INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Navegação *Turn-by-Turn* em Android

Relatório de estágio para a obtenção do grau de Mestre em Informática e Sistemas

Autor

Luís Miguel dos Santos Henriques

Orientação

Professor Doutor João Durães

Professor Doutor Bruno Cabral

Mestrado em Engenharia Informática e Sistemas

Navegação *Turn-by-Turn* em Android

Relatório de estágio apresentado para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Desenvolvimento de Software

Autor

Luís Miguel dos Santos Henriques

Orientador

Professor Doutor João António Pereira Almeida Durães

Professor do Departamento de Engenharia Informática e de Sistemas

Instituto Superior de Engenharia de Coimbra

Supervisor

Professor Doutor Bruno Miguel Brás Cabral

Sentilant

Coimbra, Fevereiro, 2019

Agradecimentos

Aos meus pais por todo o apoio que me deram,

Ao meu irmão pela inspiração,

À minha namorada por todo o amor e paciência,

Ao meu primo, por me fazer acreditar que nunca é tarde,

Aos meus professores por me darem esta segunda oportunidade,

A todos vocês devo o novo rumo da minha vida.

Obrigado.

Abstract

This report describes the work done during the internship of the Master's degree in Computer Science and Systems, Specialization in Software Development, from the Polytechnic of Coimbra - ISEC.

This internship, which began in October 17 of 2017 and ended in July 18 of 2018, took place in the company Sentilant, and had as its main goal the development of a turn-by-turn navigation module for a logistics management application named Drivian Tasks.

During the internship activities, a turn-by-turn navigation module was developed from scratch, while matching the specifications indicated by the project managers in the host entity. The system was later integrated into the main Drivian Tasks application, along with the addition of unique elements specifically developed to expand its already existing functionalities.

This report describes in detail all the stages of the project, from its planning phase, followed by its development phase, up to its final validation phase.

The described activities resulted in the construction of an independent navigation module, which may be integrated in any of Sentilant’s future Android applications. It also provided a substantial improvement on the mobile application of the Drivian Tasks product.

Keywords: Turn-by-turn; Navigation; GPS; Android

Resumo

O presente relatório descreve todo o trabalho efetuado durante o estágio de Mestrado em Informática e Sistemas, Especialização em Desenvolvimento de Software, do Instituto Superior de Engenharia de Coimbra.

Este estágio, com início a 17 de Outubro de 2017 e finalizado a 18 de Julho de 2018, teve lugar na empresa Sentilant, e teve como objetivo central o desenvolvimento de um módulo de navegação *turn-by-turn* para uma aplicação de gestão logística já existente: o Drivian Tasks.

Durante o decorrer das atividades de estágio, foi desenvolvido um módulo de navegação *turn-by-turn* completamente de raiz, correspondente às especificações indicadas pelos responsáveis do projeto na entidade de acolhimento. Este sistema foi posteriormente integrado na aplicação Drivian Tasks, em conjunto com a adição de elementos únicos, especificamente desenvolvidos para esta aplicação com a finalidade de expandir as suas funcionalidades.

Este relatório descreve em detalhe todas as fases do projeto, desde o seu planeamento, passando pela sua conceção, até à fase final de validação.

As atividades aqui descritas resultaram na construção de um módulo de navegação independente, integrável em qualquer aplicação Android que a empresa Sentilant possa desenvolver, em conjunto com uma melhoria significativa da aplicação móvel do produto Drivian Tasks.

Palavras-chave: *Turn-by-turn*; Navegação; GPS; Android

Índice

Índice de Figuras

Índice de Tabelas

Índice de Anexos

Acrónimos

API: *Application Programming Interface*. Trata-se do conjunto de funções, disponibilizadas por um sistema ou biblioteca, para que sistemas externos possam usar os seus recursos e funcionalidades.

CRUD: *Create, read, update, delete*. Refere-se às quarto operações básicas de uma base de dados: Criar, ler, atualizar e apagar, respetivamente.

DARPA: *Defense Advanced Research Projects Agency*. Agencia Norte-Americana de pesquisa militar.

GPS: *Global Positioning System*. Sistema de posicionamento que recorre a uma rede de satélites para inferir a posição geográfica de um equipamento..

ISEC: Instituto Superior de Engenharia de Coimbra.

JSON: *JavaScript Object Notation*. Um formato de dados que utiliza texto legível a humanos para transmitir dados entre sistemas.

RAM: *Random Access Memory* ou Memória de Acesso Aleatório refere-se ao tipo de memória utilizado primariamente por sistemas digitais, permitindo a leitura e escrita de dados durante a execução dos sistemas.

RESTful: *Representational State Transfer*. Convenção que define uma série de restrições a serem usadas em serviços *web*.

SICAE: Sistema de Informação da Classificação Portuguesa de Atividades Económicas.

SDK: *Software Development Kit*. Conjunto de ferramentas, bibliotecas e recursos digitais para desenvolvimento de *software*.

STOIC: *Spatial, Temporal and Orientation Information in Contested Environments*. Sistema de posicionamento geográfico inovador que recorre a sinais de referência de longo alcance e relógios extremamente precisos para inferir a posição de um equipamento.

1. Introdução

O presente relatório descreve todo o trabalho desenvolvido no âmbito da unidade curricular de Estágio ou Projeto Industrial do Mestrado em Informática e Sistemas: ramo de especialização em Desenvolvimento de Software, do Instituto Superior de Engenharia de Coimbra (ISEC). Este estágio foi realizado na empresa Sentilant durante o ano letivo de 2017 / 2018, tendo a duração de dois semestres.

O objetivo essencial do estágio refere-se ao desenvolvimento de um módulo de navegação *turn-by-turn* para um produto de *software* já existente: o Drivian Tasks. Embora este produto seja composto por um conjunto de aplicações de gestão logística interligadas, o foco essencial deste estágio é a sua vertente Android, uma aplicação utilizada pelos trabalhadores das várias empresas clientes para gerirem o seu dia de trabalho. É importante referir que parte do foco deste estágio foi alterado, e que a intenção inicial de implementar o módulo de navegação na vertente iOS da aplicação foi abandonada por razões que serão posteriormente explicadas neste documento. O resultado foi o desenvolvimento de funcionalidades mais completas e pertinentes para a vertente Android da aplicação.

Neste capítulo é apresentada a entidade acolhedora do estágio: a empresa Sentilant; assim como a entidade promotora do mesmo: o ISEC. Serão também apresentados os objetivos gerais e o plano de trabalhos previsto inicialmente. Por fim, será descrita a estrutura deste relatório.

1.1. Entidades Envolvidas

1.1.1. Entidade de Acolhimento

A entidade de acolhimento deste estágio é a empresa Sentilant (Sentilant, s.d.), uma empresa com o objetivo de desenvolver soluções de *software* disruptivas focadas em gestão logística e *Business Intelligence*. Situada na incubadora de empresas Instituto Pedro Nunes, em Coimbra, foi fundada por dois docentes da Universidade de Coimbra: O Professor Jorge Granjal e o Professor Bruno Cabral; sendo este segundo o orientador formal deste estágio na entidade acolhedora. Sendo uma empresa ainda em estado inicial, é composta, atualmente, por cinco funcionários e três estagiários.

Como estratégia de desenvolvimento, há uma separação clara das responsabilidades de cada funcionário. Cada empregado encontra-se alocado a uma parte específica de um projeto, podendo por vezes juntar-se a colegas para, em conjunto, desenvolverem um elemento específico. Geralmente os projetos são desenvolvidos de forma ágil (Cline, 2015; Tsui, Karam & Bernal 2014). A metodologia preferencialmente adotada pela empresa é o Scrum (Rubin, 2017) e, como tal, os seus funcionários são motivados a tomar a iniciativa de fazer as reuniões que considerem necessárias, idealmente seguindo os procedimentos previstos e fazendo reuniões diárias. O desenvolvimento dos projetos é acompanhado regularmente pelos responsáveis da empresa através de reuniões de revisão de cada *Sprint*, ou ciclo de desenvolvimento, assumindo estes o papel de Scrum Master.

Os projetos mais proeminentes desenvolvidos pela Sentilant até à data são o RISK.IT, um segundo projeto com três vertentes intituladas Drivian Tasks, Movismart e Entrega Drive, e um terceiro projeto, também subdividido em três variantes, denominadas Drivian, Autobud e Hit the Road.

1.1.1.1. RISK.IT

Trata-se de um projeto que visa avançar o estado da arte no que se refere à utilização de modelos de previsão e gestão de risco durante a condução. Este projeto, financiado pela União Europeia e desenvolvido em parceria com a empresa Transportes Paulo Duarte, pretende colocar ao serviço das empresas uma ferramenta que permita planear viagens e calcular rotas com o mínimo risco possível. Como tal, os fatores de risco são definidos mediante a perspetiva das empresas. O propósito último deste projeto é aumentar o nível de eficiência e segurança operacional das empresas.

1.1.1.2. Drivian Tasks, Movismart, Entrega Drive

Refere-se a um projeto com várias variantes, direcionadas a diferentes clientes, cujo objetivo é auxiliar a gestão logística das empresas clientes e aumentar a eficiência das suas operações. Este projeto permite limitar o desperdício de recursos de uma empresa, gerir a ordem de execução das tarefas dos seus operacionais e providenciar um centro de gestão de operações virtual. O resultado direto da utilização desta ferramenta é, idealmente, uma gestão eficiente dos recursos humanos e materiais de cada empresa e, consequentemente, o aumento dos seus lucros.

1.1.1.3. Drivian, Autobud, Hit the Road

Aludem a uma aplicação para *smartphone*, também com várias versões destinadas a diferentes públicos-alvo, com o propósito de inferir com a maior precisão possível o nível de segurança e economia durante a condução. Tirando partido dos sensores já presentes no dispositivo, tais como o giroscópio e o acelerómetro, é atribuída uma percentagem a cada um destes fatores. Estes projetos pretendem informar os seus utilizadores sobre as medições efetuadas, permitindo que tomem decisões de acordo, idealmente aumentando a segurança e economia da sua condução.

1.1.2. Entidade Promotora

O Instituto Superior de Engenharia de Coimbra (ISEC, s.d), doravante designado ISEC, tem como missão cultivar e difundir cultura, ciência e tecnologia através da promoção da investigação e da oferta de formação de nível superior aos seus alunos. Esta oferta formativa tem com o objetivo ensinar os seus alunos a exercer atividades profissionais no campo da Engenharia com total competência e dinamismo.

O ISEC tem como visão permanecer uma referência de excelência no campo do ensino das mais variadas vertentes da Engenharia, sendo reconhecido nacional e internacionalmente por serviços de qualidade e relevância social. Tem também como missão ser um parceiro privilegiado de organizações empresariais através da orientação eminentemente prática e profissionalizante dos seus alunos, fundamentada no desenvolvimento de um vasto corpo de conhecimento teórico. Estas duas vertentes teórica e prática, com todo o seu rigor e recetividade à inovação, são expressas em todas as atividades exercidas pelo ISEC.

Os valores fundamentais pelos quais se rege o ISEC são a cidadania, a qualidade, a busca constante da valorização, motivação e atualização pedagógica, científica e tecnológica dos seus recursos, o bom relacionamento e a disponibilidade para com os estudantes e as organizações suas parceiras e a dedicação ao desenvolvimento social e económico da região onde está inserido.

1.2. Objetivos Gerais

O estágio descrito neste documento tem como objetivo essencial desenvolver um módulo de navegação *turn-by-turn*. Este módulo deverá ser posteriormente integrado numa aplicação de gestão logística já existente para Android e iOS intitulada Drivian

Tasks. Espera-se que o módulo a desenvolver ofereça não só as funcionalidades típicas de um sistema de navegação, como também tire partido das suas características únicas para estender as funcionalidades do próprio Drivian Tasks de forma incomparável. O objetivo será o acréscimo de elementos singulares, inexistentes em aplicações da mesma natureza que, de outro modo, não seriam possíveis de desenvolver. Não se trata, portanto, de um simples módulo de navegação, mas sim de uma completa expansão das funcionalidades do Drivian Tasks.

Como aplicação de gestão logística, o Drivian Tasks para Android engloba inúmeras funcionalidades que permitem ao seu utilizador saber as tarefas alocadas para o seu dia de trabalho. Permite também gerir estas mesmas tarefas ao definir o seu estado atual de execução, saber qual a natureza de cada uma, quais os materiais por ela requeridos, e colecionar evidências do seu cumprimento. Esta aplicação móvel encontra-se interligada com uma plataforma *web*, também ela parte do produto Drivian Tasks, na qual um gestor da empresa ou o seu responsável de logística poderão definir que tarefas são necessárias para cada dia e qual a natureza das mesmas, alocá-las aos seus trabalhadores e, posteriormente, receber feedback da sua execução em tempo real. O objetivo central do produto Drivian Tasks é aumentar drasticamente a eficiência dos trabalhadores no terreno, reduzindo custos e desperdícios de tempo e recursos e, em última instância, aumentar o lucro das empresas suas utilizadoras.

O módulo de navegação *turn-by-turn* desenvolvido durante este estágio deverá integrar as funcionalidades de gestão de tarefas de forma completa, com o intuito de permitir ao utilizador cumprir todas as atividades do seu dia de trabalho sem sair do modo de navegação. Deverá também englobar os ideais de eficácia e eficiência preconizados pelo Drivian Tasks, guiando os utilizadores pelas melhores rotas possíveis, e tirando partido do facto de apresentar um mapa para integrar algumas funcionalidades adicionais e inovadoras.

Os objetivos gerais aqui definidos refletem os inicialmente apresentados na proposta de estágio disponibilizada em anexo (Anexo A), com exceção da implementação do módulo de navegação para a versão iOS do Drivian Tasks. Dado que a implementação do módulo para iOS não traria um acréscimo de valor relevante para a empresa Sentilant, devido ao reduzido número de utilizadores com este sistema operativo, os responsáveis da empresa tomaram a opção de abdicar da sua implementação. No

entanto, não se tratou de um simples abandono de uma das vertentes do módulo de navegação, mas sim de uma opção estratégica. Com efeito, durante o decorrer do estágio, surgiu a possibilidade de construir um módulo de navegação de raiz, por oposição à simples utilização de um *Software Development Kit* (SDK) de navegação e posterior desenvolvimento das restantes funcionalidades. Esta implementação de raiz permitiu construir um sistema de navegação inteiramente proprietário, com custos extremamente reduzidos, e com a possibilidade de ser integrado em projetos futuros. Uma proposta de maior valor, segundo os responsáveis da empresa. O desenvolvimento do módulo de navegação relegou-se apenas à sua implementação em Android.

1.3. Plano de Trabalhos

O plano de trabalhos inicialmente apresentado na proposta de estágio (Anexo A) contempla as seis seguintes tarefas:

- ***Tarefa 1 (T1)*** – *Definição de requisitos* – Definição de requisitos funcionais e não-funcionais;
- ***Tarefa 2 (T2)*** – *Estado da arte* – Estudo das tecnologias e técnicas para desenvolver aplicações de navegação *turn-by-turn* em Android e iOS;
- ***Tarefa 3 (T3)*** – *Programação* – Desenvolvimento, validação e verificação do código fonte para a componente de navegação;
- ***Tarefa 4 (T4)*** – *Integração* – Integração da componente desenvolvida com as aplicações de iOS e Android do sistema de gestão operacional;
- ***Tarefa 5 (T5)*** – *Validação* – Avaliação e validação da implementação;
- ***Tarefa 6 (T6)*** – *Escrita do relatório* – Escrita do relatório final do Estágio.

O período de execução de cada uma destas tarefas encontra-se devidamente alocado durante o decorrer do estágio, resultando na proposta de calendarização apresentada na Figura 1.

	Meses									
Tarefas	**Oct-17**	**Nov-17**	**Dec-17**	**Jan-18**	**Feb-18**	**Mar-18**	**Apr-18**	**May-18**	**Jun-18**	**Jul-18**
T1										
T2										
T3										
T4										
T5										
T6										

Figura 1: Calendarização das Tarefas de Estágio

Durante o decorrer do estágio, e como esperado, o plano de trabalhos e a calendarização inicialmente prevista sofreram algumas alterações. Como já exposto neste relatório, a implementação do sistema para iOS foi abandonada. Do mesmo modo, a calendarização em si não foi respeitada com absoluto rigor, tendo sido adaptada quando necessário devida a tarefas que, por variados motivos, foram atrasadas ou adiantadas. No entanto, de uma forma geral, a execução das tarefas cumpriu a previsão inicialmente estabelecida.

1.4. Estrutura do Relatório

Sendo a secção atual uma secção introdutória, nela é descrito todo o contexto de estágio, a sua organização e os seus objetivos gerais. Seguidamente, na secção de "Descrição do Problema", é apresentado o produto Drivian Tasks, o problema a solucionar, assim como as questões inicialmente previstas para o desenvolvimento do módulo de navegação *turn-by-turn*.

Após a descrição do cenário inicial, é exposta a investigação feita no contexto de "Estado da Arte". Nessa secção são descritas as várias tecnologias consideradas para a implementação do projeto, assim como a investigação feita sobre produtos similares já existentes.

Na secção seguinte, intitulada "Proposta de Abordagem", é apresentada a tecnologia de navegação selecionada para o desenvolvimento do projeto, assim como as razões que levaram à sua escolha. É também exposta a proposta da nova arquitetura de alto nível para o Drivian Tasks.

Após definidas as questões iniciais, é apresentada a secção de "Levantamento de Requisitos", onde são enumerados os requisitos funcionais e não funcionais idealizados para o sistema, assim como os procedimentos tomados para a sua validação.

A secção seguinte, "Estudo de Alternativas de Implementação", tira partido da definição dos requisitos para apresentar o estudo feito sobre duas alternativas de implementação possíveis: uma implementação do sistema com recurso a um SDK de navegação, ou uma implementação de todo o sistema de navegação de raiz. As implicações de cada uma destas abordagens são igualmente descritas. A sua viabilidade é posteriormente testada através de aplicações desenvolvidas justamente com o propósito de averiguar as limitações de cada uma das ferramentas selecionadas. As aplicações de teste, assim como a apresentação das mesmas à equipa da Sentilant, são descritas na secção intitulada "Teste de Componentes".

A partir do estudo da viabilidade de cada uma das alternativas de implementação, foram tomadas decisões sobre o desenvolvimento do módulo de navegação. Essas decisões encontram-se expostas na secção “Implementação”, assim como a descrição detalhada de todo o sistema. Esta descrição foi feita com recurso ao modelo C4 (C4 Model, s.d.). É igualmente apresentada uma descrição dos elementos desenvolvidos e considerados mais relevantes para a aplicação, assim como das dificuldades e respetivas soluções encontradas durante a fase de implementação do sistema.

Após descrita toda a fase de desenvolvimento, é apresentada a secção de validação intitulada "Validação da Solução Implementada", onde constam os testes feitos ao sistema, assim como todo o racional sobre os mesmos.

Na última secção, "Conclusões e Trabalho Futuro", é descrita a apresentação final do módulo de navegação feita à equipa da Sentilant, em conjunto com uma avaliação pessoal de todo o trabalho efetuado durante o estágio e uma breve exposição de possíveis melhorias a fazer ao sistema.

2. Descrição do Problema

O atual capítulo pretende descrever de forma geral o problema central do estágio, assim como elucidar o leitor sobre as questões essenciais envolvidas na sua execução. Nele será exposto todo o contexto apresentado no início do decorrer das atividades. Será apresentada a descrição do produto Drivian Tasks transmitida pelos orientadores do projeto na entidade de acolhimento, em conjunto com os problemas iniciais e a sua visão para o desenvolvimento do módulo de navegação em questão, o objetivo central deste estágio.

