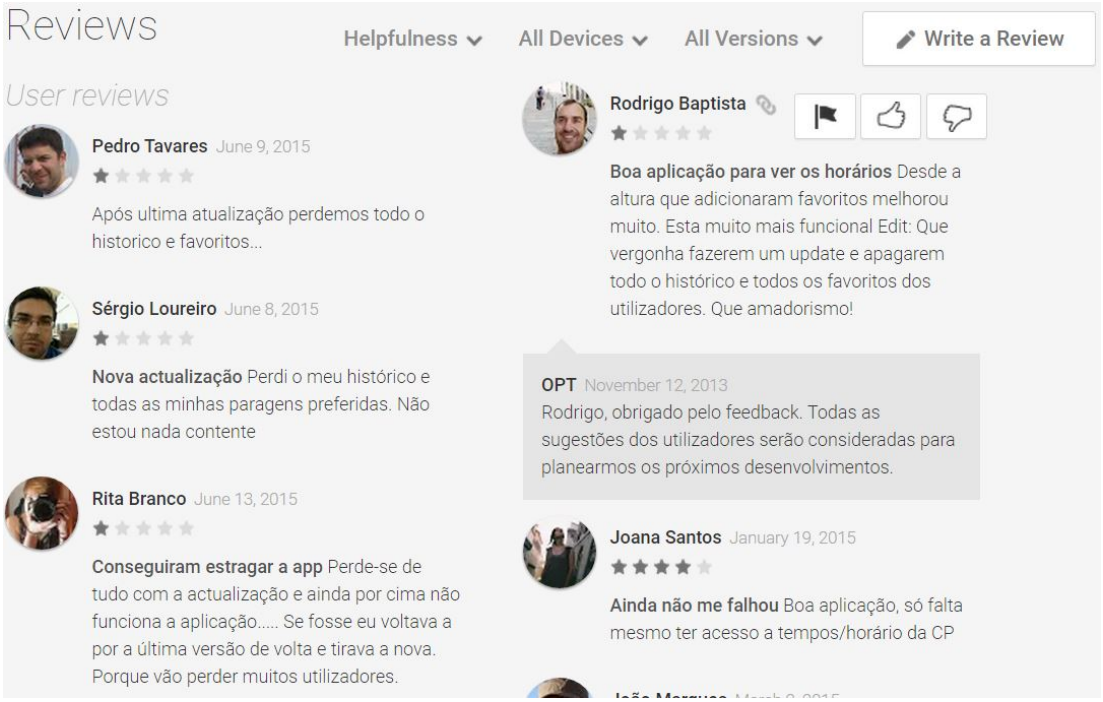
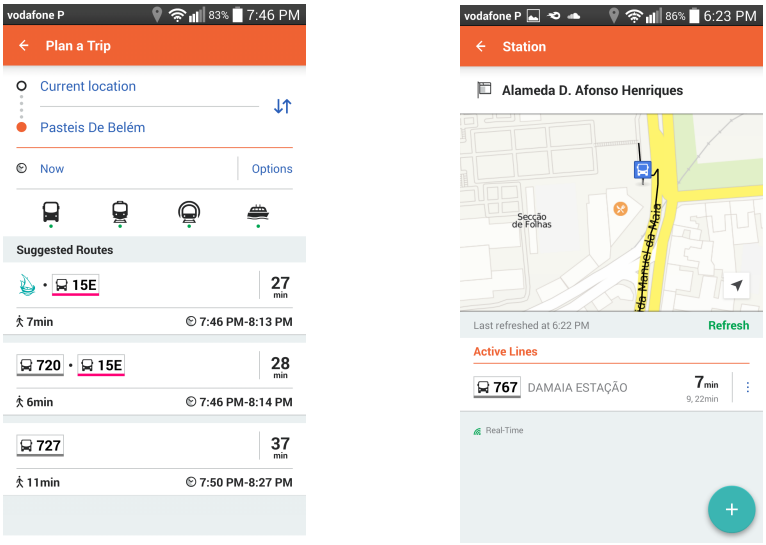


Figure 2.12: Lisboa Move-Me Reviews [16]

2.4.3 Moovit

Moovit is one of the most complete mobile transport applications available worldwide, it uses a data model that mixes “crowd-sourced data” with real-time operator provided data. Service alerts and warnings, line changes, bus stop location maps are all crowd-sourced. According to the developers this model allows the application to be improved by the millions of commuters that use the application daily. The large list of available cities supported by the application overshadows all the other analysed apps during this dissertation, by a large margin. At the time of writing the official company website boasted 600 supported cities spread over 55 countries.

The list of partner operators is also very extensive and Moovit includes a very relevant function for its users that is not supported by the previously analysed applications: a multi-operator route planner that allows users to plan or start a journey which will include not only more than one bus but also a bus and the underground as can be observed in Figure 2.13.



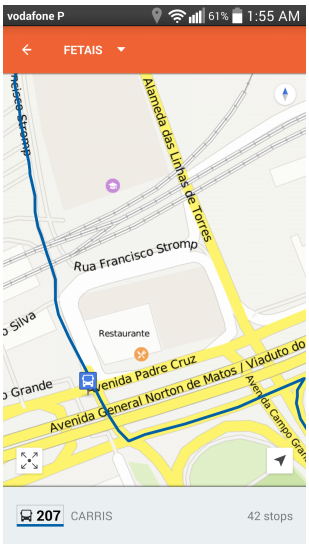
(a) Multi Modal planner (b) Bus Stop Status

Figure 2.13: Moovit example of use

Moovit is has very complete set of features, but a few issues came up during the time spent reviewing it. One would be the lack of reliability of the crowd-sourced data. Data imprecisions have the potential to affect commuters. Two such cases are presented as an example. In Figure 2.13(b) a bus stop for bus route 767 that is placed in the front parking lot of Instituto Superior Tecnico which, at least at the time of writing

is not correct. Another example would be a Figure 2.14 which both incorrectly names “Rua Cipriano Dourado” as “Rua Francisco Stromp” and also misplaces the bus stops in the area (as well as the bus course). Otherwise the application works as desired

(a) Multi Modal planner bug 1



(b) Multi Modal planner bug 2

Figure 2.14: Moovit data crowd-sourcing problems

with a constant focus on route-planning and on the address of places the user wants to reach. A better integration with current user context, location-based and map-based favourite addition could be included but it seems clear that the focus of this application is on getting the user to a destination address. Despite all of its issues with crowd sourced Moovit is still a functional application and its users show their approval in their reviews (Figure 2.15).

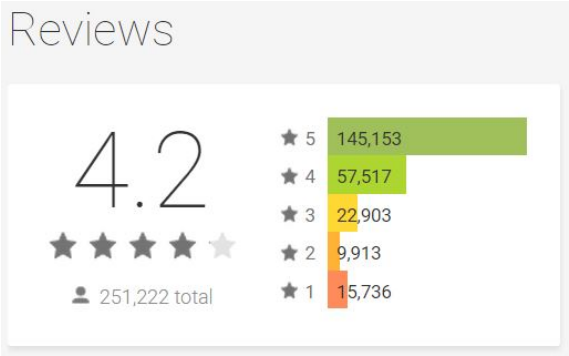
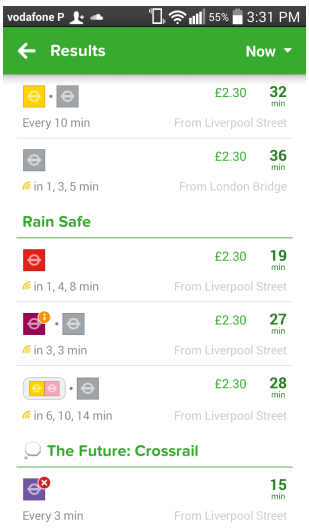


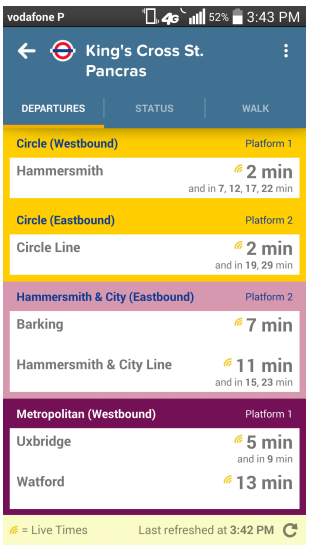
Figure 2.15: Moovit Ratings [15]

2.4.4 CityMapper

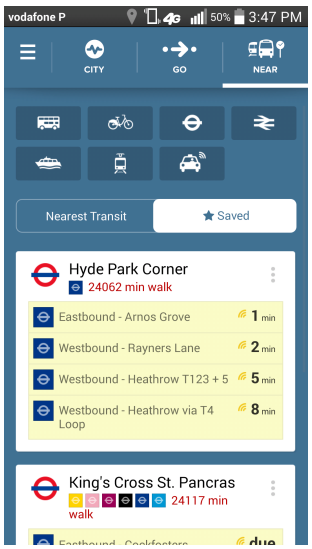
CityMapper was left as a last application to be analysed because it is considered by Google (and the public in general) as the best mobile transports application (official PlayStore ratings in Figure 2.17). The application is built around a comprehensive multi-modal (Figure 2.16(a)) route planner that includes estimated times for the same trip in multiple forms of transportation which include walking, bike, boat, underground, overground, bus, train, taxi, Uber and recent versions even include a “future mode” which includes transport by crossrail (Not available until December 2018) [2].



(a) Multi modal planner



(b) Stop status



(c) Favourite manager

Figure 2.16: CityMapper overview

[2]The version of the application reviewed for this application was for a commuter in London

It is more user context-aware than its previously analysed peers taking into account the geographical position of the user and even suggesting a “Rain Safe mode” if local weather stations indicate that it is raining. Personalization features include favourite places with separate section for commutes with automatic notifications for service disruptions affecting favourites (Figures 2.16(c) and 2.16(b)). CityMapper also provides location sharing through social networks which can be used together with user context and the multi-modal route planner to navigate a recipient to the location chosen by the sender. One downside of so many public transport details is that the application is extremely limited geographically, contrasting with Moovit which makes itself available over 600 different cities, Citymapper is available in just a little over 20 cities advertised in the application’s website and some of these cities are not yet available in the mobile app, making available a very small part of the functionality that the app is used for on the web.

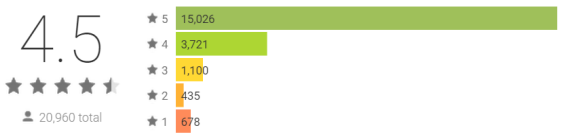


Figure 2.17: CityMapper Ratings [12]

2.5 From Literature Review to Practice

The literature review done in the course of writing this chapter shows that there is a wide variety of research being done in the field of public transportation information on mobile devices. This chapter also makes a review of the current landscape in the field of mobile transports applications available in the Google Play Store, as an introductory framing of the requirements and objectives which will be presented in the next chapter. A few distinctions must be made between what is currently available and what this project proposes to do. First and foremost this project is as much about providing real-time information to the user as it is about providing it the best way. An example would be that the user does not want to consult an extensive list of all buses provided by the operator as much as he wants to know when his commute bus is arriving - this is why the home screen always displays commute transports first. This project aims to not

only provide real-time accurate data on the buses, stops and routes but also displaying it in a logical intuitive manner.

Secondly, the XTraN Passenger Mobile app is part of an ecosystem and not a standalone product. It does have access to broader and lower level functionality than any of its “competitors”. This partnership with Tecmic - the company which engineered the buses’ onboard devices - provided the opportunity to improve the available Restful API, re-engineering small parts of the architecture improving overall performance and functionality of the application as explained over the course of the next chapters.

Page intentionally left blank

Chapter 3

Software Requirement Specification and Architecture Design

This chapter describes the first steps taken in the direction of achieving the objectives of this work. The chapter starts with an objective description of the needs of the partner company Tecmic. These requirements were debated with Tecmic during the initial 3 months of the project in the software requirements specification meetings, and are divided in two groups separated by priority. After the company's requirements for this project's prototype have been clarified the chapter advances to a base set of use cases which includes the most frequent use case scenarios for the prototype. The actual implementation of the prototype and many of the choices made during development refer back to these requirements for guidance, which will be further explored in chapter 4.

3.1 Requirements

According to the defined objectives, this work consists in the development of an android application that will allow the user to access real time information on public transportation available from the operator's XTraN passenger system. Tecmic has plans to commercialize a product based on this prototype so this section describes a list of requirements derived from the defined objectives of the project together with the results and ideas achieved during meetings with the partner company. This app will contribute to the user's well being by personalizing and filtering data into what is useful for the

user at that moment.

3.1.1 Primary Requirements

During the first months of this project over the course of several meetings with Tecmic the following software requirements specification was established:

R0: Interoperable The mobile app prototype must be compatible between different operators and countries because it is available to all the different public transport operators that use XTraN Passenger in Brazil and Portugal.

R1: Static consult The mobile app prototype will make available to users the complete static timetables and static estimated times of arrival (ETAs) of desired buses to any point in the chosen company's network. There will be multiple public transport providers to choose from.

R2: Real time Xtran access The prototype enables the user to sort through the immensity of information made available by the Tecmic Xtran system. Keeping in mind the commuter-oriented nature of the application (discerning between the features that are needed to the user and the ones that are administrator-oriented).

R3: Dynamic contextualized consult The user can know at any time which bus stops are closer to his current position, and the respective bus routes closer to that location. ETA of the next bus in each route is provided.

R4: Personalized information The prototype is able to access the huge database provided by Tecmic and sort through the information available only exposing the user to what is relevant for him/her based on the current context the user is currently in.

R5: Personalized stop status The prototype will provide notifications in the morning with the current status for the user's morning commute bus stop and bus route – integration with Favourites feature.

R6: Locate next buses The user is able to know in real time ETA's for the next buses departing from the intended point of boarding. Time estimates for the bus arrival (ETAs) are provided based on the current traffic context – available time-variants

such as peak and off-peak daily variants as well as weekday, weekend, holiday and seasonal changes allowing for precise planning of arrival at the stop. TimeEstimator service (see section 2.3.3) is a key part in obtaining realistic estimations based on thousands of buses making that same course on similar days.

R7: Trip mode 1 The user will be able to know where he is relatively to the bus planned route at all times when *trip-mode* is ON. This feature will allow a user in an unknown route to know when to press stop and leave the bus in intended stop without previously knowing the place. Warnings will be given when it is time to get off the bus.

R8: Intuitive UI The user interface designed for this application must be responsive and to present data in an intuitive manner in such a way that a new user can easily use the application for its purpose without the initial struggle typical of other applications managing complex data.

3.1.2 Secondary Requirements

A secondary set of objectives has also been agreed upon, this extra set consists of goals that are not considered critical for the successful completion of the project. The partner company suggested some of them as extra features that will only be implemented as long as the primary objectives are still accomplished. The additional set of features is a follow-up on the main work. The partner company Tecmic has invited the author to start working with them by the end of this project in order to see the prototype to the production phase. The extra set of features is as follows:

R9: Trip mode 2 The prototype will be able to access events made available to the system by the Tecmic Xtran device in real time, providing the user real time guidance and warnings relevant to his commute, traffic, delays relative to expected time of arrival. This feature will provide a real time status check of the routes considered relevant (such as favourites) to the user.

R10: Bus detection The detection of the start of the user's journey is a challenge since there is currently no direct communication between the phone and the bus.

The self-triggering of trip-mode and correct bus detection system is one of the more challenging objectives of this project.

R11: Social network Integration and interaction with social networks such as Facebook and Google+ will be provided with the objective of establishing user ratings for the chosen bus routes, keeping companies accountable for delays and other deviations from the static timetable, creating social pressure for not providing reliable bus services.

R12: RSS feed Subscription of the RSS feeds of relevant bus companies will be possible. Integration with Android notification for disruption in the operation of the network will be provided.

R13: β testing It has been the purpose of this project from the start to build an application that will go into production in the future. If the application can be developed into prototype stage in time, tests will be performed following the methodology described in section 5.1. Most of the testing done by the deadline of this report is of the functional type of all available user cases in the prototype.

Construction of Use Cases using the given Requirements

After the establishment of what are the core functions of the application in partnership with Tecmic, a varied set of use cases was drawn in order to make an application that would fulfill the needs of both the company and the user. Each use case was designed to accomplish one or more of the requirements desired by Tecmic. The use cases described focus only on one actor: the user who wishes to get information on public transports around him. The SysAdmin actor has been left out for being considered an outsider to the scope of this work on a public information application.

Favourite Selection

This use case is the first contact any user has with the application. It takes the user through the lists of the various routes and bus stops so that he can establish a set of frequently used bus stops, favourite routes that will appear from then on in his home screen.

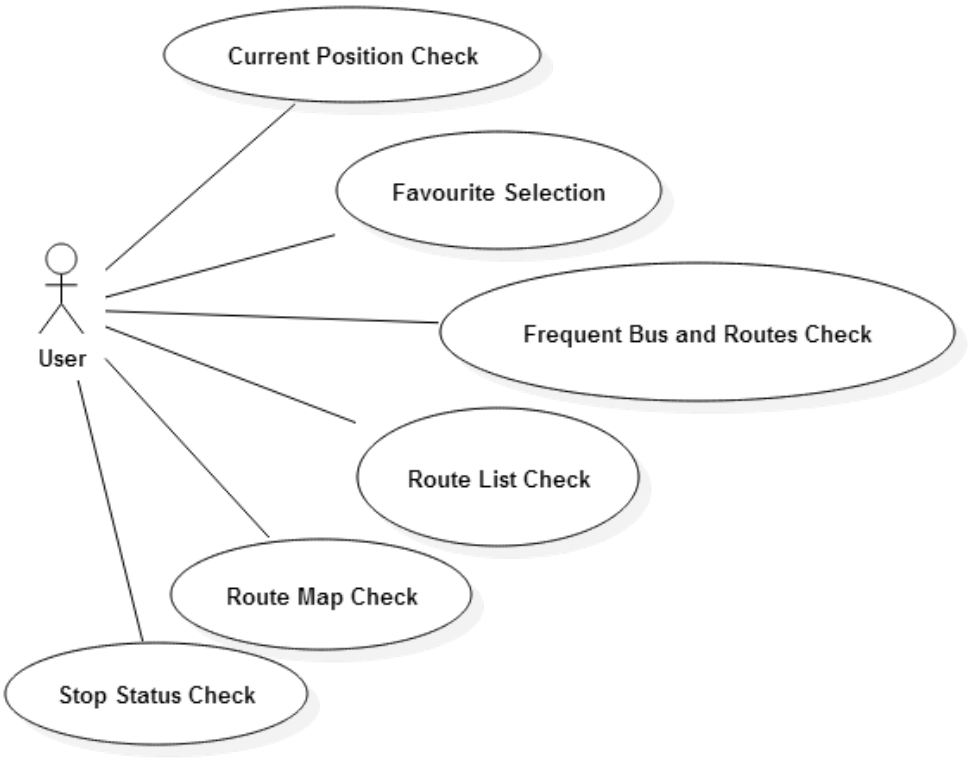


Figure 3.1: Use case diagram

1. The user starts the application.
2. The application detects it is the first time it is starting and launches a Preferences activity.
3. The user selects a city from the list presented by the application.
4. The user selects a bus route variant from the list presented by the application (Figure 3.2(a)).
5. The user selects a bus stop to get the status from in his home screen from the list presented by the application.
6. The user presses the back button to terminate the user case, close the activity and is taken to the application home screen.

