Desenvolvimento de aplicação móvel de trânsito

José Francisco Fernandes Antunes
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2013

Orientador
Engº Alexandre Gomes, InfoPortugal S.A.

Coorientador
Prof. Doutor Álvaro Figueira, DCC-FCUP

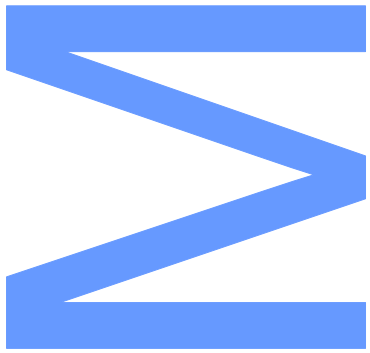
U.PORTO

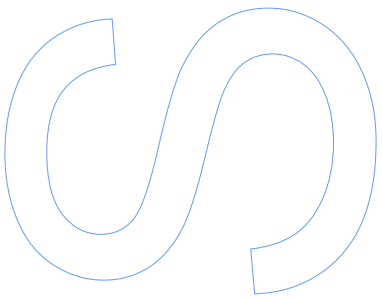
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

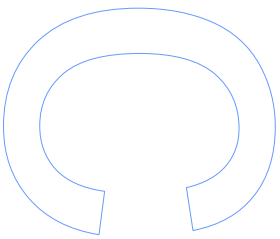
Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____







Agradecimentos

Quero agradecer ao meu melhor amigo que durante este percurso todo me deu sempre força e ânimo para continuar apesar de as coisas nem sempre terem sido fáceis, sem ti não tinha conseguido, obrigado por tudo Valentim.

À minha namorada Alexandra Azevedo pela sua dedicação e apoio em fases dificeis. Aos meus pais, à minha familia, especialmente ao meu primo João Cardoso pois é um amigo como poucos, aos meus amigos e á Alexandra Pereira.

Não podia deixar de dar uma palavra ao meu orientador Álvaro Figueira pois sempre me apoiou no decorrer do estágio e não tenho palavras para lhe agradecer todo o esforço que fez. Por fim á InfoPortugal por me ter dado a oportunidade de progredir e aos colegas com que privei na mesma empresa por me ajudarem a evoluir.

A todos um obrigado sincero.

Abstract

The goal of the present project is the development of a mobile application for Android which is capable of providing real-time information regarding traffic events in Portugal´s continental territory. The project split itself in three distinct phases. The first phase focused on the development of a transit maps server using a digital cartography system and traffic information detained by InfoPortugal. By combining these two elements, it became possible to design a system that would feed this information to the mobile application in a fast and efficient way. The second phase consisted in the actual development of the mobile application. In it, various options were included: an authentication system through Facebook and Google; a traffic observation option through map-view or through a list of the main and secondary routes; the possibility of adding your favorite routes and receiving notifications about them; and the option of easily reporting traffic events. Finally, the third consisted in the development of a server that communicates with the mobile app and allows the management and saving of the user´s preferences, like the registry, favorites and traffic reports.

Keywords

Mobile Application, Android, Traffic

Resumo

O projecto descrito neste relatório de estágio tem como finalidade o desenvolvimento de uma aplicação móvel para Android capaz de fornecer informações em tempo real sobre o estado do trânsito em Portugal Continental. O projeto divide-se em três fases. A primeira fase centrou-se na elaboração de um servidor de mapas de trânsito, recorrendo ao sistema de cartografia digital e informações de trânsito detidas pela InfoPortugal. Conjugando estes dois fatores foi possível a elaboração de um sistema, que fornecesse a aplicação com informações de trânsito de uma maneira rápida e eficaz. A segunda fase consistiu no desenvolvimento da aplicação. Foram integradas várias funcionalidades: um sistema de autenticação através da rede social Facebook e Google; a possibilidade de consulta do trânsito através de uma vista de mapa ou de uma lista de vias principais e secundárias; a possibilidade de adicionar vias favoritas e receber notificações sobre as mesmas e a facilidade de poder reportar eventos de trânsito. Finalmente para a terceira fase, foi concebido o servidor que comunica com a aplicação e que permite gerir e guardar preferências do utilizador, nomeadamente o registo, favoritos e os reports de trânsito efetuados pelo mesmo, bem como testes a confirmar a funcionalidade da aplicação.

Palavras Chave

Aplicação móvel, Android, Trânsito

Index

Lista de Figuras

Lista de Tabelas

Acrónimos

ANR Application Not Responding

API Application Programming Interface

CSV Comma Separated Values

DRY Don't Repeat Yourself

GPS Global Positioning System

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JAR Java Archive

JSF Java Server Faces

JSON JavaScript Object Notation

MP4 Moving Picture Experts Group layer 4

MVC Model-View-Controller

OSM Open Street Maps

REST Representational State Transfer

SDK Software Development Kit

SIG Sistemas de Informação Geográfica

SQL Structured Query Language

UI User Interface

URL Uniform Resource Locator

WI-FI Wireless Fidelity

XML Extensible Mark-up Language

1

Introdução

1.1 Enquadramento

A utilização das aplicações móveis no quotidiano é uma realidade cada vez mais presente. Estas conseguem suprir a necessidade constante do acesso à informação de uma forma simples e eficaz trazendo diversos benefícios tanto a nível profissional como pessoal. Numa sociedade cada vez mais marcada pela constante inovação, o uso do telemóvel tornou-se indispensável. A sua utilização passa não só pela procura da comunicação, mas também pelo auxílio nas mais diversas tarefas do quotidiano.

O aumento do consumo e consequentemente a imposição de uma sociedade cada vez mais capitalista e exigente em todos os aspetos, faz com que seja necessário uma melhor gestão e cuidado com o bem mais precioso que o ser humano possui, o tempo.

Actualmente o trânsito é um dos maiores problemas para quem vive nas cidades afetando diretamente a gestão de tempo dos seus habitantes nas suas actividades diárias.

Surge então neste contexto o desenvolvimento da aplicação de trânsito, a qual visa a disponibilização de mapas interativos, que permitem ao utilizador saber em tempo real o estado do trânsito em qualquer via principal do país. A aplicação permite também uma interação diversificada, como por exemplo, a possibilidade de reportar acidentes ou eventos, ajudando assim a disseminar a informação.

Sendo um produto com base tecnológica foi essencial perceber até que ponto seria funcional e se estaria de acordo com as necessidades dos utilizadores. Tendo como ponto de referência a aplicação anterior (ver secção 2.3.3), foi urgente identificar as carências que se fizeram sentir na sua utilização.

1.2 Descrição do problema

Devido à constante evolução das aplicações móveis despontou a necessidade da InfoPortugal atualizar a sua aplicação de trânsito de forma a poder dar uma resposta mais exigente e atual a todos os seus clientes.

Detentora de um sistema de cartografia digital e de informação de trânsito em tempo real, surgiu a necessidade de criar uma aplicação móvel que conjugasse estes dois fatores.

A problemática centra-se deste modo no desenvolvimento de um serviço móvel de utilização gratuita que permita aos condutores consultar as informações de trânsito e ao mesmo tempo possibilite que os mesmos possam contribuir com as suas próprias informações.

1.3 Objetivos

Neste projeto foram identificados como principais objectivos:

- *Background* da evolução das aplicações móveis e a sua importância no quotidiano;
- Estudo das aplicações móveis de trânsito existentes no panorama nacional;
- Desenho conceptual da aplicação móvel;
- Identificação das limitações dos dispositivos móveis;
- Selecção dos dispositivos móveis suportados pela aplicação;
- Descrição da arquitetura Cliente-Servidor;
- Identificação dos principais problemas no desenvolvimento de aplicações para dispositivos móveis;
- Desenvolvimento da aplicação.

Na fase inicial do projeto foi elaborado um Mapa de *Gantt* (ver figura A.2 no anexo A) com o intuito de auxiliar e estabelecer prazos rigorosos de forma a auxiliar a conclusão do projecto com sucesso.

1.4 Resultados esperados

No final deste projeto espera-se que a aplicação móvel desenvolvida esteja apta a ser disponibilizada a todos os utilizadores do sistema *Android*. A evolução tecnológica é uma constante e é expectável que no futuro toda a gente possa ter acesso a este género de aplicações que certamente serão uma mais-valia na gestão do tempo, citando *Honoré de Balzac*, *"O tempo é o único capital das pessoas que têm como fortuna apenas a sua inteligência"*.

1.5 Estrutura do relatório

Este relatório está dividido em 5 capítulos:

- **Capítulo 1** - Introdução: é feito um enquadramento do projeto e a sua motivação;

- **Capítulo 2** - *Background*: descrita a evolução do sistema operativo Android e a importância das aplicações móveis no quotidiano. É também efetuda uma revisão das aplicações de trânsito existentes no panorama nacional, apresentando alguns trabalhos já realizados nesta área;
- **Capítulo 3** - Desenho conceptual: é apresentado o protótipo do projeto e o tipo de interação que os utilizadores podem obter bem como uma lista de funcionalidades;
- **Capítulo 4** - Sistema desenvolvido: é abordada a solução de implementação adotada;
- **Capítulo 5** - Conclusões: é feito um resumo do processo percorrido ao longo do projeto e são dispostas ideias que ficaram para trás ou que surgiram entretanto e que poderão ser trabalhadas futuramente.

2

Background

2.1 História e Evolução do Sistema Operativo *Android*

O *Android* é um sistema operativo baseado em *Linux* e desenvolvido para ser utilizado em dispositivos móveis. Em Outubro de 2003 foi fundada em Palo Alto (Califórnia) a *Android Inc.* por Andy Rubin, Rich Miner, Nick Sears e Chris White. O intuito da *Android Inc.* segundo um dos seus fundadores Andy Rubin era a "criação de dispositivos móveis mais inteligentes e cientes da sua localização e que atendam às preferências do seu utilizador" [1].

Em Junho de 2005 a *Google* realiza a compra da *Android Inc.*, que na altura ainda não passava de uma startup promissora. O intuito da *Google* com esta aquisição era a possibilidade de entrar para o mundo dos dispositivos móveis, permitindo desta maneira divulgar os seus produtos num mercado em constante expansão.

Os dispositivos *Android* não se limitam aos *smartphones*. Atualmente são vários os dispositivos que suportam este sistema operativo, como por exemplo: *tablets*, *netbooks*, leitores de Moving Picture Experts Group layer 4 (MP4) e televisões que forneçam serviços de *Internet*. Os *tablets* têm vindo a assumir uma grande preponderância no mercado sendo considerados como um meio-termo entre um computador portátil e um *smartphone*.

Paralelamente ao desenvolvimento dos dispositivos móveis, o *Android* tem vindo também a sofrer uma grande evolução, tendo surgido desde o primeiro lançamento várias versões deste sistema operativo, as quais são caracterizadas por possuírem nomes de sobremesas.

Na figura 2.1, extraída do site develper.android.com, são mostradas as várias versões do *Android*, bem como a quota de mercado que cada uma ocupa.

Version	Codename	API	Distribution
2.2	Froyo	8	2.4%
2.3.3 - 2.3.7	Gingerbread	10	30.7%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	21.7%
4.1.x	Jelly Bean	16	36.6%
4.2.x		17	8.5%

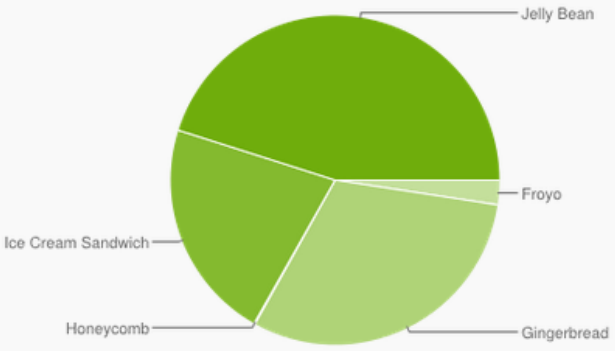


Figura 2.1: Percentagem de uso das diferentes versões Android. Dados de Setembro de 2013.[1]

Um estudo recente efetuado pela empresa *Canalys* [2], mostrou que o mercado dos dispositivos móveis, do qual fazem parte as vendas de smartphones, tablets e notebooks, cresceu

[1] Retirado de `http://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net`

para 308,7 milhões de unidades vendidas no primeiro trimestre do ano, representando um crescimento ano-a-ano de 37,4%.

O estudo mostra (ver figura 2.2) ainda que no primeiro trimestre de 2013, a plataforma *Android* representa cerca de 59,5% dos dispositivos inteligentes móveis vendidos em todo o mundo. A *Apple*, segue a plataforma da *Google* com cerca de 19,3% do mercado e por fim a *Microsoft* com 18.1%.

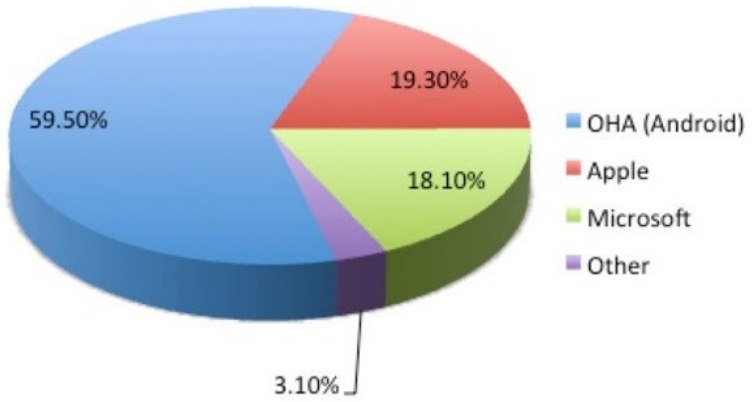


Figura 2.2: Quota de mercado do sistema operativo Android, Maio 2013.[2]

2.2 Aplicações móveis e a sua importância

As aplicações móveis estão a despoletar uma radical mudança na maneira como as pessoas abordam a tecnologia e usam os seus dispositivos móveis.