2.1. Descrição do Drivian Tasks

O presente estágio tem como foco central a expansão das funcionalidades de uma aplicação de *business intelligence* e gestão logística: o Drivian Tasks. Este produto é apresentado ao cliente como a reunião de duas aplicações chave:

A primeira vertente do Drivian Tasks refere-se a uma plataforma *web*, através da qual um gestor ou responsável de logística de uma empresa pode gerir a sua carteira de clientes e coordenar tarefas alocadas aos seus trabalhadores no terreno. Ao agendar novas tarefas, tem a possibilidade de definir o seu local e que elementos são necessários à sua realização. Pode também verificar estatísticas e informações referentes à sua execução ou referentes a outros elementos pertinentes, como por exemplo os níveis de segurança e economia da condução dos seus trabalhadores. Através da plataforma *web*, um gestor pode ainda acompanhar os seus funcionários no terreno em tempo real ou até gerir toda a frota de veículos da sua empresa. Todas estas funcionalidades têm como objetivo proporcionar dados à equipa de gestão da empresa para suportar decisões verdadeiramente informadas sobre o *modus operandi* da mesma, auxiliar a gestão dos seus vários recursos, humanos e materiais, e, em última instância, aumentar a eficiência do trabalho da empresa, prevenindo custos desnecessários e, consequentemente, aumentando o seu lucro.

A segunda componente do Drivian Tasks consiste numa aplicação móvel para Android e iOS, que permite aos seus utilizadores, nomeadamente os funcionários no terreno, gerirem as várias tarefas que têm alocadas para o seu dia de trabalho. Através desta aplicação, um utilizador consegue obter informações sobre o agendamento de cada

tarefa, os contactos dos clientes que lhe estão associados e quais os materiais necessários para a sua execução. Ao mesmo tempo tem também a possibilidade de colecionar evidências do seu trabalho, tais como assinaturas de clientes ou fotografias das suas atividades. Esta aplicação permite também ao utilizador recolher estatísticas sobre os níveis de segurança e economia da sua condução e anotar que despesas foram feitas em nome da empresa. Dependendo das permissões definidas por cada empresa para os seus funcionários, um utilizador da aplicação móvel pode ainda ter a liberdade de adicionar tarefas ao seu dia de trabalho. Estas novas tarefas serão, por sua vez, devidamente registadas e as suas informações serão integradas nos dados estatísticos da empresa.

Espera-se que o Drivian Tasks se torne uma ferramenta de suporte logístico imprescindível para os seus clientes, oferecendo funcionalidades que facilitam a organização das tarefas da sua empresa e a otimização do seu fluxo de trabalho.

2.2. Descrição Geral do Problema

Para aumentar verdadeiramente a eficiência dos trabalhadores no terreno, cujas funções implicam realizar várias viagens por dia, é essencial que o produto Drivian Tasks ofereça a possibilidade de guiar o utilizador entre as suas várias tarefas pelos percursos considerados mais eficientes. Por esse motivo, é imperativo que a sua componente móvel integre um modo de navegação *turn-by-turn* que forneça indicações precisas e pertinentes ao utilizador durante as suas viagens, enquanto o acompanha a cada momento da sua condução. Este modo de navegação não deve descurar, mas até expandir, as funcionalidades de organização logística essenciais ao Drivian Tasks no que se refere à ordem das tarefas, ao respeito pelo seu agendamento e à organização de um dia de trabalho completo.

O desenvolvimento e integração desse módulo de navegação *turn-by-turn*, bem como a criação de novas funcionalidades apenas possíveis num módulo desta natureza, são o âmbito essencial deste estágio.

2.3. Navegação *Turn-by-Turn*

"Navegação t*urn-by-turn"*, ou "navegação detalhada", alude a um sistema auxiliar de condução cujo objetivo é guiar o utilizador por rotas ou percursos, através da

disponibilização contínua de instruções. As instruções poderão ser apresentadas por imagem e/ou por voz simulada eletronicamente, e referem-se a nomes de ruas, locais para os quais o utilizador deve guiar, onde deve virar, de que forma o deve fazer, e a que distância se encontra de cada momento de decisão na sua rota.

Idealmente o sistema apresenta ao utilizador o melhor percurso possível para chegar ao seu destino, sendo continuamente atualizado para englobar fatores variáveis como trânsito, acidentes, condições de via, entre outros. O cálculo deste percurso é baseado no conhecido "Problema do Caminho Mínimo" (Ahuja et al., 2011; Cherkassy, Goldberg, Radzik, 1993), que pretende inferir qual o caminho entre dois pontos que melhor corresponda a um conjunto predeterminado de critérios. Estes critérios podem incluir, por exemplo, o cálculo do percurso com a distância mais curta, do percurso mais económico ou do percurso mais rápido, entre outros. Este cálculo é efetuado com recurso a grafos e algoritmos de exploração e procura de caminho como os conhecidos A* ou o algoritmo de Dijkstra (Koszalka, Koszalka & Kasprzak, 2015).

De uma forma geral, a grande maioria destes sistemas comercialmente disponíveis inferem a posição de um equipamento num espaço geográfico através de GPS (*Global Positioning System*), sendo alguns auxiliados por sensores de inércia para os casos em que o sinal GPS se encontra intermitente ou mesmo inexistente.

Atualmente existe um vasto leque de produtos e aplicações que permitem integrar este sistema em veículos de diversas naturezas. Este tipo de sistemas encontra-se disponibilizado para dispositivos móveis como *smartphones* ou painéis de controlo de veículos, tomando a forma de aplicações, ou integrados em equipamentos autónomos especificamente concebidos para auxiliar os utilizadores durante a condução.

2.4. Identificação dos *Stakeholders*

Assim que foi dado início ao estágio, em conjunto com a descrição do Drivian Tasks, foram também apresentadas as várias partes interessadas, ou *stakeholders*, da aplicação móvel, já identificadas previamente durante o desenvolvimento inicial da aplicação. O objetivo foi determinar padrões para os utilizadores deste produto para guiar o desenvolvimento do módulo de navegação. Identificar as necessidades de cada um destes acionistas permitiu criar uma aplicação que, de forma harmoniosa, corresponde às necessidades de cada um.

Os *stakeholders* identificados para a aplicação móvel são:

2.4.1. Operacional

O Operacional tem acesso à aplicação móvel para o apoiar durante as suas viagens. O Operacional tem a liberdade de tomar opções sobre os seus percursos e sobre as suas tarefas, dentro das restrições impostas pelo administrador. Poderá aceder a opções que lhe permitem realizar o seu trabalho de forma eficiente e eficaz.

2.4.2. Administrador

O Administrador tem acesso à plataforma *web*, na qual poderá tomar decisões, definir restrições e selecionar opções com impacto direto no funcionamento da aplicação móvel.

2.5. Requisitos Gerais

Antes de iniciar a fase de análise, planeamento e preparação, ou *pre-game,* segundo a metodologia Scrum, foram também apresentadas as intenções que os responsáveis da Sentilant tinham para o módulo de navegação. Desse modo, foi possível antever algumas questões centrais ao seu desenvolvimento. São elas:

- A integração do módulo de navegação num sistema complexo já existente (secção 2.5.1);
- O desenvolvimento de uma aplicação e correspondente arquitetura de forma escalável, que contemple a aquisição de futuros novos clientes do Drivian Tasks (secção 2.5.2);
- A possível gestão das permissões dadas aos utilizadores sobre a sua capacidade de utilizar determinadas funcionalidades da aplicação (secção 2.5.3);
- A expansão das funcionalidades já presentes no Drivian Tasks (secção 2.5.4);
- A implementação de um sistema que funcione inteiramente *offline*, para que o utilizador não perca a navegação caso não tenha acesso à *Internet* (secção 2.5.5);
- A possibilidade de construir o módulo de navegação de raiz, criando assim uma solução proprietária que poderá ser utilizada em futuros projetos da empresa (secção 2.5.6);
- A utilização de mapas e grafos devidamente atualizados (secção 2.5.7);

- O desenvolvimento de funcionalidades de forma extremamente eficiente, que tenha em conta a duração da bateria do dispositivo móvel (secção 2.5.8).

2.5.1. Integração no Drivian Tasks

Uma das particularidades mais proeminentes do projeto em questão é a integração do módulo de navegação *turn-by-turn* num sistema complexo já existente. Como tal, deve ter em conta as suas várias particularidades. Em primeiro lugar, a integração deste módulo não poderá trazer funcionalidades contraditórias às já presentes no produto atual. O módulo de navegação deverá também respeitar a arquitetura geral já definida para o Drivian Tasks. Esta arquitetura segue uma lógica servidor - cliente com três níveis: cliente, servidor e base de dados, sendo utilizada uma RESTful API (*Application Programming Interface*) para estabelecer a comunicação entre os clientes e o servidor, cruzando dados em formato JSON.

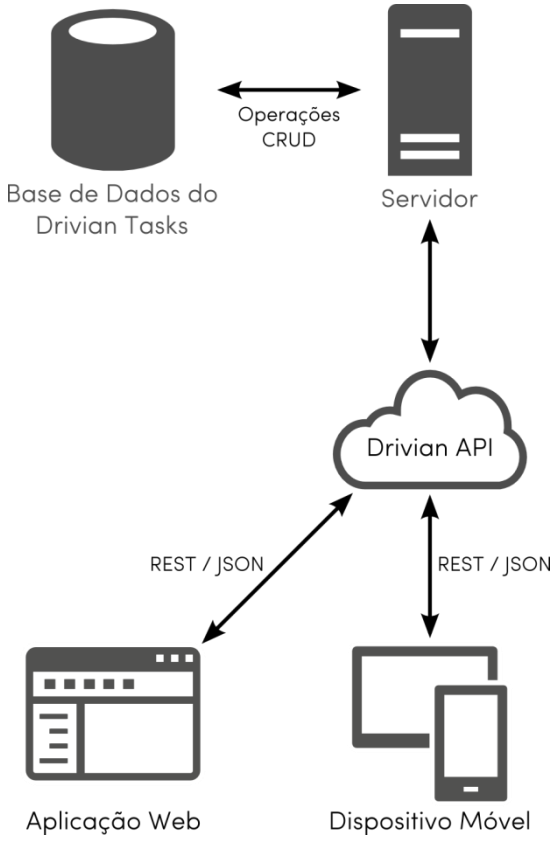


Figura 2: Arquitetura de Alto Nível do Drivian Tasks

2.5.2. Escalabilidade

Sendo o Drivian Tasks uma aplicação para empresas, o módulo de navegação a desenvolver deverá contemplar a possibilidade de o seu número de utilizadores

aumentar gradualmente. Como tal, há dois fatores que deverão ser tomados em conta ainda antes da fase de escrita do código fonte. Em primeiro lugar, a sua arquitetura deverá estar definida de forma a garantir a escalabilidade da aplicação. Dada a necessidade de integrar harmoniosamente o módulo de navegação na arquitetura já existente do Drivian Tasks, isto deverá ser alcançado sem grandes dificuldades visto a sua arquitetura estar já definida com essa mesma intenção. Em segundo lugar, deverão ser tomados em conta os custos associados à utilização de serviços externos, de terceiros, como APIs ou SDKs. Deverão ser utilizados os serviços menos dispendiosos, que assegurem a qualidade do serviço, ao mesmo tempo garantem a disponibilização das funcionalidades da aplicação na totalidade. A intenção é minimizar o impacto que estes fatores terão no preço final do Drivian Tasks, apresentado aos clientes da Sentilant.

2.5.3. Gestão de Permissões dos Utilizadores

Dadas as funcionalidades de gestão de tarefas a integrar no módulo de navegação, este deve oferecer a capacidade de ser personalizável e adaptável aos vários modelos de negócio contemplados pelo Drivian Tasks. Como tal, deve apresentar ou esconder opções mediante as permissões definidas para os trabalhadores na aplicação web.

Algumas empresas requerem que o utilizador cumpra uma ordem pré-estabelecida para as suas tarefas, com a calendarização e as horas bem definidas para cada uma. Este modelo contrasta com o de outras organizações que, por sua vez, oferecem ao trabalhador no terreno a liberdade de escolher a ordem pela qual desempenha as suas tarefas. Em acréscimo, estas segundas podem ou não permitir que os seus trabalhadores criem tarefas facilmente no momento e de forma imediata, para que estas fiquem devidamente registadas e associadas a uma localização à sua escolha.

Embora existisse a intenção de aplicar esta gestão de permissões desde o início do projeto, até ao final do estágio não foi disponibilizada na API do Drivian a possibilidade de conhecer as permissões alocadas aos trabalhadores. Como tal, optou-se por desenvolver o sistema de forma geral, de forma a corresponder a todos os modelos de negócio possíveis. No entanto o código foi estruturado para que, assim que estiver disponível, seja extremamente simples integrar esta possibilidade, e permitir ativar ou desativar as funcionalidades da aplicação móvel mediante as várias permissões.

2.5.4. Expansão das Funcionalidades

Desde o início do projeto, os responsáveis da empresa demonstraram interesse em desenvolver algumas funcionalidades particulares para o módulo de navegação.

A mais proeminente destas funcionalidades refere-se a uma opção através da qual um utilizador da aplicação móvel do Drivian Tasks, ou seja, um funcionário no terreno, consiga saber que estabelecimentos de interesse estão perto da sua área ou do seu destino. A intenção é que os possa visitar durante os momentos desocupados entre as suas tarefas para, potencialmente, adquirir novos clientes para a sua empresa.

Este requisito, relevante para empresas orientadas para *business to business*, requer que seja desenvolvido um algoritmo que permita identificar quais os potenciais clientes ao longo de um percurso ou numa determinada área, com o propósito de apresentar ao utilizador várias empresas que poderá visitar e com as quais poderá estabelecer negócio. O cálculo do valor de pertinência destes potenciais clientes depende, obviamente, da natureza do negócio da empresa, e é o alvo essencial do desenvolvimento deste algoritmo de ordenação.

Outra funcionalidade idealizada para o módulo de navegação foi a possibilidade de permitir aos funcionários criarem novas tarefas durante o desempenho das suas funções. Até ao momento, o Drivian Tasks permitia aos seus utilizadores criar novas tarefas associadas a uma morada específica de um dos seus clientes. Estes clientes teriam de já estar previamente discriminados na base de dados da empresa, pelo que a criação de tarefas estava limitada às suas moradas.

Pelo facto de integrar um mapa, o módulo de navegação traz a possibilidade de criar novas tarefas em qualquer local, sem as alocar a uma morada de um cliente já existente, alocando-as sim, de forma visual, a uma posição. Torna-se possível criar novas tarefas em qualquer lugar, associando coordenadas latitude e longitude às mesmas, e estas, por sua vez, são apresentadas no mapa. Em acréscimo, a criação destas tarefas é extremamente facilitada pelo facto de o utilizador não ter de percorrer vários menus para criar novas tarefas, associando-as à morada de um cliente. Com o mapa, ele poderá simplesmente pressionar o local onde pretende que a tarefa seja criada, e será apresentado um formulário parcialmente preenchido com alguns elementos, como a morada correspondente, por exemplo.

A construção do módulo de navegação deverá também ter a possibilidade de integrar informações de trânsito e condições das vias nos percursos apresentados ao utilizador. Estas informações devem ter impacto no cálculo dos percursos, sendo sempre ser apresentado o percurso considerado mais rápido. Do mesmo modo, devem também ser apresentados ícones no mapa correspondentes a eventos nas vias como trânsito, obras, acidentes, entre outros, para que o utilizador tome conhecimento do que vai encontrar durante a sua viagem, tal como se sucede na aplicação Waze, dada como exemplo para esta funcionalidade. Igualmente, a cor das linhas do percurso devem ser descritivas do nível de trânsito atualmente presente nos vários trechos de estrada.

Embora existam mais funcionalidades idealizadas pelos responsáveis da empresa para o módulo de navegação, estas não são consideradas complexas o suficiente para apresentar nesta secção do relatório. Alguns exemplos referem-se à possibilidade de realizar chamadas ou enviar mensagens-tipo a clientes, ou adicionar complementos como evidências e anexos às várias tarefas, e encontram-se expostas na secção de definição dos requisitos deste relatório.

2.5.5. Funcionamento *Offline*

Desde o início do projeto que os responsáveis da Sentilant expressaram interesse em que o módulo de navegação incluísse a possibilidade de funcionar inteiramente *offline*. Este requisito surgiu pelo facto de, muitas vezes, o utilizador se encontrar em zonas sem acesso à *Internet*. Neste sentido, o sistema foi desenhado com a intenção de garantir todas as funcionalidades essenciais de forma autónoma e inteiramente *offline*. Estas funcionalidades incluem, obviamente, o cálculo e ajuste de rotas, mas não só. O sistema oferece a possibilidade de guardar temporariamente as alterações referentes à gestão das tarefas, alterações que deverão ser comunicadas ao servidor assim que o acesso à *Internet* seja reestabelecido.

Embora não seja considerada uma questão inicialmente prevista, e sim apenas depois de tomada a decisão de enveredar por uma implementação de raiz para o módulo de navegação, como já referido anteriormente neste relatório, o requisito de funcionamento *offline* induziu a escolha de uma biblioteca de mapas para Android, a biblioteca Open Science Maps V™. Esta biblioteca será apresentada posteriormente neste documento, e foi escolhida precisamente por abdicar de acesso à *Internet* para apresentar os mapas ao utilizador. Como tal, foi necessário também desenvolver um sistema de transferência de

mapas para o dispositivo móvel, assim como preparar mapas e correspondentes grafos e disponibilizá-los num servidor para que sejam transferidos pelos utilizadores. Os detalhes destes procedimentos serão posteriormente descritos nas secções apropriadas deste relatório.

2.5.6. Desenvolvimento de um Sistema Proprietário

Embora também não estando inicialmente prevista, ainda durante a fase de preparação do projeto, surgiu a possibilidade de implementar o sistema de navegação totalmente de raiz.

Apesar de já existirem algumas soluções comerciais que permitem integrar um sistema de navegação completo numa aplicação Android ou iOS, nenhuma das soluções disponíveis permite a sua integração de forma gratuita para aplicações com fins comerciais. Como tal, os custos associados às ferramentas disponíveis tendem a acumular à medida que o número de utilizadores ou as suas respetivas necessidades aumentam, o que compromete a escalabilidade do negócio e condiciona o preço do produto final.

Não sendo, na sua génese, um problema inovador, dada esta possibilidade e antes da fase de implementação, foi tomada a opção de desenvolver um sistema de navegação de raiz, com o mínimo de custos associados, que apresente qualidade e funcionalidade equiparáveis ou superiores aos oferecidos pelas soluções de navegação atualmente disponíveis no mercado, com as quais vai competir. Enveredar por esta decisão permitiu criar uma solução proprietária que poderá ser utilizada em futuros projetos da empresa.

Apesar da referida presença de várias opções de navegação disponíveis no mercado, desconhecemos a existência um algoritmo definitivo, bem estabelecido e de conhecimento público, que faça o acompanhamento da condução ao longo de uma rota. Um dos desafios impostos à construção do módulo de navegação *turn-by-turn* foi o de desenvolver este sistema, que não só acompanha o utilizador durante o seu percurso, como também tem a capacidade de inferir quais os momentos pertinentes para a apresentação de instruções, também estas por sua vez apropriadas e relevantes, ao mesmo tempo que decide quando deve ajustar ou recalcular a rota atual, e de que forma este ajuste é feito. Dada a possibilidade de redução de custos, e atendendo ao facto de que o cálculo das rotas é feito, em parte, com recurso a um servidor, e visando a

possibilidade de este serviço ser contratado a terceiros (o que é a necessário para poder incluir informações de trânsito na rota), houve também a preocupação de minimizar o número de pedidos e ajustes de percurso sem prejudicar a qualidade da navegação.

As descrições de todas as particularidades deste sistema, assim como os vários algoritmos envolvidos, serão expostas nas secções apropriadas deste relatório.

2.5.7. Mapas e Grafos Atualizados

Tratando-se de uma aplicação de navegação, é extremamente importante que sejam disponibilizados aos utilizadores mapas atualizados, assim como os grafos correspondentes para o cálculo das várias rotas de forma *offline*.