Frequent Bus and Routes Check

This is the commute and favourites check feature, one of the core use cases of the application. Because it involves the most used bus routes and stops it probably is the

most used feature, so it was designed be of very quick intuitive access: it can be done in two steps.

For any use but the first time the application is started (see Subsection 3.1.2):

1. The user starts the application.
2. The user is taken to the central tab of the main screen where real time data on his favourite bus routes and stops are presented (see Figure 3.2(b)).
3. (optional) The user scrolls and taps on a specific route to get the vehicle id of the desired bus.

Current Position Check

This use case allows the user to locate himself in the local map. It is not a spider map like in the use case 3.1.2 but a detailed streets' map.

For any use but the first time the application is started (see Subsection 3.1.2):

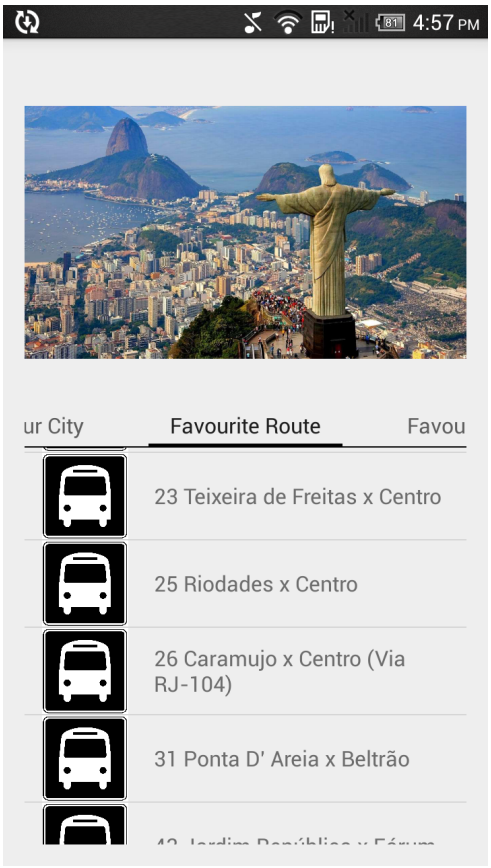
1. The user starts the application and is taken to the central tab of the main screen where real time data on his favourite bus routes and stops are presented and swipes left.
2. The user is taken to a screen with an interactive map that has marked his current position and in compatible devices shows a compass (see Figure 3.2(c)).
3. (optional) The user can scroll around in the map as well as zoom in/out.

Route List Check

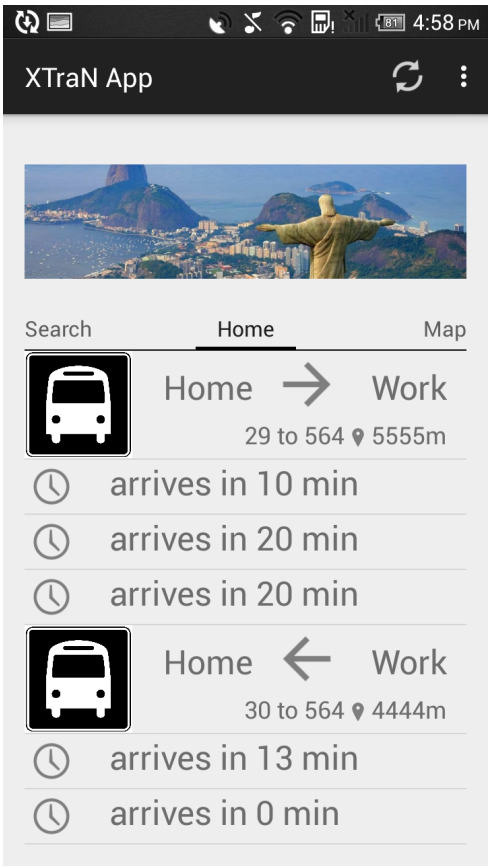
This use case was thought for someone who wants to check a list of all available route variants from the operator - buses that are not on the users favourites' list or possibly the user wants to get on one of these buses but on a different stop other that his usual commute.

For any use but the first time the app is started (see subsection 3.1.2):

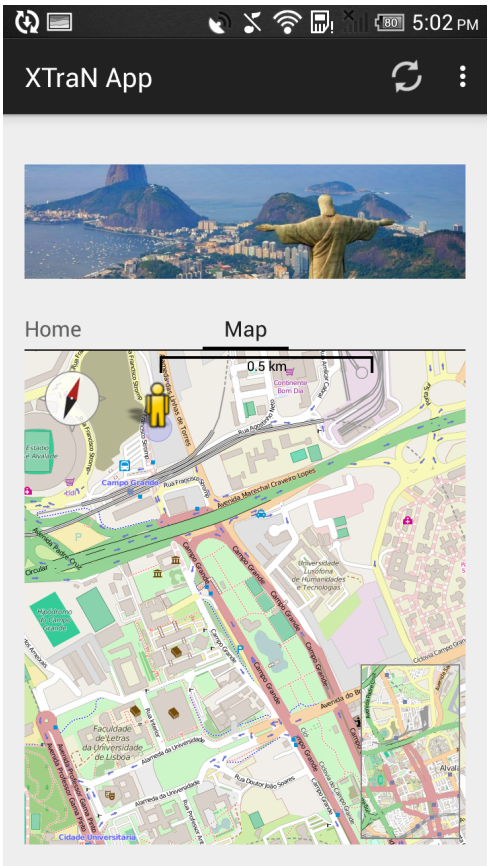
1. The user starts the application.



(a) Favourite selection - step 4



(b) Favourite checking - step 2



(c) Current position in map - step 2

Figure 3.2: Working prototype use case screenshots example 1

2. The user is taken to the central tab of the main screen and swipes the home screen to the right being presented with a full list of bus route variants (Figure 3.3(a)).

Route Map Check

A use case meant for a user who wants to check the spider map of a route variant.

The user starts by following the steps in use case “Route List Check” (Subsection 3.1.2).

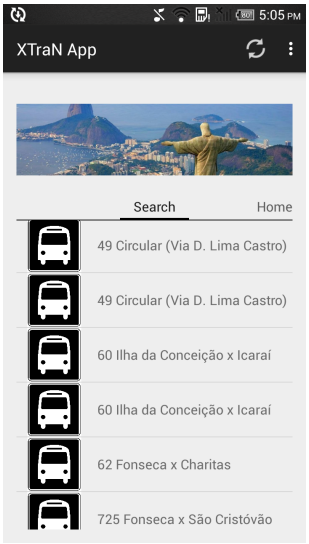
1. The user scrolls in order to find his target bus route.
2. The user is taken to the “Route Stops” screen presenting him the spider map of the bus route/line (Figure 3.3(b)).
3. (optional) The user can scroll to check the complete course of the bus or he can also swipe right being presented with a map of the opposite way of the same route variant.

Stop Status Check

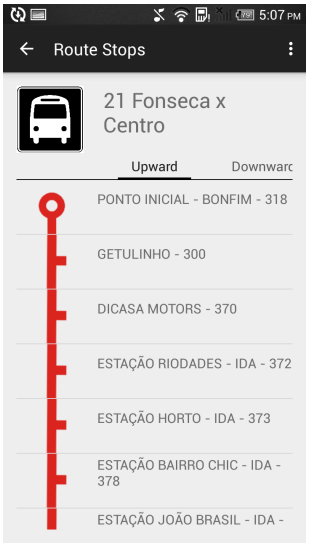
A use case meant for a user who wants to check the current status of a specific bus stop.

The user starts by following the steps in use case “Route Map Check” (Subsection 3.1.2):

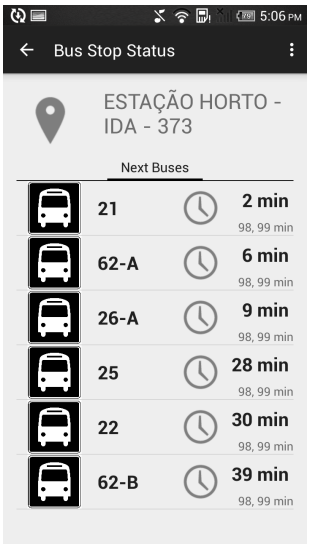
1. The user scrolls (and/or swipes) in order to find his target bus stop in the correct direction of the bus and taps the desired bus stop.
2. The user is taken to the “Bus Stop Status” screen showing him the real time data on all buses whose path includes the target bus stop (Figure 3.3(c)).
3. (optional) The user can tap any of the presented estimations to know the unique vehicle id of the bus arriving.



(a) Route List Check - step 2



(b) Route Map Check - step 2



(c) Stop Status Check - step 2

Figure 3.3: Working prototype use case screenshots example 2

3.1.3 Proposed Solution

In accordance with the software requirements specification, prioritizing the primary requirements over the secondary ones, the proposed solution will consist of an Android application designed and developed from zero. Its features will be designed to solve the needs expressed in section 3.1.1 and will include a quick way for a user to consult the current waiting times for his usual commute buses and bus stops (see R1, R2, R4) as well as other network info on his favourite routes, such as getting a spider map representation of his favourite routes with minimal effort(R1, R3, R8). The spider map of a route will be bridging the gap between the static consult of a route map and real time status of bus stops (R5, R6) of a route and trip mode (R7, R9). All data will be pulled from homogeneous RESTful services extended and updated during the course of this work, this will allow the application to be used by many public transport operators around the world maximizing the impact of this project with particular emphasis on the Brazilian and Portuguese markets (R0).

3.2 Proposed Architecture

This section describes the architecture of the project. It takes advantage of the thorough discussions on software requirements specifications previous to this phase. The current section was designed to answer the requirements specified in section 3.1 and to achieve a modular design adaptable to future changes of the prototype. Moreover the partner company Tecmic expressed the desire for an architecture modular enough that could become the basis of a commercial product, therefore an effort was made to design a prototype that followed the best practices of software architecture and design as well as Google guidelines for android applications (where applicable). Even during the development phase of the project the interchangeable characteristics of the modules proved to be crucial as more features were added.

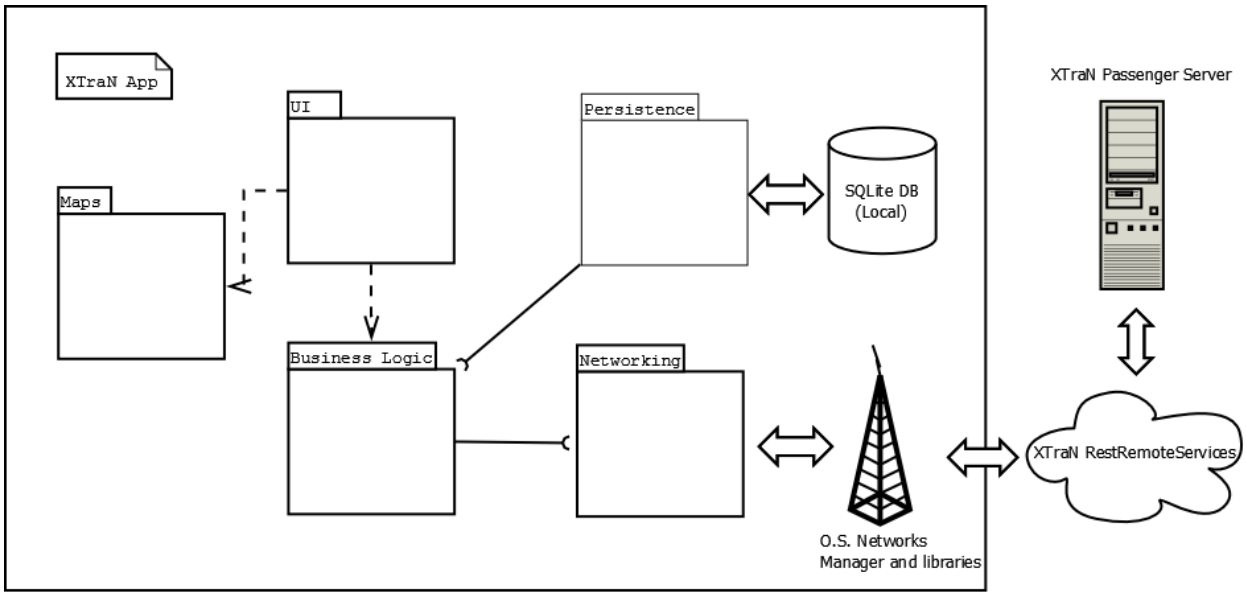


Figure 3.4: Proposed Architecture

3.2.1 Overview

The architecture of this project was designed to follow a layered structure adapted to the android operating system's operational requirements.

The modules which constitute part this prototype were designed to perform the expected features without breaking the layered architecture and without adding unnecessary coupling between parts. The modules can be logically divided in five big components: UI, Business Logic, Maps, Persistence[1] and Networking. The user interface module (UI) was one of the big investments during the project as well as its implementation, and constituted a considerable part of the "prototyping phase" of this work. The UI module of the prototype follows Google guidelines taking advantages of Google's latest developments in Activities, Adapters and Fragments so that a more fluid experience can be achieved. In situations where the components made available by Google were not considered by the author as the best fit, new ones were developed, such is the case in the "Home Screen" for example. The Business Logic module of this project was designed to be as modular as possible while still being an accurate representation of the real world of public transportation from a commuter perspective: it is composed by buses and bus routes, bus stops and waiting times in a way which allows

[1]Despite being taken into consideration for the architecture, the Persistence module was not implemented in the prototype, meaning the prototype relies on its Networking module to perform. The Persistence module will be left for a future stage prior to production.This will not be a problem since Tecmic hired me to continue developing this prototype.

an accurate modeling of the business the prototype is trying to impact. The Business Logic module also uses both the Persistence module as well as the Network module. The Network module makes use of one of Google's latest developments in networking implementations, the Volley library. This alternative to the native AsyncTask allows for faster networking over http, supports caching of requests, and is perfectly integrated with JSON requests which suits the needs of this prototype perfectly. The Maps module currently used for this prototype is the OpenStreetMaps library for android which provides complete overlay customization (which suits our needs to draw objects on the map), interchangeable tile servers so that if Tecmic wishes the prototype to use their own tile server they can just by changing a few lines of code. The Persistence module of the prototype is not currently implemented because it is not crucial for the proof of concept of this prototype but it is an important step to take before commercial launch. The current version of the prototype makes a network availability check during the launch closing "gracefully" if the check fails advising the user that the device is lacking a working network connection. For the development phase it helps that the networking module is optimized and very fast as well as the fact that the prototype during most situations only needs to request text from the server.

The top-level structure and layers include: User Interface, Business Logic, Persistence, Maps and Networking (Figure 3.4). Each section is described below:

- **User Interface** : Deals with presenting the data from the Business Model in the best way possible according to user preferences. Represented in Figure 3.4 by the "UI" module. The User Interface is described in section 3.2.2 and in depth explained in section 4.2. The class diagrams pictured in Figures 3.5 and 3.6 also provide and introduction to be followed up on chapter 4.

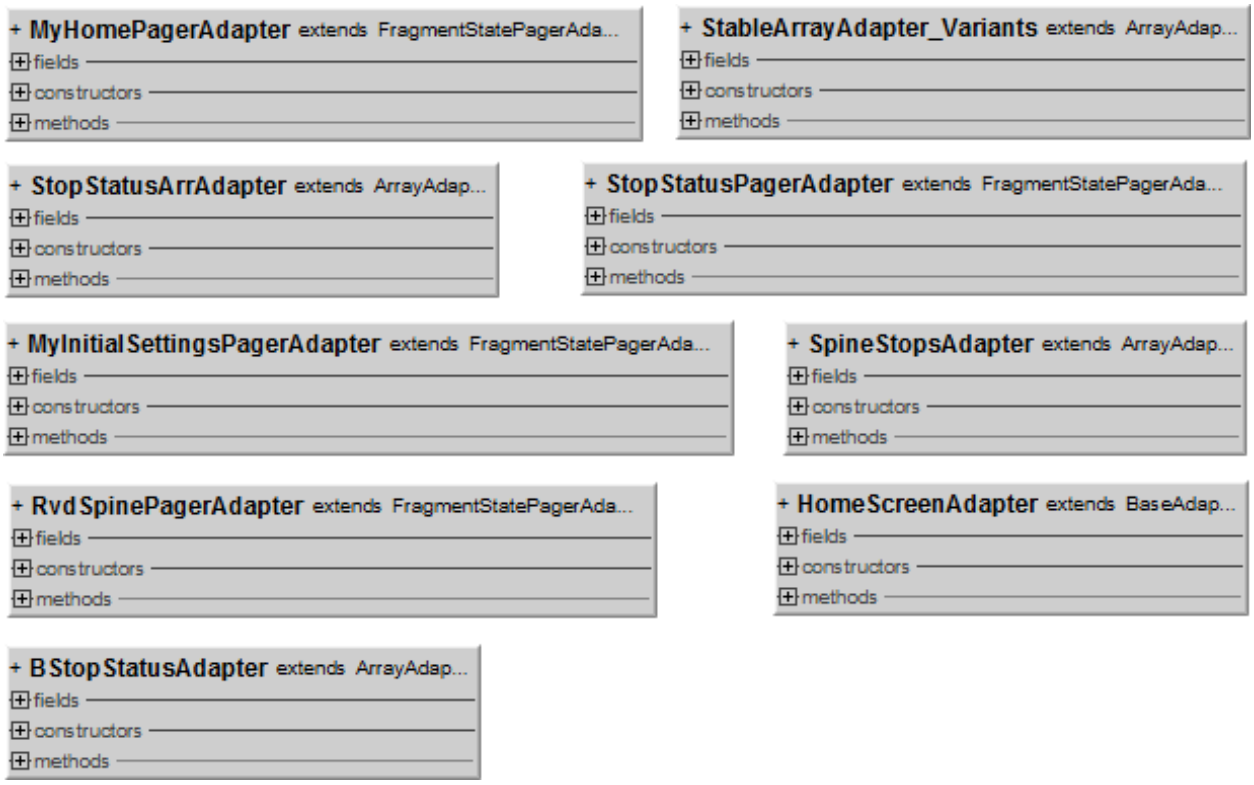


Figure 3.5: User Interface - Adapter classes

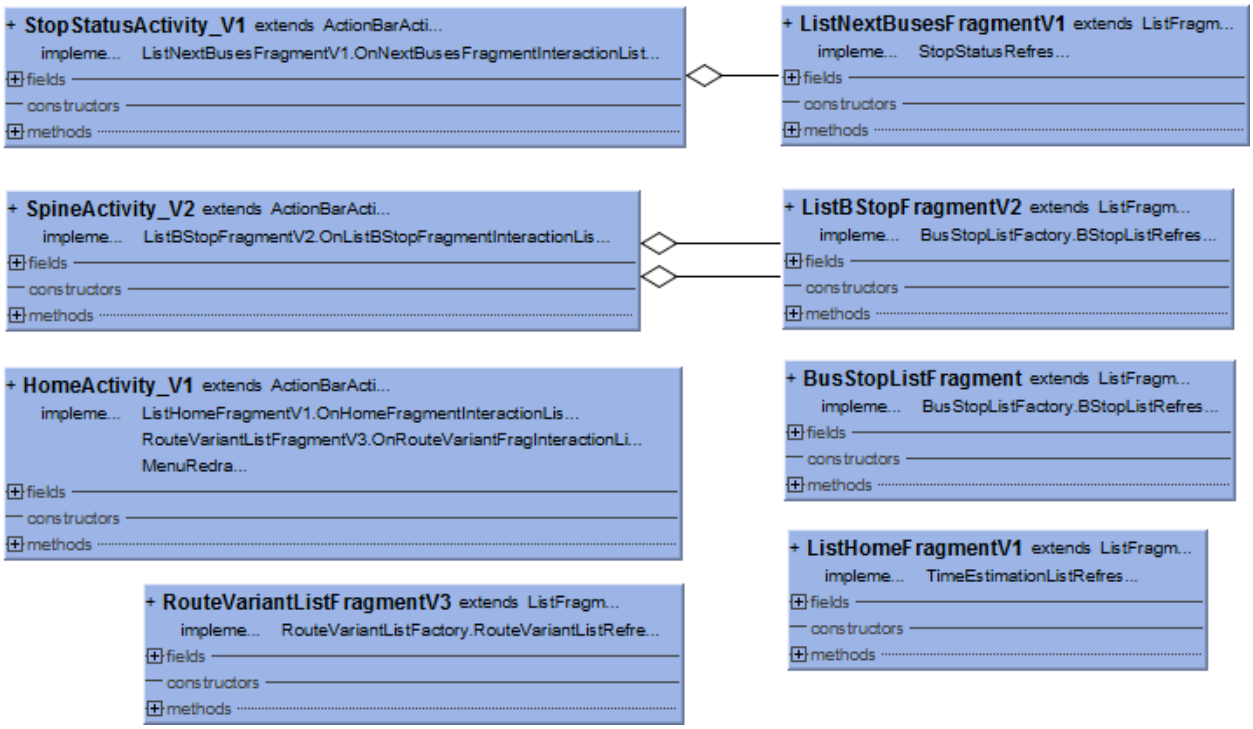


Figure 3.6: User Interface - Fragment classes

- **Business Logic**: This module deals with data structures which represent the real world public transportation companies. Represented in Figure 3.4 by the "Business Logic" module. Section 3.2.5 makes a first description of its function and section 4.3 further explains how the objective is achieved. The class diagram pictured in Figure 3.7 also provides and introduction to be followed up on chapter 4.