É notório que atualmente grande parte da população não passa sem o seu dispositivo móvel, já que com ele é possível aceder a toda a informação necessária e tudo à distância de poucos cliques. Esta constante demanda na busca de informação fez com que empresas de software e programadores apostassem em força no mercado móvel.

O conceito de "estação de trabalho"ou mesmo de "desktop"deixou de ser suficiente. Um indivíduo ou empresa, em tempo real, requer acesso imediato às suas aplicações profissionais ou pessoais, e à possibilidade de tomar decisões a qualquer hora ou em qualquer lugar com a máxima flexibilidade.

Estes requisitos deixam claro que um dos principais viabilizadores das empresas de software nos dias que correm é a mobilidade. Ao tornar móveis as suas ferramentas estas refor-

[2]Retirado de http://www.zdnet.com/android-is-crushing-apple-and-microsoft-in-the-mobile-device-market-7000015206/

çam a sua visibilidade no mercado, auxiliando e ao mesmo tempo familiarizando os utilizadores nas suas tarefas diárias com os seus produtos. [3]

Segundo a *Nielsen Smartphone Analytics* [4], um consumidor Android regular gasta 56 minutos por dia a interagir de forma ativa com a Web e com aplicações no seu dispositivo. Desse tempo, dois terços são gastos em aplicações móveis e apenas um terço é gasto na Web.

A forte aposta no mercado das aplicações móveis por diversas empresas de software ajudou a atrair mais utilizadores. Atualmente a oferta é vasta e a procura cresce exponencialmente. As aplicações móveis vão assim tomando cada vez mais espaço no nosso quotidiano, competindo ao programador encontrar possíveis necessidades nos utilizadores e conseguir supri-las através do desenvolvimento de novas aplicações que saibam atender da melhor maneira a essas necessidades.

2.3 Aplicações móveis e o trânsito

Longe vão os dias em que as pessoas aguardavam pelas informações de trânsito na rádio ou televisão. Num mundo onde a tecnologia impera e o tempo escasseia criou-se a necessidade de ter a informação à distância de uns cliques e o trânsito não é exceção.

Nas grandes cidades não existem dúvidas que o trânsito é um dos tópicos que mais preocupam todos aqueles que usam carro. O grande volume de passageiros nos transportes públicos e carros aumenta cada vez mais, afetando a fluidez do trânsito e, consequentemente, a qualidade de vida de todos.

Desta forma as aplicações que indicam o estado de trânsito em tempo real permitindo ao condutor optar por rotas alternativas têm-se revelado soluções viáveis no planeamento das viagens quotidianas, deslocações para o trabalho, escola entre outras.

No panorama nacional destacam-se três aplicações que fornecem informação de trânsito sobre as vias principais e secundárias de Portugal Continental. Elas são o *INRIX Traffic*, que usa as informações de trânsito da InfoPortugal, a Estradas.pt e a Trânsito InfoPortugal. Nos capítulos será feita uma análise e descrição destas aplicações.

2.3.1 *INRIX Traffic*

A aplicação *INRIX Traffic*, desenvolvida pela *INRIX* tem como principais funcionalidades a vista de Mapa, o ecrã de notícias de trânsito, a possibilidade de ver as câmaras de trânsito, embora em Portugal esta opção não esteja ativa e ainda oferece a possibilidade de receber

alertas, sendo que esta opção apenas faz parte da versão *Premium* a qual só pode ser ativada mediante pagamento.

Ao navegar pela aplicação, concluí que é uma solução sólida, pois oferece várias funcionalidades relevantes aos utilizadores deste género de aplicações.

As contrariedades encontradas na aplicação prenderam-se com o seu uso através de redes de dados móveis. Nem todos os utilizadores tendem a planear a sua viagem quando saem de casa com antecedência, tendo posteriormente que recorrer ao uso de internet móvel, a qual ainda é muito dispendiosa e lenta.

Na figura 2.3 é possível observar que certas vias ainda não estão coloridas com o seu estado de trânsito. Este processo revela-se por vezes demorado devido ao tempo que é necessário esperar até o mapa estar carregado com todas as informações de trânsito necessárias.

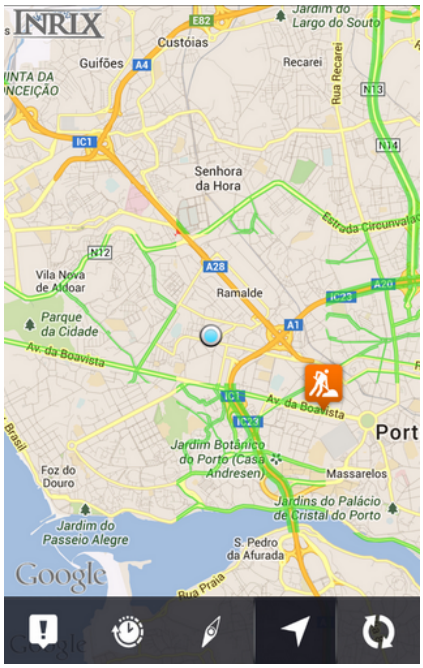


Figura 2.3: Aplicação INRIX Traffic.

2.3.2 Estradas.pt

Estradas.pt é um portal mantido pelas Estradas de Portugal que oferece informações sobre o transito. A informação aparece distribuída num mapa, onde existem vários sinais que indicam as ocorrências.

Com o aparecimento dos dispositivos móveis, a Estradas de Portugal disponibilizou uma aplicação, que permite ver, em tempo real, as mais diversas informações sobre o trânsito a

nível nacional. É possível observar a *interface* da aplicação bem como as suas funcionalidades na figura 2.4.

Figura 2.4: Interface Aplicação Estradas.pt.

A aplicação permite o acesso a câmaras, informação sobre condicionamentos de tráfego e vídeos com os boletins de trânsito diários. Para ser possível usar a aplicação é necessário registar-se dentro da mesma.

Ao usar a aplicação foi notório que o processo de autenticação nem sempre é funcional, pois foram efetuadas várias tentativas de registo e *login* sem sucesso, o que invalidava por vezes a entrada na aplicação.

Foi impossível aceder às camaras e boletins de trânsito, pois os mesmos nunca eram descarregados o que fazia a aplicação encerrar e gerar um Application Not Responding (ANR).

2.3.3 Trânsito InfoPortugal

A Trânsito InfoPortugal (ver figura 2.5) é uma aplicação que permite consultar informações e eventos de trânsito georreferenciados em tempo real. A aplicação permite também o uso da realidade aumentada para a visualização de eventos de trânsito.

A aplicação possui um vasto número de opções das quais se destacam:

- Localização da posição atual;
- Lista de todas as ocorrências com informação detalhada;
- Informação detalhada de cada ocorrência.

A localização em espaços fechados não funciona e a vista real não se revelou útil neste

Figura 2.5: Interface aplicação Trânsito InfoPortugal.

género de aplicação, pois a sua utilização não acrescenta nenhuma informação relevante ao utilizador.

A aplicação revelou ter informação útil e atualizada, contudo a aplicação está ultrapassada em termos de funcionalidades e interface. A inexistência de uma barra de ação, padrão essencial para o design de uma aplicação actualmente e a falta de funcionalidades relevantes levou a que existisse a necessidade de haver uma refundação desta aplicação.

2.4 Desafios

A usabilidade nos dispositivos móveis é um desafio recorrente para quem desenvolve aplicações móveis.

Por norma numa aplicação móvel o foco do utilizador está prejudicado, pois geralmente o mesmo encontra-se a executar outra tarefa como: ver televisão, a apreciar uma refeição, a viajar, entre outras tarefas diárias. O dispositivo móvel torna-se portanto um foco secundário, pois o utilizador habitualmente consulta o que necessita momentaneamente e retoma a sua atividade principal. A importância da aplicação possuir uma boa usabilidade de forma a prender o foco do utilizador torna-se vital.

No processo de desenvolvimento da aplicação os maiores desafios residem em tentar superar lacunas que as outras aplicações previamente estudadas possuem.

Seria pouco razoável assumir que a aplicação desenvolvida iria superar a *INRIX Traffic*, pois

esta tem por trás toda uma estrutura que a suporta e aposta no seu contínuo desenvolvimento.

Foi portanto abordado outro caminho, o caminho do poder da informação, pois a InfoPortugal é detentora de um sistema de cartografia digital similar ao *Google Maps* e é responsável pelo fornecimento da informação de trânsito a várias entidades entre as quais se encontra a *INRIX*.

O desafio principal prende-se assim em conjugar essa informação como um todo de forma a poder dar uma resposta direta a todos os utilizadores, focando a aplicação no essencial, a informação de trânsito em tempo real.

3

Desenho conceptual

O objetivo principal do estágio foi o desenvolvimento de um protótipo de uma aplicação móvel de trânsito que permitisse disponibilizar mapas interativos sobre o estado do trânsito nas principais vias do país.

O sistema foi desenhado de forma a permitir ao utilizador obter informação do trânsito em tempo real, condensando-a numa só aplicação de fácil interação ao comum utilizador do sistema operativo *Android*.

3.1 Conceitos tecnológicos

Para a compreensão do desenvolvimento do protótipo é relevante definir alguns conceitos importantes:

- ***Web service:*** é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Os *Web services* são componentes que permitem às aplicações enviar e receber dados em formato Extensible Mark-up Language (XML)/JavaScript Object Notation (JSON). Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML/JSON. [5]

- **JSON:** é um formato de dados leve para troca de dados computacionais e é um subconjunto de notação objeto de *JavaScript*, no entanto o seu uso não requer *JavaScript* exclusivamente. É utilizado em alternativa ao XML, sendo este mais fácil de escrever e percetível para o programador. [6]

- **Representational State Transfer (REST):** foi desenvolvido para permitir a comunicação entre duas aplicações quaisquer, independentemente da sua plataforma/linguagem de desenvolvimento (ver figura 3.1). Surgiu a partir de uma tese de doutoramento de *Roy Fielding*, cientista norte-americano e um dos principais autores da especificação Hypertext Transfer Protocol (HTTP). Este defende que o protocolo HTTP já possui os recursos necessários para a implementação de web services. O REST usa o protocolo HTTP para troca de mensagens. Por ser implementado através do protocolo HTTP, utiliza além dos métodos clássicos como *POST* e *GET*, métodos menos comuns como o *PUT* e o *DELETE*. [7]

[3] Retirado de http://www.devmedia.com.br/servicos-restful-com-apache-cxf-e-camel/26849

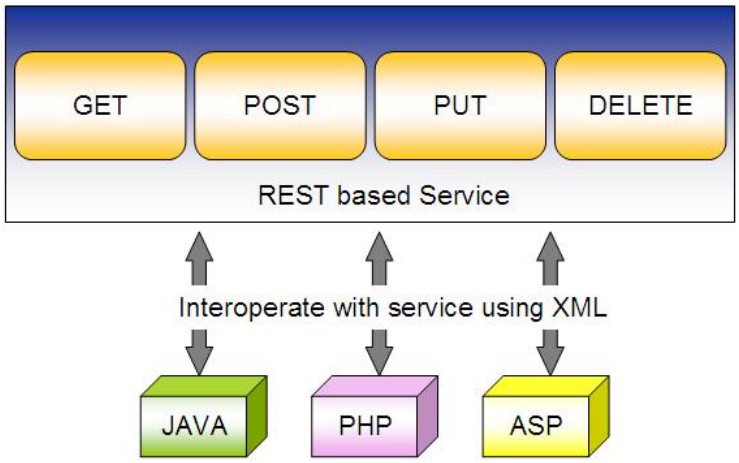


Figura 3.1: Exemplo arquitetura REST.[3]

- **Layers:** são objetos num sistema cartográfico digital que se situam no topo do mapa e consistem em um ou mais itens separados, mas que são manipulados como uma única unidade. As *layers* geralmente refletem coleções de objetos que são adicionados ao mapa para designarem uma associação comum. [8]

3.2 Servidores da aplicação

A aplicação será sustentada por dois servidores REST, o primeiro terá a função de gerar mapas de trânsito e fornecer a aplicação com estes enquanto o segundo irá promover a interação do utilizador com a aplicação.

O servidor de geração de mapas terá a função de desenhar *layers* nas vias referenciadas geograficamente pela InfoPortugal.

As *layers* irão possuir cores que simbolizem os diferentes níveis de trânsito. A introdução de um servidor com este propósito permite diminuir o fluxo de dados requerido pela aplicação e aumentar a sua performance, pois liberta o dispositivo da tarefa de desenhar as *layers* em tempo real.

O segundo servidor promove a interação do utilizador com a aplicação permitindo ao mesmo efectuar operações de login e registo, reportar eventos de trânsito e armazenar favoritos. Todos os dados referentes ao utilizador serão guardados neste servidor de forma a manter a sustentabilidade e fiabilidade dos dados do cliente.

3.3 Requisitos da aplicação

A aplicação terá suporte a partir da versão *Android* 2.4, sendo necessário preservar ao máximo o *layout* e funcionalidades entre todas as versões, dificuldade que surge devido à natural evolução do sistema operativo e a qual pode ser suprida recorrendo a diversas bibliotecas já existentes, implementadas pela própria *Google* ou por terceiros, para esse efeito.