Não seria benéfico, por exemplo, sugerir um percurso ao utilizador por uma estrada que já não existe, ou cuja sinalética foi alterada e não se reflita nas indicações apresentadas pela aplicação. Do mesmo modo, também há a preocupação de sugerir o melhor percurso possível, pelo que novos trechos de estrada deverão ser igualmente tomados em conta e incluídos nos cálculos.

Por estes motivos, uma das preocupações mais proeminentes deste projeto foi a garantia de que os mapas apresentados estão devidamente atualizados. Embora fosse uma preocupação inicial, estava previsto que a utilização de um SDK de navegação incluísse já mapas atualizados, pelo que esta dificuldade estaria naturalmente resolvida. No entanto, dada a decisão de implementar o sistema de navegação de raiz, a necessidade de encontrar mapas atualizados tornou-se uma questão de extrema importância. Trata-se de um dos pilares mais proeminentes de todo o sistema. Sem os mapas e respetivos grafos atualizados, não seria possível construir o sistema de raiz e esta opção de desenvolvimento seria inviável.

Durante a pesquisa efetuada justamente para resolver este problema, foram encontrados dois *websites* que, segundo a sua descrição, incluem mapas e grafos atualizados regularmente. São eles o repositório Mapsforge (Mapsforge Download Server, s.d.), que inclui as imagens dos mapas, e o repositório OpenStreetMaps (OpenStreetMap Data Extracts, s.d.), com os grafos correspondentes, descritos posteriormente neste relatório.

É importante referir que durante as várias viagens de teste efetuadas para comprovar a qualidade dos mapas, não surgiram quaisquer discrepâncias entre os grafos, os

percursos calculados, e as estradas. No entanto, verificaram-se pequenas discrepâncias entre as imagens dos mapas e as estradas, embora não fossem significativas por se tratar de um elemento puramente visual. O essencial, ou seja, as rotas apresentadas aos utilizadores a partir dos grafos, refletiam as alterações mais recentes das vias.

2.5.8. Eficiência

Já previsto como uma questão importante desde o início, durante o desenvolvimento do módulo de navegação houve o cuidado de fazer uma implementação eficiente das suas várias funcionalidades. Estas funcionalidades incluem não só todo o processamento necessário à gestão das tarefas, como também o cálculo e acompanhamento das rotas.

Sendo parte integrante do produto Drivian Tasks, o módulo de navegação está encapsulado num dispositivo portátil, com bateria limitada, pelo que a vida útil da carga da bateria constitui uma preocupação.

Esta eficiência refletiu-se em todas as funcionalidades do módulo de navegação, com especial enfoque no algoritmo de acompanhamento de rotas, pois este efetua cálculos sempre que o dispositivo recebe informações de GPS, o que se prevê acontecer uma vez por segundo.

2.6. Resumo

Neste capítulo foi apresentada uma descrição do Drivian Tasks, assim como a visão inicial que os responsáveis da empresa Sentilant tinham para o módulo de navegação a desenvolver. Foi também descrito o objetivo geral do estágio e apresentados os problemas inicialmente previstos para a construção do módulo, assim como algumas das considerações essenciais no que se refere à qualidade do produto final.

Embora seja apenas uma análise inicial, este levantamento proporcionou uma direção ao estudo do estado da arte, assim como à definição dos requisitos do sistema.

3. Estado da Arte

Para assegurar que o módulo de navegação a desenvolver mantém os padrões de qualidade presentes em aplicações similares, e se torna num sistema competitivo, foram realizadas duas análises iniciais: uma prospeção sobre as várias alternativas credíveis para a implementação do sistema; e uma análise sobre aplicações de navegação disponíveis no mercado. Pretendia-se explorar possibilidades para a conceção do projeto.

3.1. Tecnologias de Navegação e Posicionamento

Para assegurar a qualidade do sistema a desenvolver, foram consideradas as várias tecnologias de navegação mais proeminentes.

A tecnologia de navegação escolhida deverá permitir desenvolver um sistema robusto, fiável, com boa precisão, e com baixos custos associados, sejam estes referentes ao desenvolvimento ou ao produto final apresentado ao cliente. Deverá também minimizar a necessidade de desenvolvimento de novo *hardware* e, idealmente, tirar partido de *hardware* já possuído pelo utilizador final.

Embora exista um vasto leque de tecnologias que podem ser utilizadas para fins de navegação e posicionamento, apenas foram analisadas as consideradas minimamente viáveis ou promissoras para navegação *outdoors.* Como tal, foram excluídas desta análise tecnologias de posicionamento exclusivamente *indoor* como, por exemplo, *Bluetooth, WiFi, Radio-Frequency Identification, Ultra-Wideband,* ou *Visible Light Communication / Positioning*. Também foram excluídas tecnologias de navegação indicadas unicamente para outros meios de navegação que não veículos automóveis, como é o caso do posicionamento a partir de análise batimétrica, por radar ou por *Terrain Contour Matching,* que não seriam viáveis para o utilizador final do Drivian Tasks.

As várias tecnologias analisadas foram:

3.1.1. STOIC

O novo sistema de posicionamento global preconizado pela Defense Advanced Research Projects Agency (DARPA) dos Estados Unidos da América, intitulado STOIC

(*Spatial, Temporal and Orientation Information in Contested Environments*, ou Informação Espacial, Temporal e Orientação em Ambientes Contestados) é um sistema que utiliza sinais de referência de longo alcance, em conjunto com relógios extremamente precisos, para inferir a posição de um dispositivo. Como tal, não necessita de uma rede de satélites posicionada em órbita à volta do planeta para conseguir inferir a sua posição.

Foi inicialmente descrita no relatório bianual desta agência, intitulado *Breakthrough Technologies for National Security* (Tecnologias Disruptivas para Defesa Nacional), em 2015 (Breakthrough Technologies for National Security, 2015), e apresenta as vantagens de requerer poucos recursos e uma infraestrutura reduzida. Trata-se de uma tecnologia mais fiável que o GPS (*Global Positioning System*), a tecnologia atualmente considerada o padrão de navegação, por não sofrer de perda ou intermitência de sinal e ter um elevado nível de precisão e consistência.

Segundo as descrições feitas sobre o sistema (Lendino, 2015), é uma tecnologia que será facilmente integrada em veículos e dispositivos móveis como *smartphones* mas que, infelizmente, ainda não chegou a público, embora seja esperado que tal aconteça num futuro próximo.

Tecnologia indisponível

No entanto, por ainda não estar disponível, trata-se de uma alternativa inviável para este projeto.

3.1.2. *Signals of Opportunity*

Originada por dois artigos apresentados por uma equipa de investigação da Universidade da Califórnia, Riverside, em 2016 (Morales et al., 2016; Shamaei, Khalife & Kassas, 2016), esta tecnologia permite triangular a posição de um dispositivo através do cruzamento de diferentes sinais, tais como *WiFi* e GSM (*Global System for Mobile communications*). Esta tecnologia poderá ser utilizada como uma alternativa de navegação robusta e fiável de forma isolada ou como complemento de outras tecnologias de posicionamento. Prevê-se que esta tecnologia tenha aplicações no que se refere ao desenvolvimento de sistemas de navegação para carros autónomos ou *drones*.

Necessidade de *hardware* complexo e dispendioso

Por ser relativamente recente, os meios para a utilização desta tecnologia ainda não se encontram disponibilizados de forma acessível ao utilizador final.

Limitações geográficas

Ao tirar partido dos sinais disponíveis num ponto geográfico para inferir a posição de um dispositivo, esta tecnologia torna-se extremamente sensível a situações nas quais estes sinais não se encontram presentes. Com efeito, existem diversos locais onde estes sinais podem não existir, ou a sua quantidade ou qualidade estão limitadas. Isto não acontece num ambiente urbano, mas dado o objetivo de guiar o utilizador em qualquer zona geográfica, esta limitação pode comprometer severamente a qualidade do sistema de navegação.

3.1.3. Reconhecimento de Imagem

Uma outra alternativa para um sistema de navegação é a navegação por reconhecimento de imagem (Bruno & Osorio, 2017; Murase, 2017). Embora esta possibilidade não infira a posição de um dispositivo num espaço geográfico alargado, o que seria limitador para o desenvolvimento de uma aplicação que pretende levar o utilizador a um destino específico, ela permite que o utilizador receba instruções de navegação apropriadas através do reconhecimento da disposição do espaço circundante. Um sistema que integre esta tecnologia deverá ter a capacidade de reconhecer, por exemplo, uma rotunda ou curva eminente, e expor a instrução apropriada. Esta tecnologia é atualmente utilizada no sistema de navegação de carros autónomos para garantir uma condução segura, sendo necessariamente complementada por outras tecnologias como posicionamento por sinal GPS. As limitações mais relevantes desta tecnologia são:

Tecnologia em desenvolvimento

Esta tecnologia apresenta a grande desvantagem de ainda se encontrar em desenvolvimento, pelo que se torna uma alternativa sensível e instável. Algo considerado não ideal na construção de um sistema que pretende guiar o utilizador durante a sua condução.

Necessidade de grandes capacidades de processamento

Neste momento, a navegação por reconhecimento de imagem apresenta limitações no que se refere à necessidade de um dispositivo com boas capacidades de processamento. Este fator é extremamente importante, pois torna inviável a sua utilização em veículos

que não integrem um processador ou em dispositivos com capacidades limitadas como *smartphones* ou *tablets*.

3.1.4. *Global Positioning System*

Sendo uma tecnologia bem estabelecida, o posicionamento com recurso ao *Global Positioning System* (GPS, 2017) refere-se a um sistema de posicionamento que infere a posição de um dispositivo recetor através da triangulação do sinal de vários satélites. Para triangular a posição com boa precisão num plano a três dimensões, o dispositivo recetor necessita de receber sinal de, no mínimo, quatro satélites. É possível inferir a sua posição através da receção do sinal de três satélites, no entanto, a posição resultante é apenas uma aproximação com baixa precisão.

Cada satélite, ao transmitir, comunica o momento exato em que o seu sinal é enviado, assim como a sua posição. Ao subtrair o momento em que o sinal foi transmitido ao momento em que o sinal é recebido, o dispositivo recetor consegue calcular a distância a que se encontra de cada satélite. Por conhecer a distância a que se encontra de vários satélites em simultâneo, assim como a sua posição exata, o recetor consegue determinar a sua própria posição em três dimensões. Esta posição é posteriormente traduzida em coordenadas descritivas de latitude, longitude e altitude.

Neste momento, na Europa, é utilizada a rede de satélites dos Estados Unidos da América, intitulada NavStar, disponibilizada para utilização civil desde 1995, e atualmente composta por 31 satélites posicionados em 6 planos orbitais centrados no planeta Terra. No entanto, com a finalidade de prevenir uma grande dependência dos EUA, encontra-se em implementação uma rede de satélites própria da União Europeia: o sistema Galileo (2018).

Desde que esta tecnologia se tornou disponível ao público, tornou-se no *standard* de navegação para a indústria automóvel. É também aplicada em navegação marinha ou aérea, demonstrando aplicações em outras áreas que não estejam necessariamente relacionadas com navegação, como agricultura, mineração, atividades recreativas, entre outras.

Por se tratar de uma tecnologia bem estabelecida, a sua utilização encontra-se, de momento, muito facilitada. Os custos de integração são bastante reduzidos, a compreensão sobre a sua utilização já poderá ser considerada como estando num nível

avançado, e o *hardware* necessário para a sua utilização é acessível ao consumidor final, encontrando-se frequentemente integrado nos próprios veículos ou em dispositivos móveis como telemóveis e *tablets*.

Mesmo sendo considerada como a tecnologia de navegação padrão, esta apresenta diversas limitações:

Precisão instável

Um dos fatores mais relevantes na utilização de posicionamento com recurso a GPS é o seu nível de precisão. Dado que este sistema depende da receção de sinal de vários satélites em simultâneo, é possível que não se encontrem numa posição ideal para possibilitar ao dispositivo recetor inferir a sua própria posição. Um sistema de posição que recorra a GPS precisa de receber sinal de quatro satélites para inferir a sua posição com elevada precisão num espaço geográfico, embora possa inferir a sua posição de forma limitada a partir de apenas três satélites. O sinal de cada um dos satélites, por sua vez, está também sujeito a interferências e pode ser impedido por fatores como o espaço geográfico circundante ao dispositivo recetor ou até as condições atmosféricas.

Perda súbita de sinal

Perda de sinal resume-se ao dispositivo recetor perder a habilidade de se posicionar no espaço geográfico. Isto pode-se suceder caso entre num local onde a capacidade de receção de sinal fique comprometida, como por exemplo quando o utilizador entra num túnel ou num edifício, ou perante condições atmosféricas agressivas.

Latência

Latência de GPS refere-se à discrepância de tempo entre o momento em que o sinal é enviado a partir dos vários satélites e o momento em que o dispositivo recetor termina o cálculo da sua posição.

Quando um satélite comunica o seu sinal, envia também informação sobre o momento em que este sinal foi transmitido. Quando recebe o sinal, o dispositivo recetor calcula a diferença entre o momento em que ocorreu esta receção e o momento do seu envio. Ao recorrer a vários satélites, o dispositivo recetor utiliza os dados referentes aos momentos de envio e receção dos vários sinais para triangular a sua posição. No entanto, este cálculo poderá ser influenciado pela disparidade de momentos em que os vários sinais são recebidos. O sistema ficará a aguardar a receção de sinais de, pelo menos, três satélites para realizar o cálculo da sua posição. A qualquer momento poderá ficar parado

a aguardar a receção do sinal de um satélite para poder iniciar os cálculos da sua posição que, por sua vez, também demorarão alguns momentos a serem efetuados.

Todas estas condicionantes resultam no facto de que um sistema de posicionamento baseado em GPS consegue apenas inferir onde o utilizador esteve há momentos atrás, e não onde ele está no preciso momento em que o cálculo é terminado.

Receção intermitente

Uma das limitações inerentes a um sistema de posicionamento por GPS é a receção intermitente do sinal. Dada a natureza do funcionamento deste sistema, é impossível conceber uma receção contínua do sinal, o que poderá trazer dificuldades no que se refere à coordenação da apresentação de instruções pertinentes, assim como o facto de um posicionamento intermitente não ser algo intuitivo par o utilizador.

3.1.5. Sensores de Inércia

Sensores de inércia referem-se a sensores de movimento e direção, tais como o giroscópio, o acelerómetro e o magnetómetro, por exemplo (Inertial Navigation System, s.d.; Skog & Händel, 2005). A utilização destes sensores permite desenvolver um sistema que, através do treino de um modelo estatístico, consiga inferir os movimentos feitos por um veículo em tempo real. Desse modo, torna-se possível interpretar se o veículo alterou a sua posição ou direção, fez uma curva ou mudou de via, por exemplo. Este sistema é extremamente útil para os casos em que já existe um percurso traçado previamente, e o sistema pretende interpretar se o utilizador se encontra a segui-lo, ou se saiu da sua rota.

Trata-se de uma tecnologia bem estabelecida no mercado, sendo comum a integração deste tipo de sensores em *smartphones*, *tablets* e *smart vehicles*. No entanto, apresenta diversas limitações importantes:

Limitação preditiva

Um sistema que integre apenas esta alternativa de navegação não tem a capacidade de reconhecer a disposição e características das estradas ou vias, e, como tal, não tem a capacidade de apresentar instruções pertinentes ao utilizador. A utilização exclusiva desta tecnologia não permite construir um sistema que consiga, por exemplo, transmitir ao utilizador se este deverá virar à direita, ou sair numa saída específica de uma rotunda, por exemplo. Trata-se apenas de uma tecnologia "reativa", por oposição a "preditiva",

pois apenas consegue interpretar eventos que já ocorreram, como o movimento do veículo durante uma curva ou uma travagem.

Disparidade de sensores

Embora seja uma tecnologia bem estabelecida, existe uma enorme disparidade de sensores entre os vários dispositivos. Utilizando o exemplo dos *smartphones*, é muito frequente a ausência de um ou mais sensores necessários entre os vários modelos, pelo que se torna extremamente complexo a construção de um sistema fiável que englobe todas as possibilidades.

Incapacidade de inferir a posição inicial do dispositivo

Tal como a alternativa de reconhecimento de imagem, por si só, esta tecnologia não é capaz de posicionar o utilizador num espaço geográfico e, consequentemente, reconhecer um ponto inicial para traçar uma rota e direcionar o utilizador para o seu destino. No entanto, trata-se uma boa alternativa para inferir a posição do dispositivo após definida a sua posição inicial, pelo que se trata de um excelente complemento a um sistema de navegação, aumentando a sua fiabilidade.

3.2. Soluções Comerciais Existentes

Para complementar a análise feita sobre as várias tecnologias de navegação, em conjunto com as intenções declaradas pelos responsáveis da empresa para o módulo de navegação, foi também efetuada uma prospeção das aplicações de navegação automóvel mais proeminentes já presentes no mercado. Esta análise teve como objetivo identificar as funcionalidades essenciais de um sistema de navegação *turn-by-turn*. Do mesmo modo, também permitiu identificar funcionalidades que, não sendo essenciais, não deixam de ser extremamente relevantes para um produto com qualidade equiparável às soluções existentes.

Este estudo permitiu estabelecer um padrão de qualidade para a solução a desenvolver. É importante referir que todas as soluções analisadas utilizam navegação por GPS, dado esta ser a tecnologia padrão para navegação automóvel, com o ocasional complemento de navegação com recurso a sensores de inércia como o giroscópio ou o acelerómetro.

No total, foram analisadas quinze aplicações de navegação. A sua comparação encontra-se exposta na tabela 1.

Dada a enorme disparidade de sistemas existentes, não foi definido um critério de pesquisa robusto. Foram apenas analisadas as aplicações com maior número de utilizadores e com melhor reputação, sendo terminada a pesquisa assim que se considerou que não traria nada de novo e relevante.

Os elementos da tabela foram definidos a partir das funcionalidades consideradas mais relevantes após uma primeira pesquisa superficial. Por "relevante" entenda-se funcionalidades que são importantes para delinear uma aplicação competitiva. Foram também sendo acrescentados campos durante a pesquisa, à medida que novas funcionalidades foram sendo consideradas como relevantes.

O resultado obtido é importante pois permite acrescentar novas sugestões ao projeto a implementar baseado em boas práticas e definir características que são consideradas essenciais a um sistema desta natureza. Deste modo, é possível assegurar que funcionalidades fundamentais não foram esquecidas, e que o sistema é atual e competitivo.