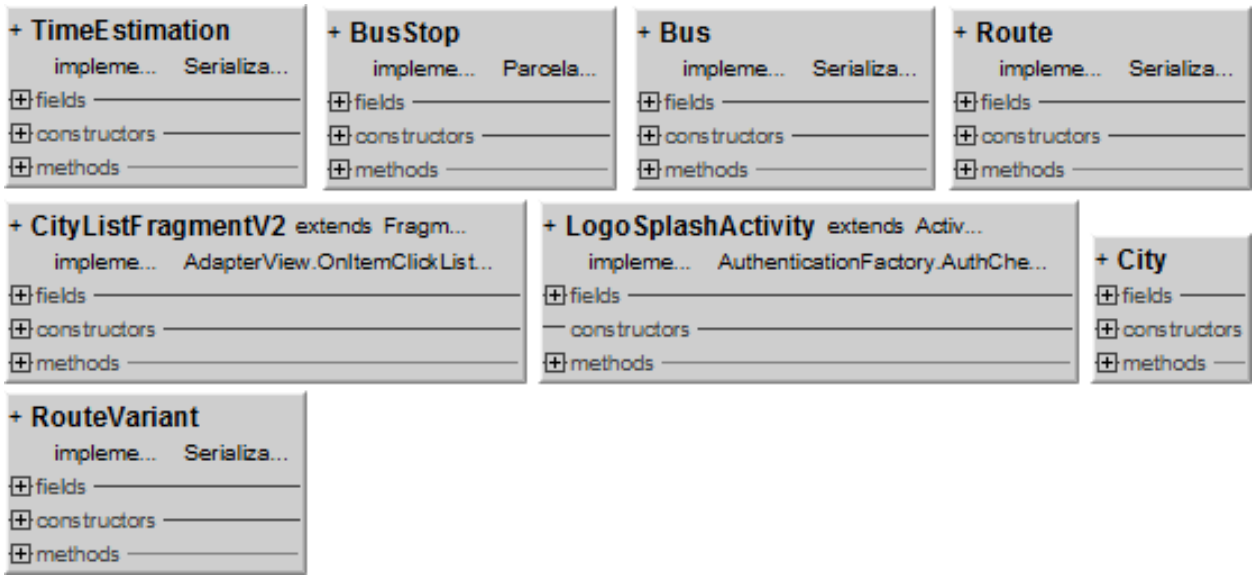


Figure 3.7: Business Logic - Business Logic classes

- **Persistence**: The persistence module deals with local storage following the data access object pattern (DAO) to access a SQLite database. Represented in Figure 3.4 by the "Persistence" module and the SQLite database. The module is described in section 3.2.4.

- **Maps**: Module intended to deal with bringing geographical information to the user. This important module fulfills the application's need for Maps. It uses Open Street Maps for android (OSMdroid) in order to represent user location and other geographical data such as bus stop locations in an easily understandable visual language. It is represented in Figure 3.4 by the "Maps" module. A brief description is made in section 3.2.3, while several important considerations and in depth description are made in section 4.4. The class diagram pictured in Figure 3.8 also provides and introduction to be followed up on chapter 4.

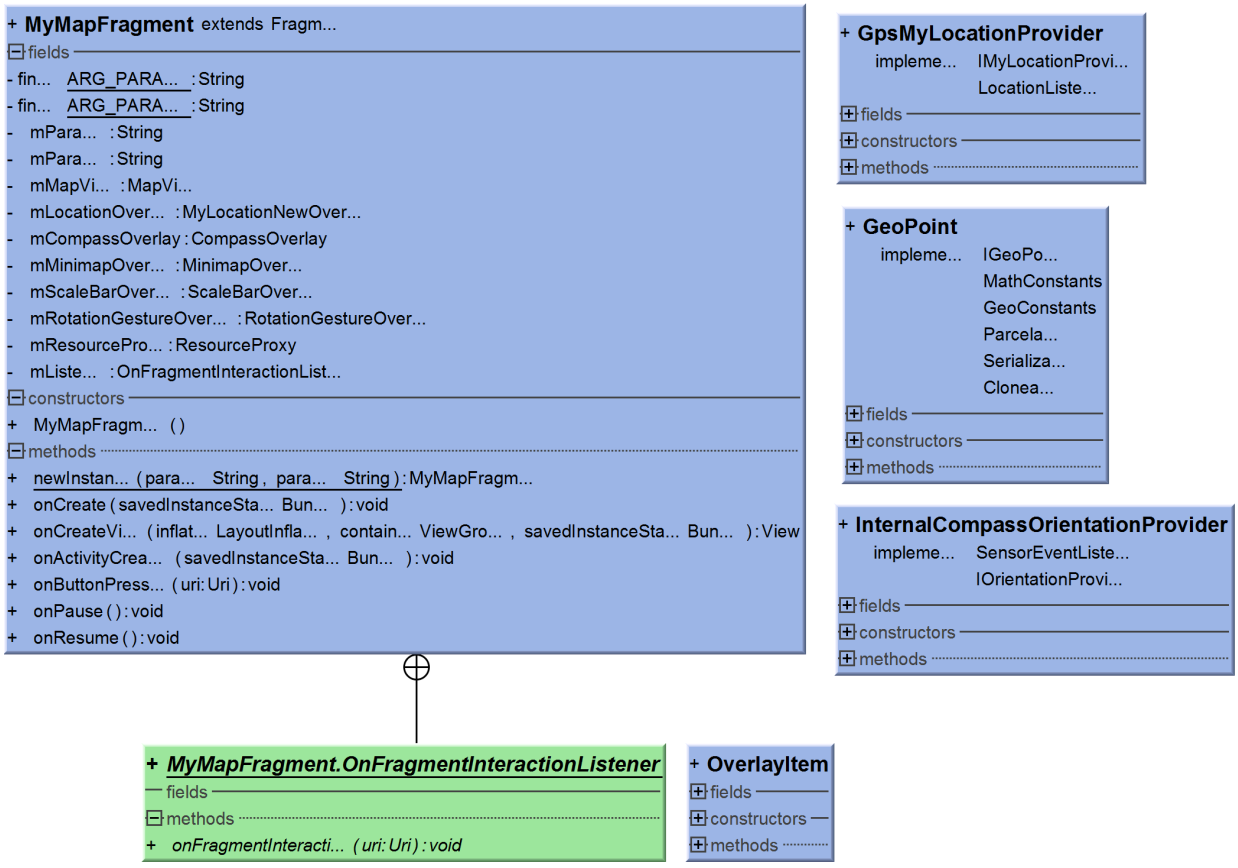


Figure 3.8: Maps - Maps classes

- **Networking**: The Networking module deals with the crucial task of optimizing communications on top of the operating system's networks manager and libraries allowing for asynchronous communication, coherent cache management and other features offered by android Volley. It also deals with the inherent setbacks that library brings. Networking is represented in Figure 3.4 by the module "Networking". It is first described in section 3.2.6 with a more detailed analysis on section 4.5. The class diagram pictured in Figure 3.9 also provides and introduction to be followed up on chapter 4.

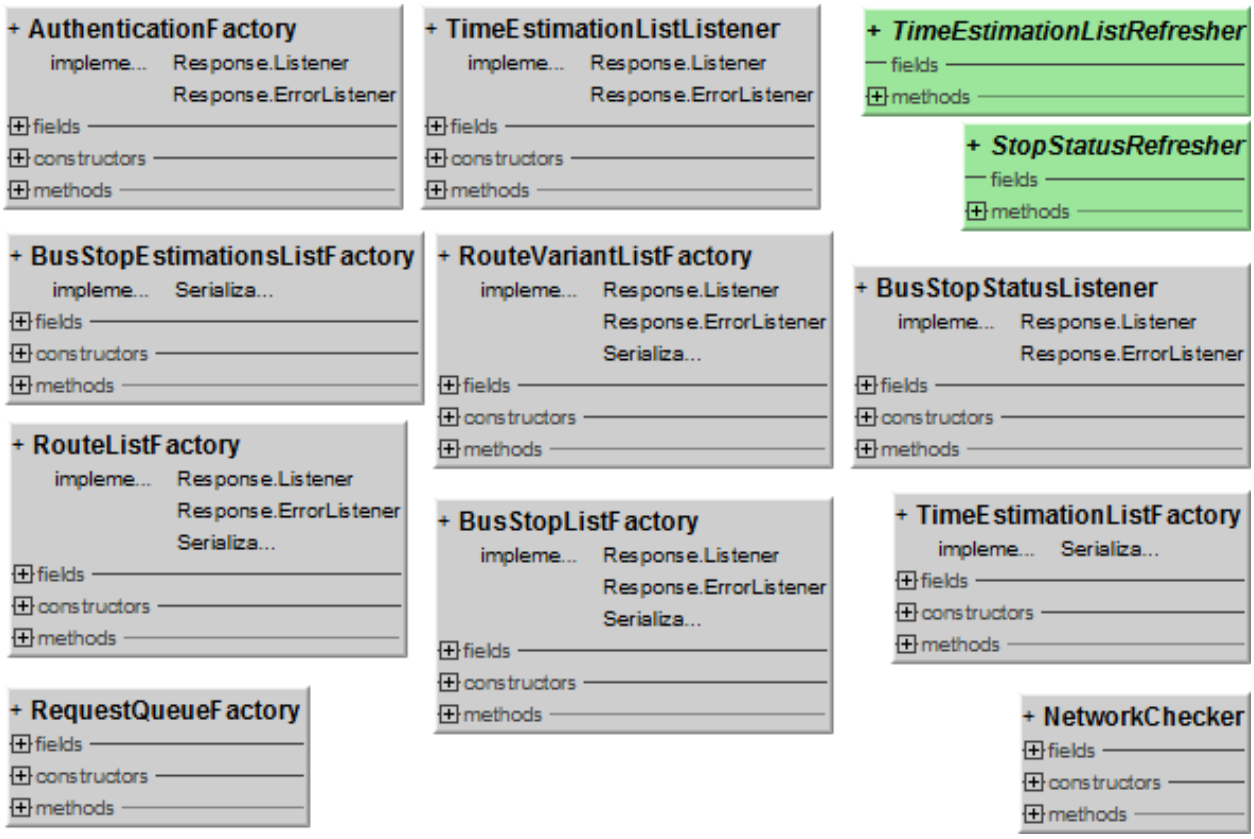


Figure 3.9: Networking - Networking classes

3.2.2 User Interface

The “User Interface” module performs all the tasks related to presenting the data to the user and dealing with user interaction in the application which includes the building of an interface customized to present the right information, as well as the performance of the prototype’s interface itself. This section makes a high level description of the module while section 4.2 goes deeper into the implementation details of the classes responsible for customization and optimization of the user interface in this prototype.

The XTraN Passenger system has the ability to provide varied types of meta data on each bus of the fleet it is installed on. Messages such as “engine-start”, “open-doors” or “outside planned itinerary” are just examples of the virtually infinite amount of data which can be loaded into the mobile application in real time. A large part of the bus meta data available will not be of interest for the public transportation passenger, as it was designed for purposes of fleet management. Nevertheless even among the requests which are meaningful to the passenger, the blobs of data output by the system are so complex that make data-filtering a mandatory requirement so the application can be used by an ordinary consumer. The purpose of the final prototype has been, from the beginning, to provide a strong basis for the development of a commercial product for the partner company Tecmic. With this objective in mind and according to the needs of the situation the following base components of the android framework were studied and used: Activity, Fragment, ViewPager, Adapter, Intent, AsyncTask, Menu and Layout.

Despite the typical good performance of these Google provided components, it became apparent that they could not completely fulfill the needs of this prototype. Given the complexity of the data received for many of the implemented use cases, most of the original android adapters were simply not fit to use them as data sources. Additionally, in order to avoid a proportional increase in the difficulty of use of the prototype, a necessity for more customized UI layouts became a priority. New components were implemented sometimes only using abstract base classes in need of complete manual implementation with serious impact on performance of the application. Therefore the customization of android base components during the course of this work had two main objectives:

- **Customization** Android provides a set of basic components such as activities, fragments and adapters to deal with the most basic needs of the prototype in order to simplify the developer's task and increase his productivity. But in a business with such an abundance of blob data, bringing the most relevant information closer to the user makes the distinction between a useful application from a prototype that will not make it to production. The necessity to show the user the correct information in a clear, intuitive layout when the data source is complex dictates the need for layouts "tailor-made" to suit the situation. These components are described in details in sections 4.2.1 - 4.2.3, and 4.2.9, 4.2.12.

- **Performance** Components such as adapters which are the liaison between the model and the user interface are elements of key importance to the personalization of the application by transforming blobs of data into UI view elements. However they also have an important role in the performance of the interface components hosting these views (AdapterViews). A common AdapterView such as the Android ListView can only be as fast as the performance of the underlying adapter allows it to. While the frameworks' specialized adapters designed for "typical situations" have a very good performance overall, they will not be suitable for the application's "tailor-made" layouts. Therefore the building of our own adapters also requires an effort so that performance of the attached AdapterViews can be optimized. These components are described in sections 4.2.4 - 4.2.8 and 4.2.10 - 4.2.11.

3.2.3 Maps

This module is intended to deal with bringing geographical information to the user. In order to perform the task of presenting the user with data he can easily understand and relate to, the UI module includes carefully developed extensions to the original UI components built into the Android operating system. These classes are described in sections 3.2.2 and 4.2) but the actual presentation of geographical data remains a challenge to be addressed in this section and in section (4.2.9). This module uses the open source external library OSMdroid for rendering the data inside the application's Activities and Fragments and Open Layers and Overlays to draw layers of geometries

such as bus stops on the map.

When an application needs to access user location for its operation it will make a request for the users' location to the operating system. This can be done by first asking the O.S.'s location services for one or more "location providers" which represent the different ways available to the operating system and the developer to get location data. The "location providers" typically present in an android mobile device are: GPS provider, Network provider (with or without WiFi access point data) and the Passive provider.

As these different "location providers" are considered, additional considerations and points of failure will be added and need addressing, issues such as being inside of a building or in fast movement will have an impact on the use of location services. A few commonly found challenges in applications that make use of current user location are [9]:

- GPS signal availability
- Location accuracy and multiple location providers
- Wi-Fi signal availability
- Network (GSM/UMTS) signal availability
- Changing location
- Power consumption

The challenges mentioned above can be solved by evaluating for each situation and deciding on an equilibrium between performance and power saving as well as with leveraging Google's vast database of user data from the well known Google location services. Any application which needs access to precise and broad user location data will need to request these permissions in the applications' manifest, access fine location and access coarse location respectively, the strategic part is on "how" to make an optimized use of all the location providers in order to get the "location fix" while dealing with the constraints.

Even having both the mentioned permissions GPS signal may not be available, or it may be taking a long time to get the initial fix in a new location so there is a list of different location providers available for use, with different advantages and drawbacks.

3.2.4 Persistence

The persistence module's purpose is to store a variety of data in a non-volatile form, meaning if the device is turned off, the user can still access the data upon reboot. In situations when the user does not have access to a WiFi or when cellular data connection fails this layer could also be used to offer partial functionality. There are several options for persistent storage in Android: file storage, the SharedPreferences API and a local database storage through the use of SQLite, each of these types of storage has its advantages and setbacks. They can be analyzed according to the desired application.

3.2.5 Business Logic

The business logic module models the real world into java classes and objects. Regarding the business logic, from the start, the approach taken with this prototype was to use the knowledge gained in the area of public transportation business, make an analysis of the model built by Tecmic for the fleet management application filtering out the parts that could be dispensable for a public domain application such as this one. The result became a much lighter model that still represents the real world public transportation business from the point of view of a bus passenger. This reduction on the model will also alleviate load on the local storage SQLite DB when it is implemented. Another point that took some thought to put into project and implement was the adaptation layer between the data and the UI the customized layer of adapters built to fulfill the requirements of the prototype, the weight of the choice of showing the most data on a certain subject versus showing the the most relevant data is a process most relevant in the building of the data structures and representative model in this prototype.

3.2.6 Networking

The search for the right networking tools holds a high degree of importance in a world where distributed systems are everywhere including our mobile phones. The particular circumstances of the prototype belonging to a large distributed commercial system highlights the need for a standard, simple, reliable network layer that will simplify the communication between the mobile application and Tecmic's RESTful API. Given that

the purpose of this application is to become the basis for a production prototype, one of the main requirements of this project is keeping the app's UI highly responsive. This requirement forces all network requests to be made asynchronously so that the user does not see the interface "freezing" while waiting for the network response. This is one of the most important requirements but it is not the only one, the list of operational requirements of the ideal Networking layer for this prototype is as follows:

1. Asynchronous requests.
2. Designed to use HTTP.
3. Raw JSON support.
4. Request cancellation support.
5. Native caching mechanism.
6. Access to more than just the HTTP payload.
7. Native authentication mechanism.

These requirements and the available options for a solution are further explored in section 4.5.

3.3 RESTful API expansion

The real time nature of this project implies that it relies on server-side infrastructure. The capabilities of this infrastructure do, in more ways than one, set the limits of what the application is capable of giving the final user. When this project started the RESTful API RestRemoteServices from Tecmic was already in existence and working (see Figure 2.5) the issue was that it provided nowhere near the desired functionality and data the prototype required in order to satisfy the software requirement specification established with the partner company. Since the start of this project that both sides knew the existing API would not suffice, therefore the only solution was to expand it, so a set of new features and methods was established to be worked on in partnership with Tecmic developers:

1. Get stops belonging to a certain route.

2. Get time estimation for a bus.

3. Get time estimation for a route.

4. Get static timetable for a bus stop.

5. Get all publicly available RouteVariants.

6. Get all RouteVariant's static timetables for local storage.

7. Get closest bus stops to a user's location.

Most of these previously established API requirements were worked on together with Tecmic developers and became deployment-ready during the course of the project. My role here included the design of the required specification since at the time my access to production servers was limited.