É essencial que o dispositivo permita a localização e intercâmbio de dados em tempo real, através de Global Positioning System (GPS), Wireless Fidelity (WI-FI) ou dados móveis. Este ponto é fulcral, pois para a aplicação ser totalmente funcional é necessário existir uma constante troca de informação entre a mesma e vários serviços de forma a manter a aplicação com a informação sempre atual.

O dispositivo terá que fornecer uma determinada capacidade de armazenamento, a qual será importante para guardar os mapas transferidos pelo servidor à aplicação.

A prioridade no desenho da aplicação centrou-se em conseguir interagir com o condutor durante a navegação da forma menos intrusiva possível. Tentou-se assim limitar, tanto quanto possível, o recurso à interface gráfica. Procurou-se mantê-la simples o suficiente para não perturbar o condutor com informação visual desnecessária, mas tentando sempre que este tenha acesso à informação que precisa para a navegação.

3.4 Interação utilizador/aplicação

Para iniciar a navegação e dispor das funcionalidades da aplicação o utilizador só necessita de estar autenticado, sendo o processo de registo efetuado automaticamente, com base nas informações fornecidas pelo provider escolhido. Após a autenticação o utilizador é automaticamente georreferenciado no mapa não necessitando de navegar até à sua posição atual. É assim possível consultar as condições de trânsito envolventes de um modo célere e consequentemente tomar decisões mais imediatas e eficazes. A figura 3.2 mostra o diagrama de estados da aplicação.

De forma a diminuir a interação do utilizador com a aplicação, foram implementadas as soluções a seguir descritas.

Para reportar eventos o utilizador apenas necessita, se achar conveniente, de indicar com maior precisão a posição do evento, pois ao aceder a esta funcionalidade a sua posição volta a ser atualizada. É facultada, no mesmo ecrã, a opção de ligar diretamente para o número de emergência nacional, caso o condutor detete gravidade no evento ocorrido e possua disponibilidade para o fazer.

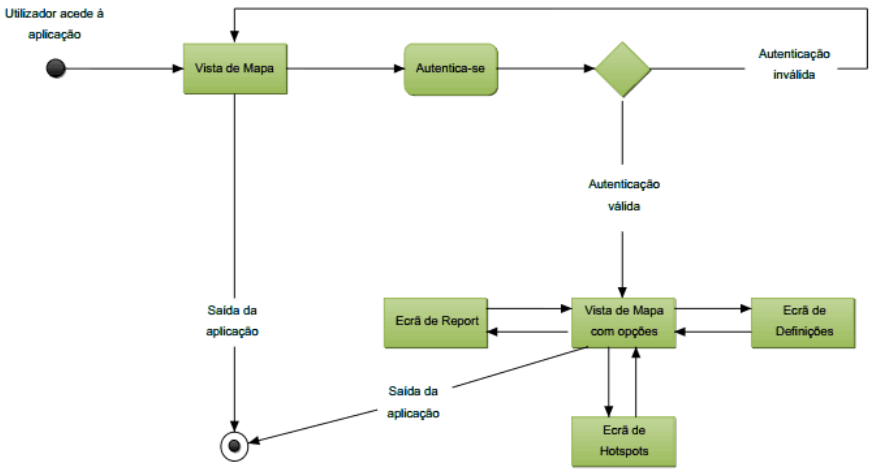


Figura 3.2: Diagrama de estados da aplicação.

O menu lateral,elemento presente na aplicação que permite ao interagir com a aplicação, pode ser ocultado a qualquer momento, apenas com um toque, permitindo uma visualização imediata do ecrã corrente. O menu foi customizado consoante o tipo de registo usado na aplicação, havendo um maior foco em relação ao *Facebook*, onde será possível, após autenticação, visualizar a foto de perfil e o nome do utilizador no topo do menu. No caso de a autenticação ser efetuada pela *Google*, existirá uma imagem padrão da *Google* acompanhada pelo email utilizado no registo.

O menu agrega uma lista de favoritos que o utilizador selecionou previamente no ecrã de *hotspots*. Os favoritos encontram-se organizados por proximidade geográfica ao condutor, facultando deste modo uma rápida navegação até aos mesmos, exclusivamente através de um toque. Este ponto é importante, pois, por norma, os favoritos são lugares de passagem frequente por parte do utilizador, sendo assim necessário que estes estejam sempre acessíveis e visíveis no momento em que a aplicação é iniciada.

3.5 Funcionalidades

No primeiro ecrã da aplicação, o utilizador pode optar por fazer login (ver figura 3.3(a)). Para isso deve possuir uma conta registada no *Facebook* ou *Google*. A escolha preferencial recaiu sobre o *Facebook* por ser a rede social mais utilizada e também por oferecer determinadas funcionalidades que seriam úteis na promoção e divulgação da aplicação. A autenticação através de conta *Google* surge como uma alternativa secundária, pois permite a todos aqueles

que não se quiserem autenticar via Facebook ou não possuírem conta no mesmo, poderem utilizar uma conta *Google* associada ao dispositivo, a qual terá necessariamente de existir para ser possível aceder ao *Android Market* e descarregar a aplicação.

(a) Autenticação. **(b)** Vista menu.

Figura 3.3: Autenticação e menu personalizado

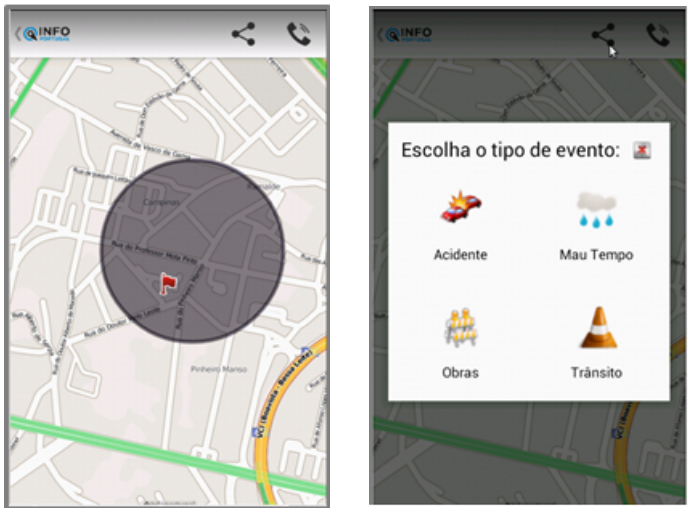
Após a validação dos dados é possível aceder a um menu personalizado da aplicação (ver figura 3.3(b)).

No menu previamente ilustrado encontramos as seguintes funcionalidades:

- **Reportar:** o utilizador pode reportar eventos de trânsito após a sua localização num raio pré-definido de duzentos e cinquenta metros. Dentro da área delimitada é possível escolher o local com maior precisão e posteriormente indicar o tipo de evento que se deseja reportar: acidente, trânsito, mau tempo e obras. (ver figuras 3.4(a) e 3.4(b))

- **Hostspots:** é apresentado ao utilizador uma lista navegável, a qual também possui pesquisa integrada, de todas as vias que fazem parte da base de dados da aplicação. Cada elemento da lista é ilustrado com as cores padrão dos níveis de trânsito, sendo possível apenas pela lista apresentada verificar o estado de trânsito de cada via e complementarmente adicioná-la à lista de favoritos, ou navegar diretamente até à mesma no mapa (ver figuras 3.5(a) e 3.5(b)).

- **Definições:** nas preferências da aplicação é possível escolher se é desejável visualizar o report de outros users na vista de mapa e também definir o alarme para a receção do



(a) Raio de acção do report.

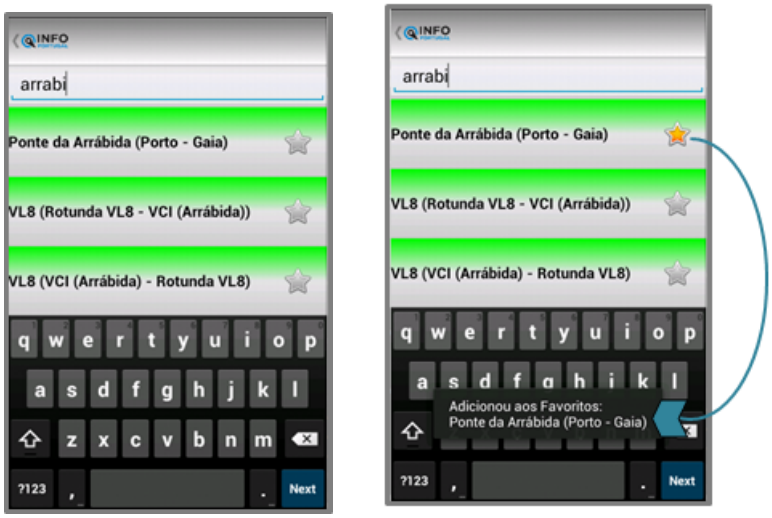
(b) Escolha de evento a reportar.

Figura 3.4: Reportar

evento de verificação do trânsito. Se esta última opção estiver selecionada, o utilizador vai receber sempre a determinada hora, uma notificação para verificar o trânsito do top cinco dos seus favoritos, sendo esta seleção efetuada por ordem de proximidade ao mesmo (ver figuras 3.6(a), 3.6(b) e 3.6(c)).

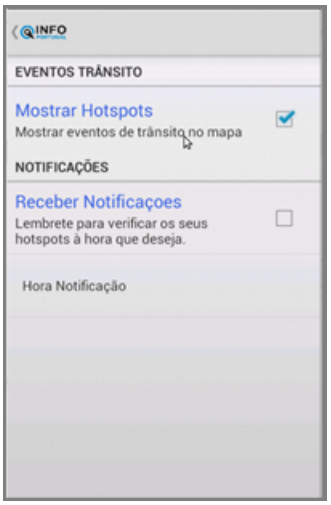
- **Lista de favoritos:** é fornecido ao utilizador uma lista de favoritos que é possível adicionar através do ecrã de *hotspots*. Encontra-se situada no menu do utilizador, facilitando assim a acessibilidade do mesmo aos seus locais favoritos (ver figura 3.7)

- **Vista de Mapa:** funciona como ecrã principal da aplicação e permite a navegação pelas vias de Portugal Continental consultando o estado de trânsito das mesmas. Ao utilizador autenticado é também possível verificar reports de outros utilizadores, que tenham efectuado um report num intervalo de tempo máximo de quarenta e cinco minutos (ver figuras 3.8(a), 3.8(b) e 3.8(c)).

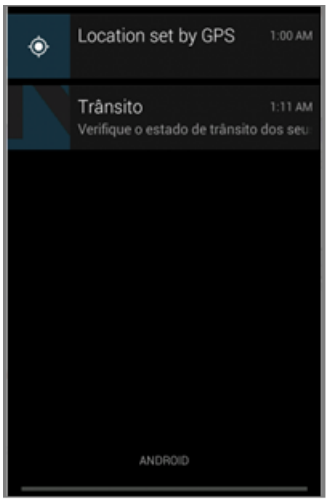


(a) Lista de Hotspots. **(b)** Adição de favoritos.

Figura 3.5: Hotspots



(a) Ecrã de Notificações.



(b) Notificação recebida.

(c) Ecrã de verificação do estado de trânsito dos favoritos do utilizador. Na imagem o 1 e o 2 são os únicos favoritos do utilizador, e estão ordenados por ordem de proximidade ao mesmo.

Figura 3.6: Definições

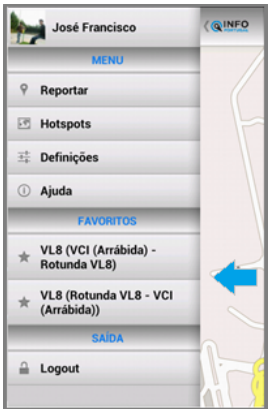
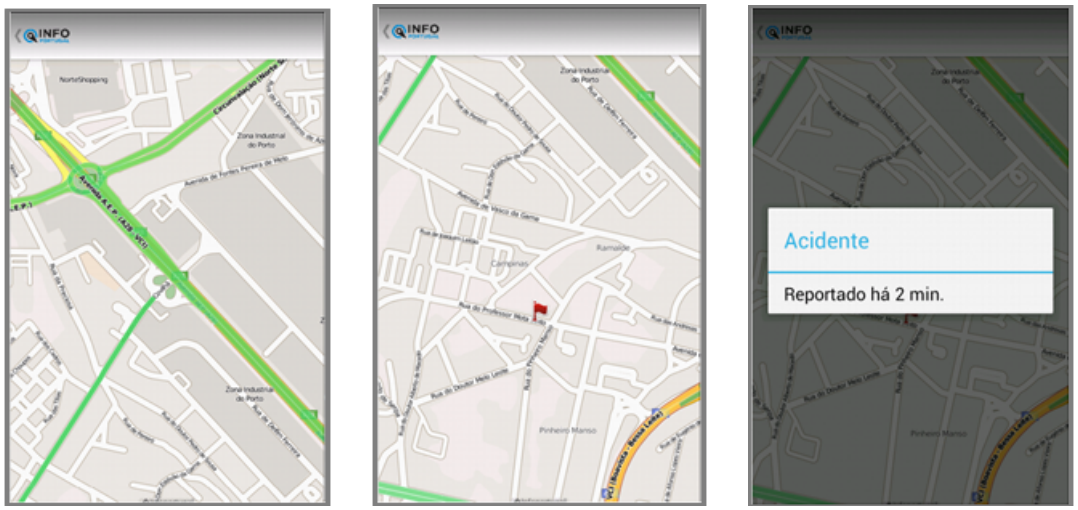


Figura 3.7: Lista de favoritos.



(a) Vista de mapa.

(b) Evento reportado.

(c) Descrição do evento reportado.

Figura 3.8: Vista de mapa.