	 GPS Navigation	 GPS Navigation & Tracker	 GPS Route Finder	 Karta GPS	 MotionX GPS Drive (iPhone)	 MapQuest GPS Navigation	 Magellan RoadMate	 Waze
Navegação 3D	✔	✔	✔	✔	✔	✔	✔	✔
ETAs e condições de trânsito em tempo real	✔	✔	✔	✔	✔	✔	✔	✔
Participação e alertas da comunidade								✔
Recálculo de rotas baseado nas condições de trânsito	✔		✔	✔		✔	✔	✔
Assistência de voz	✔		✔	✔	✔	✔	✔	✔
Sugestão de escolha de vias	✔			✔			✔	
Pontos de interesse		✔	✔	✔	✔	✔	✔	✔
Localização de polícia, operações stop, câmaras de velocidade	✔			✔			✔	✔
Mapas offline	✔			✔	✔		✔	
Estimativas de custos de viagem (combustível, portagens, …)						✔		
Opções de rota (evitar auto-estradas, …)	✔	✔	✔	✔	✔	✔	✔	✔
Liga ao painel do carro							✔	✔
Rotas para contatos da lista telefónica			✔	✔				
Partilha de informação (ETA, posição, …) com social media	✔		✔	✔	✔	✔	✔	✔
Apresentação de preços de combustível	✔					✔	✔	✔
Integração com calendário								
Procura de parques de estacionamento próximos				✔	✔		✔	✔
Navegação pedestre	✔		✔	✔	✔		✔	

Tabela 1: Comparação de Soluções de Navegação

 Google Maps	 TomTom GPS Navigation Traffic	 Sygic	 INRIX Traffic	 HERE WeGo - Offline Maps & GPS	 CoPilot GPS	 Route Planner by ViaMichelin	
	✔	✔	✔	✔	✔	✔	Navegação 3D
✔	✔	✔	✔	✔	✔	✔	ETAs e condições de trânsito em tempo real
	✔	✔	✔			✔	Participação e alertas da comunidade
✔	✔	✔	✔	✔	✔	✔	Recálculo de rotas baseado nas condições de trânsito
✔	✔	✔	✔	✔	✔	✔	Assistência de voz
✔	✔	✔			✔		Sugestão de escolha de vias
✔	✔	✔		✔	✔	✔	Pontos de interesse
	✔	✔					Localização de polícia, operações stop, câmaras de velocidade
	✔	✔		✔	✔		Mapas offline
		✔		✔		✔	Estimativas de custos de viagem (combustível, portagens, …)
✔	✔	✔	✔	✔	✔	✔	Opções de rota (evitar auto-estradas, …)
		✔					Liga ao painel do carro
	✔						Rotas para contatos da lista telefónica
	✔	✔	✔				Partilha de informação (ETA, posição, …) com social media
		✔					Apresentação de preços de combustível
			✔				Integração com calendário
	✔	✔	✔	✔			Procura de parques de estacionamento próximos
✔		✔		✔		✔	Navegação pedestre

Tabela 1: Comparação de Soluções de Navegação (Continuação)

Em acréscimo ao estudo comparativo das várias soluções de navegação, foram analisados vários comentários e as avaliações de alguns utilizadores. Esta análise teve como objetivo compreender as funcionalidades mais apreciadas pelos mesmos, os problemas que consideram mais significativos, e as funcionalidades mais desejadas mas inexistentes nas soluções comparadas.

Também com o objetivo de idealizar outras funcionalidades pertinentes para o utilizador final, foram exploradas várias aplicações que, não tendo necessariamente uma componente de navegação, apoiam o utilizador na sua condução.

Para esse fim, foi feita uma breve exploração das funcionalidades de várias aplicações de apoio à condução para Android e iOS, das quais são apresentadas as catorze mais relevantes. Dada a natureza variada destas aplicações, torna-se impossível conceber uma grelha que permita comparar visualmente as características de cada uma. Como tal, foram elaboradas pequenas descrições das suas funcionalidades mais interessantes ainda não presentes nas aplicações de navegação já analisadas.

Como critério de pesquisa, foram analisadas as aplicações com melhor reputação e maior número de utilizadores. O objetivo foi o de não ir com uma ideia pré-concebida sobre as funcionalidades de uma aplicação de apoio à condução, e evitar reduzir demasiado o foco da pesquisa, fazendo, simplesmente, um levantamento das características mais valorizadas pelos utilizadores. A pesquisa foi terminada assim que se considerou que as ideias mais relevantes estavam exaustas, e que sua continuação não revelaria nada de novo ou significativo.

3.2.1. Car Dashdroid

- Lê, com conversor de texto para áudio, mensagens de texto recebidas durante a viagem. Esta funcionalidade tem suporte para as aplicações de mensagens mais utilizadas (WhatsApp, Telegram, Facebook Messenger, entre outros).

3.2.2. Drivemode

- Lê, com conversor de texto para áudio, mensagens de texto recebidas;
- Permite respostas automáticas, predefinidas, a mensagens de texto recebidas;
- Permite desativar notificações durante a viagem.

3.2.3. iOnRoad

- Utiliza a câmara do telemóvel em conjunto com o GPS para monitorizar a distância de segurança relativamente à viatura da frente;
- Monitoriza se o condutor está a sair dos limites da sua via;
- Emite sinais sonoros para avisar o condutor e evitar potenciais acidentes;
- Deteta sinais de limite de velocidade.

3.2.4. Car Mode

- Integra conversor de texto para áudio para ler as mensagens de texto recebidas durante a condução;
- Integra conversor de voz para texto, permitindo enviar mensagens apenas com a voz;
- Memoriza o lugar de estacionamento para o condutor voltar facilmente para a sua viatura.

3.2.5. Roadtrippers

- Possibilita o planeamento de viagens com duração de semanas. Permite marcar hotéis e restaurantes e encontrar pontos de interesse apenas recorrendo à aplicação;
- Estima as despesas de combustível;
- Permite aplicar filtros às estimativas de combustível como gasóleo, gás ou gasolina.

3.2.6. TollSmart

- Permite calcular as despesas com combustível e portagens desde o início ao fim da viagem;
- Permite criar perfis de veículos com impacto nas estimativas.

3.2.7. Glympse

- Permite partilhar a localização e a estimativa de chegada ao destino (ETA), através do envio de mensagens com um link.

3.2.8. Taxmileage

- Permite registar a quilometragem da viatura;
- Gera relatórios detalhados das viagens.

3.2.9. Fuel Buddy

- Apresenta estimativas das despesas em combustível;
- Apresenta estatísticas e gráficos com os vários indicadores.

3.2.10. GasBuddy

- Permite encontrar o posto de abastecimento mais próximo e mais barato, baseado na posição do utilizador;
- Os preços são reportados pela comunidade de utilizadores;
- Recompensa os utilizadores que partilham informações sobre os preços de combustível.

3.2.11. Max Speed

- Avisa o utilizador com um sinal sonoro se este passar os limites de velocidade;
- Monitoriza a condução e avisa o condutor se este sair dos limites da sua via.

3.2.12. MotionX GPS

- Guarda o histórico dos trilhos e caminhos pelos quais o utilizador passou;
- Permite que outras aplicações estejam a ser utilizadas em primeiro plano;
- Apresenta gráficos descritivos dos dados recolhidos.

3.2.13. Automatic Classic

- Monitoriza os hábitos de condução do utilizador;
- Avisa se o condutor passar o limite de velocidade;
- Monitoriza o combustível da viatura;
- Em caso de acidente, faz automaticamente uma chamada de emergência.

3.2.14. Carcorder

- Permite gravar as viagens em vídeo para que o utilizador tenha evidências concretas em caso de acidente.

3.3. Conclusões

A pesquisa do estado da arte permitiu identificar componentes que, sem prejuízo de outras, são consideradas essenciais a um sistema de navegação *turn-by-turn*. São elas:

- Apresentar previsões de chegada e condições de trânsito em tempo real;
- Recalcular rotas dependendo das condições de trânsito;
- Assistir o condutor através de indicações com voz;
- Alertar se o condutor conduzir acima da velocidade permitida;
- Apresentar pontos de interesse pertinentes;
- Permitir navegação em modo *offline*;
- Disponibilizar opções de percurso;
- Permitir procurar parques de estacionamento próximos do destino;
- Permitir procurar pontos de abastecimento no percurso.

De igual modo, a partir da análise das aplicações de apoio à condução, foi possível destacar algumas funcionalidades consideradas um acréscimo de valor relevante a um sistema de navegação:

- Permitir desativar notificações durante a viagem;
- Permitir a leitura e envio de mensagens de texto de forma simples e integrada;
- Memorizar o lugar de estacionamento da viatura;
- Estimar as despesas de viagem;
- Permitir aplicar filtros às estimativas de combustível como gasóleo, gás, gasolina;
- Permitir criar perfis de veículos, com impacto nas estimativas;
- Em caso de acidente, realizar chamadas de emergência.

3.4. Resumo

Com a exploração do estado da arte foi possível adquirir uma maior compreensão sobre as várias tecnologias de navegação atualmente disponíveis para a navegação automóvel. Foi também possível aferir as vantagens e desvantagens associadas a cada uma, assim como a sua pertinência para o projeto.

Através da análise de várias aplicações de navegação e de apoio à condução, foi feito um levantamento de funcionalidades pertinentes e características relevantes que não

foram previstas inicialmente. A intenção foi a de considerar a pertinência de cada uma destas funcionalidades, ou ideias delas derivadas, para o módulo de navegação a desenvolver. Com esta análise, foi também possível averiguar a qualidade dos sistemas de navegação atualmente disponíveis, oferecendo uma noção da qualidade mínima para o sistema a desenvolver.

Em última instância, este estudo permitiu tomar opções estratégicas informadas e idealizar um sistema de navegação que corresponde à qualidade das soluções atualmente disponíveis no mercado.

4. Proposta de Abordagem

Com o conhecimento adquirido sobre as várias tecnologias disponíveis para a construção do sistema de navegação, em conjunto com a sondagem feita sobre os sistemas de navegação disponíveis no mercado, encontravam-se reunidos os elementos necessários para definir a direção geral do projeto.

A partir da informação recolhida, foi possível optar por uma tecnologia de navegação, prever as suas limitações e definir os vários componentes essenciais ao desenvolvimento do módulo de navegação para o Drivian Tasks.

4.1. Tecnologia Escolhida

Dando seguimento à pesquisa efetuada sobre as várias tecnologias aplicáveis ao projeto, foi possível pesar os pontos a favor e contra a utilização de cada uma. Após esta análise, concluiu-se que um sistema baseado em GPS é a opção mais credível para o desenvolvimento de uma aplicação *turn-by-turn*.

Como já discutido durante a análise feita sobre as várias tecnologias, trata-se de uma opção monetariamente acessível ao consumidor final, com poucos custos de desenvolvimento associados, e encontra-se atualmente integrada num grande número de automóveis e dispositivos móveis. Este fator é extremamente relevante para o desenvolvimento deste projeto, atendendo ao objetivo de integrar o sistema de navegação numa aplicação já existente para *smartphones*. Esta tecnologia tem também a vantagem de estar já bem estabelecida no mercado e encontrar-se num nível de desenvolvimento avançado, sendo, inclusivamente, considerada o *standard* da navegação automóvel. Como tal, a sua utilização encontra-se muito bem documentada, o que significa que já existem soluções bem definidas para os seus problemas mais comuns, e os meios para a sua utilização são relativamente simples de adquirir.

Um possível complemento considerado para o sistema de navegação foi a navegação com recurso a sensores de inércia. De momento, estes sensores encontram-se também presentes num número considerável de dispositivos móveis, embora não em todos. No entanto, muitos dispositivos integram-nos apenas de forma parcial, pelo que esta diversidade é um fator relevante durante a implementação. Devido ao facto de a utilização exclusiva destes sensores não permitir posicionar o utilizador num espaço

geográfico, necessitando sempre da definição da sua localização inicial, esta alternativa não poderá ser assumida como a tecnologia central para o projeto. Considerando a sua baixa precisão, a necessidade de dedicar um tempo considerável ao treino dos modelos estatísticos necessários e ao desenvolvimento desta funcionalidade, o facto de não se tratar de algo inovador o suficiente para corresponder às intenções do estágio, e visto tratar-se apenas de um complemento para aumentar a precisão do sistema de navegação, foi decidido que, caso se tomasse a opção de enveredar por uma implementação de raiz de todo o sistema de navegação, esta não seria uma opção viável para a conceção do sistema. Considera-se que os potenciais ganhos de implementar algo meramente complementar, que depende de sensores que não se encontram disponíveis de forma completa nem uniforme em todos os dispositivos móveis, não compensam o risco de abdicar de funcionalidades essenciais previstas para o módulo de navegação por falta de tempo. Caso se optasse pela utilização de um SDK de navegação, esta tecnologia poderia, ou não, estar já integrada, dependendo da ferramenta escolhida.

Desse modo, dada a opção de implementar o sistema de navegação de raiz, o foco do projeto foi exclusivamente a navegação com recurso a GPS.

4.2. Soluções Iniciais para os Problemas Inerentes à Tecnologia Escolhida

Embora o GPS seja uma tecnologia já muito bem estabelecida, existem diversos problemas associados que deverão ser tomados em conta aquando da sua utilização. Estes problemas, inerentes à própria tecnologia, poderão trazer dificuldades ao desenvolvimento do sistema de navegação, e deverão ser devidamente previstos para poder formular possíveis soluções.

Deste modo, os problemas previstos, inerentes à utilização de posicionamento por GPS são: a possibilidade de perda de sinal por disposição inconveniente da rede de satélites, dificuldades de precisão provenientes de reflexos de sinal no espaço circundante ao dispositivo recetor, latência na receção do sinal, e o facto de, por natureza, ocorrer uma receção intermitente do sinal. Esta tecnologia tem também limitações no que se refere a posicionamento *indoors*, embora isso não seja um fator limitativo para o projeto em questão.

As soluções encontradas para cada um dos problemas proporcionaram uma direção para a implementação do sistema de navegação:

4.2.1. Precisão

O sistema deverá contemplar um determinado raio como margem de erro quando inferir a posição do dispositivo recetor. A sua sensibilidade deverá ser facilmente ajustável para que possa ser calibrada. Deverá também acondicionar a informação da precisão recebida pelo próprio sinal de GPS, que poderá ser variável.

4.2.2. Perda súbita de sinal

O sistema de navegação deverá continuar a apresentar o trajeto ao utilizador, mesmo que a sua capacidade de apresentar instruções fique comprometida. Deste modo o utilizador continuará com o trajeto espelhado na aplicação.

4.2.3. Latência

A apresentação de indicações deverá contemplar algum espaço de manobra, e garantir que apresenta as instruções com antecedência suficiente para minimizar o impacto da latência natural do GPS.

4.2.4. Receção intermitente

A visualização deverá apresentar ao utilizador uma navegação aparentemente fluida e ininterrupta, animando, no dispositivo, o movimento do utilizador entre os pontos do percurso onde este é localizado.

4.3. Tecnologias e Componentes Necessárias

Para desenvolver um sistema de navegação *turn-by-turn*, são necessários vários elementos que deverão interagir entre si. São eles:

4.3.1. *Hardware* do Utilizador

Para poder tirar partido do sistema de navegação *turn-by-turn* a desenvolver, o utilizador deverá ter em sua posse um dispositivo recetor de sinal GPS. Este dispositivo deverá também ter acesso à *Internet* e permitir a interação com as várias funcionalidades do sistema através de um *touchscreen*.

Dada a integração do sistema num ambiente já existente, o Drivian Tasks, o *hardware* definido para o utilizador deverá corresponder ao já definido para esta aplicação. Como

tal, os dispositivos compatíveis deverão ser, na sua grande maioria, *smartphones*, mas poderão contemplar quaisquer outros dispositivos que preencham estes requisitos, tais como *tablets* ou painéis de controlo de veículos.

4.3.2. Servidor

Atendendo ao objetivo de integrar várias funcionalidades de gestão de tarefas no módulo de navegação, é essencial disponibilizar um servidor que, por sua vez, faça a gestão das várias bases de dados necessárias. Este servidor deverá também contemplar a possibilidade de incorporar funcionalidades de cálculo de rotas, ou disponibilizar ficheiros adicionais que poderão ser necessários ao funcionamento da aplicação a desenvolver, como por exemplo ficheiros de mapas. Deve também integrar uma API que permita realizar alguma gestão de tarefas a partir dos dispositivos móveis.

4.3.3. API para Cálculo de Rotas

De modo a apresentar ao utilizador final a melhor rota possível para o seu destino, será necessário que o *hardware* do utilizador comunique com uma RESTful API online cuja responsabilidade seja o cálculo de rotas entre dois pontos geográficos. Com efeito, esta funcionalidade não deverá estar inteiramente associada ao *hardware* do utilizador, dado que este terá, em princípio, capacidade de processamento e bateria limitadas. Atendendo ao objetivo de aumentar a eficiência do trabalho dos utilizadores do Drivian Tasks, este cálculo deverá, idealmente, considerar fatores que tenham impacto na eficiência da rota tais como trânsito, condições das vias, limites de velocidade de cada trecho de estrada, entre outros, o que não é exequível num dispositivo portátil que efetue estes cálculos localmente. Ao recorrer a um API para integrar estas funcionalidades, a responsabilidade destes cálculos é transferida para um servidor com maior capacidade de processamento. O cálculo de rotas efetuado localmente, sem recurso a uma API *online*, deverá ser utilizado apenas de forma pontual.

4.3.4. API para Gestão de Tarefas

Para permitir a cada utilizador gerir as tarefas alocadas ao seu dia de trabalho através de um dispositivo móvel, é essencial disponibilizar uma RESTful API com vários terminais, ou *endpoints*, relativos à gestão de tarefas. Deste modo, deverão ser disponibilizados *endpoints* que permitam obter informações referentes aos clientes de uma empresa, assim como sobre as próprias empresas em si, os seus veículos e os seus

empregados utilizadores da aplicação. Igualmente, deverá também existir um *endpoint* para a gestão das várias tarefas, que permita coordenar o seu ciclo de vida, receber informações, adicionar informações ou ficheiros, entre outras funcionalidades. Estas alterações, por sua vez, deverão refletir-se nas respetivas bases de dados.

4.3.5. Bases de Dados

Para gerir todos os dados recolhidos referentes aos vários clientes ou ao ciclo de vida de cada tarefa, será necessário acesso a várias bases de dados. Atendendo à organização já existente no sistema do Drivian Tasks, o sistema a desenvolver deverá receber e transmitir dados a uma base de dados de utilizadores através de uma API, onde constam dados sobre os próprios utilizadores do Drivian Tasks. Do mesmo modo, deverá igualmente contemplar uma base de dados com informações das empresas clientes do Drivian Tasks e dos seus respetivos clientes. Deverá existir também uma base de dados onde sejam guardadas informações sobre as tarefas, assim como os vários elementos que lhe estão associados.

4.3.6. Aplicação *Web*

Para permitir que um gestor ou responsável de logística de uma empresa consiga gerir as tarefas alocadas aos seus trabalhadores e obter dados estatísticos ou informações específicas sobre as várias tarefas, o Drivian Tasks inclui também uma aplicação *Web*, acessível a partir de qualquer *browser*. Esta aplicação permite analisar o desempenho dos seus empregados, gerir a sua frota de veículos, e acompanhar em tempo real a execução das tarefas, sendo assim considerado como o centro de gestão e informação de uma empresa no que se refere à sua logística.

O módulo de navegação a desenvolver deverá permitir a um trabalhador no terreno gerir as suas tarefas, e esta gestão deverá refletir-se, por sua vez, nas informações apresentadas na aplicação web, através do cruzamento dos dados presentes nas bases de dados.

4.4. Arquitetura

Na sua essência, a arquitetura do sistema espelha a já definida inicialmente para o Drivian Tasks: uma arquitetura com uma lógica servidor-cliente, na qual os vários clientes enviarão e receberão dados referentes à gestão de tarefas a partir de um

servidor. Esta comunicação é feita através de uma RESTful API que disponibilizará os dados em formato JSON. O servidor, por sua vez, tem a responsabilidade de assumir o processamento sobre a gestão das tarefas. Esta lógica permite também interligar o servidor a uma aplicação web, também esta cliente, onde um responsável pela gestão logística de uma empresa pode coordenar as várias ações no terreno.

Para o módulo de navegação a desenvolver, pretende-se que este comunique também com uma API com a responsabilidade de disponibilizar as rotas necessárias ao cumprimento das várias tarefas. Isto permite libertar recursos do lado dos dispositivos cliente, que efetuarão o cálculo de rotas de forma autónoma, *offline*, apenas em último recurso, caso a comunicação com o servidor esteja obstruída, ou em alguns outros casos pontuais que serão especificados posteriormente.

É importante referir que a aplicação *web*, assim como a lógica do servidor, ultrapassam o âmbito deste projeto, que se centra apenas no desenvolvimento do módulo de navegação da aplicação móvel do Drivian Tasks.