Chapter 4

XTraN App

4.1 Introduction

This chapter further explores the development of this project continuing after chapter 3. Chapter 4 specifies an implementation for the prototype and discusses the options taken, their advantages and setbacks, and takes advantage of the study done during previous chapters to justify and make these choices.

During the implementation phase of the development of this project a “Package by Layer” approach was taken. Instead of having one package for each feature on the root folder of the project the organization of the project is as described in Figure 3.4. Organizing the code in layers, any developer with knowledge of the inner workings of android can pick up this project and continue development. When compared with a pure “Package by Feature” approach the chosen option is less modular than the alternative but it suits some of Tecmic’s needs better in this project. One example would be that in the future more developers could quickly pick up the project, more developers are familiar with software layers than with a unique structure drawn from specific features of the product each with an individual package and each with its set of files. Another reason for the choice of this organization was that if a literal “Package by Feature” approach had been followed, another choice would have to be made. Before evolving into a more user friendly product, the prototype used a more homogeneous UI for most features, activities, fragments, which presented as “Package by Feature” is translated into either breaking the pattern by using code external to the feature package and therefore breaking the “pure modularity” offered by package by feature, or by

replicating code in each package hindering all development of the UI. This would prove to be an even more serious problem with business model related classes because there are many features using entities such as buses, routes, bus stops. A common solution to this problem is also breaking pattern by making an exception in the package organization putting all model related classes in a separate package. After initially considering a “Package by Feature” approach the referred problems and the consequences of their discussed common solutions were disadvantageous when compared to a clear “Package By Layer” organization in this project that will allow any android developer to continue this work with less overhead and time consumed understanding the alternative feature structure.

4.2 User Interface

The User Interface module includes components which can be organized in two groups: Customization of the User Interface and Performance of the User Interface. This section is organized in subsections, the first introduce the specific problems to be solved (4.2.1), followed by subsections which describe the classes that solve these problems. The following sections describe implementations proposing solutions related to User Interface Customization:

- 4.2.2 - Custom Screen Implementation
- 4.2.3 - Heterogeneous ListView and BaseAdapter Implementation
- 4.2.6 - Data Feeding the Adapter: Data Source Merging
- 4.2.7 - Returning the correct ViewGroup in an Heterogeneous AdapterView
- 4.2.8 - RouteVariantListFragment and StableArrayAdapterVariants
- 4.2.9 - MapFragment (and Drawing Overlays)
- 4.2.11 - BusStop Status Implementation and ArrayAdapter use

The next list of sections was written in order to describe optimizations made to the use of the reduced resources available to mobile devices such as memory and processing power:

- 4.2.4 - Optimizations to the Custom Adapter's Behaviour I
- 4.2.5 - Optimizations to the Custom Adapter's Behaviour II
- 4.2.7 - Merging DataSources and Differentiation between data types
- 4.2.10 - Activity as a host to a ViewPager Container
- 4.2.12 - Route Spine Activity Implementation and ArrayAdapter use

All of the components in this list are further described in the sections below.

4.2.1 Building the User Interface

In an Android project each "window" is an activity which contains other UI elements. In this project, in order to better organize the architecture and keep each layer separate, each application feature is mapped to its own activity or activity fragment in cases where better usability demands that closely related features be put together in one activity. This choice of organization improves modularization by taking some ideas from a "Package by Feature" approach without the setbacks of actually organizing packages in this way, as described in (Figure 3.4). The presentation layer is mostly composed by activities, fragments, layouts and drawable resources. Taking a simple case as an example (see Figure 4.1) where one activity groups together the most used features in swipe view tabs, these components work in sets: each activity uses a layout and drawable resources, activities may also contain fragments with their own lifetime cycle which in turn also have their own layouts that may use drawable resources. The Presentation layer is loosely coupled with the Business Logic layer, and Persistence layer and all code here aims at making a better use experience and displaying the information provided by the Business Logic layer in a way that makes sense for the user. In this project the addition or removal of a feature will usually be translated into adding or removing the corresponding activity or fragment with the respective layouts and resources, leaving persistence and business logic untouched, although in this case there will probably be some unused code and tables in the Business Logic layer and Persistence layer respectively, but the remaining application functionality will most likely not break with the removal of features.

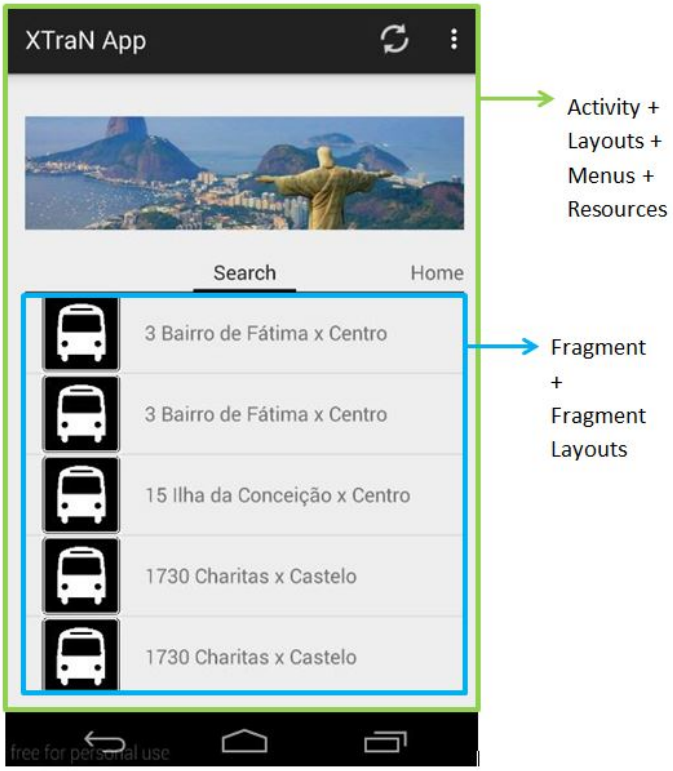


Figure 4.1: User Interface example: activity hosting a fragment that features all the available bus routes

The building of a typical Android interface starts with the coding of the XML starting layout. The main elements of this schema are View and ViewGroup objects. Views are user interactive interface elements and ViewGroups are interface elements which can host other View and ViewGroup elements. Resulting in a hierarchical structure such as the one seen in Figure 4.2. Both Views and ViewGroups and their behaviour and state can be defined in XML and dynamically changed in code, and according to Google this is the correct and easiest way to implement an android interface.

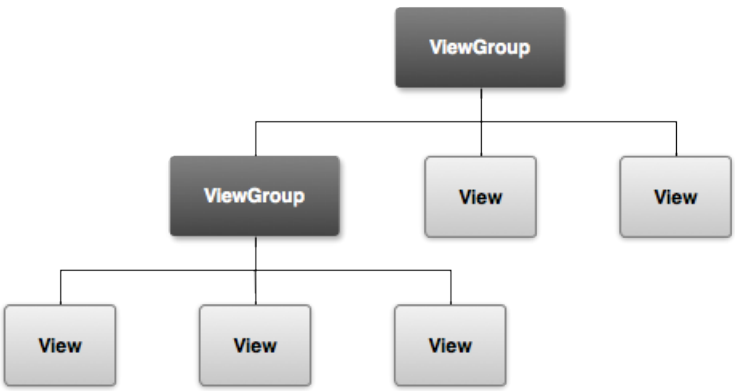


Figure 4.2: Interface layout - views and view groups [7]

The framework includes its own extensions to the View and ViewGroup objects sparing the developer a lot of boilerplate code in the most common situations (e.g. linear layouts, buttons, text fields). This setup allows a typical android user interface to be built in two phases: Declaration of elements in XML - which provides a solid base hierarchical structure Views and ViewGroups - followed by runtime instantiation of these elements making them eligible for being manipulated or queried at the developers discretion. A specific example can be seen in Figure 4.3 taken from this project in Android Studio layout preview mode, where the views were not yet manipulated by code. What can be seen is approximately what could be expected of the BusStopStatus activity if there was not any runtime manipulation of the instantiated views. The base structure of the layout in this activity is composed of:

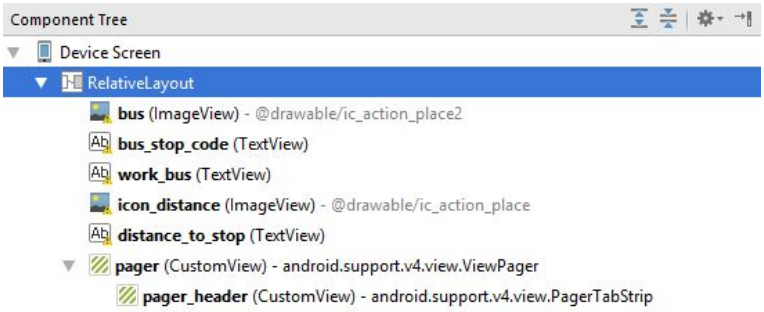
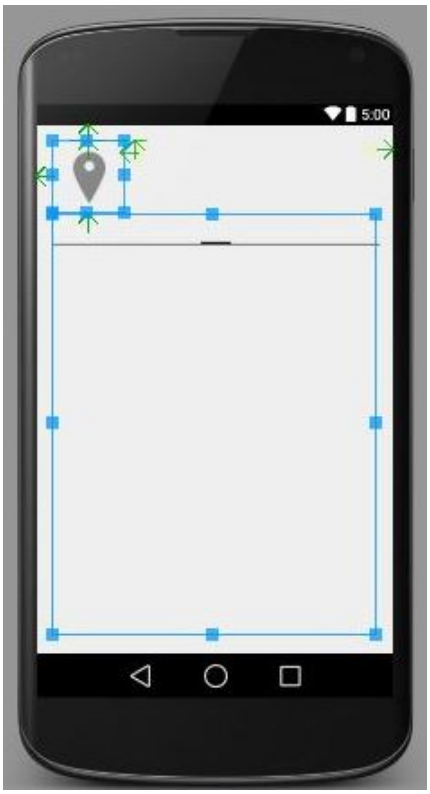


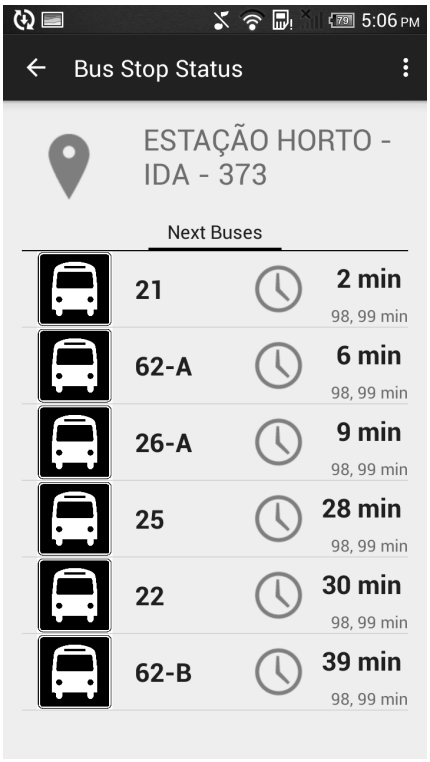
Figure 4.3: Interface layout - base XML layout

- **RelativeLayout** Subclass of ViewGroup (therefore also of View) in which the hosted child Views are described in relation to each other or in relation to the parent View (the RelativeLayout itself)
- **Several TextViews** For the purpose of displaying information in string format.
- **ImageView** For the purpose of displaying information in the form of an icon.
- **ViewPager** A child View which is a direct subclass of ViewGroup, like the layout, therefore it will host child views of its own, in this specific case other ViewGroups and one PagerTabStrip to simplify navigation.

It becomes obvious from Figures 4.4(a) and 4.4(b) that the creation of a working production-ready interface in Android requires a bit more work than instantiating the components through inflation in XML. Exclusively from the UI customization perspective (ignoring all the work done in other architecture layers), in order to go from Figure



(a) Interface layout - Hard-coded XML layout visual representation



(b) Interface layout - After dynamic manipulation in a running activity

Figure 4.4: Visual representation of hard-coded base XML layout VS dynamically manipulated layout

4.4(a) to Figure 4.4(b) the required data-sets must go through an adaptation layer which was designed to transform these abstract data-sets into interactive UI elements displayed in accordance to a customized layout.

4.2.2 Custom Screen Implementation

One example of deep interface customization in order to meet the requirements of the application is the implementation of the home screen of the application, particularly its main SwipeView. By studying exclusively the layout of the its main fragment, nothing would indicate any kind of customization it is a simple ListView(see Figure 4.5).

But looking at the final result of the working application it becomes evident that there is some other factor of customization at play in this case. Taking into account the complexity and diversity of the data-set in question (from bus numbers to bus stop ids to time estimations and distances) two options were available:

1. Converting the various possible objects to string values which would fill each

Figure 4.5: Generic ListView representation [8]

row with chosen pieces of data from the object in question, using a standard Google provided adapter. Resulting in a uniform typification of each row (all string-type rows), despite the diversity of the information to be displayed. The different sources of data would also need to be fused into a single data structure to feed the adapter which would then use it to fill the ListView. Taking this option would save a lot of work, but would also make the application virtually impossible to use.

2. Acknowledging that the complexity and diversity of the underlying data structures require dedicated customized display elements in order for the prototype to be of some value other than academic. This was the option taken.

By acknowledging that the provided tools included in the android framework do not fit the requirements of the prototype the project of the interface was expanded to include several steps (for each of the required screens):

- Setting individual Views in customized rows suitable for the entity model it represents.

- Implementing a customized adapter built to deal both with the desired model entity and with the individual Views inside the specialized row mentioned above. This is a non-trivial matter since depending on how much specialization is needed

the rows in the list, memory and processing power management are relevant issues for the mobile device.

- Perform the necessary optimizations to the CustomAdapter so that it can deal with large data-sets and still behave with fluidity.

- (Repeat for all the customized ViewGroups in the screen).

In section (4.2.10) these steps will be discussed for the specific case of the main SwipeView of the Home Screen.

4.2.3 Heterogeneous ListView and BaseAdapter Implementation

Choosing to implement the abstract class BaseAdapter is a commitment to manually do the heavy lifting code the framework usually does for the developer. It requires a deep understanding of working mechanisms behind the Adapter and the corresponding AdapterView and the manual implementation of the pair's interactions and posterior behavior optimizations. The discarding of the frameworks' provided adapters is a debatable choice due to the considerable extra work it requires and it could be argued that partial overrides of existing adapter implementations could be used but the degree of freedom in customization allowed by implementation of the base abstract class is much broader than that of partial overrides of framework adapters. The only disadvantage being the extra workload of manually implementing all the required methods and the "optional" performance optimization mechanisms.

Desired behaviour of CustomAdapter and the corresponding AdapterView

The choices mentioned in the last section make the actual implementation of the Adapter the next step of the process. Knowing the desired form of AdapterView will be a ListView to be filled with customized rows leads to thinking of the behaviour of simpler listViews and the Google's take on their implementation. In order to keep the UI fluid and responsive, no more rows than the ones that fit the screen (plus a small offset) will be kept in the mobile device's memory. This is true independently of the size of the data-set feeding the underlying adapter.

Sparing the mobile device's limited memory through this method has a serious consequence for performance from the processing power point of view. One of the most expensive operations (for the mobile processor) is the process of "inflation" of the interactive UI Views hosted by the AdapterView from the XML files. In this case the inflation of each row, itself a heavily customized ViewGroup, involves the inflation of several child Views per row including three ImageViews for icons, so the problem is exponentially worse than with a simple ListView composed of single TextView rows. Making the call to GetView() can become one of the heaviest operations to hinder the responsiveness of a complex interface such as this, and this is the callback method invoked each time a row is scrolled into the screen. Worse case scenario the user could make a quick scroll up or down a large list - quickly draining the ability of the processor to keep the inflation speed up with the rate at which new rows are coming into the screen.

4.2.4 Optimizations to the Custom Adapter's Behaviour I

One possible solution to the challenge presented in section 4.2.3 is to reduce the cost of each calling the GetView() method. A particularly effective strategy is to target its most expensive operation, the call that trigger XML file inflation: Inflater.inflate(R.layout.home_work_h
This can be done by only inflating enough Views to fill the screen (once), and reusing them through a process called "recycling" which involves the views which are scrolled out of screen being referenced and reused in the calls to GetView() of the rows coming into the screen. A good illustration of this process by L.Rocha [17] can be seen in Figure 4.6.

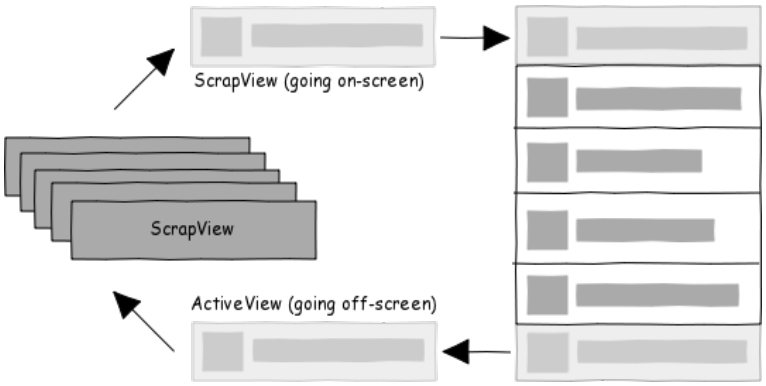


Figure 4.6: Generic representation of the view recycling process in ListView [17]

4.2.5 Optimizations to the Custom Adapter's Behaviour II

Another costly operation frequently used even after the optimization in section 4.2.4 is the finding of inner Views inside inflated layouts, with a cost proportional to the amount of child Views in each ViewGroup. For reference, instead of a single TextView standard case for ListView rows, the rows heading each favourite section in the main SwipeView of the home screen include: 3 ImageViews and 4 separate TextViews which could result in 7 calls of FindViewById for each row scrolling into screen.

One possible solution for this problem is implementing a pattern known as the ViewHolder pattern. This pattern is meant to drastically reduce the number of FindViewById calls. The ViewHolder is a container class which will hold the references to the rows' inner views avoiding repeating the expensive calls to the FindViewById() method. After the first inflation of the ViewGroup layout a first and only call to FindViewById() occurs and the references to inner row views are stored in the ViewHolder object for future use.

4.2.6 Data Feeding the Adapter: Data Source Merging

The role of the the customized adaptive layer can not be described in detail without mentioning the data structures feeding the adapter (and the corresponding AdapterView). Once again the particular circumstances of this adapter require some extra design and implementation work. The android framework assumes both homogeneity in the model entities being passed to the adapter and the view types representing those entities resulting in an AdapterView such as the general representation seen in Figure 4.5. In the case of more complex customized interfaces this assumption does not apply: a strategy is needed in order to deal with representing different entities in the same AdapterView with a distinct interface (distinct rows with different child Views) without breaking the View recycling mechanism (see section 4.2.3).