4

Sistema Desenvolvido

4.1 Arquitetura

Neste capítulo é feita uma descrição da forma como foi idealizada e desenvolvida a arquitetura do servidor de trânsito, bem como do servidor de interação com a aplicação. Posteriormente é descrita a arquitectura de uma aplicação *Android* e a forma como esta foi programada de forma a cumprir com os requisitos expostos no capítulo 3.

4.1.1 Arquitetura do servidor de trânsito

Foi imperativo implementar o servidor que gerasse mapas de trânsito de forma a sustentar a aplicação.

O servidor é responsável por adicionar *layers* aos mapas conferindo aos mesmos uma representação gráfica do estado de trânsito actual. Para a elaboração das *layers* de trânsito foi necessário recolher informação precisa de dois *providers* já existentes na empresa.

O primeiro *provider*, intitulado de *Probes*, recolhe informação através de um sistema de posicionamento global, conseguindo definir um padrão das velocidades e quantidade de carros, dos locais georreferenciados pela InfoPortugal, estabelecendo posteriormente uma resultante entre estas duas variáveis, a qual se pode quantificar em três níveis:

- **Nível um:** trânsito fluido;
- **Nível dois:** trânsito intermédio;
- **Nível três:** tráfego elevado.

Cada registo recolhido nas vias tem associado um ID único, sendo os dados provenientes de aparelhos *GPS* alocados aos carros, os quais possuem um identificador único. Este fato faz com que seja possível agrupar um conjunto de pontos de forma a determinarmos diversos trajetos efetuados por vários veículo.

Na figura A.1, localizada no anexo A, podemos visualizar o exemplo do resultado da geração de uma probe:

Os campos *STATE* e *NUM_PASS*, respectivamente o estado da via e a quantidade de carros que passaram num intervalo de tempo de 10 minutos, foram utilizados com o intuito de definir um padrão que estado da via. Sendo considerados, por ordem da InfoPortugal, os dados onde o *NUM_PASS* fosse superior a 6.

O segundo *provider* designado de Trânsito baseia-se num sistema gerido por vários funcionários da InfoPortugal, os quais são incumbidos de actualizar a sua informação à medida que recolhem informação em tempo real. A recolha é baseada em diversas fontes, sendo o

rádio e a televisão as fontes mais fidedignas. Não foi necessário qualquer verificação dos dados provenientes deste *provider*, pois a sua inserção era supervisionada, não havendo assim necessidade de uma posterior revisão.

O servidor de trânsito, implementado na linguagem *Python*, funcionou como núcleo da aplicação tendo sido desenvolvido para a longo prazo se comportar de forma independente. A sua construção e desenvolvimento foi apoiada no servidor base de mapas da InfoPortugal que possui tiles específicas para cada coordenada e nível de zoom. A tile é uma porção de um mapa, normalmente um quadrado com resolução de 512x512 *pixels* [9], que conjuntamente com outros tiles complementares constituem um mapa. Para diferentes níveis de zoom existem tiles diferentes. Na figura 4.1 podemos encontrar um exemplo desta funcionalidade.

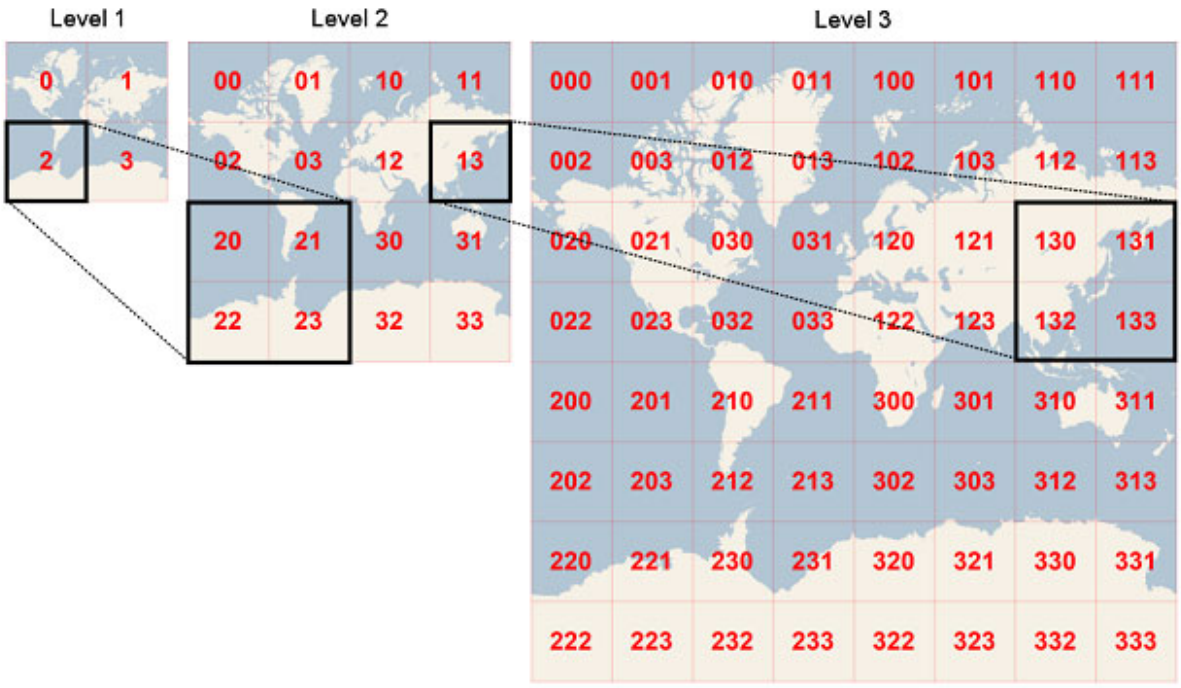


Figura 4.1: Exemplo de tiles que constituem um mapa com diferentes níveis de zoom.

Para tornar o servidor de trânsito autónomo, este foi dotado de um sistema de cache, servindo-se apenas do servidor base caso não possua uma determinada tile, em cache, aumentando assim a sua performance, pois, progressivamente deixaria de ser necessário descarregar as tiles pedidas pelo cliente ao servidor estando estas já armazenadas localmente.

4.1.1.A Desenho de *layers* no mapa

As *layers* são desenhadas em cada *tile*, numa estratégia *on-demand*, sendo o desenho baseado na informação obtida pelos *providers* previamente referidos e efetuados pelo *Mapnik*, que funciona como um motor de renderização do OpenStreetMap's.

O *OpenStreetMap* é um projeto fundado em 2004 com o objetivo de criar e disponibilizar dados geográficos gratuitos. Num espírito colaborativo, milhares de utilizadores de todo o mundo recolhem informação sobre estradas, edifícios, linhas de comboio, florestas, rios e muita outra informação habitualmente visível em mapas. Como os dados são recolhidos diretamente, e não copiados de outras fontes, não existem limitações na sua utilização [10]. Foi baseado neste projeto que os mapas da InfoPortugal foram desenvolvidos. Conjuntamente com o *Mapnik*, é possível editar estilos associado ao Open Street Maps (OSM), que são controlados definindo datasources e regras de estilo num esquema XML específico ao *Mapnik*.

A base de dados do servidor de trânsito é posteriormente usada pelo *Mapnik* na ilustração das vias. Esta é composta por um sistema *PostgreSQL* com extensão *PostGIS* que permite o uso e referenciação de objetos Sistemas de Informação Geográfica (SIG), nomeadamente o conjunto de pontos que constituem uma via. Estes conjuntos encontram-se guardados na base de dados como geometrias. De referir que a base de dados já existia na empresa, tendo sido apenas necessário adicionar um campo à tabela, que indicava o estado de cada via.

Quando a informação é extraída dos *providers Probes* e Trânsito é necessário atualizar a base de dados do servidor de trânsito, sendo o único ponto comum o ID da via. Após a informação ser tratada pelo servidor, o nível de trânsito associado a cada ID poderá ser alterado, e é com base neste nível de trânsito que o mapa é posteriormente desenhado, representando o verde o nível um, o amarelo o nível dois e o vermelho o nível três. É importante também referir que a informação vinda do *provider* Trânsito tinha prioridade no processo de inserção na base de dados, pois esta, como já foi anteriormente referido é sempre supervisionada e não depende de fatores estatísticos como é o caso das *Probes*.

4.1.2 Arquitetura do Servidor de aplicação

Para sustentar a aplicação, garantindo que a mesma seja capaz de ter uma tolerância a falhas foi implementado um servidor REST que permite ao utilizador a salvaguarda dos seus favoritos em caso de apagar a aplicação ou a memória do telemóvel e efetuar reports que possam ser visualizados por todos que usem a aplicação.

Para o desenvolvimento do servidor foi escolhida uma *Framework* de forma a acelerar a sua criação. Uma *Framework* é uma arquitetura "padrão" que fornece várias ferramentas

comuns a vários tipos de projetos facilitando e agilizando o processo de escrita de código. O seu uso proporciona uma arquitetura extremamente dinâmica, extensível, flexível e de fácil manutenção.

A escolha recaiu sobre o *Django*, por ser norma da empresa onde me encontrava a estagiar e também por possuir uma documentação bem definida e estruturada, o que me permitiu aprender rapidamente o seu funcionamento.

A *Framework Django* utiliza um sistema Don't Repeat Yourself (DRY) [11] que promove a reutilização de código existente e possui as seguintes características:

- **Mapeamento Objeto-Relacional** – O utilizador pode definir os seus modelos inteiramente em *Python* podendo assim manipular e gerar tabelas de bases de dados sem recorrer ao Structured Query Language (SQL). No entanto, caso assim pretenda, é também possível utilizar SQL;
- **Interface de Administração Automática** - É gerada automaticamente uma interface de administração completamente funcional baseada nos modelos definidos pelo utilizador;
- **Design Elegante de URL** – Permite a criação de Uniform Resource Locators (URLs) simples e personalizáveis;
- **Sistema de Templates** – Possui uma linguagem de templates poderosa, extensível e amigável que permite separar o conteúdo, design e código *Python*.

O *Django* segue o padrão *MVC* e fornece um suporte sólido para o desenvolvimento *Web*. A figura 4.2 ilustra o funcionamento do padrão Model-View-Controller (MVC).

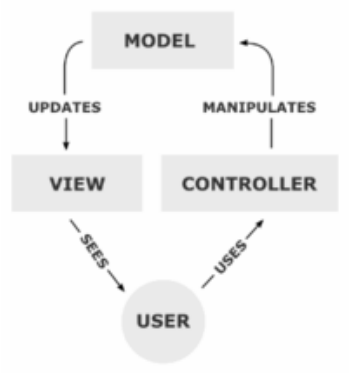


Figura 4.2: Padrão MVC.

O funcionamento do *Django* pode assim resumir-se nos seguintes passos:

- O model (models.py) define a informação em *Python* e interage com ela. Esta informação geralmente está contida em bases de dados relacionais, no entanto, é possível utilizar outros mecanismos tal como o XML;
- O *URLdispatcher* mapeia o pedido efetuado para uma função da view;
- A função na view recolhe os parâmetros fornecidos, se existirem, e executa o pedido, que por norma envolve leitura/escrita na base de dados;
- Depois de processar o pedido, a própria view retorna a resposta sob a forma de um objeto HTTP response.

4.1.2.A Implementação URLdispatcher

No URLdispatcher foram definidos padrões de URLs. Os padrões funcionam através do reconhecimento de palavras-chave passadas no URL. Uma chamada a http://myexampleserver.com/login corresponde ao primeiro elemento da lista de urlpatterns (ver figura 4.3). O Django ao receber e detetar o padrão do URL chama a função favoritos.views.loginJSON, que se encontra implementada na *view*.

```
from django.conf.urls import patterns, include, url
from django.conf import settings
from favoritos.views import *

urlpatterns = patterns('',
    url(r'^login$','favoritos.views.loginJSON'),
    url(r'^mFavorites$','favoritos.views.manageFav'),
    url(r'^report$','favoritos.views.Report_user'),
    url(r'^getReport$','favoritos.views.Report_get'),

)
```

Figura 4.3: Ficheiro de padrões URL.

Cada padrão definido possui uma função específica inerente à aplicação as quais passo a descrever seguidamente:

- `url(r'^login$','favoritos.views.loginJSON')`: Autenticação/criação de um utilizador;
- `url(r'^mFavorites$','favoritos.views.manageFav')`: Permite atualizar a lista de favoritos do utilizador na base de dados;

- `url(r'^report$','favoritos.views.Report_user'):` Permite efetuar reports de eventos, armazenando-os na base de dados;

- `url(r'^getReport$','favoritos.views.Report_get'):` Devolve todos os *reports* que a base de dados possui de todos os utilizadores.

Existem três casos particulares onde os pedidos são acompanhados por objetos *POST*, como é o caso do *login*, *mFavorites* e *report*.

4.1.2.B Implementação Views

Uma view, é simplesmente uma função *Python* que recebe uma requisição *Web* e retorna uma resposta *Web*. Esta resposta pode ser um conteúdo Hypertext Markup Language (HTML) de uma página, um redireccionamento, um erro 404, um documento XML, uma imagem, ou qualquer outro tipo de resposta que o programador deseje retornar. A *view* em si contém qualquer lógica arbitrária que é necessária para retornar uma resposta. Este código, por uma questão de boa prática, deve situar-se num ficheiro chamado views.py. [11]

No servidor da aplicação foram definidas quatro *views*:

- `favoritos.views.loginJSON:` Serve para autenticar o utilizador na aplicação. Recebe um objeto *POST* proveniente e verifica se este contem as chaves email, que vai funcionar como *username* do utilizador, e tipo, que indica o provider usado para efetuar o login na *app*, neste caso *Google* ou *Facebook*. Se o utilizador já se encontrar registado no sistema recebe a sua lista de favoritos, caso contrário é efetuado um registo automático usando os parâmetros recebidos no objeto de *request*. Retorna o ID de registo do utilizador o qual é guardado na aplicação para futuras trocas de informação;

- `favoritos.views.manageFav:` Faz a gestão em tempo real dos favoritos do utilizador. Sempre que existe uma adição/remoção esta é comunicada pela aplicação ao servidor o qual se encarrega de manter os favoritos atualizados;

- `favoritos.views.Report_user:` Permite ao utilizador enviar um *report*, o qual é inserido posteriormente na base de dados do servidor;

- `favoritos.views.Report_get:` Retorna todos os *reports* efetuados pelo utilizador num espaço temporal inferior a quarenta e cinco minutos.