A lógica aqui definida permite não só centralizar os dados como libertar os clientes da responsabilidade de processamentos mais complexos, como será o caso do cálculo de rotas ou a própria gestão de tarefas inerente a todo o projeto. Pretende-se, desta forma, limitar o consumo de bateria dos dispositivos móveis, e, consequentemente, aumentar a usabilidade da aplicação a desenvolver.

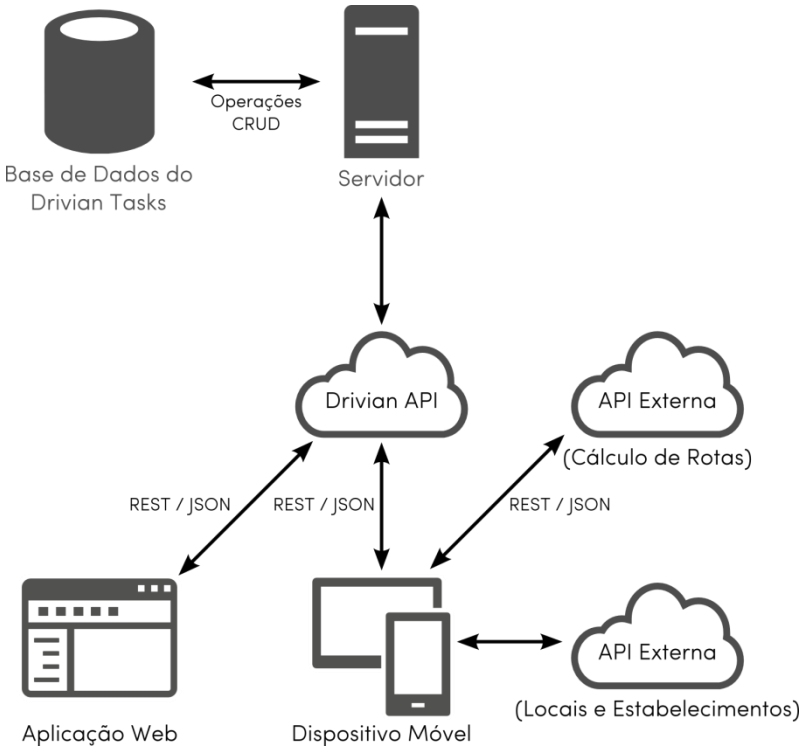


Figura 3: Proposta de Arquitetura

4.5. Resumo

Nesta secção foram identificadas as componentes consideradas essenciais para a conceção do sistema.

Foi decidido utilizar o sistema de GPS como estratégia central de navegação no terreno, dada a sua facilidade de aquisição por parte do cliente, assim como a sua facilidade de utilização da perspetiva de desenvolvimento. Foram também identificados os problemas mais proeminentes inerentes à utilização desta tecnologia, cujas implicações estarão refletidas na definição dos requisitos do sistema.

Por fim, foram especificados os vários elementos do sistema. Muitos destes elementos espelham, necessariamente, a estrutura já definida para o Drivian Tasks, dado o objetivo de integração do módulo de navegação numa aplicação já existente.

Esta especificação resultou na proposta da arquitetura de alto nível para o sistema e numa proposta estratégica inicial para o seu desenvolvimento.

5. Levantamento de Requisitos

A partir da análise feita sobre as funcionalidades das várias soluções de navegação disponíveis no mercado, em conjunto com a definição da arquitetura de alto nível e a definição de uma proposta estratégica inicial, foi possível definir os vários requisitos funcionais e não funcionais do sistema.

Estes requisitos foram construídos através do cruzamento das funcionalidades essenciais de um sistema de navegação *turn-by-turn,* identificadas aquando do estudo do estado da arte, em conjunto com a discussão das funcionalidades esperadas por parte dos orientadores do projeto.

Infelizmente, dada a natureza do projeto, não foi possível levantar os requisitos com os utilizadores finais, os vários trabalhadores no terreno ou as suas empresas. Este impedimento foi contrabalançado pela experiência dos orientadores do projeto, dado terem um bom conhecimento das várias necessidades dos utilizadores finais, assim como uma ideia bem definida das funcionalidades que gostariam de integrar para expandir o Drivian Tasks através do sistema de navegação a desenvolver.

5.1. Ambiente Operacional

A intenção inicial, descrita na proposta de estágio, solicita a implementação do módulo de navegação nos dois sistemas operativos já contemplados pelo Drivian Tasks: Android e iOS.

Com efeito, aquando da definição dos requisitos para o módulo de navegação, mantinha-se a intenção de implementar o sistema nestes dois sistemas operativos. No entanto, a pedido dos responsáveis pelo projeto, este objetivo inicial foi alterado, e a implementação do sistema em iOS foi abandonada. As razões para esta decisão serão posteriormente expostas na secção adequada deste relatório.

A implementação, inicialmente prevista para Android e iOS, passou, numa fase posterior, a contemplar o desenvolvimento do sistema apenas para Android.

5.2. Requisitos de Implementação e *Design*

Sendo, na sua génese, uma componente de navegação que auxiliará o condutor durante a sua condução, a necessidade de não distrair o condutor é uma das preocupações centrais. Por esse mesmo motivo, foram definidas as restrições para o módulo de navegação, presentes na tabela 2.

Referência	Requisito de Implementação e Design
RID-001	Todas as opções possíveis deverão ser acessíveis em menos de três toques.
RID-002	Não deverão existir opções que tapem a visibilidade do condutor enquanto este estiver em navegação.
RID-003	Todas as opções deverão estar condensadas em menus discretos, que só serão apresentados ao utilizador mediante a sua solicitação ou caso seja absolutamente necessário.

Tabela 2: Requisitos de Implementação e Design

Todo o *design* da aplicação, assim como a definição dos casos de uso, deverão refletir estas decisões.

5.3. Pressupostos e Dependências

Sendo o objetivo deste estágio a implementação de um módulo para uma aplicação já existente, as dependências do projeto prendem-se, naturalmente, com as dependências já identificadas para o Drivian Tasks. Note-se que, como descrito anteriormente, até ao momento de definição dos pressupostos e dependências, mantinha-se a intenção de implementar o módulo de navegação em iOS. As dependências identificadas para o módulo de navegação a desenvolver são:

- O utilizador deverá possuir um *smartphone* com o sistema operativo Android 4.4 (KitKat) ou iOS 7, ou uma versão superior;
- O *smartphone* do utilizador deverá ter integrado um sistema de localização por GPS;
- O utilizador deverá ter acesso contínuo à *Internet* através dos dados móveis do seu tarifário móvel ou através de uma rede *WiFi*.

5.4. Requisitos Funcionais e Casos de Uso

Os vários requisitos funcionais foram especificados com recurso a *user stories*, ou descrições das ações de um utilizador. Este formato tem como principal objetivo a compreensão dos requisitos por parte do utilizador final, de uma forma que lhe seja intuitiva e lógica. As *user stories* foram elaboradas com o seguinte formato:

- Enquanto <ator do sistema>
- Quero <ação>
- De forma a <proposta de valor>

Para garantir que a aplicação resultante deste projeto chegaria ao fim do seu ciclo de desenvolvimento com, pelo menos, as suas funcionalidades essenciais, foi atribuído a cada requisito funcional um nível de prioridade. Estes níveis foram definidos a partir de uma escala com dois níveis, m*ust* e *should*, já utilizada pela empresa. Foi adicionado um terceiro nível, *nice to have*, para englobar os requisitos que melhor se referem a preferências não essenciais.

- *Must*: requisito essencial para a aplicação conseguir cumprir os objetivos a que se propõe;
- *Should*: o requisito deverá ser implementado. No entanto, em circunstâncias particulares e após ponderação das suas implicações, poderão existir razões para que não seja implementado;
- *Nice to Have*: O requisito deverá ser implementado caso o orçamento e o tempo alocado o permitam. No entanto, não é uma prioridade, sendo considerado uma preferência.

Por fim, os requisitos funcionais foram separados em duas categorias essenciais: Requisitos para a aplicação móvel e para a aplicação *web*, sendo estes segundos os complementos necessários às funcionalidades presentes na aplicação móvel.

Os requisitos expostos neste relatório correspondem apenas ao título de cada requisito, sendo a sua especificação e detalhes omitidos para salvaguardar a confidencialidade do documento de requisitos escrito para a empresa.

Os requisitos funcionais delineados para o módulo de navegação do Drivian Tasks encontram-se expostos nas tabelas 3 e 4, correspondentes à aplicação móvel e à aplicação web, respetivamente.

Referência	Requisito	Prioridade
RF-AM-001	Conhecer o percurso para o dia.	*Must*
RF-AM-002	Conduzir para o próximo destino.	*Must*
RF-AM-003	Ver a lista de indicações do percurso atual.	*Should*
RF-AM-004	Ouvir alerta para condução acima da velocidade permitida.	*Should*
RF-AM-005	Selecionar percurso de volta.	*Must*
RF-AM-006	Escolher opções de percurso.	*Nice to Have*
RF-AM-007	Corrigir destino.	*Should*
RF-AM-008	Ver informações sobre tarefas.	*Must*
RF-AM-009	Ver opções de tarefa.	*Must*
RF-AM-010	Adicionar complementos de tarefa.	*Nice to Have*
RF-AM-011	Receber novas tarefas.	*Must*
RF-AM-012	Fazer chamadas para o cliente atual.	*Must*
RF-AM-013	Enviar mensagens automáticas para o cliente atual.	*Should*
RF-AM-014	Fazer chamadas para a central.	*Must*
RF-AM-015	Receber sugestões de visita espontânea.	*Should*
RF-AM-016	Telefonar a cliente de visita espontânea.	*Should*
RF-AM-017	Pesquisar clientes.	*Nice to Have*
RF-AM-018	Criar novas tarefas.	*Should*
RF-AM-019	Incluir um ponto de abastecimento no percurso atual.	*Nice to Have*
RF-AM-020	Ver estatísticas de condução.	*Nice to Have*
RF-AM-021	Reportar imprevistos.	*Nice To Have*
RF-AM-022	Procurar parques de estacionamento.	*Nice to Have*

Tabela 3: Requisitos da Aplicação Móvel

Referência	Requisito	Prioridade
RF-PW-001	Ativar possibilidade de recusar novas tarefas.	*Should*
RF-PW-002	Ativar possibilidade de criar novas tarefas no terreno.	*Must*
RF-PW-003	Definir mensagens-tipo.	*Nice to Have*

Tabela 4: Requisitos da Aplicação *Web*

5.5. Requisitos Não Funcionais

Na fase de definição de requisitos foi também definido um conjunto de requisitos não funcionais, ou atributos de qualidade, que deverão ser respeitados durante o desenvolvimento do sistema. Estes requisitos encontram-se expostos na tabela 5.

Referência	Requisito Não Funcional
RNF-001	O sistema deverá respeitar as versões mínimas dos SDKs já definidas para aplicação Drivian Tasks.
RNF-002	O sistema deverá minimizar a utilização da *Internet* com o intuito de poupar dados móveis ao utilizador.
RNF-003	O sistema deverá minimizar a utilização da memória RAM do dispositivo móvel, visto estar previsto que seja utilizado em dispositivos não atuais.
RNF-004	O sistema deverá minimizar o processamento necessário às funcionalidades de navegação. Prevê-se que a posição do utilizador seja atualizada uma vez por segundo. Como tal, os cálculos associados à navegação também ocorrerão uma vez por segundo. Deverão ser otimizados para garantir a durabilidade da bateria do dispositivo móvel.
RNF-005	A aplicação deverá estar otimizada de forma a preparar mapas e carregar grafos em memória dentro de um período de tempo aceitável. De preferência, menor que cinco segundos.

Tabela 5: Requisitos Não Funcionais

5.6. Elaboração de *Mockups*

Em conjunto com a definição dos requisitos para o sistema, foram também elaborados *mockups*. A construção destes *mockups* teve como finalidade guiar e acelerar o desenvolvimento do módulo de navegação, assim como permitir a partilha da visão para a aplicação entre os vários elementos da empresa.

No anexo B constam alguns dos *mockups* elaborados, não sendo apresentados na sua totalidade, não só para salvaguardar os interesses da empresa Sentilant, como também por serem considerados desnecessários para os fins deste relatório.

5.7. Validação de Requisitos

Após a definição dos requisitos da aplicação e da elaboração dos respetivos *mockups*, foi feita uma apresentação da aplicação a todos os elementos da empresa. Nesta apresentação foram discutidas as várias implicações de cada requisito individualmente e em detalhe, tanto da perspetiva de desenvolvimento como da perspetiva de negócio. O desenho da aplicação foi aprovado por toda a equipa por unanimidade, e o seu *feedback* foi devidamente anotado e integrado na definição do sistema.

Os diapositivos da apresentação estão expostos no anexo C deste relatório.

5.8. Resumo

Na fase de preparação descrita neste capítulo, foram definidos os vários requisitos do sistema. O desenho detalhado da aplicação permitiu criar uma imagem mais precisa do sistema e adquirir uma visão concreta do resultado e das suas funcionalidades finais.

Este estudo resultou numa apresentação feita na empresa Sentilant, após a qual o *feedback* dado pela equipa foi integrado nos requisitos. Com esta apresentação, os requisitos idealizados para o sistema foram considerados completos e aceites. Deu-se assim início à próxima fase de desenvolvimento: a análise das várias alternativas de desenvolvimento e a pesquisa das componentes necessárias para a construção do sistema de navegação.

6. Estudo de Alternativas de Implementação

Definidos os requisitos para o módulo de navegação do Drivian Tasks, tornou-se necessário explorar as várias alternativas de implementação possíveis, e quais as vantagens e limitações associadas a cada uma. Nesse sentido, foram contempladas duas possíveis abordagens: uma primeira com recurso a um *Software Development Kit* (SDK) de navegação, sem acesso ao código fonte (*black box)*, e cujas responsabilidades de acompanhar o utilizador, dar instruções apropriadas, calcular rotas, disponibilizar mapas e gerir os mapas no dispositivo móvel, são inteiramente delegadas a esse mesmo SDK; e uma segunda abordagem que corresponde a uma implementação de raiz de todo o sistema de navegação, apenas com a responsabilidade de cálculo de rotas com informações adicionais (trânsito, condições de via, entre outras) delegada a uma API externa.

Independentemente da abordagem escolhida, e com o objetivo de implementar o requisito com a referência RF-AM-015 (Receber sugestões de visita espontânea.), foi necessário desenvolver um algoritmo de seleção e ordenamento de sugestões de visitas a potenciais clientes. Para implementar este algoritmo foi selecionada uma API de Locais e Estabelecimentos que disponibiliza dados sobre os mesmos.

6.1. Implementação com Recurso a *Software Development Kits* (SDKs)

A primeira abordagem explorada para implementar todos os requisitos definidos para o módulo de navegação do Drivian Tasks foi a implementação com recurso a um *Software Development Kit* (SDK). Com efeito, recorrer a um destes sistemas modelo apresenta duas grandes vantagens:

- Permite uma implementação simples e rápida;
- A qualidade do sistema de navegação é quase garantida.

Por outro lado, esta abordagem apresenta diversas desvantagens:

- O preço associado à utilização do SDK inflaciona o preço final do Drivian Tasks e compromete a escalabilidade da aplicação;
- Torna-se impossível escolher a API de cálculo de rotas a utilizar;
- Impossibilita a gestão do número de pedidos feitos à API;
- Impossibilita a adição de outros elementos de navegação não incluídos no SDK.

6.1.1. Levantamento de SDKs de Navegação

Foi explorado um total de onze SDKs de navegação. Para cada uma das empresas que disponibiliza um destes SDKs, foram enviados *emails* com dúvidas específicas ou efetuadas chamadas em áudio e videoconferência. Estes contactos foram estabelecidos com o intuito de compreender se cada um destes produtos correspondia à totalidade dos requisitos definidos para o sistema.

O resultado deste estudo encontra-se na tabela 6.

	MapBox	Here Maps	Navmii	Skobbler	Sygic
Cobertura em Portugal	✔	✔	✔	✔	✔
Rotas cientes de trânsito	✔	✔	✔	?	✔
Navegação com voz em Português	✔ (Suporte para Amazon Polly)	✔	?	?	✔
Navegação 3D	✔	✔	✔	✔	✔
Instruções de via (não apenas de estrada)	✔				✔
Suporte para mapas *offline*	✔	✔	✔	✔	✔
Opções de rota (evitar autoestradas, …)		✔	?	?	✔
Custos de portagens		✔	?	?	
Localização de radares	✔ (Previsto para finais de 2018)	✔	?	?	✔
Rotas para veículos pesados		✔	✔		✔
Opções para perfis de veículos pesados		✔	?		✔
Android	✔	✔	✔	✔	✔
iOS*	✔	✔	✔	✔	✔
Versão de avaliação	< 50.000 utilizadores mensais ativos	90 dias 100K transações mensais	?	Não disponível	Entre 7 a 14 das, dependendo do SDK
Preços	€0.50 por cada 500 utilizadores mensais (após os 50.000 iniciais)	€21 p/ ano para ligeiros €22.80 p/ ano para pesados + *Back office*: €20.30 / €29 p/ano	?	Não disponível Não aceitam novos clientes	Volume de negócio mínimo: €1000 por ano

Tabela 6: *Software Development Kits* de Navegação

Carto	MapQuest	MapZen	Tally Go	AntaresNav	
?	✔	✔			Cobertura em Portugal
✔	***	(Transitland)	✔	?	Rotas cientes de trânsito
?	?	?			Navegação com voz em Português
✔	✔	✔		?	Navegação 3D
					Instruções de via (não apenas de estrada)
✔		✔**	✔	?	Suporte para mapas offline
?	✔	✔	✔	?	Opções de rota (evitar autoestradas, …)
?				?	Custos de portagens
?	?			?	Localização de radares
	?		✔		Rotas para veículos pesados
	?				Opções para perfis de veículos pesados
✔	✔	✔	✔	✔	Android
✔	✔	✔ (Swift)	✔ (Swift)	✔	iOS*
30 dias	< 15.000 transações mensais	< 100 / 25.000 pedidos (dependendo do tipo de pedido)	20,000 viagens	?	Versão de avaliação
€149 p/ mês Ou €1639 p/ ano	< 30.000 transações mensais: €99 < 75.000: €199 < 200.000: €399 ...	€0.50 por cada 1000 pedidos	Tudo gratuíto menos pedidos para navegação turn by turn: €2.50 por cada 1.000 viagens	?	Preços

Tabela 6: *Software Development Kits* de Navegação (continuação)

Legenda	
✔	O SDK inclui a funcionalidade;
?	Não foi possível esclarecer as dúvidas com o vendedor e a informação não se encontrava presente no *website*;
*iOS	Assume-se Objective C por defeito;
**	Até à data, no website do Mapzen encontra-se a informação de que a equipa irá implementar navegação offline no seu SDK. De momento é possível utilizar navegação *offline*, mas de forma pouco intuitiva e de implementação complexa.
***	Não tem tráfego ciente de trânsito em Portugal. Apresentam alternativas em: https://developer.mapquest.com/documentation/samples/directions/v2/route/

6.1.2. SDKs Não Considerados

Em acréscimo aos SDKs apresentados na tabela 6, foram também explorados outros que, por variadas razões, não se revelaram relevantes para a implementação do sistema de navegação. São eles:

6.1.2.1. WazeSDK

Embora seja um dos SDKs mais promissores, as suas políticas de utilização são impeditivas. O WazeSDK declara, explicitamente, que não poderá ser utilizado em aplicações comerciais para gestão de frotas (About the Waze Transport SDK, 2018).

6.1.2.2. Wikitude Navigation

O sistema de navegação é apenas parte de um SDK maior de realidade aumentada. Como tal, torna-se extremamente caro para as funcionalidades que são realmente necessárias para o módulo de navegação a desenvolver.