4.2.7 Merging DataSources and Differentiation between data types

The current version of the main swipe view of the home screen inflates and recycles 4 different types of views depending on the type of favourite needs to be displayed. Accompanying these different View types there are 3 different collections with distinct

data types representing model entities which are needed to feed the adapter for this AdapterView. Focusing on the data types which come from different sources two options could be taken: monitor and manage as many different data sources as necessary or merge the different collections into a new List of unspecified Objects and perform the necessary management on this single structure. The latter option was the one taken. While it can be argued that this abstraction could create bugs for the whole AdapterView if one of the dataSources stops working, it is also true that the position indexing in one single abstract structure is more suitable to the use of patterns followed by the framework, and provides a guideline for the use of efficient View recycling patterns that would be broken otherwise. If the data sources can successfully be merged into a single one the next challenge to overcome is the adaptation of the View retrieval and recycling mechanism to an heterogeneous data source, in other words the AdapterView needs to know what type of ViewGroup will be needed next upon user scrolling.

Returning the correct ViewGroup in an Heterogeneous AdapterView

Being a base implementation of the abstract class BaseAdapter any kind of indexing would be possible. Taking into account that arranging the display in a easily understandable way would be better for the user, the AdapterView will display the custom rows grouping them by type. This strategy reinforces and takes advantage of the way the data sources were merged, View retrieval is done in accordance to the global position of the needed data relative to the merged single structure, with the objective to fill the correct row type in the ListView. As a result, independently from the number of entries originally merged into the global data structure the correct View type to be retrieved will always be correctly indexed to its data type through pairing index sets with the correct View types.

The way the adapter was designed and built, more types of favourites can be added, meaning the current implementation of the adapter could easily be changed to accommodate new View types and new merged data sets into the global merged data structure feeding the adapter.

View Retrieval Implemented (getView())

As a conclusion to this section of the text a review of the view retrieval mechanism is in order.

As described in the previous sections the manual implementation of the abstract base class brought several advantages such as the ability to completely personalize the way information is displayed to the user, as well as bring the different ViewGroups as heterogeneous rows in this ListView. The quantity of planning and code it required might seem dissuasive at first, but the level of control achieved by going "behind the curtain" of the typical use of the framework is impossible to achieve with the provided components. Moreover it allowed a complete rewrite of the interactions between Adapter and AdapterView which was a necessity for the optimizations described in the past sections.

4.2.8 RouteVariantListFragment and StableArrayAdapterVariants

This fragment is the part of the interface responsible for bringing the user the features of checking all currently available RouteVariants. This is also the departure point for the user to verify each route's synoptic map representation and each bus stop's real time status.

Adapter and AdapterView

The customizations needed in order to provide this feature were not deep as in the last sections, one of the reasons for this fact is the homogeneity of the ViewGroups constituting each row placed in the AdapterView (see Figure 4.7). Each row's layout was customized as a ViewGroup constituted by an ImageView (the icon) and a TextView (with the routeVariant description).

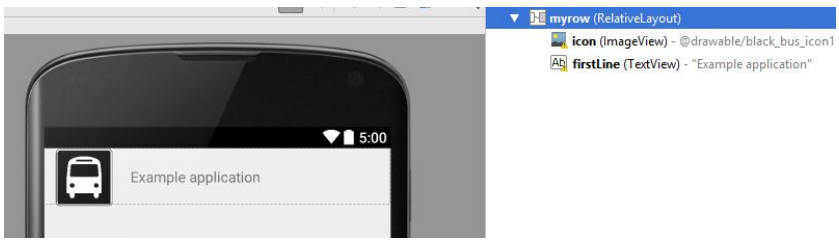


Figure 4.7: Graphical representation of the bus row layout

Taking into account that the entities being represented are always bus routeVariants, the imageView (icon) can be preloaded through the XML file because it will never need dynamic manipulation in the code (once the parent ViewGroup is inflated on a ListView row). This way each time the getView() method of the adapter is called only once is the FindViewById() method called. The calls to getItemViewType() and ViewTypeCount() are reused form the ArrayAdapter superclass, notice that even though each row is customized to show an icon and a text description, the AdapterView hosted by the ListFragment assumes only a single ViewGroup and all calls to getItemViewType() return the same value, as well as ViewTypeCount() which always returns the value 1 like in the ArrayAdapter superclass.

4.2.9 MapFragment (and Drawing Overlays)

The rightmost fragment hosted by the home screen Activity is the part of the application which can offer context information if the user knows nothing of his whereabouts.

It is a fragment composed by:

- MapView
  - MyLocationNewOverlay
  - CompassOverlay
  - MinimapOverlay
  - ScaleBarOverlay
  - RotationGestureOverlay
- ResourceProxy

The use of these classes implies the import of OSMdroid libraries a topic further explored in section 3.2.3. The fragment's interface is defined by the presence of the above elements, the MapView is inflated upon fragment layout creation, followed by the instantiation of each of the Overlays listed above. Some of these overlays require enabling and set-up such is the case with MiniMapOverlay, MyLocationNewOverlay and ScaleBarOverlay as well as the tile scaling to the correct DPI. The next operation is a step that will be partially changed, the request to the Location provider happens as it is

supposed to resulting in the MapView centering its position in the location returned by the location provider and drawing the MyLocationNewOverlay on top of the MapView. This is where one step is missing, the request for all bus stops within a certain radius of that position. In its current version due to server-side maintenance circumstances outside of the author's control that specific request cannot be made. An alternative way to demonstrate the feature was built the following way: the author looked for the coordinates of some of the bus stops which serve ISEL and hard-coded this set of coordinates in the application so that while external problems can't be overcome, the final outlook can still be simulated in the prototype. This is done by instantiating an ItemizedIconOverlay object, which will work as a holder layer for the OverlayItems which will be representing individual bus stops on top of the MapView. Each of the OverlayItems will then be associated with the ItemizedIconOverlay and an icon will be attributed to this Overlay, which will then be added to the MapView. MapView needs to be refreshed for the changes to become visible. With the objective of saving resources such as power, some of the fragment life-cycle callback methods were overridden so that they would only use resource-hungry hardware when the map fragment comes into focus, an example would be the calls to disableCompass() and disableMylocation() methods.

4.2.10 Activity as a host to a ViewPager Container

After examining the three swipe-able fragments in the home screen, one android architecture pattern becomes noticeable: the HomeScreenActivity does not hold any feature specific logic, it works as a context provider component to the ViewPager it holds. The ViewPager relays the activityContext to the hosted fragments. Communication between the activity and the hosted fragments is held through callback-interfaces developed for that specific purpose.

4.2.11 BusStop Status Implementation and ArrayAdapter use

The layout of this activity was visually modeled together with the home screen favourite buses so that the user can be familiarized with the application functionality without much effort. The layout follows the logic of one top section that identifies the purpose of the activity through several Views (TextViews and ImageViews) which in addition to

a ViewPager container which hosts a single fragment listing the times, and routes of all the currently expected buses.

The homogeneity of each row allows the customization of the ViewGroup layout to be almost completely done in static XML, the exception being the changing of waiting times and bus RouteVariant Ids.

4.2.12 Route Spine Activity Implementation and ArrayAdapter use

This activity has the objective of graphically representing the bus stations which compose a specific RouteVariant. The graphical representation chosen was a synoptic graph similar to the one a passenger might find in any given bus stop around the world, differentiating between starting, intermediate and terminal bus stops.

Taking into account the changing nature of a route's composition, and the objective of minimizing network traffic, a decision was made that these synoptic graphs would have to be dynamically drawn by the application and not downloaded as static images. The implementation of this feature benefited from the knowledge gained during the heavy customization of layouts needed by other activities. The activity itself is a context provider for a ViewPager hosting only one fragment. This single fragment makes use of a modified ArrayAdapter which receives the RouteVariant's composition as a dataset and dynamically builds a listView composed by customized rows which themselves are ViewGroups but with only two Views each: an ImageView and a TextView. The TextView obviously shows the name of the bus stop but the ImageView has a little more to it than the obvious. The objective is to build a synoptic representation of the line, through row customization, so differentiation between start points, intermediate stops and terminals is needed. In order to avoid previously described costly processing in the getView() method the layout is originally inflated as if all the dataset stops were intermediate stops, and only in the case of the first and last station more expensive processing performed.

Row Customization and ArrayAdapter use

The rows corresponding to the bus entity from the model were customized in order to obtain a simplified representation similar to what a passenger would get when waiting

for the bus at the bus stop: an icon referencing a bus, an identifier for the bus number and a estimated time of arrival identified by a clock icon (see Figure 4.8), the user can use the real world knowledge he has and instead of looking at a static table at the bus stop he sees real time information on the application.

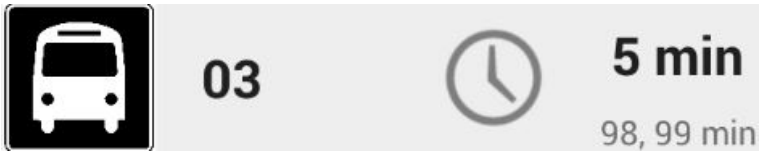


Figure 4.8: Graphical representation of the customized row in BusStopStatus Adapter-View

This row layout is dynamically changed through the use of an extension to the ArrayAdapter provided by the android framework, performance mechanisms are the ones implemented by the superclass. The inflation mechanism inside the getView() method was left in its simplest form: one inflation per item per getView() call. The reason for this implementation choice was that the number of active buses scheduled to go by a certain bus stop will never be sufficient as to allow the user to trigger a massive call to getView() by quickly scrolling, in most cases the number of expected buses will not even fill the screen which will limit the number of calls of getView() to just 1, the first (and mandatory) inflation. The implementation of this activity also follows the activity as a context provider for a container, despite only hosting one fragment.

4.3 Business logic

The business logic in the application was planned to fit the needs of a real world public transportation business, while simultaneously simplifying whenever it was possible to do so. One of such examples was the model and the equivalent factories needed to obtain the correct object instances. Most instances of the model and then used by the adapters referenced in past sections as parts of the datasets feeding the adaptation layer sitting between the business model / business logic and the presentation layer.

4.3.1 Model

The XTraN Passenger system offers a comprehensive set of features and entities describing real world public transportation business. The system was designed to serve the needs of fleet owners, managers and planners which includes much a much larger amount of meta-data and results in a respectively more complex model than the one that needs to be publicly exposed or that is needed for the scope of a public domain application. For the purpose of building this prototype a task of slimming down the existing model was performed (the bus commuter does not need an app entity representing each bus driver for example). In order to fulfill the the needs of the prototype's use cases the following model and entities are used:

- Bus
- BusStop
- City
- Route
- RouteVariant
- TimeEstimation

These entities are instantiated by the corresponding factories which make an optimized use of network requests through the addition of an intermediate network layer (using Android Volley).

4.3.2 Factories and Volley

The current form of the distributed system: a RESTful API, is extremely favourable to the use of JSON based constructors. But the foundation for these "distributed constructors" needs to be solid enough to work under a large range of circumstances and resilient to generalized failure (not propagating failure to other parts of the application) if possible. The networking library of choice for the implementation of this prototype was Android Volley for the reasons described in section 3.2.6. The factory architectural pattern used together with Volley for the production of entity describing objects in

the application. Some of the model entities listed in section 4.3.1 have an associated factory capable of retrieving the JSON data needed for the respective constructor. The following factories were implemented:

- BusStopListFactory
- RouteListFactory
- RouteVariantListFactory
- TimeEstimationListFactory

Everything went the way it was advertised (or close enough) for implementation and testing of the first three factories. They were designed for asynchronous JSON requests the following way given in example with the BusStopListFactory. The class implements interfaces respective to successful response callbacks and errors. The requests go through a singleton static instance of a queue exclusive to these requests and the JSON formatted responses come through the implemented successful callbacks method. With the dataset client-side the factory would then synchronize with the Activity UI through a refresher callback back in the main thread. This is the best case scenario, a virtually perfect behaviour, only differing from the advertised for two (important) details:

- Authentication
- Connection state management

These two factors derive from the same origin: the simplicity offered by the Volley library is often broken by the technical needs that pushed other solutions to less simple implementations. In the case of authentication Google does not provide a standard way to deal with the issue, the solution being several different approaches coming from different sources depending on the authentication layer implemented server-side. In this case the solution better adapted to the authentication method implemented by the API was to implement an authentication factory which would also deal with the second problem: state management. As a consequence of the non-existent native support of an adequate type of authentication mechanism, the default behaviour of this networking layer was changed: the default form of state management for most libraries lies in

the http client instance used for the requests. Volley instantiates as many clients as needed in an independent and transparent manner (as a default behaviour without the developer explicitly coding it) and does not sync any kind of connection state, despite the advanced caching mechanism. Here the true nature of the problem surfaces it is not as “seamless” as advertised to use volley in real world products, the overrides and customizations needed are a considerable amount of work, and every time the technical requirements of real world engineering come into play, the developer has to dive into the underlying layers of the volley library with insufficient documentation and frequently with restrictions which would not apply to other networking libraries. Which serves as an introduction to a worse case scenario where on top of the extra work already mentioned one could have a situation where the JSON payload received was not enough, a situation where some meta data from the connection were also relevant. This also implies overriding code in volley. In an architecture model which is not truly event-oriented the following problem can happen: a certain screen that shows data from heterogeneous sources can implement volley “listeners” that will not be specifically registered for events triggering erroneous behaviour from listeners and once more the solution became the re-implementation methods. The result became an adaptation networking layer on top of the native android networking libraries.

4.4 Maps

This section makes a more in depth analysis of the Maps module and the different forms of acquiring a user location fix with different constraints.

Location Providers

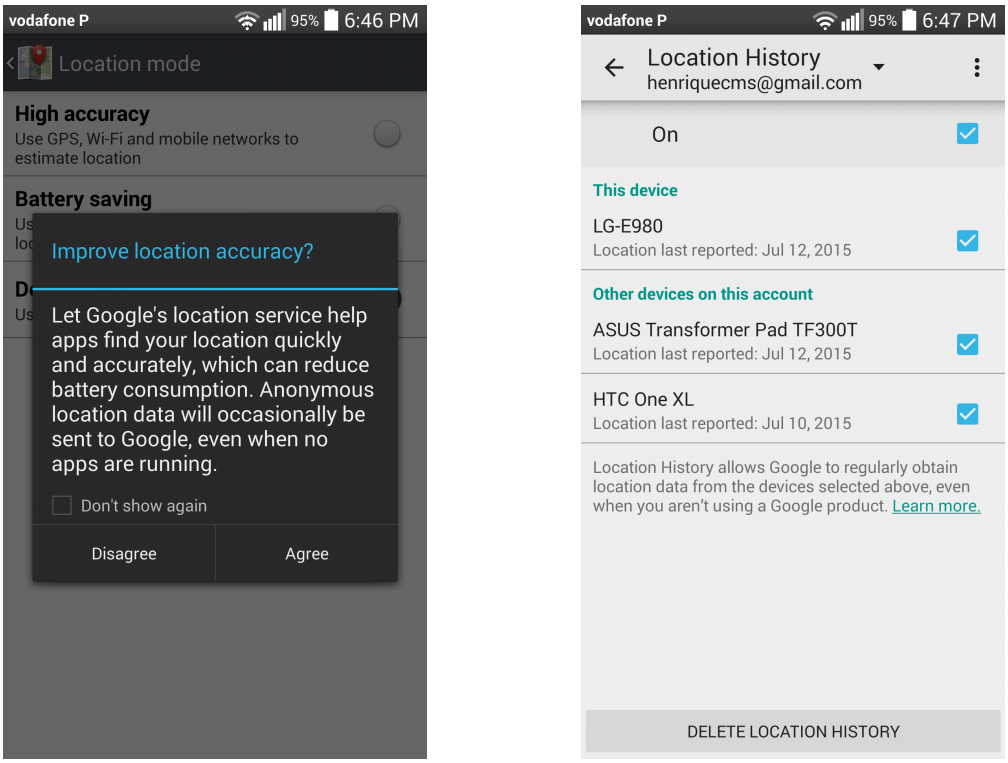
As discussed in section 3.2.3 different situations with different constraints such as low power reserves, indoor operation or outdoor operation in an urban canyon environment require different location providers. The location providers offered are: GPS provider, Network provider with or without WiFi access point data, and the Passive provider. The developer must evaluate the constraints and decide on which “criteria” to use and decide which location provider is best for his purposes. This is a matter of relevance to this prototype: ideally a user would should be able to use the context-aware features

of the application even if he is still at home (inside a building).

As described in the android developers documentation GPS provider is the most accurate, it gets the most precise location fix, but it needs a clear line to the satellites as well as a longer time to get a first fix (TTFF) and it implies higher power consumption. It requires the permission android.permission.ACCESS_FINE_LOCATION. This provider will be useless in most cases where the user is not outside, but if he is, the GPS provider but can get approximately 2-5 m accuracy. In the context of this application the GPS provider would be the most indicated for situations where the user needs to know his precise location, such as when looking for the closest bus stops. An alternative would be to use the Network provider which uses assisted GPS (AGPS) cell tower Id and WiFi MAC id (if using wifi assisted Google location services). This provider will get a faster initial fix by using AGPS and still using the GPS chip, but it drastically reduces power consumption while keeping it's accuracy at a high level. Moreover, this provider includes the possibility of using WiFi to further assist in getting a location fix. In populated areas with an increased amount of WiFi access points and cell towers, it is possible to get an acceptable even with the GPS disabled, this works because Google location services record user location data crowd-sourcing WiFi MAC addresses from all devices, an example can be seen in Figure 4.9.

An interesting comparison experiment by M.Fasel in [14] regarding this matter shows the difference in performance when using different providers.

The comparison shows some of the advantages and shortcomings of both providers, in Figure 4.10 the "accuracy circle" by the GPS provider is typically smaller (i.e. more accurate) than the Network provider counterpart. A few remarks which are relevant to our prototype application must be made from this experiment. This experiment was performed in Melbourne in what can be described as an "urban canyon" which is the typical city scenario in which the prototype is meant to be used, this is a relevant point to justify that in Figure 4.10 even though there is a better accuracy the position is sometimes drawn inside the buildings instead of on top of the black line which represents the actual route taken. If the received GPS signal is reflected of buildings instead of a direct line from the satellite it will affect the triangulation of the position frequently resulting the user being positioned inside of the building instead of the sidewalk, an example would be the point marked as 1 in Figure 4.10. If the prototype uses only



(a) Crowd-sourced location data 1 (b) Crowd-sourced location data 2

Figure 4.9: Crowd-Sourcing user location data: WiFi assisted location service

the GPS provider such inaccuracies are to be expected affecting the search of closest bus stops for example. Another problem in using just the more accurate GPS provider is illustrated by point 2, the researcher reported losing signal going through a narrow street and only recovering it near the end of the journey. This would pose a problem for the typical user of our application as well.

In a third experiment the researcher used the network provider with a very high accuracy which can be observed in Figure 4.12. This change comes from the WiFi assisted Google Location Services which uses the collected MAC address information gathered by all the users of the service. Figure 4.12 also illustrates a problematic occurrence when one access point was moved from location 1 to location 2, since location 1 was already mapped by many users, the access point triggered false positive fixes from the moment it was moved. This could be problematic for the prototype of our application if it was not such a rare occurrence.