4.1.2.C Implementação de Models

Um model é essencialmente constituído por classes *Python*, onde cada classe representa uma tabela da base de dados e cada atributo dessa mesma classe representa um campo na base de dados.

```
class UserApp(models.Model):
    email = models.CharField(blank=False, max_length=100, null=False)
    tipo = models.CharField(max_length=10, blank=False, null=False)
    def __unicode__(self):
        return self.email
    def save(self, *args, **kwargs):
        super(UserApp, self).save(*args, **kwargs)

class Favorite(models.Model):
    user_id=models.ForeignKey(UserApp)
    id_favoritos=models.IntegerField(blank=False,null=False,max_length=10)
    def save(self, *args, **kwargs):
        super(Favorite, self).save(*args, **kwargs)

class Report(TimestampsBaseModel):
    user_id=models.ForeignKey(UserApp);
    lat=models.FloatField(blank=False,null=False,max_length=60)
    longitude=models.FloatField(blank=False,null=False,max_length=60)
    desc=models.CharField(max_length=100,blank=False,null=False)
    def save(self, *args, **kwargs):
        super(Report, self).save(*args, **kwargs)
```

Figura 4.4: Models.

A figura 4.4 ilustra os *models* criados para o servidor da aplicação. A *classe UserApp* serve para guardar o email e o tipo de conta associada àquele email. Sempre que é inserido um novo utilizador é gerado um ID único. Na *classe Favorite* é armazenado o par user_id/id_favoritos. Cada favorito, constitui uma via específica, logo também possui um ID único o qual é emparelhado com o ID do utilizador. Por fim a classe *Report*, permite armazenar todos os *reports* efetuados por um determinado utilizador armazenando a sua localização através do lat e longitude e uma determinada descrição que será armazenada no *desc*.

4.1.3 Arquitetura da Aplicação

4.1.3.A Bibliotecas auxiliares

A aplicação móvel foi construída em *Java* e como tal orientada para ser utilizada pelo sistema operativo *Android*. Foi necessário porém, recorrer a várias bibliotecas já implementadas:

- IpMaps*Android*: biblioteca que integra o sistema de mapas da InfoPortugal, *IpMaps*, num ambiente *Android*. Foi necessário alterar o endereço de servidor de recolha de tiles para o do servidor de trânsito, de forma a serem providenciados os mapas pretendidos;

- *ActionBarSherlock*: Uma das principais características no desenvolvimento atual de aplicativos para *Android* é a presença de uma *Action Bar*, sendo a *Google* uma das principais impulsionadoras deste procedimento. O problema é que a *Google* só oferece uma Application Programming Interface (API) nativa para criar *Action Bars* a partir da versão 3.0 do *Android*, deixando assim de oferecer suporte a 60% dos utilizadores. Com esta biblioteca é possível suprir essa mesma falha; [12]

- *Facebook* Software Development Kit (SDK): Permite a integração da aplicação com o *Facebook*;

- *OpenCSV*: Biblioteca que permite a leitura e escrita de ficheiros em formato Comma Separated Values (CSV) em ambiente *Android*;

- Jackson: *Parser* de JSON, reconhecido por ser atualmente o mais eficaz e rápido. Extremamente útil para receber e tratar a informação de trânsito dos *providers* e para envio de informação para o servidor da *app* (ver figura 4.5).

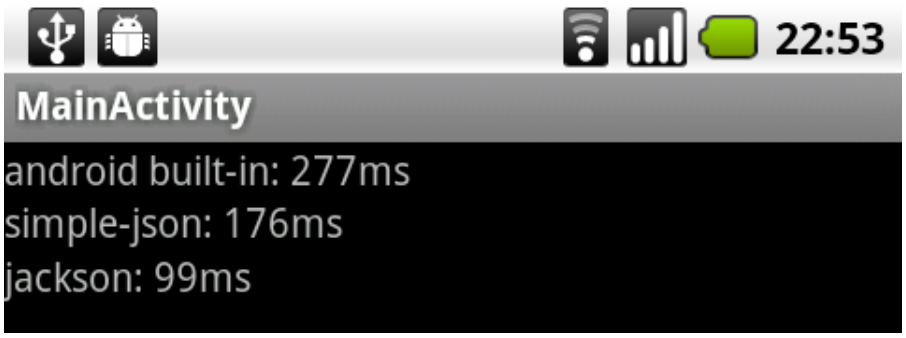


Figura 4.5: Eficiência da biblioteca Jackson.

4.1.3.B Estrutura da aplicação

Uma aplicação *Android* possui uma estrutura específica, a qual é composta pelos seguintes componentes (ver figura 4.6):

Figura 4.6: Estrutura de uma aplicação *Android*.

- src: Código fonte da aplicação;
- gen: Código fonte gerado automaticamente;
- assets: Ficheiros estáticos que serão incluídos na aplicação;
- bin: Nesta pasta são guardados os ficheiros depois de compilados;
- libs: Nesta pasta são guardados todos os Java Archives (JARs) (bibliotecas) necessárias;
- res: Pasta onde se encontram recursos como ícones, definição de *layouts* e imagens. Os recursos da aplicação estão guardados em subpastas as quais correspondem a um determinado tipo de recurso. Nestas pastas são guardados os ficheiros XML com a definição de *layouts*, *strings* e cores. Esta organização de informação permite a definição de recursos alternativos para diferentes configurações específicas dos equipamentos, da linguagem e da resolução do ecrã. O acesso aos recursos através do código é realizado através da classe R que é criada no momento da compilação do projeto. A classe R contém subclasses para cada tipo de recursos, desde que exista pelo menos um recurso desse tipo [13].

Cada projeto *Android* inclui um ficheiro *Android*Manifest.xml, guardado na raiz do projeto. Neste ficheiro fica definida a estrutura, os meta dados e os componentes da aplicação. É um elemento crucial de qualquer projeto *Android* que funciona como uma "tabela de conteúdos"do

projeto. É nele que são declaradas as *Activities* que definem cada um dos ecrãs que o utilizador poderá visualizar, os *Services*, *Receivers*, a versão do SDK e as respetivas permissões que a aplicação terá ao ser instalada no dispositivo móvel.

Uma *Activity* é um ecrã numa aplicação *Android*. Cada *activity* é implementada como uma única classe que se estende a partir da classe base *Activity*. Essa classe irá mostrar uma interface composta por Views e a qual irá responder a eventos por parte do utilizador.

A mudança de um ecrã para outro é realizada através do início de uma nova *Activity*. Quando um novo ecrã abre, o seu antecessor fica em *standby* e é colocado numa pilha, permitindo ao utilizador navegar entre eles. Para realizar uma mudança de ecrã o *Android* utiliza uma função *startActivity* que pode incluir como parâmetro uma instância de uma classe especial chamada *Intent*. No *Intent* podemos definir parâmetros que sejam necessários para a inicialização da nova Activity, como por exemplo *strings*, booleanos e estruturas.

Um *Service* é um componente da aplicação que permite realizar operações de longa duração em background de forma a não afetar a interface do utilizador. Um determinado componente da aplicação pode iniciar o serviço e o mesmo irá continuar a correr em background independentemente do utilizador ter a aplicação a correr ou não.

Um *Receiver* é um componente do *Android* que responde a determinadas notificações enviados pelo sistema. O *Android* envia uma mensagem para todo o sistema quando determinados eventos ocorrem, e esta mensagem pode ser respondida por aplicações que tenham interesse em recebê-las. Esta mensagem é chamada de *Broadcast* e o componente responsável pela resposta é chamado de *Broadcast Receiver* ou *Receiver*.

É importante também referir outros conceitos que foram comummente utilizados no decorrer da programação da aplicação:

- Interface: A interface é um recurso muito utilizado em *Java*, bem como na maioria das linguagens orientadas a objetos. Permite que um determinado grupo de classes possa ter métodos ou propriedades em comum num determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente.

- Context: A classe *Context* permite aceder a configurações e recursos compartilhados entre os vários ecrãs (*Activities*) da aplicação. Cada *Activity* possui o seu próprio *Context*.

- View: Uma *View* pode ser um componente gráfico da aplicação como por exemplo: botões, *checkboxes* e imagens. Pode também atuar como um gerenciador de layout possuindo a função de organizar outras *Views* mais simples (ver figura 4.7).

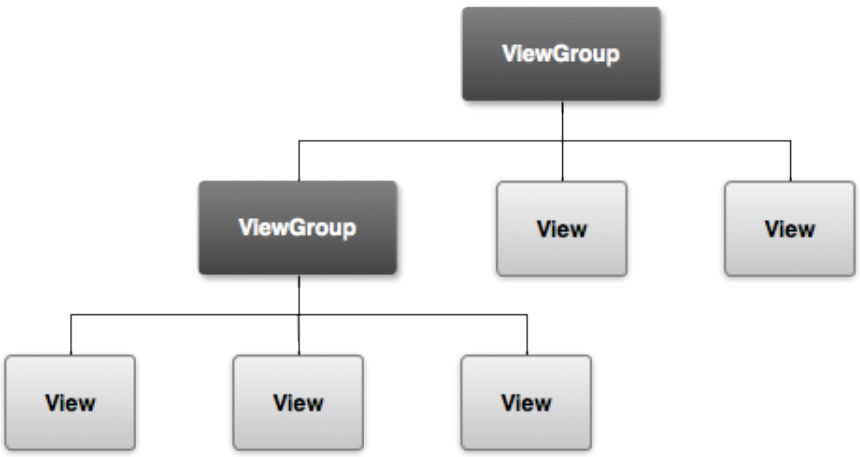


Figura 4.7: Hierarquia de uma View.

- Multitasking: É comum, em aplicações mais complexas ser necessário realizar tarefas mais "pesadas", as quais podem demorar um tempo considerável até terminar a sua execução. Pode ser uma requisição *Web* ou até uma operação mais complexa na base de dados. Este tipo de tarefas requer uma atenção especial, pois não é recomendável que sejam executadas como parte do processo principal, pois isso impede que o estado da aplicação mude, ou seja, não será possível atualizar o ecrã corrente ou receber nenhum comando de entrada. A aplicação ficará em *standby* durante o processo, e no caso do *Android*, se isso for frequente, esta poderá encerrar automaticamente.

Parte da solução é utilizar outras *threads* para diversos processamentos complexos, e a *thread* principal fica apenas responsável pelas operações de *INPUT/OUTPUT*. Existe um senão no uso destas *threads*, pois apenas a *thread* que iniciou a UI pode alterá-la, ou seja, não é possível criar uma nova *thread* e obrigá-la a atualizar algo no ecrã. Para contrariar esse detalhe o *Android* fornece alguns métodos:

- Activity.runOnUiThread e View.post: Métodos que permitem a qualquer *thread* adicionar um *Runnable (Interface Java)* para ser executado na *thread* da UI;
- AsyncTask: Encapsula o processo, por norma complexo fornecendo métodos para modificar a UI antes, durante e após a execução;
- Adapter: É um padrão desenho que serve para adaptar duas interfaces incompatíveis. No caso do *Android*, serve para adaptar uma lista de objetos para uma lista de elementos da interface gráfica, como as linhas de uma *ListView* (exibe os componentes em forma de lista). O *Android*, através da classe *ListView* solicita, para cada linha da lista um novo

objeto *View*. A função do *Adapter* é criar um objeto *View* que represente visualmente o objeto da posição específica da lista de objetos.

- Shared Preferences: A classe *Shared Preferences* fornece uma *Framework* geral que permite salvar e recuperar pares de dados primitivos do tipo chave-valor. Os mesmos continuam armazenados, mesmo após a aplicação ter sido encerrada. Esta classe foi particularmente útil no gerenciamento das sessões dos utilizadores na aplicação.

4.1.4 Programação da Aplicação

O código da aplicação foi subdividido em dezassete pacotes. A divisão promoveu a organização do código em funções específicas da aplicação. A figura 4.8 ilustra como foi processada essa divisão:

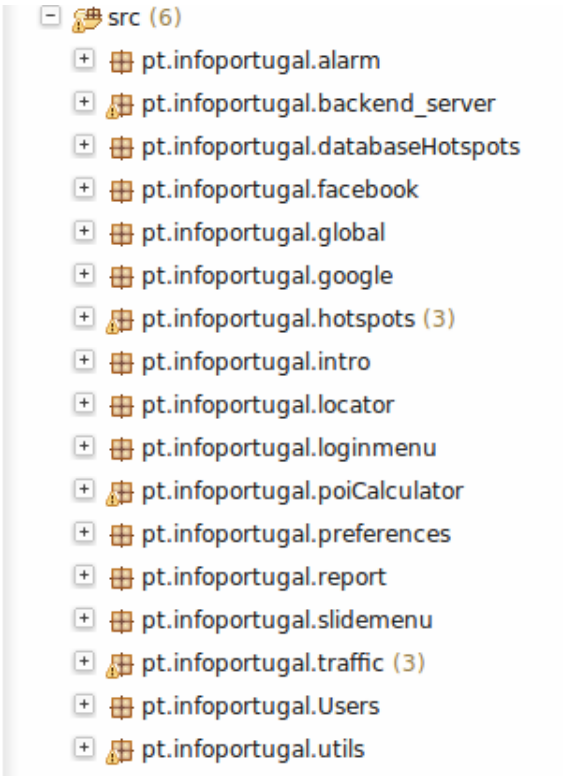


Figura 4.8: Packages da Aplicação.