6.2. Implementação de Raiz

A segunda alternativa de implementação para módulo de navegação do Drivian Tasks refere-se a uma implementação de todo o sistema de raiz. A implementação de um sistema desta natureza apresenta as seguintes vantagens:

- Permite controlar os custos associados ao desenvolvimento e manutenção do sistema;
- Possibilita a escolha da API de cálculo de rotas a utilizar;
- Permite a alteração da API de cálculo de rotas a utilizar após o lançamento do produto, caso seja necessário. Isto poderá ser feito com recurso à implementação de adaptadores;
- Possibilita a gestão do número de pedidos feitos à API de cálculo de rotas;
- Permite desenvolver um módulo de navegação proprietário, que poderá ser aplicado a outros projetos da empresa.

Esta abordagem apresenta também as seguintes desvantagens:

- A implementação do sistema torna-se complexa e demorada;
- Obriga a empresa a adquirir regularmente mapas e grafos atualizados;
- Responsabiliza a empresa pela qualidade do sistema de navegação e por todos os inconvenientes que este possa suscitar.

Uma implementação de raiz implica a reunião de uma série de elementos. São eles:

- Uma API de cálculo de rotas que inclua informação de trânsito em tempo real para calcular rotas otimizadas. Também deverá disponibilizar informações adicionais sobre condições das vias, acidentes, obras na estrada, entre outros dados considerados pertinentes;
- Uma biblioteca de mapas para disponibilizar mapas ao utilizador, e com a possibilidade de funcionar inteiramente *offline*. Esta necessidade surgiu devido ao facto das bibliotecas nativas de Android e iOS apresentarem diversas restrições legais não concordantes com os objetivos do projeto. Algumas destas restrições contrastam com a necessidade de guardar uma grande quantidade de imagens de mapas no dispositivo para garantir a sua apresentação *offline*, ou com a necessidade de apresentar vários elementos no mapa, como marcadores por exemplo, provenientes de outros serviços que não os preconizados pelas empresas criadoras dos sistemas operativos;
- Mapas atualizados de todos os países do Mundo. As várias bibliotecas de apresentação de mapas *offline* necessitam sempre das imagens dos mapas com diferentes níveis de aproximação;
- Um algoritmo que permita o cálculo de rotas *offline*, caso o acesso à *Internet* esteja limitado;
- Grafos atualizados referentes às estradas de todos os países do Mundo, para que possam ser carregados para a memória do dispositivo e calcular percursos *offline*;
- Um sistema de acompanhamento de rotas, que acompanhe o utilizador no seu percurso, faça ajustes de rota conforme necessário, e apresente instruções apropriadas, pertinentes, no momento oportuno.

É importante referir que durante a recolha dos vários elementos a utilizar, houve a preocupação de verificar as suas licenças de utilização. Dado o objetivo de desenvolver um sistema proprietário para a empresa Sentilant, caso fosse tomada a opção de enveredar por uma implementação de raiz, houve a preocupação de apenas selecionar como potenciais adições ao projeto os elementos que incorporam licenças permissivas.

6.2.1. Levantamento de APIs de Cálculo de Rotas

Para uma implementação do sistema de navegação feita de raiz, foram exploradas doze APIs e as suas respetivas características. O resultado deste levantamento encontra-se exposto na tabela 7.

	Google Maps	Graphhopper	Openroute	Bing Maps	Via Michelin	TomTom
Cobertura em Portugal	✔	✔	✔	✔	✔	✔
Rotas cientes de trânsito	✔ (apenas *premium*)	✔ (dados históricos TomTom)	?	✔	✔	✔
Opções de rota (evitar autoestradas, …)	✔	✔	✔	✔	✔	✔
Resposta com texto das indicações do percurso	✔	✔	✔	✔	✔	✔
Cálculo de custos de portagens		✔			✔	
Localização de radares						
Rotas para veículos pesados		✔	✔	✔	✔	
Opções para perfis de veículos pesados		2 perfis tipo	✔		✔	
Permite fazer pedidos de rota com vários pontos obrigatórios?	Limite de 23 *waypoints* com o serviço *standard*	Limite de pontos depende da package. = 1 pedido	✔	Sim. Pontos ilimitados = 1 pedido	✔	Sim. = 1 pedido Máximo de 50 *waypoints*. Máximo de 20 com pedido de rota otimizada.
Disponibiliza mapas				✔ (mas não permite a sua transferência)	✔	✔
Permite pedir dados de trânsito isolados				✔		✔ Dados de transito ou imagens codificadas
Versão de avaliação	2.500 solicitações por dia	500 pedidos e 5 rotas por dia 1 veículo	2.500 solicitações por dia	< 125,000 transações por ano	< 1000 pedidos 45 dias	
Preços	2.500 solicitações diárias gratuitas + €0.50 por cada 1000 solicitações	*Basic*: €39 / mês por 5.000 pedidos diários (máx. 2 veículos) *Standard*: €109 por 15.000 (máx. 10 veículos)	€55 / mês por <10.000 pedidos diários €105 por <20.000	Quando se trata de seguir *assets*, o preço tem de ser por *asset*. Min €2,200 p/ ano	€5000 p/ ano por 0 a 500.000 solicitações anuais	Pedidos de rota: 50.000 pedidos = €199 = € 0,00398 p/ pedido 100.000 pedidos = €379 = € 0,00379 p/ pedido

Tabela 7: APIs de Cálculo de Rotas

Here	MapBox	Sygic Maps	MapQuest	RouteXL	Routific	
✔	✔	✔	✔	✔	✔	Cobertura em Portugal
✔	✔	✔				Rotas cientes de trânsito
✔	✔	✔				Opções de rota (evitar autoestradas, …)
✔	✔	✔	✔ (sem instruções em Português)			Resposta com texto das indicações do percurso
✔						Cálculo de custos de portagens
		✔				Localização de radares
✔		✔				Rotas para veículos pesados
✔		✔				Opções para perfis de veículos pesados
Sim. Pontos ilimitados = 1 pedido	Sim. = 1 pedido Limite de 3 pontos com informação de trânsito Limite de 25 pontos sem informação de trânsito	✔	✔	✔ (máx. 10 no plano de *trial*)	✔	Permite fazer pedidos de rota com vários pontos obrigatórios?
✔	Restrições sobre o tempo que podemos manter os mapas em cache. Máx. 6000 tiles offline					Disponibiliza mapas isolados
✔ Imagens codificadas	✔ Imagens codificadas					Permite pedir dados de trânsito isolados
90 dias	Até 50.000 pedidos mensais	20 dias	Até 15.000 pedidos mensais	✔	30 dias	Versão de avaliação
Pedidos de rota: 50.000 pedidos = €199 = € 0,00398 p/ pedido 100.000 pedidos = €349 = € 0,00349 p/ pedido Ou **€30.50** por *asset* p/ ano	Restrictions = plano regular >250 users = licence customizada		30.000 pedidos mensais = $99 75.000 pedidos mensais = $199 200.000 pedidos mensais = $399 500.000 pedidos mensais = $799	€35 por mês. Pedidos ilimitados.	€150 p/ mês = 1000 visitas mensais €275 p/ mês = 2000 visitas mensais 3 meses de prova de conceito, com a API sem limites por €150 / mês	Preços

Tabela 7: APIs de Cálculo de Rotas (continuação)

6.2.2. Outras APIs Não Consideradas

Para além das APIs apresentadas na tabela 7, foram também contempladas oito outras que simplesmente não eram indicadas para os objetivos do projeto ou cujas empresas, infelizmente e mesmo após alguma insistência, não responderam às tentativas de contacto nem disponibilizaram o seu serviço de forma autónoma. Estas APIs foram:

- GBB;
- Route4me;
- LogVRP;
- TrackRoad;
- Mapsme;
- Locus Map;
- Navitia;
- OsmAnd.

6.2.3. Projetos para Referência

Com o intuito de recolher alguns dos elementos necessários e explorar várias possibilidades para o desenvolvimento do módulo de navegação, foi analisado um conjunto de aplicações de código aberto relacionadas.

Foram recolhidos vários projetos a partir da plataforma GitHub (GitHub, s.d.), com o objetivo de estudar a implementação de funcionalidades presentes nestas aplicações. Intuía-se que seria possível analisar abordagens de implementação, perceber limitações e recolher possíveis ferramentas como bibliotecas, por exemplo, que pudessem ser úteis no desenvolvimento de um módulo de navegação de raiz. Note-se que esta exploração não serviu para definir requisitos, pois estes já se encontravam definidos previamente.

No total, foram analisados vinte e quatro projetos de código aberto, expostos na tabela 8.

	Projeto	Endereço
1	AndRoad	https://github.com/gkfabs/AndRoad
2	Google Map Direction Turn by Turn	https://github.com/haylammmd/Google-Map-direction-turn-by-turn
3	MyRoute	https://github.com/maksumon/myRoute
4	TurnByTurnNavigation	https://github.com/KangSeungIl/TurnByTurnNavigation
5	Turn By Turn Navigation	https://github.com/sameershaik1996/turn-by-turn-navigation
6	TurnByTurnNavi	https://github.com/difriend/TurnByTurnNavi
7	PocketMaps	https://github.com/junjunguo/PocketMaps
8	Mobile Android Samples for Carto SDK	https://github.com/CartoDB/mobile-android-samples
9	RouteOffline	https://github.com/nikhilgiseng/RouteOffline
10	OfflineMap	https://github.com/konzaho/OfflineMap
11	Graphhopper Routing	https://github.com/graphhopper/graphhopper https://github.com/graphhopper/graphhopper-ios
12	Hellomap3d-android	https://github.com/nutiteq/hellomap3d-android
13	OfflineRouteMatcher	https://github.com/Icgemu/offline-route-matcher
14	OfflineMapAndroid	https://github.com/parthdave93/OfflineMapAndroidDemo
15	OfflineMap	https://github.com/yuviii/OfflineMap
16	Cunavigator	https://github.com/tobydigz/cunavigator
17	GarphHopper	https://github.com/konsoletyper/teavm-graphhopper
18	Offline Routing Java	https://github.com/garys-esri/offline-routing-java
19	OSM Routing App	https://github.com/jgrunert/Osm-Routing-App
20	Osmand	https://github.com/osmandapp/Osmand
21	OsmDroid	https://github.com/osmdroid/osmdroid
22	OfflineRoutingSample	https://github.com/lassana/offline-routing-sample
23	Mapsforge	https://github.com/mapsforge/mapsforge https://github.com/medvedNick/Mapsforge_iOS
24	Navit	https://github.com/navit-gps/navit

Tabela 8: Projetos de Código Aberto

Após o levantamento destas aplicações verificou-se que, infelizmente, a grande maioria encontra-se incompleta ou simplesmente descontinuada. Como tal, não foi possível recolher projetos robustos. No entanto este estudo permitiu agregar algumas ideias e ferramentas pertinentes para melhorar o sistema de navegação.

Especificamente, a partir destes projetos foi possível:

- Recolher uma biblioteca que implementa vários algoritmos de cálculo de rotas *offline*, intitulada Graphhopper, e que permite colocar grafos em memória para efetuar as operações necessárias ao cálculo de uma nova rota;
- Descobrir vários sistemas para a implementação de imagens de mapas *offline*, sem recurso às bibliotecas nativas de mapas já presentes em Android e iOS. A exploração destes projetos permitiu a recolha de três bibliotecas: V™, OsmDroid e MapsForge;
- Recolher um projeto de navegação que, embora incompleto e com problemas de implementação, representa uma boa base de desenvolvimento para um projeto mais robusto. Este projeto, o Navit, tem a vantagem de implementar um algoritmo de cálculo de rotas *offline*, que poderia ser estudado, adaptado, e implementado. No entanto, este projeto tem a desvantagem de possuir licenças GPLv2 e LGPLv2 pouco permissivas.

6.2.4. Reunião dos Elementos Necessários

Em adição aos vários elementos reunidos a partir da análise dos projetos, foi também efetuada uma exploração dos restantes elementos em falta, nomeadamente mapas, grafos e um sistema de acompanhamento de rotas.

Foi continuada a pesquisa sobre bibliotecas adicionais para a apresentação de mapas para Android e iOS, em conjunto com um levantamento de soluções para o cálculo de rotas *offline.* Esta pesquisa não revelou quaisquer resultados credíveis para além dos já recolhidos a partir da análise dos projetos.

Acerca da recolha de mapas, e após alguma exploração, foi possível encontrar um repositório das imagens dos vários mapas em http://download.mapsforge.org/maps/ (Mapsforge Download Server, s.d.). Foi também encontrado um repositório para os vários grafos em http://download.geofabrik.de/ (OpenStreetMap Data Extracts, s.d.).

Tratam-se de repositórios atualizados regularmente, o que constitui uma vantagem e elimina um dos problemas inerentes a uma implementação do sistema feita de raiz.

Não foi possível recolher o último elemento em falta, o sistema de acompanhamento de rotas, pelo que este terá de ser desenvolvido.

6.3. Seleção de uma API de Locais e Estabelecimentos

Para implementar a funcionalidade de apresentação de sugestões de visita a potenciais clientes, foi utilizada uma API de Locais e Estabelecimentos. Esta API tem a responsabilidade de disponibilizar informações referentes a vários estabelecimentos, que podem incluir, entre outros elementos, dados sobre a sua reputação, o número médio de visitas de cientes diárias, o seu horário de funcionamento, ou a que categorias corresponde. Estas métricas são necessárias para averiguar a relevância de cada estabelecimento para o negócio do utilizador.

Após o levantamento inicial, concluiu-se que, para garantir que seria apresentado um número aceitável de estabelecimentos em Portugal, só existiam realmente três opções credíveis: Google Places API, Foursquare Places API e Bing Places for Busness API. Uma análise mais cuidada revelou que as políticas da Google Places API, até à data de escrita deste relatório, não permitem a utilização das suas informações em mapas que não sejam os disponibilizados pela própria Google (Places API Policies, 2018). Esta limitação contrasta com a utilização de mapas de outras entidades, que, por utilizar mapas *offline*, é o caso deste projeto.

Um pequeno teste feito às duas outras APIs revelou que, infelizmente, a Bing Places for Business API não apresentava um número suficiente de estabelecimentos.

A escolha ficou limitada apenas à Foursquare Places API. Felizmente revelou ser uma boa opção, visto que, para além de incluir todos os elementos necessários à implementação deste requisito, disponibilizava um número maior de pedidos gratuitos que as outras opções.

6.4. Resumo

Durante o levantamento de alternativas de desenvolvimento foi possível traduzir a análise efetuada em duas abordagens de implementação distintas: uma primeira com

recurso a um SDK de navegação e uma segunda que corresponde a uma implementação de raiz de todo o sistema. Foi efetuada uma recolha de todos os componentes necessários e disponíveis para cada uma.

A primeira abordagem, por recorrer a um serviço de terceiros, não necessita de elementos adicionais para a navegação para além do próprio SDK. Cada um dos SDKs é uma ferramenta completa e robusta, pelo que já integra todas as funcionalidades necessárias à construção de um sistema de navegação.

Em contraste, a segunda abordagem, por se tratar de uma implementação feita de raiz, necessitou da reunião de um conjunto de elementos como APIs de cálculo de rotas, bibliotecas, algoritmos, e ficheiros de mapas e grafos que, por sua vez, teriam de ser compatíveis entre si e apresentar licenças de utilização permissivas.

Foi também feita uma análise sobre APIs de Locais e Estabelecimentos com o intuito de implementar o requisito referente à apresentação de sugestões de visita do Drivian Tasks.

7. Teste de Componentes

Reunidos os vários elementos considerados pertinentes para a construção do sistema, tornou-se necessário averiguar as verdadeiras possibilidades de cada uma das duas abordagens, seja a implementação do sistema de raiz, ou a implementação com recurso a um SDK. Simultaneamente também foi necessário averiguar a qualidade e o potencial de cada um dos elementos selecionados para a implementação do sistema de raiz.

Estes testes de componentes possibilitaram compreender qual das abordagens teria a maior capacidade para implementar os requisitos do projeto na totalidade, com a qualidade desejada, e de forma acessível, possibilitando tomar uma decisão verdadeiramente informada sobre a direção do projeto e a viabilidade da abordagem escolhida.

Até à data deste estudo, nenhum dos funcionários da empresa tinha conhecimentos ou experiência sobre as ferramentas aqui apresentadas, pelo que este estudo constituiu uma contribuição interessante para projetos futuros da Sentilant.

A pedido dos responsáveis pelo projeto, e com o intuito de acelerar o processo de decisão, a implementação destas aplicações de teste foi feita exclusivamente para Android, e apenas sobre os elementos considerados mais pertinentes. Os testes de componentes para iOS foram ignorados já neste fase por os orientadores entenderem que, provavelmente, não seria feita uma implementação do módulo de navegação para este sistema operativo.

Os vários elementos testados foram:

- Biblioteca de mapas Google Map, nativo do SDK para Android;
- Biblioteca de mapas V™;
- Biblioteca de mapas Osmdroid;
- Biblioteca de mapas MapsForge;
- Google Maps API;
- Bing Maps API;
- Projeto Navit;
- Biblioteca de cálculo de rotas Graphhopper;
- Mapas Mapsforge;

- Grafos Geofabrik;
- SDK de navegação Here Maps;
- SDK de navegação Mapbox;
- SDK de navegação Tally Go.

7.1. Desenvolvimento de Projetos de Teste de Componentes

Para averiguar qual das alternativas de implementação seria a mais promissora, foi construído um projeto de teste para cada uma das alternativas. Esses projetos traduziram-se no desenvolvimento de oito aplicações Android, que incluíram todos os SDKs considerados viáveis, assim como os vários conjuntos de bibliotecas que poderiam resultar no desenvolvimento de um sistema de navegação robusto implementado de raiz. Os vários protótipos funcionais desenvolvidos foram:

7.1.1. TestGoogleMaps

Aplicação desenvolvida para testar as possibilidades oferecidas pela biblioteca de mapas nativa do sistema Android, em conjunto com a utilização da API de cálculo de rotas disponibilizada pela Google, a Google Maps API (Google Maps Platform, s.d.).

7.1.2. TestBingMaps

Protótipo criado com o propósito de testar as possibilidades oferecidas pelo cruzamento da biblioteca de mapas Osmdroid (Open Street Maps for Android) (Osmdroid, 2018) com a API de cálculo de rotas Bing Routes API (Bing Routes API, 2018).

7.1.3. TestHereMaps

Projeto desenvolvido com o objetivo de analisar as possibilidades do SDK de navegação disponibilizado por HERE Maps (Here Maps SDK, 2018).

7.1.4. TestMapBox

Projeto criado para analisar as funcionalidades oferecidas pelo SDK de navegação MapBox (MapBox SDK, 2018).

Embora promissor, e apesar de disponibilizar navegação *offline*, este SDK apresentava a desvantagem de necessitar de acesso à *Internet* para estabelecer uma nova rota.

7.1.5. TestNavit

Projeto criado com recurso a uma aplicação de exemplo já existente intitulada Navit (2018). O projeto base apresenta diversos problemas de configuração, funcionalidade e navegação que o tornam inviável para o desenvolvimento do módulo de navegação para o Drivian Tasks. Embora fosse possível reconstruir estes elementos, o próprio projeto em si não demonstrou ser muito promissor. O facto de englobar uma licença GPL também foi considerado como fator limitador. Apesar de todos estes obstáculos, foi concebido um protótipo de navegação para testar a viabilidade desta abordagem.

7.1.6. TestGraphhopper

Protótipo desenvolvido com o intuito de analisar as capacidades da biblioteca de mapas V™ (2018) e a biblioteca de cálculo de rotas Graphhopper (2018). Neste projeto foram também testados os grafos do repositório OpenStreetMaps (OpenStreetMap Data Extracts, s.d.).

7.1.7. TestMapsforge

Projeto criado com o objetivo de testar as capacidades da biblioteca de mapas Mapsforge (2018) assim como as imagens de mapa provenientes do repositório Mapsforge (Mapsforge Download Server, s.d.).