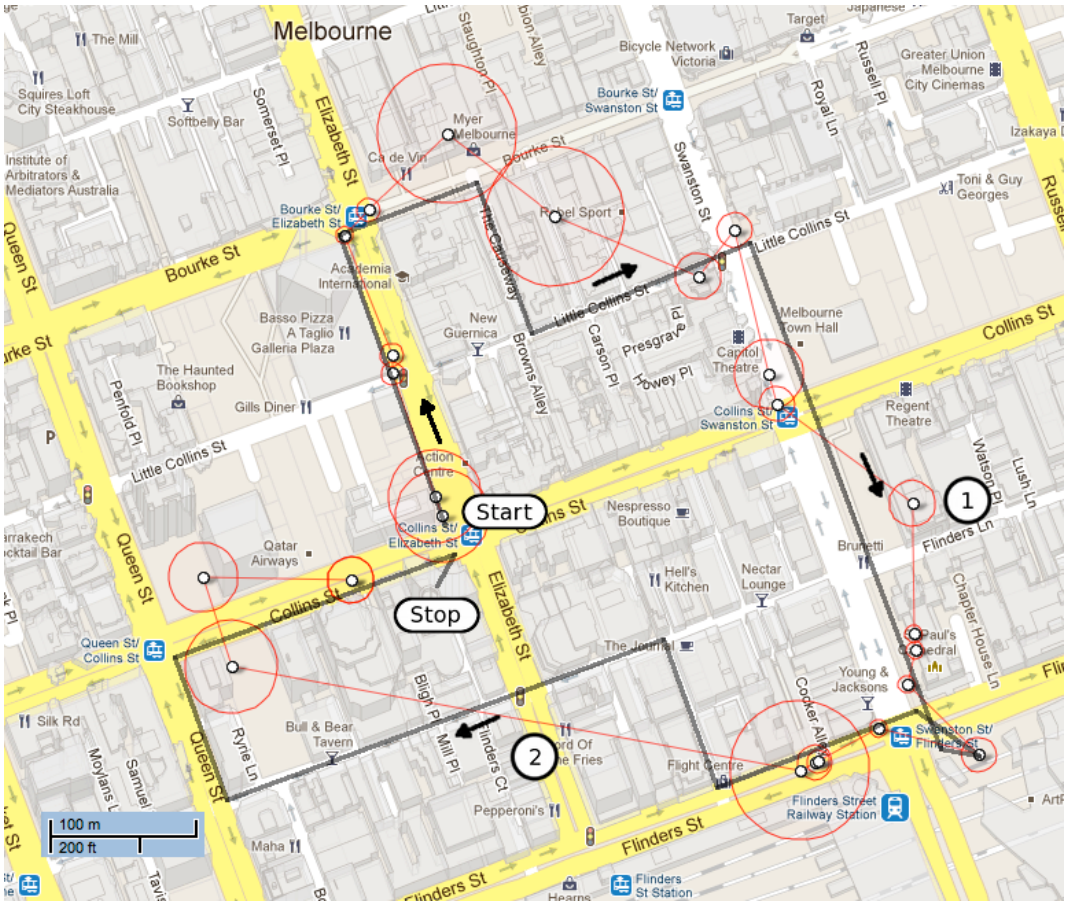


Figure 4.10: Comparison of Android Location Providers: GPS [14]

4.4.1 Strategies for efficient Location

In regard to power saving and user location strategies there are several parameters which can be used in our favor in order to better achieve the goals of the prototype while minimizing setbacks that would be prejudicial in a production environment. While addressing the challenges mentioned in section 4.4 some of the parameters and requirements which can be leveraged are:

- **location accuracy vs power saving**
- **Time to first fix (TTFF) vs location accuracy**
- **Request update location speed vs power saving**
- **Is indoor operation required?**
- **Is the application most likely to be used in a densely populated area?**

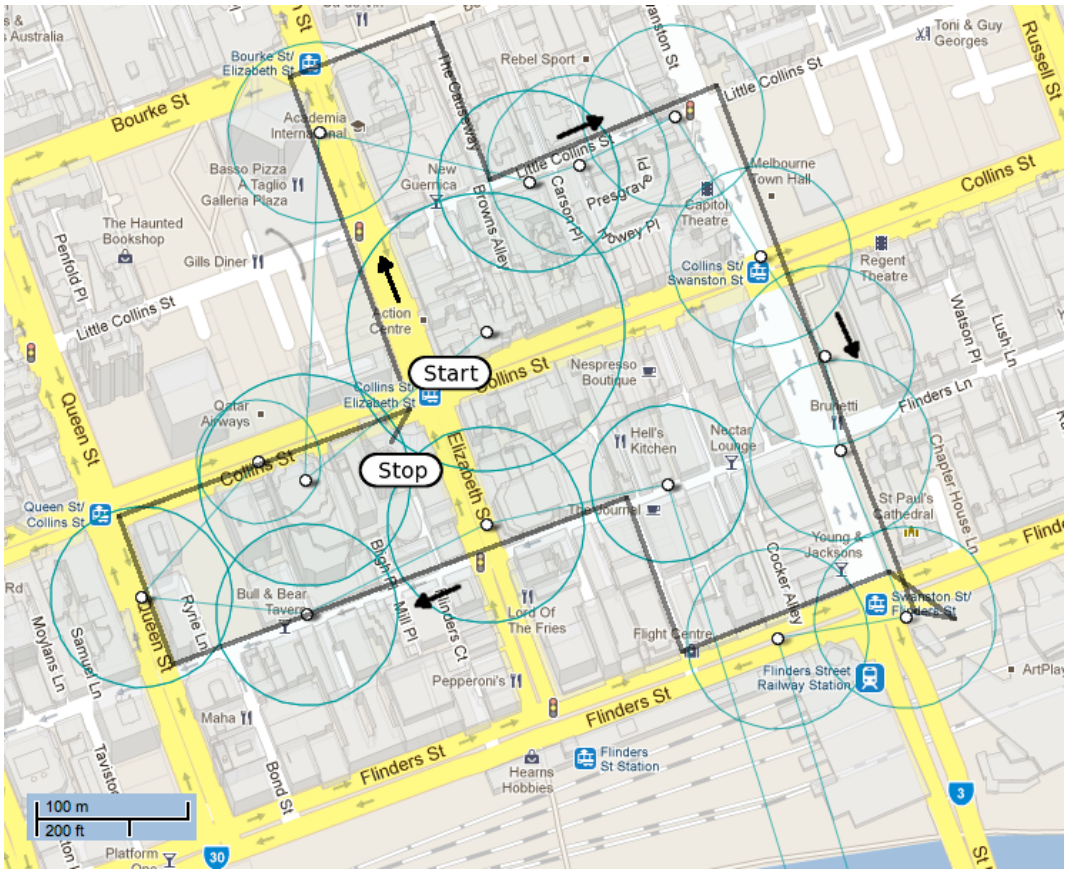


Figure 4.11: Comparison of Android Location Providers: Network [14]

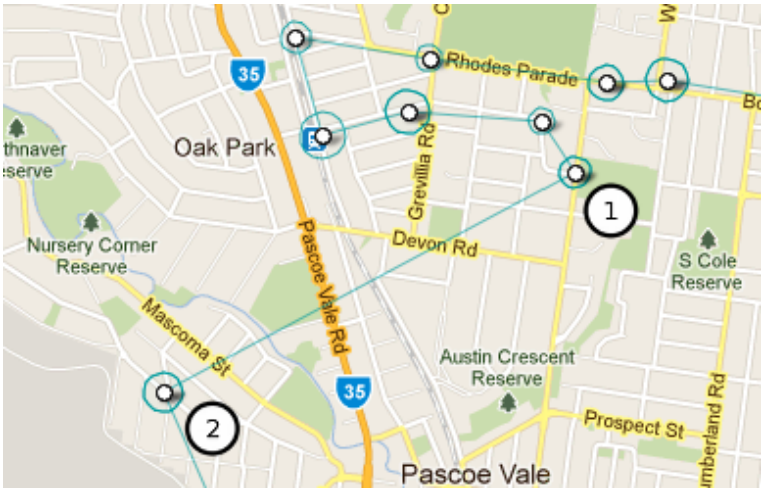
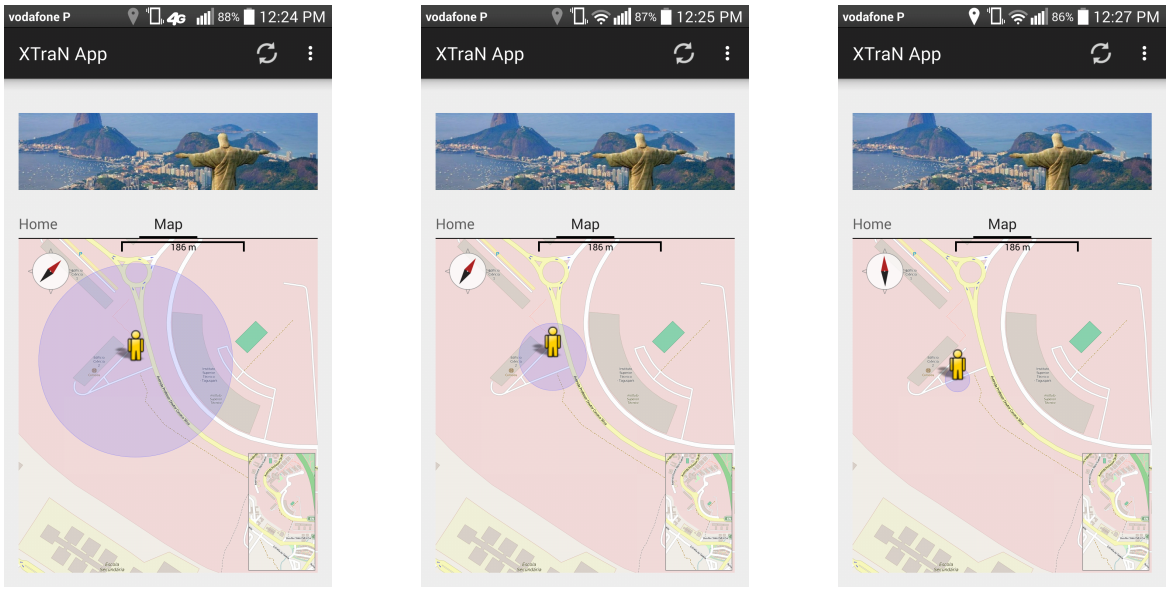


Figure 4.12: Comparison of Android Location Providers: Network (with WiFi assisted Google Location Services [14]

The calibration of the criteria used for deciding on the location provider used may differ for different situations. In the case of our prototype the criteria has been to prioritize speed to acquire a location fix (even if at the initial expense of accuracy minimizing TTFF) this is done by getting an initial fix from the Network provider. Reflecting on

the first topic of the list above, leveraging location accuracy versus power saving, it becomes clear that obtaining the first fix will consume less power by using the mobile network instead of waiting for an accurate fix from the GPS chip. An important requirement for a mobile transports application is that it should work in most situations, the user may choose to plan his trip from the street but also from home, so there is a constraint that the prototype application should work indoors. The dual use of the Network provider first then the GPS provider, will address this requirement. Sacrificing accuracy as a trade off for indoor operation and a short waiting time for a fix may appear as a bad optimization, but not all factors are yet accounted for. It is a fair assumption that if the user is using this public transports application he finds himself in the same urban environment where the transports operate. This will introduce two factors: the possibility of the user finding himself in a street resembling an urban canyon situation, and a possibility that even if he is not at home, there are WiFi access points in the vicinity. An urban canyon situation will minimize the benefits that would come from using the GPS provider, by getting wrong location readings due to GPS signal reflection on the surrounding buildings, and an even longer waiting time for the first fix. On the other hand, being a densely populated area the Network provider will maximize the benefits of the surrounding WiFi access points increasing the accuracy of the Network provider by a large factor.

Comparing the results obtained with the prototype in Figure 4.13 when forcing the application to use only the Network provider and disabling WiFi leaving the cell tower ID to pinpoint the user's current location. The accuracy was the worst result of the three measurements (figure 4.13(a)) as expected but was instantaneous ($<$ 1s) and with minimal battery use. After enabling WiFi, the location fix was much sharper as per the results in figure 4.13(b). The time to first fix was almost instantaneous ($<$ 5s) and consumption was kept at a low level by not using the GPS module. The last test used the deployed settings of the prototype without forcing any of the providers on/off. By analyzing figure 4.13(c) a few conclusions can be drawn. The TTFF increased to 1̃min even with the assistance of the networking module (AGPS), but the accuracy also increased by a large factor as did the power consumption due to the usage of the GSP chip. An interesting point to be made is that the actual location where the experiment was performed is outside of the circle supposedly displaying the accuracy



(a) Network provider w/cell tower ID

(b) Network provider w/cell tower ID and WiFi

(c) GPS provider

Figure 4.13: Prototype operating with different location providers, measured from the same location

of the measurements. This fact can be explained by the experiment being made on a high floor balcony, with half of expected satellites covered by the building a multipath effect can be expected with the device receiving GPS signal reflected off the ground where the map is getting the fix [27].

4.4.2 Addition of a layer on top of OSM Mapnik

The addition of a layer on top of open street maps Mapnik (the slippy map implementation used) is used to represent data such as bus stops, adapting the typical unreadability of textual geographical data into human readable info-graphics on top of the map layer.

4.5 Networking

This section analyzes the implementation of the networking layer classes which assure the communication of the prototype with the server-side XTraN infrastructure. The available technological options are described in depth with their pros and cons for each situation and the choices made are also justified here. The following options were considered to fulfill the communications requirements mentioned in section 3.2.6:

- Using Java concurrency utils such as ThreadPools/Executors.
- Native http client performing requests on an Android AsyncTask.
- Native http client performing requests on an Android AsyncTaskLoader.
- Integrating Android Volley modified listeners for use with model entity factories.

The use of Java threads appealing as it may seem from the familiarity point of view has a few drawbacks when used in Android: Synchronization issues on processing conclusion become a problem due to the nonexistent native mechanism of posting and synchronizing the result of parallel processing, not to mention the issue of “orphan threads” when the parent context disappears. Using AsyncTask is the default recommended way to escape the single thread model whenever necessary in Android. Up until recently the standard procedure to perform http requests involved the creation of AsyncTasks and some implementation of http client which would perform the requests in a parallel thread avoiding the frozen UI while waiting for the thread to conclude its non-UI related work. AsyncTask natively includes the creation of background threads and a specific method for syncing with the main UI thread, for these reasons it became the long time first choice of Google and many developers for performing asynchronous processing in android. One of its disadvantages though is still the handling of android configuration changes. This problem was finally solved by the appearance of LoaderManager and the AsyncTaskLoader which provided the tools to deal with these setbacks, although simplicity and re-usability is not one of its strong points. Rethinking the challenge and taking into account that what is needed to solve this problem is a simple, reliable form of performing asynchronous http requests leads to looking into one of Google’s new networking library.

Android volley is the only framework native library capable of autonomously dealing with http requests without hindering the main interface thread. It was built to become Google's official substitute for Apache's http client, while making it asynchronous giving it valuable technical capabilities more difficultly obtained before. After the release of this new library, it became possible to natively prioritize and requests, the caching mechanisms already available in Java clients became more advanced with coherence between different instances of the underlying http client while keeping the raw JSON support which was one of the main uses of the old clients.

Ideally, volley would turn into a possibility the following: the author would implement a set of factories for the needed model entities which would turn the JSON responses from http request into objects of the respective classes, making it look seamless to start with an http request and finish with the object you need. The reality of the situation is slightly different as will be explained in section 4.3.2.

4.6 Persistence

4.6.1 SharedPreferences

The SharedPreferences API is perfect for the storage of small key-value pairs [5]. The framework allows for the definition and use of several of these files making the required distinctions on access policies for each of them, and together with the API provided management methods for primitive data types, SharedPreferences becomes the tool of choice for saving user preferences and favourites like the name implies.

4.6.2 File storage

The choice for a file based persistent storage should be made if the amount of data is quite large. File objects are written in byte ordered sequence, using the java.io API. As an example, the need to locally store multimedia files would justify the use of file based storage in android [4]. Another choice to be made in case of file based storage is on whether to use internal or external file storage. The two main factors to consider here are: availability, the user can remove external storage (in a lot of device models) making it impossible for the application to access it. The other factor that should be

taken into account here is security, the external storage is by definition “world-readable” which is a deal-breaker for many applications where data confidentiality matters.

4.6.3 SQLite Database

The use of a database, even if only a small local SQLite database requires more than the two previous options. SQLite databases are used with structured data sets and one particular example in Android is the contacts database. The definition of a database schema should be described in a contract class including the tables, columns and URIs [6].

Given the needs of the prototype, and the technical specifications of the three solutions available, two options hold the most advantages: SharedPreferences for saving all types of favourites and other user preferences, and a local SQLite database for a large set of almost static timetables for basic offline operation[1].

[1] SQLite persistent storage has been taken into account during the project phase of this work but was not implemented. For exterior reasons to the author it was not yet possible to request public transportation static timetables.

Chapter 5

Evaluation

This chapter will evaluate the specific goals meant to be met during the course of this project. This chapter will also describe the methodology followed by the tests performed in order to evaluate prototype functionality, as well as real world results' validation and the prototype's capability to fulfill the proposed base use cases.

5.1 Methodology

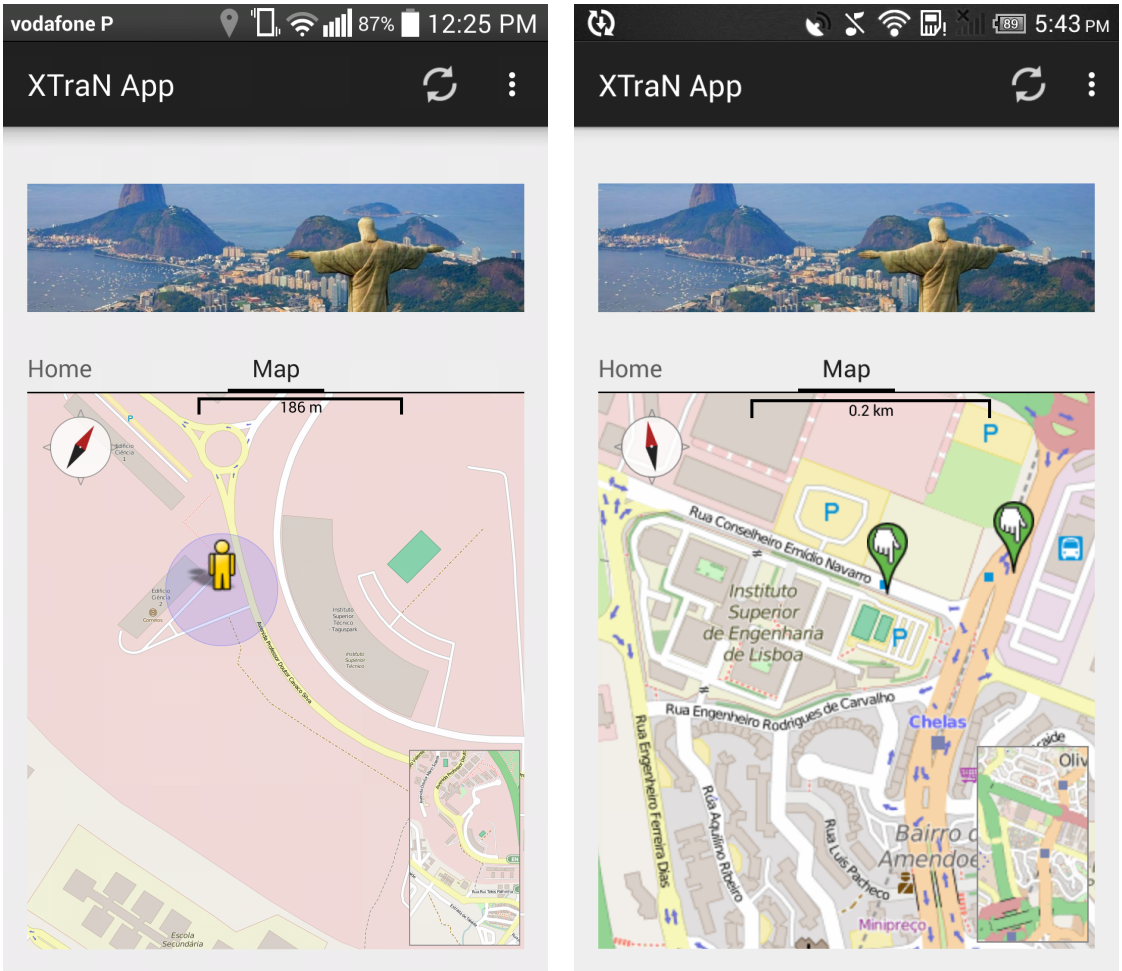
The methodology followed in the validation of results obtained with the prototype consists in two phases. The first is a detailed walk-through of the steps in each use case to validate the operability of the prototype. Phase two is designed validate the results obtained by the prototype in a commercially deployed setting of the XTraN Passenger system. The company chosen for these tests is "Auto Lotação Inga" [28] in Rio de Janeiro and Niteroi, Brazil. Section 5.3 validates the data presented to the user by matching it against data as seen in Inga's operations' center in Brazil.