O desenvolvimento da aplicação centrou-se em diversas funcionalidades que serão descritas nos subcapítulos seguintes. De referir que para a construção de determinadas funções como a vista de mapa e o *slidemenu* recorreu-se a bibliotecas anteriormente já referidas, pelo que a sua implementação se resumiu ao uso da API dessas mesmas bibliotecas.

Finalmente, é também importante dar ênfase ao pacote pt.infoportugal.global que possui o ficheiro Global.java que usa o padrão de software *singleton*, que garante a existência de apenas uma instância de uma classe, mantendo assim um ponto global de acesso ao objeto [14]. Dentro deste objeto foram declaradas quatro variáveis estáticas que eram acedidas sempre que necessário em qualquer parte da aplicação. Para criar este padrão foi necessário extender o objeto Global a toda a aplicação como podemos ver no código a seguir apresentado.

```
1  public class Global extends Application{
2
3      private static SparseArray<TrafficObject> favoritos_slide=null;
4
5      private static ListView slideView=null;
6
7      private static SessionManager session=null;
8
9      private static PoiDegree real_time=null;
10
11 }
```

4.1.4.A Alarme e Preferências

Ambas as funcionalidades estão situadas em dois pacotes distintos, embora estejam relacionadas.

O pacote *pt.infoportugal.preferences* é composto por dois ficheiros: *Preferences.java* e *TimerClock.java*. O *Preferences.java* é uma activity que estende a classe *PreferenceActivity*. A classe *PreferenceAcitivity* permite criar um ecrã de preferências, guardando todos os valores selecionados pelo utilizador automaticamente nas *SharedPreferences*. Para definir as preferências foi necessário criar um ficheiro .xml que é automaticamente carregado sempre que o ecrã de preferências é acedido.

```
<CheckBoxPreference
            android:defaultValue="false"
            android:key="semana"
            android:summary="Lembrete para verificar os seus hotspots à hora
            que deseja."
            android:title="Receber Notificaçoes" />
```

```
        <pt.infoportugal.preferences.TimerClock
            android:defaultValue="12:00"
            android:key="horario_semana"
            android:dependency="semana"
android:summary="Hora Notificação" >
</pt.infoportugal.preferences.TimerClock>
```

A tag <CheckBoxPreference> indica que será mostrado ao utilizador uma *checkbox*, que pode ser ativada ou desativada pelo mesmo. Dentro da tag existem elementos que definem os valores padrões da opção. O android:title indica o título da opção no ecrã de preferências e o summary a sua descrição, ambas são visíveis ao utilizador e têm como intuito auxiliar na decisão de ativar ou desativar a opção.

O *defaultvalue* indica o valor por omissão da opção, neste caso false, ou seja, a primeira vez que o utilizador acede ao ecrã das opções a *checkbox* estará desativada. O value pode ter vários tipos desde *string, long, int, float ou boolean* como é o caso. A *key* indica o nome da chave da opção. É particularmente útil na ótica do programador pois permite aceder à opção por código em qualquer instância da aplicação. A seguir é apresentado o código usado para aceder à opção que tem a *key* "semana":

```
1 SharedPreferences prefs  PreferenceManager.getDefaultSharedPreferences(
    getApplicationContext())
2 prefs.getBoolean(semana, false)
```

O método *getBoolean* recebe como parâmetros *(key,defaultValue)*, e devolve como resultado o valor atual da preferência independentemente do *defaultValue*.

As opções condicionam a aplicação visto ativarem ou desativarem determinadas funcionalidades, sendo assim importante poder aceder ao seu valor.

A *tag* <pt.infoportugal.preferences.TimerClock> foi uma opção que teve que ser implementada, ao contrário da *CheckBoxPreference* que é uma opção pré-definida em *Android*. Foi necessário implementar a classe *TimerClock* a qual devolve uma *widget* do tipo *TimePicker*, que permite ao utilizador selecionar a hora e minutos que deseja receber a notificação. Ao confirmar a hora, esta é salva nas *SharedPreferences*.

Esta opção só se encontra disponível caso a opção semana esteja selecionada. Dentro da *tag* <pt.infoportugal.preferences.TimerClock> existe um campo android:dependency="semana", que indica que esta opção depende da opção semana para ser editável ao utilizador.

O pacote pt.infoportugal.alarm contém o alarme da aplicação. Dentro do pacote está definida uma *Activity* e dois *Broadcast Receivers*.

O primeiro *receiver* é o *ReceptorAlarme* que permite à aplicação lançar uma notificação ao sistema operativo para a hora em que o alarme está definido.

O segundo *broadcast receiver* denominado de *ReceptorBoot* tem a função de reprogramar o alarme da aplicação. Sempre que o telemóvel é reiniciado, o alarme, que funciona por *PedingIntents* (*intents* que ficam em modo espera até que um determinado evento ocorra) perdem-se. É assim necessário reconfigurar o alarme quando um *reboot* ao sistema ocorre. É possível detetar um reiniciar do sistema através das seguintes declarações no manifest.xml:

```
<receiver android:name="pt.infoportugal.alarm.ReceptorBoot">
          <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
          </intent-filter></receiver>
```

Quando a notificação é ativada e apresentada ao utilizador, o mesmo ao clicar sobre ela faz com que seja lançada uma nova Activity, que vai iniciar um ecrã com vista de mapa e com os favoritos do utilizador ordenados por ordem de proximidade.

4.1.4.B Base de Dados

Para mostrar todas as vias no ecrã de *Hotspots* foi necessário recorrer à base de dados já existente na empresa. Esta era constituída por uma tabela que continha o ID da via, nome da via, as suas coordenadas e o seu nome.

A base de dados foi extraída para um ficheiro .csv (ver figura 4.9), que foi guardado na pasta *assets*, permitindo desta maneira ser sempre possível restaurar a base de dados caso a mesma tivesse sido apagada da memória da aplicação. Na figura a seguir encontra-se um excerto do ficheiro .csv guardado na pasta *assets*.

O pacote pt.infoportugal.databaseHotspots contém todos os ficheiros necessários à realização das tarefas relativas ao acesso e criação da base de dados. É composto pelos seguintes ficheiros: *DBCreator, DBTraffic e TrafficObject.*

No ficheiro *DBTraffic* foram criados vários métodos de interação com a base de dados. A criação, a leitura do ficheiro .csv e posterior inserção e a verificação de existência da base de dados no sistema foram os principais métodos implementados.

A base de dados na aplicação era constituída por uma tabela idêntica à estrutura do ficheiro .csv, possuindo os campos de ID(chave primária), latitude, longitude e nome.

id	lat	lon	name
989	-9.138447	38.763222	2ª Circular (Aeroporto - Calvanas)
988	-9.123269	38.774311	2ª Circular (Aeroporto - Encarnação)
1006	-9.199699	38.741318	2ª Circular (Benfica - Pinamanique)
993	-9.162715	38.757797	2ª Circular (Calçada de Carriche - Colégio Alemão)
990	-9.143373	38.762695	2ª Circular (Calvanas - Aeroporto)
991	-9.151364	38.760921	2ª Circular (Calvanas - Calçada de Carriche)
992	-9.151501	38.760704	2ª Circular (Campo Grande - Calvanas)
994	-9.163467	38.757496	2ª Circular (Colégio Alemão - Campo Grande)
995	-9.167004	38.757324	2ª Circular (Colégio Alemão - Eixo Norte-Sul)
1002	-9.186056	38.753704	2ª Circular (Colégio Militar - Estádio da Luz)
1003	-9.188467	38.750961	2ª Circular (Colégio Militar - Fonte Nova)
996	-9.168338	38.757248	2ª Circular (Eixo Norte-Sul - Colégio Alemão)
997	-9.175191	38.757732	2ª Circular (Eixo Norte-Sul - Torres de Lisboa)
987	-9.123599	38.776203	2ª Circular (Encarnação - Aeroporto)
986	-9.120204	38.784832	2ª Circular (Encarnação - Sacavém (A1))
1001	-9.185976	38.753975	2ª Circular (Estádio da Luz - Colégio Militar)
1000	-9.181911	38.756294	2ª Circular (Estádio da Luz - Torres de Lisboa)
1004	-9.191479	38.746288	2ª Circular (Fonte Nova - Benfica)
1005	-9.191969	38.745922	2ª Circular (Pinamanique - Benfica)
1007	-9.200253	38.740856	2ª Circular (Pinamanique - Benfica)
985	-9.117724	38.787277	2ª Circular (Sacavém (A1) - Encarnação)
998	-9.175514	38.757576	2ª Circular (Torres de Lisboa - Eixo Norte-Sul)
999	-9.181869	38.756466	2ª Circular (Torres de Lisboa - Estádio da Luz)
415	-8.795647	38.913284	A10 (A13 - Benavente)
411	-8.98396	39.017052	A10 (A1 - Arruda dos Vinhos)
412	-8.941979	39.014618	A10 (A1 - Benavente)
408	-9.075526	38.921974	A10 (A9 - Arruda dos Vinhos)
410	-9.057413	38.979256	A10 (Arruda dos Vinhos - A1)
409	-9.062024	38.958744	A10 (Arruda dos Vinhos - A9)
413	-8.843138	38.967052	A10 (Benavente - A1)
414	-8.819601	38.944065	A10 (Benavente - A13)
421	-8.507551	41.537548	A11 (A3 - Barcelos)
422	-8.490284	41.532856	A11 (A3 - Braga)
433	-8.263304	41.300575	A11 (A42 - Felgueiras)
434	-8.260973	41.287941	A11 (A42 - Lixa)
439	-8.206135	41.235191	A11 (A4 - Marco de Canaveses)
427	-8.334762	41.433689	A11 (A7 - Guimarães)
428	-8.260337	41.396378	A11 (A7 - Vizela)

Figura 4.9: Ficheiro .csv da base de dados.

O ficheiro *DBCreator* é uma *AsyncTask* que funciona como verificador da existência da base de dados na aplicação. Caso a base de dados não exista na memória a mesma será criada, recorrendo para o efeito ao método *createEntry* existente no ficheiro *DBTraffic*.

O método *createEntry* tem a tarefa de ler o ficheiro .csv e até o seu término inserir os dados na base de dados da aplicação.

A operação de verificação ocorre sempre que a aplicação é inicializada, num ecrã introdutório, comumente chamado de *splash screen*, e situado no pacote pt.infoportugal.intro.

```
1 entry = new DBTraffic(targetCtx);
2 entry.open();
3 if(entry.exits()==false)
4 {
5 entry.createEntry();
6 }
7 entry.close()
```

4.1.4.C Localização

A localização do dispositivo, foi obtida através de GPS ou de redes de dados a que o utilizador esteja ligado. A primeira localização obtida era posteriormente analisada, caso não cumprisse determinados requisitos, era descartada e a procura continuava até que fosse encontrada uma localização satisfatória. Caso não fosse detetada nenhuma localização o utilizador era avisado.

Para obter a melhor localização até ao momento era usado o método pré-definido em *Android*, *getLastKnowLocation* para os dois *providers*, GPS e *NETWORK*, sendo usada a mais recente, ou seja, a que foi obtida há menos tempo, como podemos observar no código a seguir.

```
1  locationGPS=locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER)
       locationNetw=locationManager.getLastKnownLocation(LocationManager.
       NETWORK_PROVIDER)
2
3  if(locationGPS==null && locationNetw==null)
4              return null;
5
6           if(locationGPS==null)
7              return locationNetw;
8           else
9              if(locationNetw==null)
10                return locationGPS;
11
12           if(locationGPS.getTime()>locationNetw.getTime())
13             return locationGPS;
14          return locationNetw;
```

Caso a melhor localização obtida até ao momento fosse uma localização mais antiga do que a que obtida era efetuado um teste de precisão à nova localização. Se a precisão da localização ultrapassasse os 400 metros a nova localização era descartada e a procura continuava.

A procura consiste num *Timer* de três minutos, tempo necessário em média para obter uma localização precisa através de GPS. Se nesse espaço de tempo nenhuma localização cumprisse os requisitos estabelecidos o utilizador seria informado que não foi possível atribuir-lhe uma localização.

A localização situa-se no pacote pt.infoportugal.locator e é constituída pelo ficheiro *LocationActivity* que trata do processo de localização descrito acima e por uma interface *Changed*

que permite estender os métodos de alteração/falha de localização a outros componentes da aplicação.

```
1 public interface Changed {
2
3    void locationChanges(Location location);
4    void locationFailed();
5 }
```

4.1.4.D Autenticação

A autenticação/registo é processada por dois métodos como já foi anteriormente referido: *Facebook* e *Google*. Ambas estão situadas em dois pacotes pt.infoportugal.*Facebook* e pt.infoportugal.google.

O pacote pt.infoportugal.Users contém o ficheiro SessionManager.java que cria um ficheiro de *Shared Preferences* chamado Users o qual é usado para gerir a sessão do utilizador enquanto este estivesse autenticado.

Para garantir a consistência dos dados sempre que uma operação de autenticação/registo ocorre é estabelecida uma comunicação com o servidor que se encarrega de armazenar os dados do cliente.