7.1.8. TestTallyGo

Aplicação desenvolvida com o intuito de experimentar as várias funcionalidades disponibilizadas pelo SDK TallyGo (TallyGo SDK, s.d.).

7.2. Apresentação dos Projetos de Teste à Equipa

Reunidas as várias aplicações de teste, foi feita uma apresentação a todos os elementos da empresa na qual foram discutidos os pontos fortes e fracos das aplicações protótipo mais promissoras. O conteúdo da apresentação encontra-se disponível no anexo D.

Simultaneamente, foram também apresentadas as várias APIs de cálculo de rotas consideradas mais relevantes, os custos associados, e as suas várias características. Esta comparação foi feita com recurso à tabela apresentada previamente neste relatório (Tabela 7).

Do mesmo modo, também foi apresentada a tabela comparativa de SDKs (Tabela 6), na qual constam as várias características dos vários SDKs de navegação.

Munidos de todas as evidências necessárias, foi assumida pelos responsáveis da empresa a tarefa de tomar uma decisão final sobre qual abordagem seria a mais indicada para o desenvolvimento do módulo de navegação.

7.3. Resumo

Durante o teste de componentes para o módulo de navegação do Drivian Tasks, foram desenvolvidas várias aplicações para Android. O desenvolvimento destas aplicações teve o propósito de tomar uma decisão sobre qual das alternativas seria a mais indicada para o desenvolvimento da aplicação final.

O teste de componentes culminou numa apresentação feita a todos os elementos da empresa. Nessa apresentação foram ponderadas as características de cada uma das alternativas consideradas viáveis. Durante a mesma, os responsáveis da empresa recolheram as informações que consideraram mais pertinentes para, em conjunto com o documento de requisitos definidos para o sistema, tomarem uma decisão final sobre a abordagem a seguir.

8. Implementação

Com os projetos de teste desenvolvidos, e a potencialidade de cada uma das ferramentas devidamente comprovada, foi dado início à fase de implementação. No seu início, houve um pequeno período no qual os responsáveis da empresa tomavam decisões de negócio com impacto direto no projeto. Apenas após tomadas essas decisões se iniciou a escrita do código da aplicação em concreto.

8.1. Decisões Gerais de Desenvolvimento

Já desde a apresentação feita à equipa sobre as várias alternativas de implementação que estava pendente a decisão sobre se a implementação do sistema de navegação para iOS traria realmente algum acréscimo de valor ao Drivian Tasks. Com efeito, o número de utilizadores do Drivian Tasks com este sistema operativo é extremamente reduzido. Aquando do início da fase de desenvolvimento, foi decidido que a implementação do sistema para iOS seria abandonada. Esta decisão teve também como finalidade permitir uma implementação mais completa do que a inicialmente prevista, visto que nesse momento os responsáveis da empresa consideravam a possibilidade de desenvolver um sistema de raiz. Com efeito, uma implementação de raiz traria um acréscimo de valor mais acentuado à própria empresa, pois seria algo proprietário da mesma e teria a possibilidade de ser aplicável a futuros projetos, eliminando dependências e custos associados à utilização de uma ferramenta de navegação externa. Deste modo, o desenvolvimento do módulo de navegação focou-se apenas em Android.

Após a apresentação das alternativas de implementação, foi também pedido que fosse elaborado um pequeno plano de previsão que comparasse uma implementação do sistema de navegação feita de raiz com uma implementação recorrendo a um SDK, para que os responsáveis pudessem tomar uma decisão final sobre a direção da implementação.

A partir deste plano de previsão, foi decidido que o sistema de navegação seria, realmente, implementado de raiz, reconstruindo todos os elementos necessários e utilizando componentes gratuitas com licenças permissivas. Esta decisão fundamentou-se no facto de os SDKs atualmente disponíveis no mercado apresentarem limitações contrastantes com os requisitos previamente definidos. Este é o caso do MapBox SDK,

que, não funcionando inteiramente *offline*, necessita de acesso à *Internet* para pedir o cálculo inicial de um percurso. Outro motivo relevante para esta decisão foi o facto da utilização de um SDK ter custos associados que condicionariam algumas decisões de negócio inerentes ao Drivian Tasks, como o preço apresentado ao cliente ou a escalabilidade da aplicação.

Como metodologia de trabalho, foi estabelecido que o desenvolvimento da aplicação seria feito de forma ágil, baseado em Scrum, mas com algumas diferenças importantes: Apesar de existirem *sprints* com uma duração de duas semanas, após as quais um subconjunto de funcionalidades deveria estar implementado na totalidade, este não resultaria, necessariamente, numa versão funcional do produto, embora tal se sucedesse a grande maioria das vezes; não existira também uma equipa de trabalho, sendo este desenvolvido de forma individual e, como tal, não existiria Scrum Master ou reuniões diárias. Estas foram substituídas por reuniões semanais para que o Professor Dr. Bruno Cabral acompanhasse o desenvolvimento do projeto.

Dado algumas das funcionalidades previstas não se encontrarem ainda implementadas do lado do servidor, e sendo estas atribuídas aos membros da equipa da Sentilant responsáveis pelo mesmo, foi também decidido que as funcionalidades que dependessem desta implementação seriam relegadas para a fase final do projeto. Infelizmente, não tendo sido possível implementar as funcionalidades do lado do servidor até ao fim do projeto, não foi possível desenvolver algumas das funcionalidades previstas. Este foi o caso da atribuição de permissões aos operacionais no terreno, da escolha de opções de percurso para os operacionais, que seria feita na aplicação *web*, e da possibilidade de reportar imprevistos.

Por fim, foi estabelecido que o módulo de navegação do Drivian Tasks seria desenvolvido de forma autónoma, numa aplicação própria e isolada, mas composta visando a sua futura integração num sistema maior. Deste modo foi possível realizar facilmente testes de navegação no terreno e fazer alterações, otimizações e melhorias constantes, até o sistema apresentar robustez suficiente para ser integrado.

8.2. Ferramentas Utilizadas

Para a implementação do sistema de navegação do Drivian Tasks, foi escolhido um conjunto de elementos a partir dos vários elementos já testados. São eles:

8.2.1. Navegação baseada em GPS

Como descrito anteriormente, foi decidido que o sistema de navegação seria construído com recurso ao sistema de posicionamento geográfico por GPS. As razões para enveredar por esta opção já se encontram expostas nas secções de Estado da Arte e Proposta Estratégica deste relatório.

8.2.2. Android Studio

Atendendo ao facto da implementação para iOS ter sido descontinuada, foi definido que o ambiente de desenvolvimento da aplicação seria exclusivamente o Android Studio. Esta decisão teve como base não só a facilidade de integração no Drivian Tasks, também este desenvolvido em Android Studio, mas também simplificar o seu desenvolvimento, tirando partido do facto de este IDE ter sido desenvolvido com o exclusivo propósito de desenvolver aplicações para Android, pelo que todas as suas funcionalidades estão alinhadas nesse sentido.

8.2.3. Biblioteca de Mapas V™

Para incluir mapas no módulo de navegação do Drivian Tasks, foi decidido utilizar uma biblioteca de mapas de código aberto e licença permissiva intitulada V™ (2018). Esta biblioteca, vertente de um projeto maior, também ele de código aberto, o Open Science Maps, permite integrar mapas em dispositivos móveis para utilização inteiramente *offline*. Trata-se de uma biblioteca que disponibiliza classes para importação de mapas a partir de ficheiros .map, assim como classes que permitem a sua apresentação em aplicações, ou a construção de mapas e respetivas informações por camadas.

Esta biblioteca apresenta a grande vantagem de permitir um funcionamento inteiramente *offline*, um dos requisitos essenciais do módulo de navegação do Drivian Tasks, ao contrário da biblioteca de mapas nativa em Android que, pelas limitações impostas pela sua empresa, a Google, até à data, é muito restritiva no que se refere à transferência de mapas para utilização *offline*. Por outro lado, apresenta a desvantagem de necessitar que o utilizador ocupe o seu dispositivo com os ficheiros de mapas, para além de requerer o desenvolvimento de um sistema de gestão de mapas na aplicação com o propósito de lidar com estas limitações.

8.2.4. Mapas Mapsforge

Para adquirir ficheiros de mapas para disponibilizar aos utilizadores do Drivian Tasks, foram utilizados os mapas do repositório Mapsforge (Mapsforge Download Server, s.d.). Este repositório contém um conjunto de mapas referente a todo o Mundo para utilização livre e gratuita. É também compatível com a biblioteca de mapas V™, pelo que se torna uma opção ideal para o projeto.

8.2.5. Bing Routing API

A Bing Routing API (Bing Routes API, 2018) foi utilizada apenas para desenvolver o sistema de roteamento e acompanhamento de rotas com todas as suas funcionalidades. Esta decisão teve como fundamento o facto de a API de roteamento disponibilizada pela empresa Sentilant não incluir dados relativos a trânsito e a eventos nas estradas aquando do desenvolvimento destas funcionalidades do módulo de navegação. Pelo facto de um dos requisitos do sistema contemplar a inclusão de informação de trânsito nas rotas, foi decidido implementar um adaptador que permita à empresa utilizar qualquer API na aplicação, com a possibilidade acrescida de incluir este tipo de informação. Nesse sentido a Bing Routing API permitiu conceber um sistema inicial que inclui a possibilidade de apresentação de informação de trânsito, condições de vias e imprevistos nas estradas. Este sistema foi, posteriormente, adaptado à API da empresa. O sistema fica assim com algumas capacidades de inclusão de trânsito que não ficam diretamente em utilização, mas que estão previstas para o futuro, assim que a empresa adquirir dados dessa natureza.

8.2.6. Drivian Routing API

Concluído o sistema de roteamento com inclusão de informação de trânsito, e depois dos respetivos requisitos estarem devidamente implementados, o sistema foi adaptado à API disponibilizada pela empresa.

Esta API baseia-se no projeto de contribuição comunitária Open Source Routing Machine (OSRM) (OSRM Backend, 2018) integrado num servidor da empresa Sentilant com a finalidade de ser utilizado pelos seus próprios projetos e clientes.

Esta API, tal como a Bing Routing API descrita anteriormente, permite criar rotas entre dois pontos geográficos, mediante determinadas especificações, embora não inclua, até à data, informações relativas a trânsito e condições de vias.

8.2.7. Motor de Cálculo de Rotas Graphhopper

Para realizar o cálculo de rotas de forma inteiramente *offline*, um dos requisitos essenciais previstos para o módulo de navegação do Drivian Tasks, foi utilizado o motor de cálculo de rotas de código aberto Graphhopper (2018). Este motor permite importar grafos para a memória do dispositivo e utilizar os mesmos para calcular rotas entre dois pontos geográficos. Deste modo, é possível limitar o número de pedidos feitos pela aplicação à API de roteamento, e permitir uma utilização do módulo de navegação em locais onde não haja ligação à *Internet*.

Para o cálculo dos percursos *offline* foi escolhido o algoritmo A*, implementado no motor de cálculo de rotas Graphhopper, sendo considerado uma variação informada do algoritmo de Dijkstra. Esta opção tem como fundamento o algoritmo A* considerar pesos atribuídos aos vários nós do grafo, conseguindo, dessa forma, que o resultado apresentado se aproxime mais do resultado ótimo. Embora recorra a mais memória e necessite de mais operações por nó que o algoritmo de Dijkstra, pelo facto de considerar os pesos de cada nó, acaba por, em última instância, explorar muito menos nós que o algoritmo de Dijkstra, que por sua vez explora todas as possibilidades de igual forma. Por esse motivo, o A* é mais rápido e apresenta melhores resultados, o que o torna a escolha óbvia para a construção do sistema.

8.2.8. Grafos para Cálculo de Rotas do OpenStreetMaps

Os grafos utilizados para cálculo de rotas, importados para a aplicação pelo motor Graphhopper, foram recolhidos a partir do repositório OpenStreetMaps (OpenStreetMap Data Extracts, s.d.). Este repositório contém ficheiros de grafos referentes à disposição das estradas em todo o Mundo. É também atualizado regularmente, o que o torna numa escolha ideal para o projeto. Após adquiridos os ficheiros, estes são tratados para poderem ser importados para o módulo de navegação. Mais uma vez, trata-se de um elemento com licenças permissivas, sem quaisquer encargos legais ou financeiros.

8.3. Descrição das Várias Componentes do Sistema

8.3.1. Modelo C4

De forma a melhor apresentar a estrutura do módulo de navegação e descrever a sua integração na aplicação móvel do Drivian Tasks, será utilizado o padrão de arquitetura

C4 (C4 Model, s.d.). Foi tomada a opção de utilizar este modelo porque permite focar gradualmente cada uma das componentes da aplicação a partir do seu contexto maior, o que é ideal no caso da descrição de um módulo para uma aplicação já existente. Composto por quatro níveis, este modelo tem como vantagens definir diagramas simples, o que facilita a sua leitura e reduz a distância entre o *design* do sistema e a sua implementação. Ao mesmo tempo, permite também o detalhe minucioso de cada um dos componentes do sistema. Os diagramas apresentados foram elaborados nas aplicações yEd (yEd, s.d.) e Inkscape (Inkscape, s.d).

8.3.1.1. Nível 1: Diagrama de Contexto

O primeiro nível do modelo C4 refere-se ao diagrama de contexto. Este diagrama permite apresentar o funcionamento geral de toda a aplicação a partir da perspetiva dos seus utilizadores. Neste primeiro nível, o detalhe não é considerado importante, dado que a finalidade deste diagrama é a sua apresentação a indivíduos que não têm necessariamente conhecimentos técnicos na área de informática.

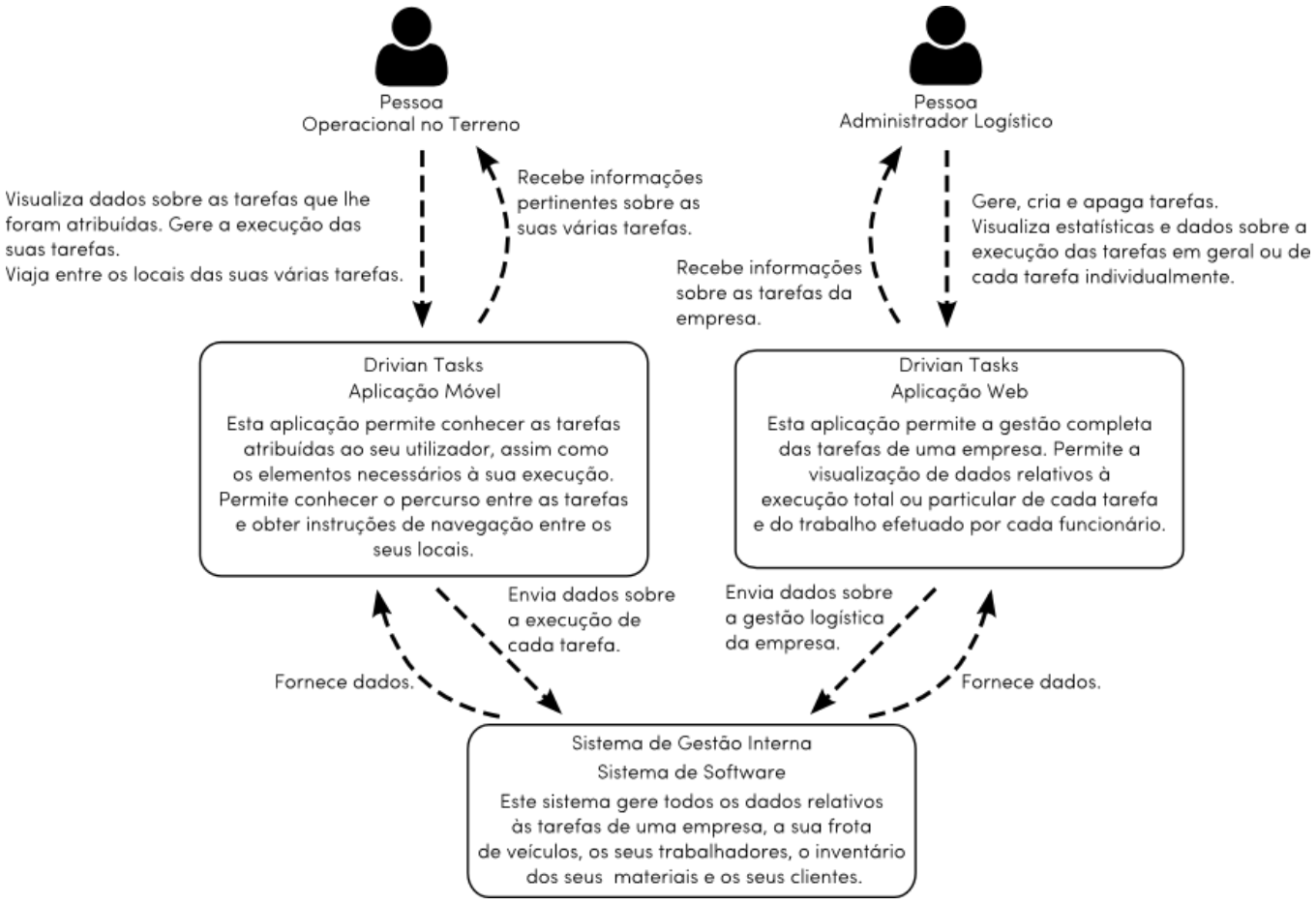


Figura 4: Diagrama de Contexto

A partir do diagrama apresentado, é possível verificar que, como já descrito nas secções anteriores deste relatório, o sistema do Drivian Tasks é composto por duas aplicações: uma aplicação web, a partir da qual um utilizador responsável pela organização logística de uma empresa pode organizar as várias tarefas dos seus funcionários; e uma aplicação móvel, através da qual um funcionário no terreno consegue obter informações sobre as suas tarefas e, através da mesma, relatar todos os detalhes da sua execução ao sistema central.

No centro do diagrama encontra-se um sistema de gestão interna cuja responsabilidade é sincronizar os vários dados entre as aplicações. Esta centralização da informação permite que cada um dos utilizadores tome conhecimento do estado de execução das várias tarefas, limitando a disparidade entre as informações apresentadas a cada um.

8.3.1.2. Nível 2: Diagrama de *Containers*

No segundo nível do modelo C4 encontra-se o diagrama de *containers*. Este diagrama tem a responsabilidade de expor visualmente a estrutura geral do sistema, apresentar as decisões de alto nível sobre as tecnologias a utilizar, definir de que forma as responsabilidades estão distribuídas pelas várias componentes, e, essencialmente, descrever de que forma os vários *containers* comunicam entre si.

Por *container* entende-se algo que envolve código ou dados. Por essa definição, um *container* é algo que precisa de estar ativo para o sistema funcionar.