5.2 Use case tests

In this section each of the use cases will be walked through for validation of the prototype's operability.

5.2.1 Testing use case 1: Current position check

By testing this use case the application shall be tested for the capability to offer the user information on his whereabouts. The data presented will provide graphical location information in a map (Figure 5.1). The first position check was performed in Tecmic's offices near IST's Tagus Park campus showing the user location and his surroundings (Figure 5.1(a)). Despite the external server-side problems outside the control of the author, the closest bus stops will be shown in the map interface when the API is fully working. A working example is shown in Figure 5.1(b) with the closest bus stops to ISEL being drawn using the same mechanism (Open Overlays) that will draw all locations upon local bus stops location request.



(a) Crowd-sourced location data 1 (b) Crowd-sourced location data 2

Figure 5.1: Testing use case - Finding Current Location

5.2.2 Testing use case 2: Route List Check

This is the use case that shows the user the complete list of RouteVariants provided by the public transport operator. The test shows that the user can get the prototype to show this complete list with one swipe, a partial list is shown in Figure 5.2.

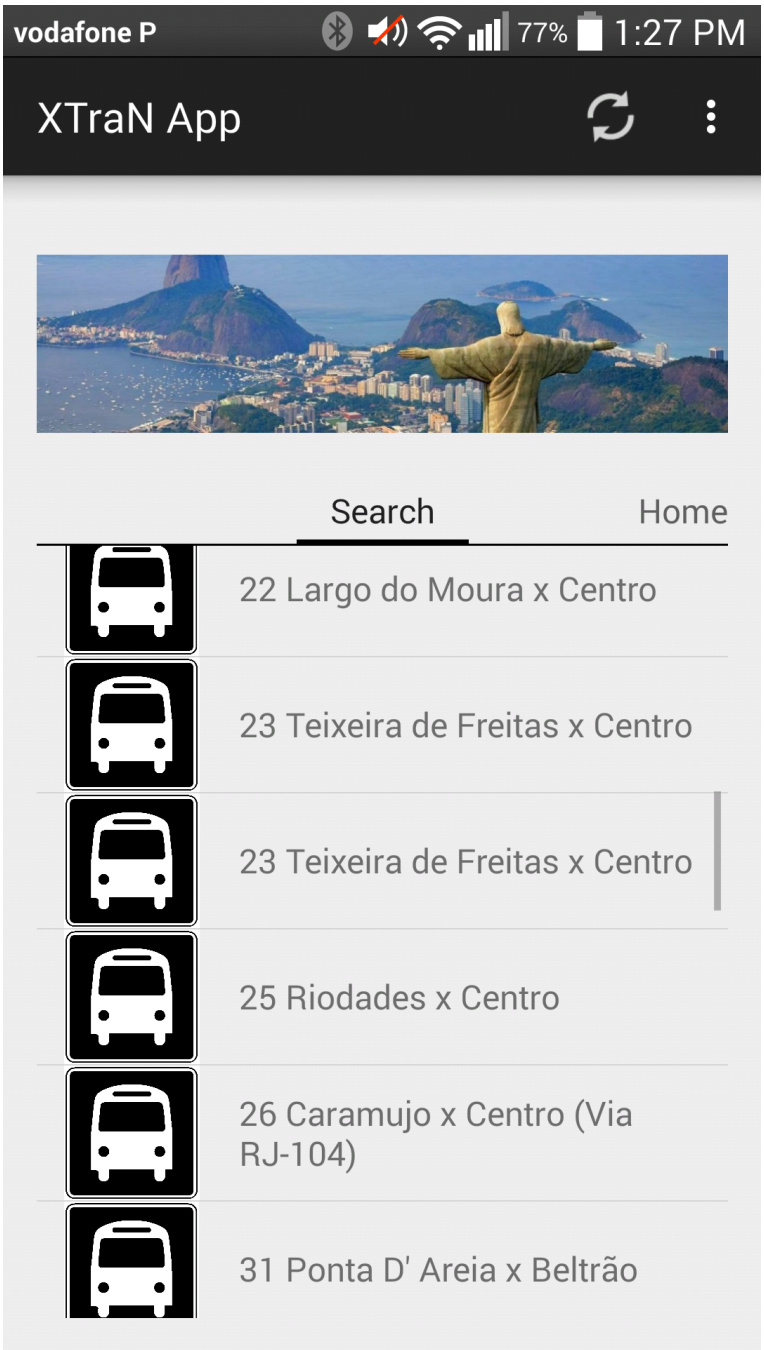


Figure 5.2: Testing use case - Route List Check

5.2.3 Testing use case 3: Route Map Check

This test, uses test 5.2.2 as a starting point. As Figure 5.3 shows, with a single click the prototype shows a synoptic representation of all of the routeVariant's bus stops, with only another swipe the user can check the composition of the return way of the routeVariant which is frequently composed of different stops due to one way streets and other transit regulations.

Figure 5.3: Testing use case - Route Map Check

5.2.4 Testing use case 4: Stop Status Check

Starting where test 5.2.3 left off, this prototype screen shows an interactive real-time version of the bus stop timetable. Showing how long is the expected waiting time for the next bus, and also identifying the different routeVariants currently passing through that stop. This feature is working as expected, but if looking for anomaly cases, tests show that the prototype also shows buses which are registered to as “not yet in voyage”. These vehicles are shown with a real position relative to normal working buses but always with a 0 waiting time associated to the bus (a real case of one of these anomalies can be seen in Figure 5.4). From this test it is also observable that the layout and adapter are ready to accept multiple buses registered to a single routeVariant, but as of the current date the API was not optimized for this UX usability feature.

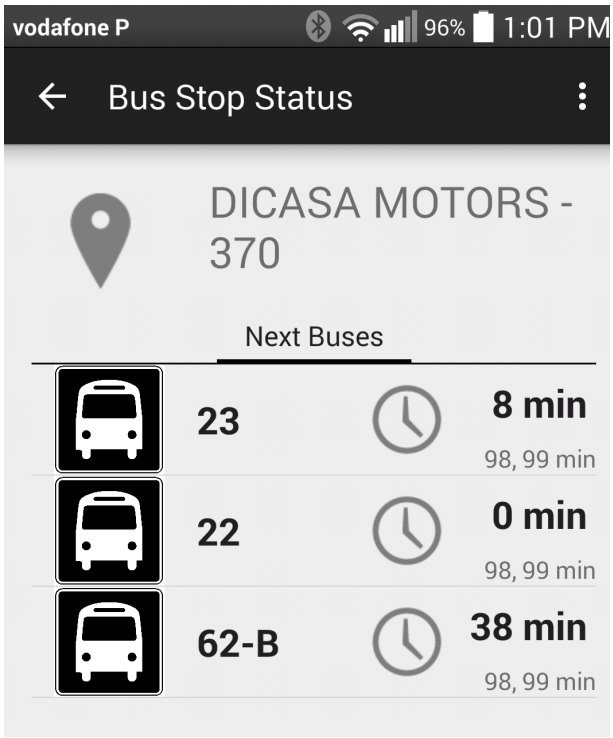


Figure 5.4: Testing use case - Stop Status Check

5.2.5 Testing use case 5: Frequent Bus and Routes Check

This test represents the predicted most used feature of the application - the user simply opens the app and is quickly shown real-time data on his most frequently used buses, his favourite bus stops and public transportation lines, all in a single scrollable screen. The test shows that the performance mechanisms implemented in each of the different adapters work almost flawlessly allowing the user to scroll customized rows at any speed that he likes. Even though there is a high degree of customization, there is very little to no noticeable lag with the exception of some transitions between tabs (Home Screen – Map Screen – Search Route screen). This is a point to improve on but should not hinder the usability of the application.

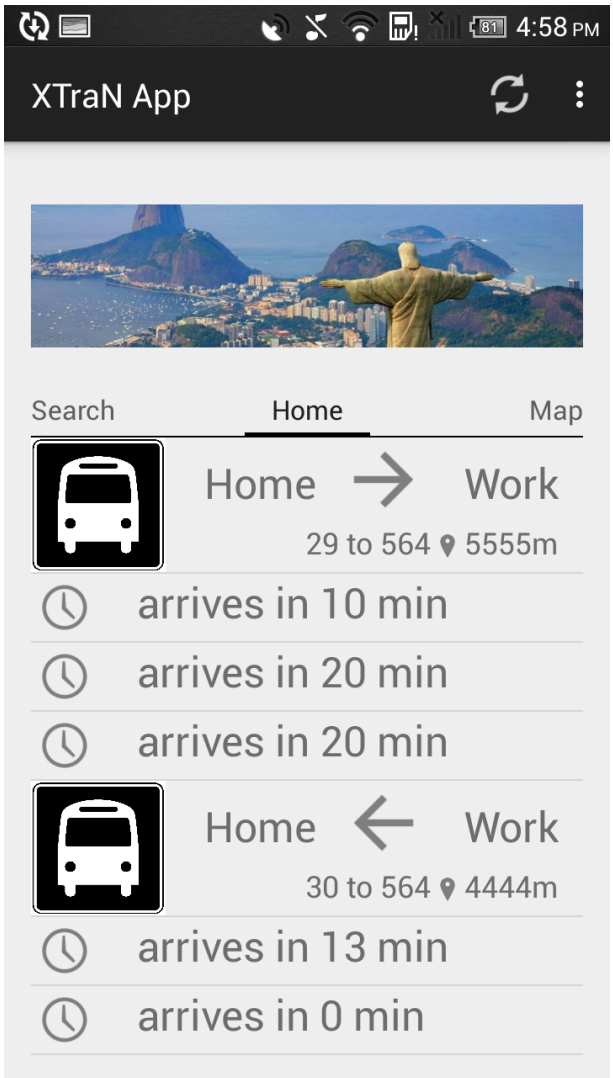


Figure 5.5: Testing use case - Frequent Bus and Routes Check

5.2.6 Testing use case 6: Frequent or Favourite selection

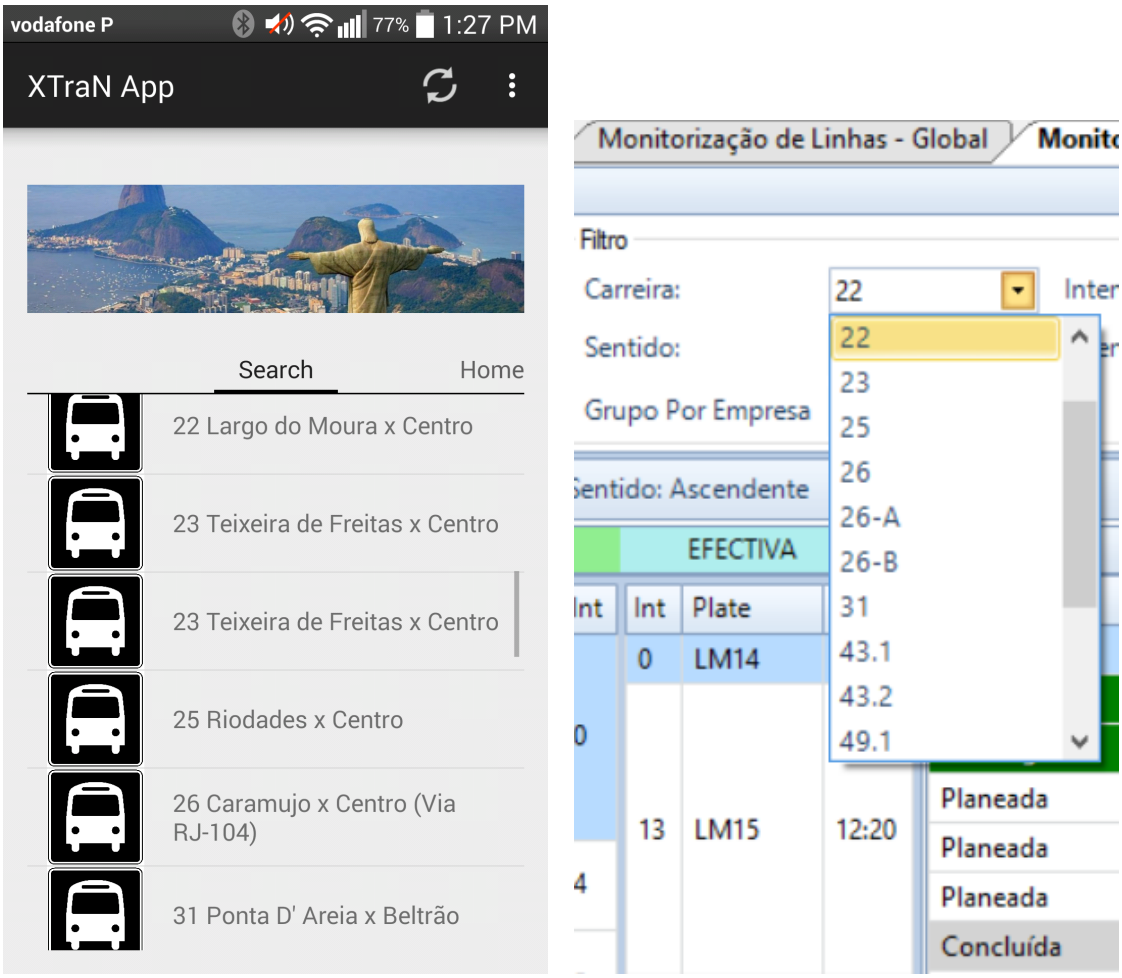
This test performs an action necessary for the customization of the home screen, and the use of persistent data in the form of SharedPreferences (explained in section 4.6.1). For the prototype the preferences are hard-coded but the SharedPreferences mechanism is in use and working.

5.3 Real world tests

The XTraN passenger system is deployed in many public transportation companies distributed by several countries including Portugal, but because the infrastructure is continuously being improved the target of this work has focused on one client with one of the most recent infrastructure updates already deployed. As soon as the infrastructure is updated in Lisboa, the same set of tests can be performed here. This set of validation tests used a VPN connection to the Brazilian servers of public transportation company “Auto Lotação Inga” [28] for direct access to the commercially deployed XTraN passenger system. Through this connection real world experimental data was obtained which could confirm or deny the results obtained with the prototype mobile app. This test is the best feasible approach to results’ validation since the buses are running in Brazil. A desirable complementary method which will be applied before production includes users personally testing the prototype “on the ground” in Rio de Janeiro. Each of the features of the application will be verified against results from the fleet management functions of INGA’s operator deployment of XTraN Passenger. These results will be used in order to validate what is being shown in the mobile app.

5.3.1 Validating Route List Check

This section puts side by side the route variant search feature of the mobile application prototype with the XTraN Passenger deployment in “Auto Lotação Inga”. Although the screen shots only show partial lists all the route variants are represented. Upon attentive observation there are routes which are apparently replicated in Figure 5.7(a), this is a result of the prototype showing all available, registered route variants unfiltered which is the case of the results in Figure 5.8.

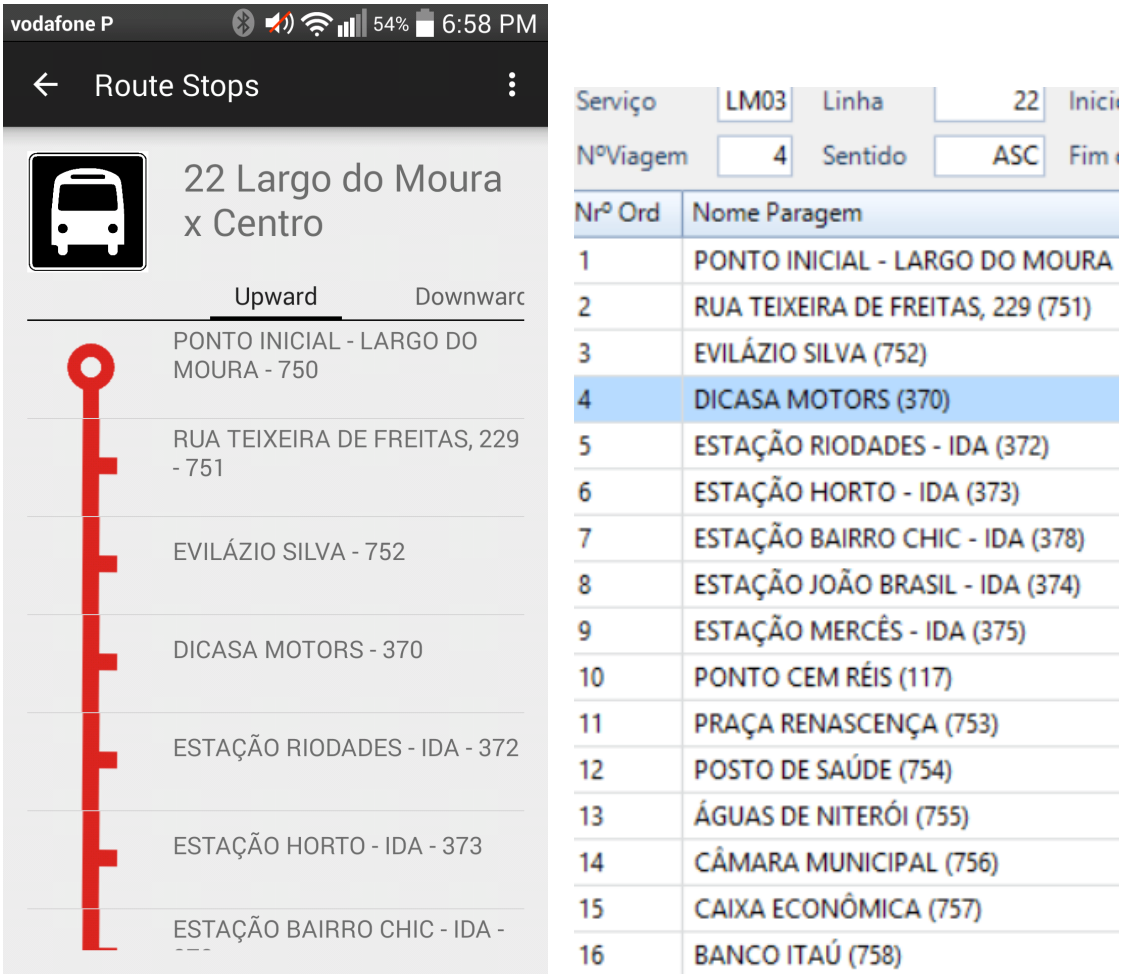


(a) as seen in the mobile app (b) as seen in the operations center in Brazil

Figure 5.6: Validation - Routes offered by the Public transportation provider

5.3.2 Validating Route Map Check

This section puts side by side the synoptic map representation of a bus route variant with its respective counterpart verified in the operations' center software. The list in Figure 5.7(a) is only partially visible but includes all the stops displayed in Figure 5.7(b).