Após o processo de autenticação/registo o utilizador recebe um ID de registo proveniente do servidor, que é armazenado na aplicação e posteriormente utilizado nas comunicações com o servidor.

A autenticação pela *Google* é efetuada através do *AccountManager*. Todas as contas de um determinado tipo, neste caso *Google*, presentes no dispositivo podem ser acedidas através do *AccountManager*.

```
1        private static AccountManager mngr;
2        mngr=AccountManager.get(context);
3        acts=mngr.getAccountsByType("com.google");
```

Após serem detetadas, estas são apresentadas ao utilizador que poderá selecionar qual delas é que pretende associar à aplicação. Após a seleção é feita a verificação da mesma através do seguinte método:

```
1 getAuthToken(selected_account, "ah", null, (Activity) mcontext, null, null);
```

Através do método descrito acima é obtido um token de autenticação. O método recebe como parâmetros principais a conta selecionada e o parâmetro “ah”. O parâmetro “ah” indica que a conta vai ser validada através do *Google App Engine*.

O *Google App Engine* suporta a integração de uma aplicação com as contas da Google para autenticar um utilizador. A aplicação permite que um utilizador faça login com uma conta da *Google* e posteriormente seja possível aceder ao endereço de e-mail e ao nome de exibição associados à conta.

A autenticação através do *Facebook* é feita recorrendo à biblioteca oficial do *Facebook*. Ela permite extrair diversas informações do utilizador e fornece uma caixa de autenticação para utilizadores que não tenham a aplicação do *Facebook* instalada no seu dispositivo.

Para implementar a autenticação foi necessário recorrer à variável *Session* pertencente à biblioteca do *Facebook*. A variável *Session* permite que seja efetuada a autenticação e posteriormente seja obtido um token referente ao login efetuado. Após a variável *Session* ter sido inicializada é possível abrir a mesma para leitura e extrair os dados necessários para autenticar o utilizador no servidor da aplicação. Por questões de privacidade do *Facebook*, não foi possível extrair o e-mail do utilizador sendo assim extraído o *Facebook* ID que é sempre único para cada utilizador.

```
1 Session session = new Session(mcontext);
2 Session.setActiveSession(session);
3 session.openForRead(new Session.OpenRequest(mactivity).setCallback(callback));
4 Request.executeMeRequestAsync(session, new Request.GraphUserCallback() {
5 public void onCompleted(GraphUser user, Response response) {
6             email = user.getUsername();
7             name = user.getName();
8             id=user.getId();}
```

Após os dados recolhidos é extraída a imagem do utilizador através do seu ID no *Facebook*. A imagem é descarregada a partir do url `https://graph.\protect\unhbox\voidb@x\bgroup\def.{Facebook}\let\futurelet\@let@token\let\protect\relax\protect\edefn{it}\protect\afterassignment\edef10.95{10.95}\afterassignment\edef13.6pt{8.59023ptplus1.14534pt}\edef{1.5}\let1.5\def\size@update{\baselineskip13.6pt\relax\baselineskip\baselineskip\normalbaselineskip\baselineskip\setbox\strutbox\hbox{\vruleheight.7\baselineskipdepth.3\baselineskipwidth\z@}\let\size@update\relax}\xdef\T1/phv/m/it/10.95{\T1/phv/m/n/10.95}\T1/phv/m/it/10.95\size@update\enc@updateFacebook\egroup.com/<id_user>/picture` e posteriormente guardada num ficheiro, o que permite reutilizá-la sempre que necessário sem ter que voltar a fazer o seu download.

Após a extração do *e-mail* no caso da autenticação ser feita pela Google ou do *Facebook User* ID se a autenticação for efetuada pelo *Facebook*, é enviado ao servidor um pedido de autenticação/registo. A forma como os dados são armazenados no servidor encontra-se ilustrada na figura 4.10.

Figura 4.10: Lista de utilizadores autenticados na aplicação.

Se um destes parâmetros (*e-mail*/*Facebook User* ID) não existir é criada uma nova conta. Cada conta possui um tipo associado, que identifica o provider de forma a poder distinguir o tipo de autenticação efetuada pelo utilizador.

Se o utilizador já possuir um registo, é verificado se o mesmo possui favoritos adicionados, os quais serão enviados através do seu ID na resposta ao pedido de autenticação/registo. Os favoritos são armazenados no servidor conforme é ilustrado na figura 4.11.

User id	Id favoritos
jose.francisco.5437	1563
jose.francisco.5437	1562
jose.antunes.externo@infoportugal.pt	1006
jose.antunes.externo@infoportugal.pt	990
jose.antunes.externo@infoportugal.pt	989
jose.antunes.externo@infoportugal.pt	988
geno.spereira@gmail.com	1006
geno.spereira@gmail.com	990
geno.spereira@gmail.com	989

Figura 4.11: Favoritos do utilizador.

4.1.4.E Favoritos

Os favoritos na aplicação são objetos do tipo TrafficObject. A classe é constituída por quatro campos, o ID, a latitude, a longitude e o nome do favorito, neste caso, da via.

Os favoritos são constituídos por uma lista *TrafficObject* a qual se encontra ordenada e apresentada ao utilizador no seu menu lateral por ordem de proximidade. Para ordenar a lista foi necessário recorrer à posição atual do utilizador e ter em conta as variáveis lat e lon presentes em cada favorito.

Para obtenção da distância e consequentemente obter uma proximidade, foi usado o mé-

todo *static void distanceBetween (double startLatitude, double startLongitude, double endLatitude, double endLongitude, float[] results)*. Este método devolve uma distância em metros, entre duas posições o que possibilitou a interação sobre a lista de favoritos e organizá-los consoante a sua distância ao utilizador.

4.1.4.F Possíveis falhas de sincronização de favoritos

No processo de remoção/adição de favoritos além de o favorito ser removido da lista, é efetuada uma comunicação com o servidor onde é enviado o ID do favorito conjuntamente com o ID do utilizador (armazenado nas *Shared Preferences*).

Por vezes não é possível estabelecer conexão com o servidor, devido a vários problemas, como a falta de *internet* ou problemas de conexão ao servidor. Para suprir possíveis falhas foram desenvolvidos dois métodos:

- Sempre que existe uma falha na comunicação com o servidor, é guardado nas *Shared-Preferences* uma variável que indica que os favoritos não estão sincronizados. A sincronização é posteriormente efetuada quando o dispositivo consegue voltar a comunicar com o servidor;
- Todos os favoritos são guardados e acedidos através de um ficheiro criado em memória. A aplicação apenas recorre ao servidor quando o utilizador faz login e é necessário descarregar os favoritos, ou se existiu uma operação de remoção e adição que precisa de ser comunicada de forma a manter a integridade dos favoritos do utilizador.

Para a escrita no ficheiro foi usada a biblioteca *Jackson*, que converte objetos *Java* para JSON e executa também a operação contrária.

```
1 ObjectMapper mapper = new ObjectMapper();
2 File file = new File(_context.getFilesDir(), "favoritos");
3 if (!file.exists())
4     file.createNewFile();
5 mapper.writeValue(file, favoritos);
```

No código anterior é mostrado o processo de como os favoritos são guardados no ficheiro. É criado um objeto *ObjectMapper* e verificado se o ficheiro de favoritos já existe, caso contrário será criado. Posteriormente é passado no método *writeValue* o ficheiro e a lista de favoritos.

Para obter os favoritos do ficheiro basta referenciar o tipo de objeto que queremos ler e o ficheiro onde os favoritos estão guardados, como é mostrado no código a seguir:

```
File file = new File(_context.getFilesDir(), "favoritos");
List<TrafficObject> list = mapper.readValue(file, new TypeReference<List<
    TrafficObject>>(){});
```

4.1.4.G Hotspots

O ecrã de *Hotspots* é composto por uma lista e por uma *search box*, que permite a procura de elementos da lista, situada no topo do ecrã. Cada item presente na lista era objeto do tipo TrafficObject, possuindo os campos id, latitude, longitude e um nome.

Na *UI* é apresentado o nome da via, o estado da via e uma checkbox que permitia ao utilizador adicionar aquela via aos seus favoritos.

Para atribuir um estado à via foi necessário recorrer a um serviço externo, desenvolvido posteriormente pela InfoPortugal, o qual devolve um JSON com o estado das vias que possuam algum tipo de incidente. Nesse mesmo JSON também era possível receber eventos que não possuíam ID, mas estes foram descartados pois não estavam associados com nenhuma via.

Para manter o ecrã de hotspots com o estado de trânsito atual das vias foi necessário implementar uma thread que, de cinco em cinco minutos, fizesse um pedido ao servidor para consultar a informação.

Após a informação ser descarregada esta era guardada numa estrutura *Hash* que possui como chave o point_id e como valor correspondente o estado desse ponto dado pelo event_degree. Na figura 4.12 é possível ver como a informação estava estruturada.

Figura 4.12: JSON dos eventos de trânsito.

Apenas eram considerados os eventos que possuíssem um ID, ou seja, que estivessem referenciados e tivessem um estado de trânsito superior a um (ver tabela 4.1). Para colorir a lista foi usado como referência o ID de cada *TrafficObject* da lista. Se o ID fosse uma chave do HashMap construído anteriormente era obtido o seu valor e consoante o valor era atribuída uma cor ao elemento da lista. Se o ID não se encontrasse no *HashMap* era devolvido o valor 1, indicando que não existia trânsito naquela via. Após obter o estado de trânsito da via a mesma seria decorada com um *background* que o identificasse esse mesmo estado.

Para a criação dos Hotspots foi criada uma *ListActivity*. Uma *ListActivity* é uma classe filho da Activity cujo objetivo é mostrar ao utilizador uma Lista (*ListView*). Em suma, é uma *Activity* com alguns métodos para gerenciamento de listas, criada com o intuito de facilitar a criação de ecrãs com essa configuração, os quais são muito comuns nas aplicações *Android*.

Cor da via	Numero de estado da via		
	1	2	3
Verde	X		
Amarelo		X	
Vermelho			X

Tabela 4.1: Tabela de estados de via - Nível 1: Trânsito fluído; Nível 2: Trânsito intermédio; Nível 3: Trânsito elevado.

4.1.4.H Report

O Report foi criado através de uma *Activity*, funcionando assim como um ecrã novo. Sempre que o ecrã de *Report* é acedido é efetuada uma nova localização da posição do utilizador. Após esta ser detetada é construído um perímetro à volta da localização do utilizador a partir do qual o *Report* pode ser efetuado.

Para evitar um abuso no sistema, através de *reports* excessivos foi criada uma variável nas *Shared Preferences* do utilizador que indica a última hora que o utilizador tinha efetuou um *report*. Se o tempo do *report* for superior a 5 minutos e existir uma conexão com o servidor e o *report* é permitido.

Para o envio do *report* a aplicação envia o *User ID*, a latitude e longitude correntes da

posição do utilizador e uma descrição.

User id	Lat	Longitude	Desc	Created on
jose.francisco.5437	41.16685	-8.65156666667	Acidente	Aug. 31, 2013, 2 a.m.
jfantumes@gmail.com	41.1778624692	-8.68463845192	Obras	May 14, 2013, 10:17 a.m.

Figura 4.13: Reports efetuados por utilizadores.

A descrição era enviada consoante o tipo de evento que o utilizador selecionava quando fazia o *report*. A figura 4.13 ilustra a forma como os dados relacionados com um *report* se encontram armazenados no servidor.

4.1.5 Testes

Para testar a aplicação foram usados dois programas: o *Monkey Test* [13] e um plugin do *Eclipse* chamado *FindBugs* [15]. O *FindBugs* é uma ferramenta de análise estática, que examina as classes existentes no projeto procurando possíveis problemas durante o desenvolvimento. É um projeto criado pela Universidade de *Maryland*. O *FindBugs* é utilizado por projetos como o Java Server Faces (JSF), *Ebay* e *Google*.

A análise estática de código é uma análise automatizada realizada por um ferramenta (software) sem que seja preciso executar o programa a ser analisado. A ferramenta procura no código-fonte da aplicação erros ou prováveis causas de erros no programa. Os erros devem posteriormente ser verificados pelo programador, que deve decidir se o código necessita de ser ou não corrigido.

Os analisadores estáticos de código, como é o caso do *FindBugs*, permitem aos programadores detetar diversos erros nos estágios iniciais do processo de desenvolvimento de software.

O *FindBugs* trabalha, basicamente, com as seguintes categorias de *bug*:

- Correção: O código está a fazer algo que o programador não pretendia. Por exemplo, referenciar um *null pointer*;
- Más práticas: O código viola alguma boa prática;
- Erros de concorrência;
- Potenciais problemas de desempenho;
- Vulnerabilidade de código malicioso.

Esta ferramenta foi particularmente útil no desenvolvimento da aplicação, pois permitiu a construção de um código rigoroso e bem estruturado. Após concluído o projeto apenas foram

obtidos seis *warnings*, os quais, não interferiram com a performance e segurança da aplicação. Após a aplicação ser testada a nível de código foi também necessário testar a performance do sistema e da interface gráfica como um todo. Para efetuar esse teste foi usada a ferramenta *Monkey*, a qual corre a partir da linha de comandos e vem pré-instalada juntamente com o SDK do *Android*. O seu funcionamento consiste no envio de eventos aleatórios para o dispositivo efetuando um teste de stress da User Interface (UI). Os eventos são simulações de ações no sistema, como por exemplo: *KeyEvent*, evento que simula o clique de botões do telemóvel, como o *Home*, *Back*, escrever no teclado; *MotionEvent*, evento que simula toques no *touchscreen*; *FlipEvent*, simula a rotação do ecrã.