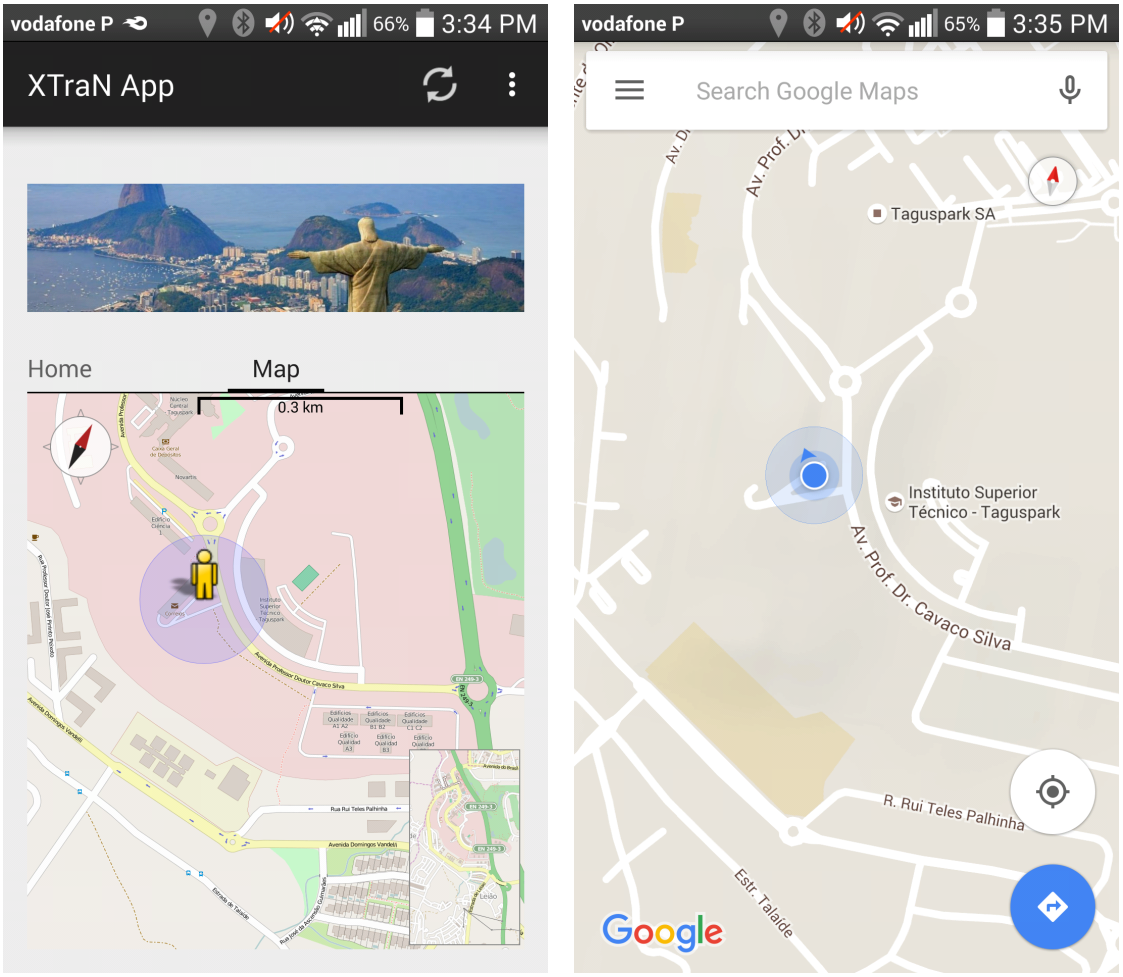


(a) as seen in the mobile app (b) as seen in the operations center in Brazil

Figure 5.7: Validation - Stops constituting a route synoptic map

5.3.3 Validating Current position check

This validation is the only which did not require a remote connection to Inga's VPN. Instead the geographical position validation was performed in Tecmic's offices in Tagus park by comparing the position indication by the maps module in the prototype (Figure 5.8(a)) with the readings made at the same place by the Google Maps application (Figure 5.8(b)).



(a) Testing - Current Location Tecmic Tagus Park campus with Prototype

(b) Testing - Current Location Tecmic Tagus Park campus with GoogleMaps

Figure 5.8: Validation - Prototype (with OSMdroid) VS. GoogleMaps

5.3.4 Validating Stop Status Check

This validation test was performed by accessing the live feed in the new generation public information panel screens at Inga's bus stops (Figure 5.9) and comparing the estimated times with what is shown in the XTraN mobile application prototype (Figure 5.10). All the times from buses which are actively in voyage (estimated times) are shown in the application correctly, furthermore, the prototype shows estimated times for other "in voyage" buses which are are not represented in the physical panel but that also go through the "Moinho Atlantico" bus stop.

5.4 Scenario for final test

What would constitute the most accurate final test is a typical usage scenario where the actor is a frequent user of the public transportation system of his city to commute to work and to occasionally travel to other places he is less familiar with. In the test he uses this application to make the journey effortless. This final test scenario develops as follows:

In the morning before leaving for work the user opens the application to check the waiting times at his home-to-work commute bus stop (set as a "favourite"). As soon as he opens the app, the main tab of the home screen immediately shows him what he wants to know. The user now knows if can stay at home for a few more minutes, if he must leave right now, or even if he can afford to wait for the next bus. During his lunchtime the user wants to go to downtown and try a new restaurant with a coworker. He can skim through the list of available routes using the left tab of the home screen, he can then click to check the synoptic map of the route and the waiting times at the bus stop he is planning to use. At the end of the day instead of going home he is invited for coffee by a friend who gives him a ride to the place. Once the coffee is over he does not know the area very well, so he can use the "map tab" to look for the closest bus stops and the available bus routes. He can then check the waiting times at each bus stop and decide one would serve him best.

This final test not only shows and tests a good part of the prototype's features as well as the added value for the consumer of the public transportation company, making it one very good scenario for testing and demonstration of the prototype.

5.5 Results' Analysis

The conclusions which can be drawn from this data validation experiment are very encouraging. The results revealed by the comparison between the prototype's displayed values and the data available in the fleet management center in Brazil match perfectly and the prototype can offer more relevant information to the user as intended from the start.

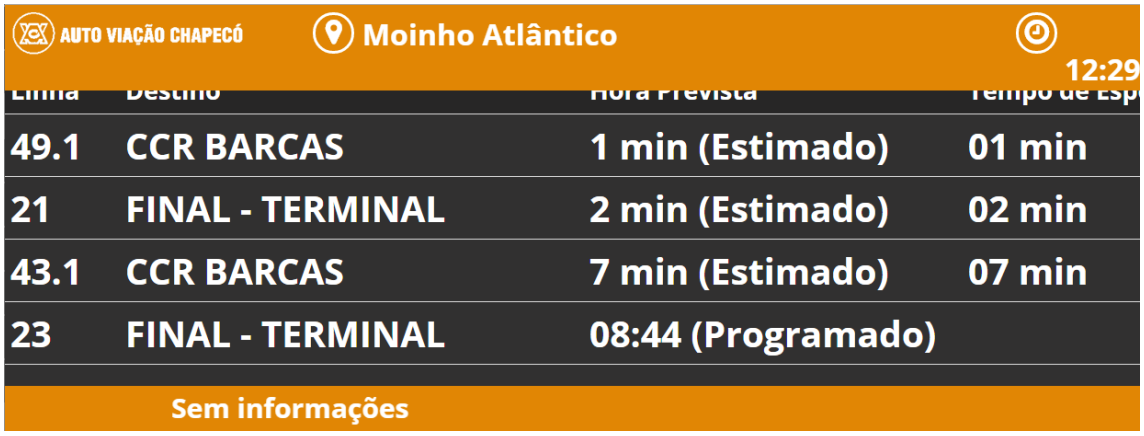


Figure 5.9: Validation - Waiting times at bus stops (as seen in the public information panel at the bus stop in Brazil)

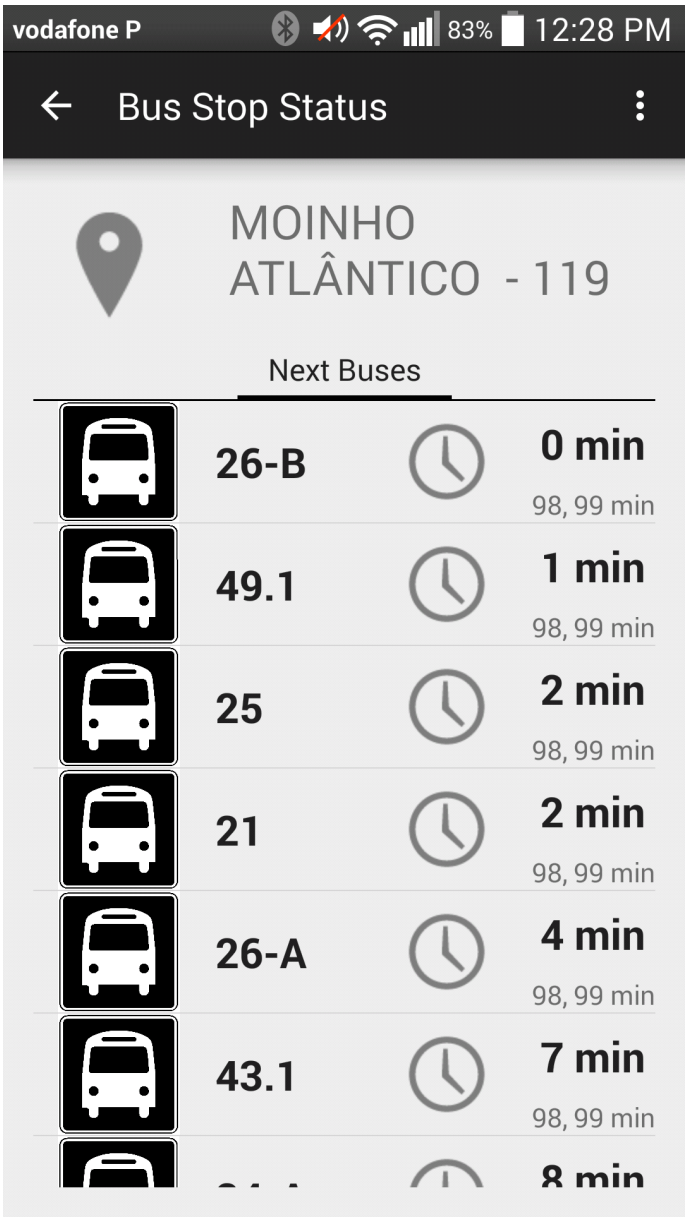


Figure 5.10: Validation - Waiting times at bus stops (as seen in the mobile app)

Page intentionally left blank

Chapter 6

Conclusions and Future Work

6.1 Conclusions

During the course of this work, the intelligent transportation system XTraN Passenger was studied and expanded with a mobile application. That objective required a deep understanding of the workings of the XTraN modular ITS system, and careful planning between the author and the partner company Tecmic so that the final prototype would meet the needs of the company. A software requirements specification was first established for the prototype based both on the features desired by Tecmic and on the results of many discussions with the advisor Professor on what were the most interesting problems to solve during the course of this work with the prototype. Departing from these conclusions on the SRS, research was made on some of the most important applications the industry currently has to offer. The chosen applications were tested for the purpose of comparing not only the available features in these applications but also their degree of usability, graded by user satisfaction rating recorded on the Google Play store. Between all the analyzed applications this significant contribution was taken: the combined outputs of the author's own testing together with the public's praise and criticism of these apps helped to mold the specifics of each of the use cases chosen for the prototype's final version. Another of the objectives of this prototype was to implement the results of these use cases in a way that would translate into a personalized user experience where the user's own input has a significant part in the resulting output of the application. Inputs such as the users preferences on his most used transports, and his current geographical position are relevant factors for the use of the application and tak-

ing the most benefits out of it. Important work was also done in the field of optimization of Android resources and base classes such BaseAdapters and the interaction with their AdapterViews and in the field of customization of existing user interface classes and views. All of these developments resulted in an overall different and improved experience for the user, which will lead to a better future commercial product.

6.2 Future work

Due to the fact that this prototype is meant to be the basis of a commercial product there are many developments needed in different areas of the project.

- **Persistent SQLite Storage**: This is one component to implement before taking the prototype to the production stage. The main idea of this prototype does not improve a lot with the existence of the SQLite database (because there is already persistent storage for basic preferences using android SharedPreferences) but for specific large structured data-sets like the ones required when locally storing static timetables the performance of the SQLite database is a requirement.

- **Static timetable queries**: This is one feature with significant impact for users without data-contracts on their mobile phones. Although limiting and denying most of the benefits of the real-time benefits of the app, some users

- **Trip mode**: Adaptation of the synoptic RouteVariant representation into a real-time progress along the line with warnings of destination stop.

- **Repair closest bus stops query**: This requires a new deploy with the bus company INGA in Rio de Janeiro, Brazil. The partner company Tecmic has already repaired the service but has not yet made a deploy (the new version has not yet been fully tested).

- **Location sharing**: Real time location sharing between users is one way to provide functionality that most other competitors do not offer by mixing it with transports information making it easier for people to set up encounters and meeting points.

- **Social networks integration**: The networking effects of social sharing on the part of users is one of the future ideas suggested by Tecmic both to spread the use of the application and to publicly rate public transportation companies' assiduity.
- **Testing in Lisbon, PT (After local infrastructure update)**: These tests can be performed as soon as infrastructure is updated, and will first be performed using the methodology described in section 5.2, the next round of tests will be aimed at validation following methodology in section 5.3 and finally a real world usage scenario such as the one described in section 5.4.

6.3 Commercial interest and viability

As of September 30th 2015, one commercial client has already shown interest in the prototype and is scheduled to have a first contact and test drive the application during the months of October/November/December depending on the full availability of the Tecmic's API. Upon further developments the final tests can be scheduled immediately following the scenarios in section 5.2. Tecmic's personnel on the ground will perform the first use cases' with the client's deployment of the xtran passenger system. Followed by tests with the client's own personnel with a scenario such as the one in section 5.4 which show the full range of application of the prototype in the real world. From there the client's branding will be added to the opening sequence of the application and the application will be submitted to Google for public release.

References

[1] Carris. Information for passenger apps. URL http://carris.transporteslisboa.pt/pt/informacao-ao-passageiro/. [Online; accessed 5-July-2015].

[2] ESRI. What is gis? URL http://www.esri.com/what-is-gis. [Online; accessed 5-July-2015].

[3] ESRI. An overview of arcweb services, 2004. URL http://downloads.esri.com/support/whitepapers/ws_/853arcweb-services.pdf. [Online; accessed 5-July-2015].

[4] Google. Saving files, . URL http://developer.android.com/training/basics/data-storage/files.html. [Online; accessed 30-August-2015].

[5] Google. Saving key-value sets, . URL http://developer.android.com/training/basics/data-storage/shared-preferences.html. [Online; accessed 30-August-2015].

[6] Google. Saving data in sql databases, . URL http://developer.android.com/training/basics/data-storage/files.html. [Online; accessed 30-August-2015].

[7] Google. Developer api guides - user interface overview, 2014. URL http://developer.android.com/guide/topics/ui/overview. [Online; accessed 21-August-2015].

[8] Google. Developer api guides - listview, 2014. URL http://developer.android.com/guide/topics/ui/layout/listview.html. [Online; accessed 21-August-2015].

[9] Google. Location strategies, 2014. URL http://developer.android.com/guide/topics/location/strategies.html. [Online; accessed 22-August-2015].

[10] IBM. Web services architecture overview - the next stage of evolution for e-business, 2000. URL http://www.ibm.com/developerworks/library/w-ovr/. [Online; accessed 5-July-2015].

[11] IZIMOOVE. Izi carris. URL https://play.google.com/store/apps/details?id=pt.izimoove.carris. [Online; accessed 5-July-2015].

[12] C. Limited. Citymapper - real time transit. URL https://play.google.com/store/apps/details?id=com.citymapper.app.release. [Online; accessed 5-July-2015].

[13] X. Lu. Develop web gis based intelligent transportation application systems with web service technology. In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pages 159–162, June 2006. doi: 10.1109/ITST.2006.288823.

[14] M.Fasel. A good look at android location data, 2011. URL http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/. [Online; accessed 20-August-2015].

[15] Moovit. Moovit: Next bus and train info. URL https://play.google.com/store/apps/details?id=com.tranzmate&hl=en. [Online; accessed 5-July-2015].

[16] OPT. Lisboa move-me. URL https://play.google.com/store/apps/details?id=pt.izimoove.carris. [Online; accessed 5-July-2015].

[17] L. Rocha. Performance tips for android's listview, 2012. URL http://lucasr.org/2012/04/05/performance-tips-for-androids-listview/. [Online; accessed 22-August-2015].

[18] W. Tang and J. Selwood. *Connecting our world : GIS Web services*. ESRI Press, Redlands, Calif., 2003. ISBN 1589480759 (pbk.). 2003013719 .b13787664 (OCoLC)52471578 Winnie Tang and Jan Selwood. col. ill., col. maps ; 23 cm.

[19] Tecmic. Xtran passenger busdvr, . URL http://www.tecmic.pt/wp-content/uploads/2014/05/BusDVR_EN.png. [Online; accessed 5-July-2015].

[20] Tecmic. Carris utiliza o xtran passenger – sistema de ajuda à exploração e informação aos passageiros, . URL http://www.tecmic.pt/carris-utiliza-o-xtran-passenger-sistema-de-ajuda-a-exploracao-e-informacao-aos- [Online; accessed 5-July-2015].

[21] Tecmic. Xtran passenger counter, . URL http://www.tecmic.pt/en/portfolio/xtran-passenger/counter. [Online; accessed 5-July-2015].

[22] Tecmic. Xtran passenger eco-driver, . URL http://www.tecmic.pt/wp-content/uploads/2014/05/Eco_Driver_EN.png. [Online; accessed 5-July-2015].

[23] Tecmic. Xtran passenger infopublic, . URL http://www.tecmic.pt/wp-content/uploads/2014/05/InfoPublic_EN.png. [Online; accessed 5-July-2015].

[24] Tecmic. Xtran passenger - operating support systems for passenger transport, . URL http://www.tecmic.pt/en/portfolio/xtran-passenger/. [Online; accessed 5-July-2015].

[25] Wikipedia. Geographic information system — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Geographic_information_system&oldid=668929592. [Online; accessed 5-July-2015].

[26] Wikipedia. Microsoft mappoint — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Microsoft_MapPoint&oldid=665216524. [Online; last accessed 5-July-2015].

[27] Wikipedia. Error analysis for the global positioning system — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Error_analysis_for_the_Global_Positioning_System&oldid=671016803. [Online; accessed 16-July-2015].

[28] Wikipédia. Auto lotação ingá — wikipédia, a enciclopédia livre, 2015. URL https://pt.wikipedia.org/w/index.php?title=Auto_Lota%C3%A7%C3%A3o_Ing%C3%A1&oldid=42972973. [Online; accessed 19-setembro-2015].