Se a aplicação falhar ou receber uma exceção de execução, o *Monkey* vai parar o seu funcionamento e reportar o erro na linha de comandos. Se a aplicação gerar um ANR, o *Monkey* também vai parar e reportar esse mesmo erro.

Ao correr este aplicativo em dois dispositivos diferentes nomeadamente um *Samsung Galaxy S3* e num *HTC Wildfire* os resultados foram positivos, não ocorrendo nenhum erro.

5

Conclusões

Ao desenvolver a aplicação procurou dar-se uma resposta a um problema diário e esta funcionar como um facilitador do quotidiano. Sendo uma aplicação personalizada e orientada para um mercado constantemente em expansão e onde encontramos diversas aplicações desprovidas de diferenciação, este projeto destaca-se pela simplicidade da interface e por responder às necessidades de todos aqueles que diariamente perdem o seu tempo no trânsito.

5.1 Objetivos realizados

O objetivo do estágio consistia na criação de uma aplicação móvel do estado de trânsito. Todas as funcionalidades pedidas foram implementadas e a aplicação encontra-se funcional e estável.

A realização e conclusão deste projeto subdividiu-se em 3 etapas:

- Desenvolvimento do servidor de trânsito;
- Desenvolvimento do servidor da aplicação;
- Desenvolvimento da aplicação.

5.2 Trabalho futuro

A aplicação deverá passar por um processo de revisão de layout, de forma a tornar-se mais apelativa. Apesar de se revelar uma boa alternativa à atual, penso que há certos aspetos visuais (cores,ícones) que poderiam ser melhorados tornando-a mais apelativa.

A integração da aplicação com o Facebook deve ser mais explorada, de forma a dar mais visibilidade à mesma. A partilha de eventos de trânsito através do Facebook ou de todos os que se registaram na aplicação seria muito benéfico, pois iria atrair diversas pessoas que não estão tão familiarizadas com o mercado das aplicações.

O servidor da aplicação deveria ser provido de uma maior segurança nas comunicações efetuadas com os utilizadores de forma a prevenir diversos tipos de ataque, sendo essencial num futuro próximo dotar o mesmo com um sistema de chaves para efetuar as comunicações.

Bibliografia

[1] “Buisness Insider,” http://www.businessinsider.com/history-of-android-2013-8?op=1 Acedido em 2013-07-01.

[2] “Canalys,” http://www.strategyanalytics.com/default?mod=pressreleaseviewr&a0=5403 Acedido em 2013-08-05.

[3] “Teleco,” http://www.teleco.com.br/promon/pbtr/Mobilidade_4Web.pdf Acedido em 2013-08-20.

[4] “Nielsen Norman Group,” http://www.nmgroup.com/articles/mobile-usability-update Acedido em 2013-09-03.

[5] “W3schools,” http://www.w3schools.com/webservices/ Acedido em 2013-02-11.

[6] “Mozilla Developer Network,” http://developer.mozilla.org/en-US/docs/JSON Acedido em 2013-04-11.

[7] “RESTful Web Services: The basics,” http://www.ibm.com/developerworks/webservices/library/ws-restful/ Acedido em 2013-01-25.

[8] “Google Map Layers.”

[9] “Map tile system,” http://msdn.microsoft.com/en-us/library/bb259689.aspx Acedido em 2012-11-15.

[10] “Open Street Maps,” http://www.openstreemap.org/en/1.5 Acedido em 2012-12-10.

[11] “Django Documentation,” http://docs.djangoproject.com/en/1.5 Acedido em 2013-01-12.

[12] “ActionBarSherlock,” http://actionbarsherlock.com Acedido em 2013-01-23.

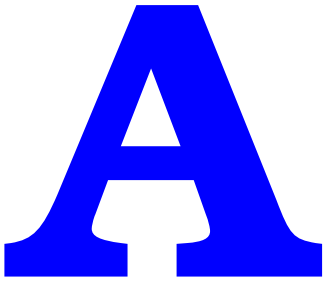
[13] “Android Developpers,” http://developer.android.com Acedido em 2013-03-03.

[14] “Object Oriented Design,” http://www.nmgroup.com/articles/mobile-usability-update Acedido em 2013-04-15.

[15] “FindBugs,” http://http://findbugs.sourceforge.net/ Acedido em 2013-02-05.

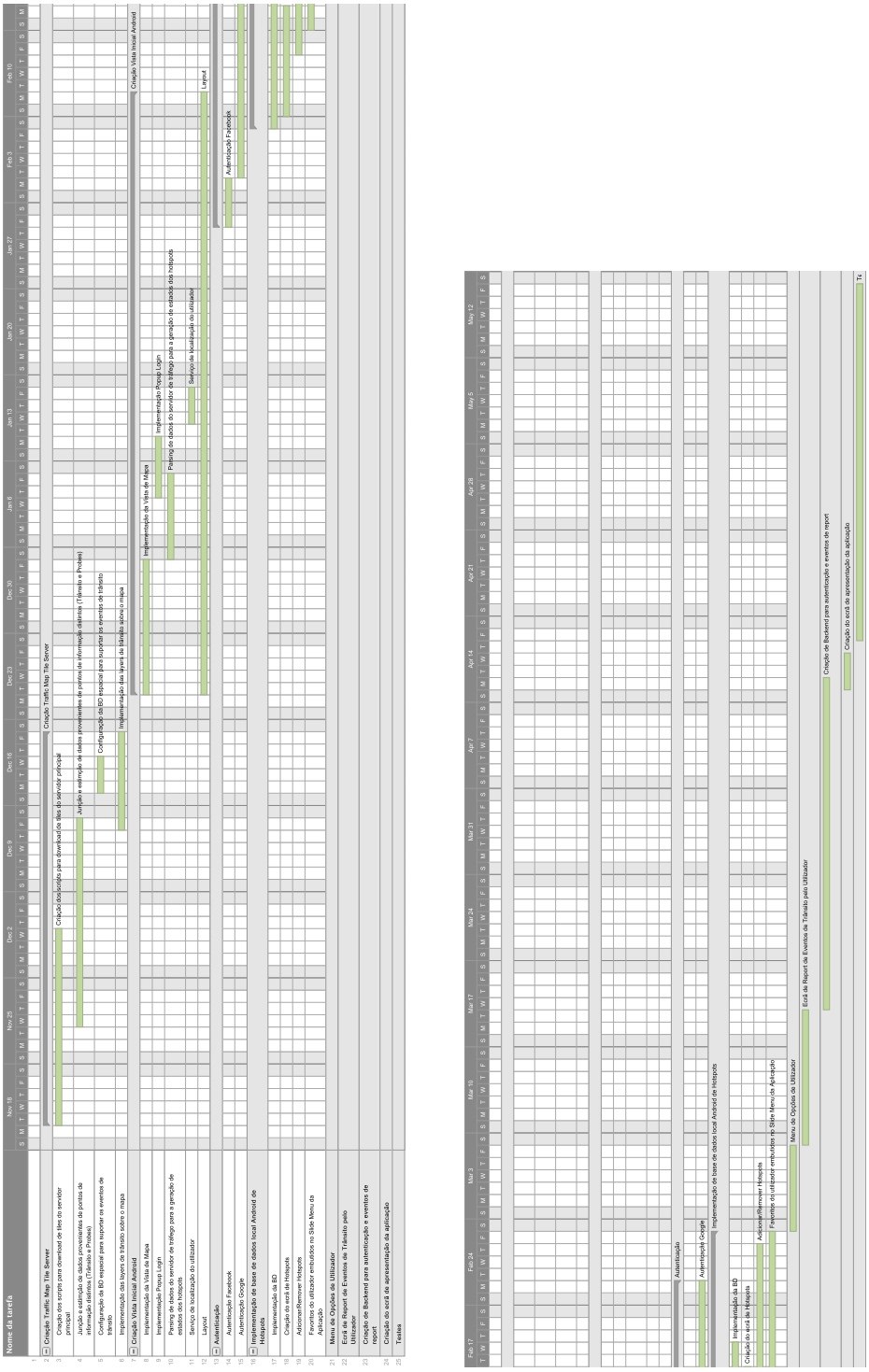
[16] “Mapnik,” http://github.com/mapnik/mapnik/wiki Acedido em 2012-11-05.

[17] “Jackson JSON processor,” http://jackson.codehaus.org/ Acedido em 2013-02-13.



Anexo A

	POINT_ID	NAME	LENGTH(m)	OPTIMAL_SPEED(km/h)	OPTIMAL_TRAVEL_TIME(s)	NUM_PASS	AVG_SPEED(km/h)	DELAY_TIME(s)	TRAVEL_TIME(s)	STATE
3	922.0	VCI (Bessa Leite - Francos/Matosinhos)	1282.09761090821	90.0	51	23	31	97	148	3
4	116.0	Portagens da Ponte 25 de Abril	1976.01344616651	114.2	62	19	38	125	187	3
5	737.0	A28 (IC1) (Leixões - Matosinhos (A4))	895.536195763255	120.0	26	17	23	114	140	3
6	735.0	A28 (IC1) (Matosinhos (A4) - Rotunda AEP)	1706.92334165105	100.0	61	15	22	218	279	3
7	926.0	VCI (Via Norte/Carvalhido - Francos/Matosinhos)	1741.97832210072	90.0	69	12	41	83	152	3
8	1910.0	EN10 (Arrentela - Aldeia de Paio Pires)	1874.34817510716	59.0769230769231	114	11	28	126	240	3
9	1557.0	Avenida A.E.P. (VCI - A28)	2126.76413461083	64.4444444444444	119	10	30	136	255	3
10	15.0	Rotunda da Boavista	650.185178885604	44.0	53	9	21	58	111	3
11	1009.0	Eixo Norte-Sul (Camarate - Túnel do Grilo)	736.892337046083	91.0	29	9	35	46	75	3
12	1039.0	Calçada de Carriche (Lumiar - Eixo Norte-Sul)	366.459685659201	91.0	14	9	39	19	33	3
13	1638.0	EN14 (Trofa - Ribeirão)	1753.46718753385	55.35	114	9	11	459	573	3
14	933.0	VCI (A3 - Antas)	1133.03581332984	90.0	45	8	38	62	107	3
15	1037.0	Calçada de Carriche (Eixo Norte-Sul - Telheiras)	555.349016499277	91.0	21	8	25	58	79	3
16	676.0	A25 (Carvoeiro - Talhadas)	10266.2918584896	120.0	307	8	48	462	769	3
17	1635.0	EN14 (Bougado - Vila do Coronado)	4065.45788509694	63.5	232	8	28	290	522	3
18	2145.0	IC8 (Cumeada - Cabeçudo)	4222.47643569168	91.0	167	7	41	203	370	3
19	2056.0	EN125 (Montemor - Quelfes)	1403.26173018493	61.8181818181818	82	6	27	105	187	3
20	621.0	A24 (Carvalhal - Arcas)	4746.11300585469	120.0	142	6	54	174	316	3
21	1329.0	IC16 [Sintra] (Terrugem - Mem-Martins)	692.720904631293	91.0	27	6	22	86	113	3
22	1041.0	Calçada de Carriche (Paço Lumiar - Lumiar)	418.011319474436	91.0	16	6	32	31	47	3
23	1903.0	EN10 (Cruz de Pau - Santa Marta)	1351.20749473331	69.75	70	6	24	132	202	3
24	954.0	Circunvalação (Monte dos Burgos - Norte Shopping)	2246.02682123344	60.4	134	5	25	189	323	3
25	1651.0	EN13 (A28 - Modivas)	887.78129667469	60.0909090909091	53	5	18	124	177	3
26	2279.0	Rua de Nossa Senhora de Fátima (Ramada Alta - Rotunda da Boavista)	591.438688341289	36.0	59	5	16	74	133	3
27	2375.0	Avenida António Augusto de Aguiar (Corte Inglês - Túnel do Marquês)	854.559947612486	44.0	69	5	19	92	161	3
28	1264.0	IP4 (Macedo de Cavaleiros - Vale de Prados)	4223.39623009645	91.0	167	5	34	280	447	3
29	1575.0	Avenida da República (General Torres - Serra do Pilar)	217.340129425841	44.0	17	5	17	29	46	3
30	1650.0	EN13 (Modivas - A28)	2047.51691960867	59.1785714285714	124	4	24	183	307	3
31	2432.0	Rua de Damião de Góis (Clube Ténis do Porto - Túnel de Faria Guimarães)	768.480481191912	44.0	62	4	20	76	138	3
32	905.0	A44 (Madalena - Coimbrões (IC2))	1197.48598548479	120.0	35	4	54	44	79	3
33	739.0	A28 (IC1) (Leça da Palmeira - Leixões)	792.104251631132	120.0	23	4	50	34	57	3
34	2326.0	Avenida da Liberdade (Praça do Marquês de Pombal - Praça dos Restauradores)	1116.6639030946	50.0	80	4	18	143	223	3
35	2456.0	Bessa Leite -> VCI	547.103282009811	44.0	44	4	18	65	109	3
36	1843.0	IC2 (Condeixa-a-Nova - Condeixa-a-Velha)	1261.81334868445	91.0	49	4	32	92	141	3
37	1373.0	Avenida de Ceuta (Cais de Alcântara - Ponte 25 de Abril)	390.714105144336	44.0	31	4	20	39	70	3
38	46.0	Ponte da Arrábida (Gaia - Porto)	882.517461790114	100.0	31	4	38	52	83	3
39	1493.0	EN117 (Alfragide - Cabos D Ávila)	1453.36243659342	70.8	74	4	30	100	174	3



(a) Parte I **(b)** Parte II

Figura A.2: Mapa de gantt.