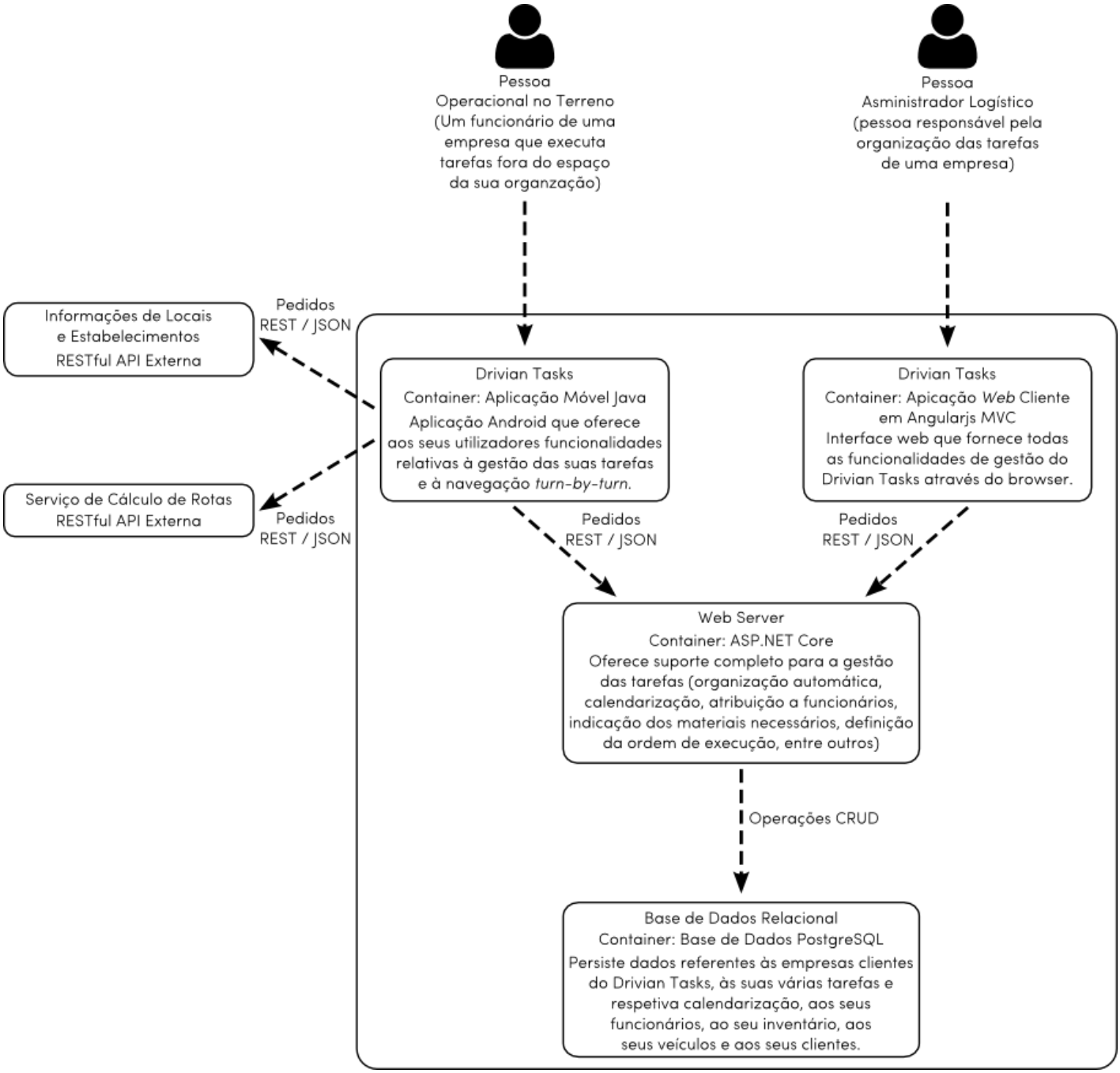


Figura 5: Diagrama de *Containers*

O diagrama apresentado revela que a comunicação entre as várias componentes do sistema será feita com recurso a objetos em formato JSON (*Javascript Object Notation*) transmitidos através de três RESTful APIs: uma primeira que disponibiliza um serviço de roteamento, recebe pedidos de rotas e devolve dados que serão transformados num percurso a ser interpretado pela aplicação móvel; uma segunda para aquisição de dados sobre estabelecimentos para apresentação das sugestões de visita no mapa da aplicação;

e uma terceira API, que serve como porta de acesso ao servidor *web* responsável por receber e transmitir todos os dados do Drivian Tasks.

Simultaneamente, o servidor *web* deverá ter acesso a uma base de dados, onde poderá persistir dados através das operações habituais de *Create*, *Retrieve*, *Update* e *Delete*.

Esta estrutura integra-se harmoniosamente com a já existente para o Drivian Tasks, já apresentada neste relatório.

8.3.1.3. Nível 3: Diagrama de Componentes

O terceiro nível do modelo C4 expõe os vários componentes de cada um dos *containers*. O seu objetivo é decompor cada um dos *containers* nas suas partes essenciais e, consequentemente, melhor identificar os vários elementos necessários ao seu desenvolvimento. Do mesmo modo, pretende traduzir o conteúdo de cada um desses componentes, quais a suas responsabilidades e, se necessário, especificar alguns detalhes de implementação.

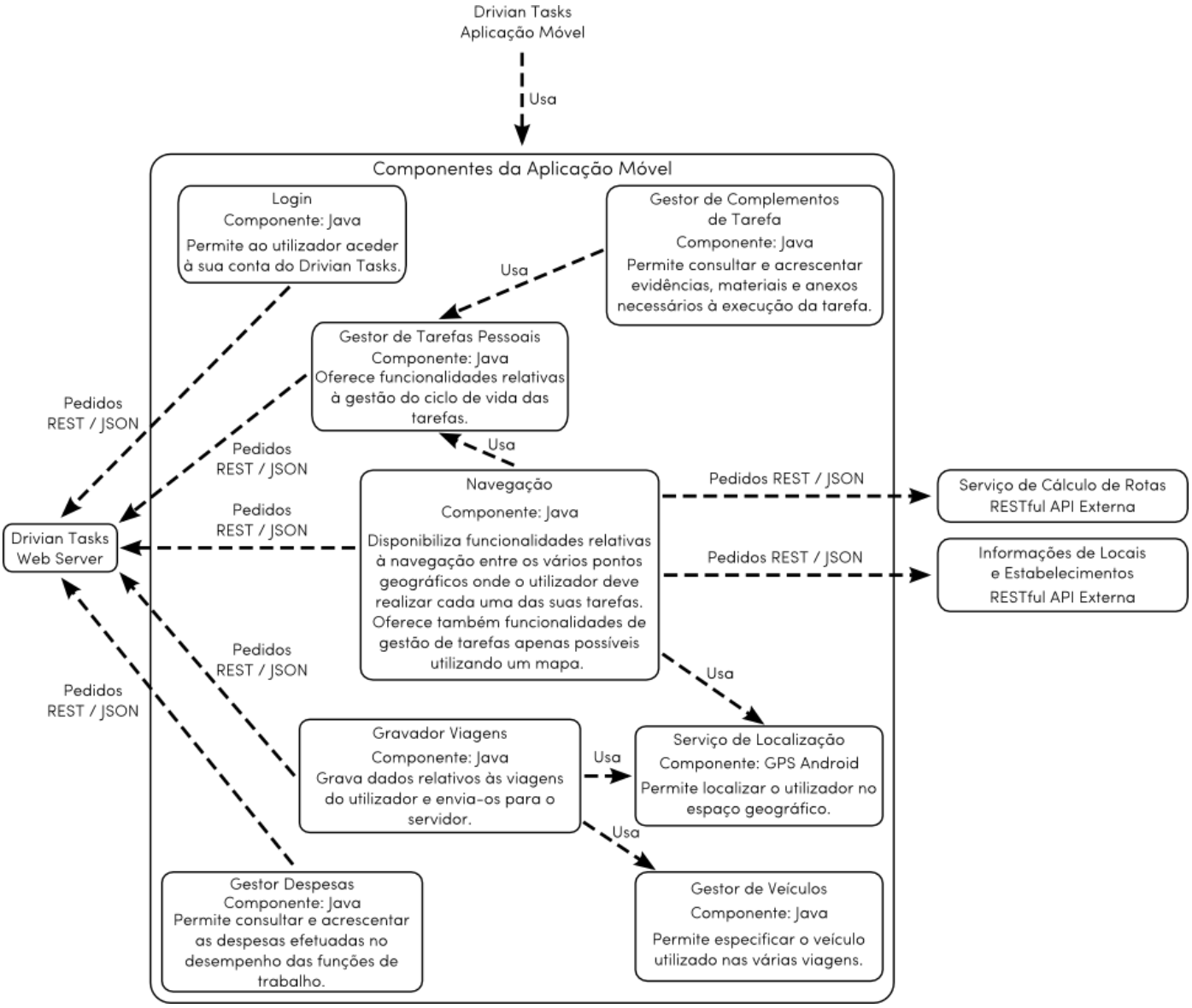


Figura 6: Diagrama de Componentes

O diagrama apresentado permite destacar todos os elementos que compõem a aplicação móvel do Drivian Tasks. Com efeito, esta aplicação contempla diversos módulos, entre eles o módulo de navegação, foco central deste estágio, e que será explicado em detalhe no próximo diagrama. Os restantes módulos referem-se a uma componente de *Login*, cuja responsabilidade é restringir o acesso a utilizadores não clientes do Drivian Tasks; um Gestor de Tarefas que apresenta ao utilizador as suas várias tarefas com as respetivas características como calendarização e estado de execução; um Gestor de Complementos de Tarefa, através do qual um utilizador poderá verificar que materiais são necessários à execução de uma determinada tarefa, ou acrescentar evidências do seu cumprimento como fotografias do trabalho efetuado ou assinaturas dos clientes; um

Serviço de Localização, responsável por localizar o utilizador no espaço geográfico e inferir as suas coordenadas de latitude e longitude; um Gravador de Viagens, responsável por transmitir ao servidor web os percursos efetuados pelo utilizador; um Gestor de Veículos, que permite ao utilizador selecionar o veículo que está a utilizar; e um Gestor de Despesas, para que o utilizador possa registar as suas despesas durante o desempenho das suas funções profissionais.

Através do diagrama, é possível verificar que diversos módulos formulam pedidos e transmitem dados para o servidor do Drivian Tasks. Apenas o módulo de navegação utilizará o serviço externo de cálculo de rotas.

8.3.1.4. Nível 4: Diagrama de Código

O último nível do modelo C4 refere-se ao Diagrama de Código, equiparável ao habitualmente utilizado Modelo de Domínio. Nele constam todas as classes que compõem cada um dos módulos apresentados no Diagrama de Componentes, o nível anterior do modelo C4. Dado que o foco deste estágio é o desenvolvimento de um módulo de navegação para a aplicação móvel do Drivian Tasks, apenas este será representado neste relatório.

A figura 7 pretende espelhar a estrutura do módulo de navegação, implementado na aplicação móvel do Drivian Tasks.

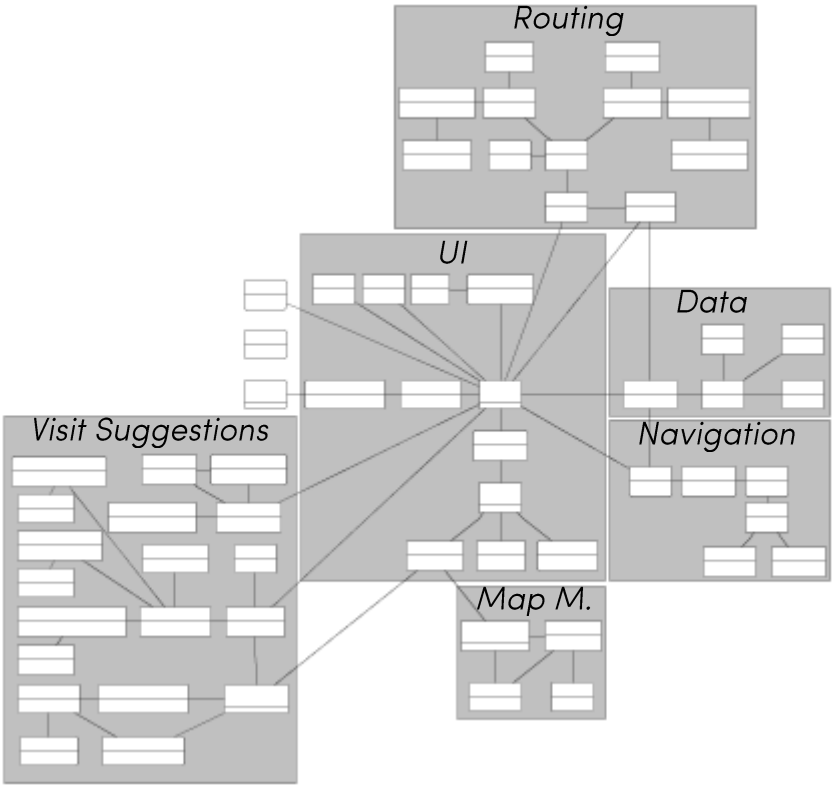


Figura 7: Diagrama de Código

Embora não conste no modelo C4, para melhor organizar a estrutura do módulo de navegação, este foi dividido em seis subsistemas: *UI*, *Data*, *Routing*, *Navigation*, *Map Management* e *Visit Suggestions*. Cada um destes subsistemas contempla um leque de funcionalidades próprio que será descrito na próxima secção, a Descrição Detalhada do Sistema.

8.3.2. Descrição Detalhada do Sistema

Com o objetivo de a facilitar a organização e a descrição do sistema, foi tomada a opção de agrupar as várias classes apresentadas no Diagrama de Código em seis subsistemas. Cada um dos subsistemas representa um conjunto de classes com um foco específico e com responsabilidades exclusivas.

Em acréscimo à descrição detalhada do sistema, existem também alguns elementos centrais que, embora apenas sejam mencionados, serão relatados em pormenor na próxima subsecção do relatório. Estes elementos incluem, entre outros, os algoritmos desenvolvidos e ferramentas construídas durante o desenvolvimento do projeto.

Existem também três classes adicionais que não estão integradas nos subsistemas mencionados. Estas classes não deixam de ser fundamentais e, como tal, serão igualmente descritas.

8.3.2.1. Classes Adicionais

A primeira das três classes adicionais refere-se à MainActivity. Sendo uma aplicação desenvolvida para Android, os elementos de interface deverão estar, necessariamente, encapsulados numa classe Activity. Dado o objetivo de integração do módulo de navegação no Drivian Tasks, o sistema foi concebido a partir de um objeto Fragment que, segundo a estrutura nativa do Andorid, deverá estar integrado numa classe Activity. Este é o ponto de entrada do sistema.

A segunda classe refere-se a um serviço de GPS provisório. Inicialmente, no módulo de navegação desenvolvido de forma isolada, foi criado um Android Service responsável por receber e processar o sinal de GPS recolhido pelo dispositivo. Após a integração com o sistema completo do Drivian Tasks, este Service foi substituído com o já presente no Drivian Tasks.

Por fim, a classe NavSettings regista todas as opções tomadas pelo utilizador sobre o funcionamento do sistema de navegação. As opções incluem, por exemplo, ligar ou desligar edifícios em 3D no mapa; voltar a focar posição do utilizador N segundos após este mexer no mapa; ligar ou desligar modo de chamadas em alta voz durante a condução; apresentar ou não apresentar velocidade de condução recomendada. Esta classe é também responsável por persistir as opções do utilizador e por restituir as mesmas sempre que o utilizador voltar a ligar a aplicação.

8.3.2.2. *UI Subsystem*

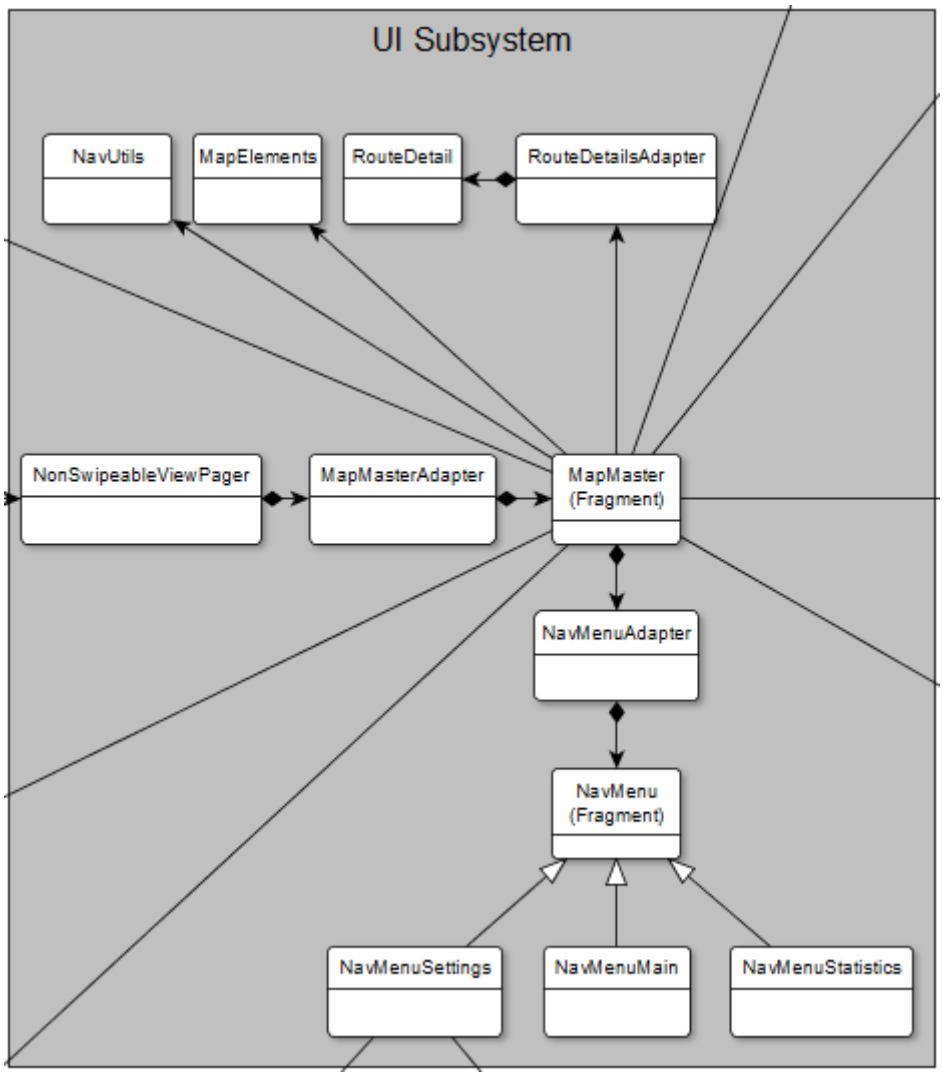


Figura 8: UI *Subsystem*

O subsistema de UI (*User Interface* ou Interface de Utilizador) encapsula todas as funcionalidades sobre os vários elementos apresentados ao utilizador. Como tal, inclui menus, botões e o próprio mapa, assim como todos os elementos nele contidos. Este sistema funciona como porta de entrada para os vários *inputs* do utilizador e como porta de saída para os vários elementos que lhe deverão ser apresentados.

Dado o requisito de fácil integração do módulo de navegação no Drivian Tasks, o elemento essencial deste sistema, ou seja, a interface de utilizador que contém o mapa, foi desenvolvida a partir de um Fragment de Android. Isto permite que este fragmento, a classe MapMaster, seja simplesmente adicionado ao projeto existente que já se encontrava construído com um fragmento temporário pronto a ser substituído. Esta decisão facilita também a integração deste módulo de navegação em outras aplicações.

Sendo a maior classe de todo o sistema (com perto de 5 KLOCs, ou 5000 linhas de código), a classe MapMaster contempla imensos comportamentos impossíveis de isolar, tais como todas as interações do utilizador com o mapa, ou todas as representações visuais a apresentar no mapa. É a criação deste fragmento que dá origem à instanciação de todos os outros elementos necessários ao sistema de navegação. A atribuição desta responsabilidade é essencial, visto que nenhum dos elementos de navegação será instanciado na aplicação até o utilizador utilizar o modo de navegação pela primeira vez.

Os vários elementos que poderão ser apresentados no mapa encontram-se encapsulados na classe MapElements para permitir a sua fácil alteração, caso desejado.

Os vários menus que poderão ser apresentados incluem a classe NavMenuMain que apresenta todas as funcionalidades do menu principal previstas para o módulo de navegação. A partir deste menu, o utilizador poderá navegar para o menu de opções, correspondente à classe NavMenuSettings, responsável por apresentar as várias opções do módulo de navegação passíveis de serem alteradas pelo utilizador, persistindo e processando as suas escolhas através da classe NavSettings. O terceiro submenu, NavMenuStatistics, é responsável por apresentar ao utilizador estatísticas referentes ao nível de segurança e economia da sua condução sob a forma de gráficos circulares e histogramas.

8.3.2.3. *Data Subsystem*

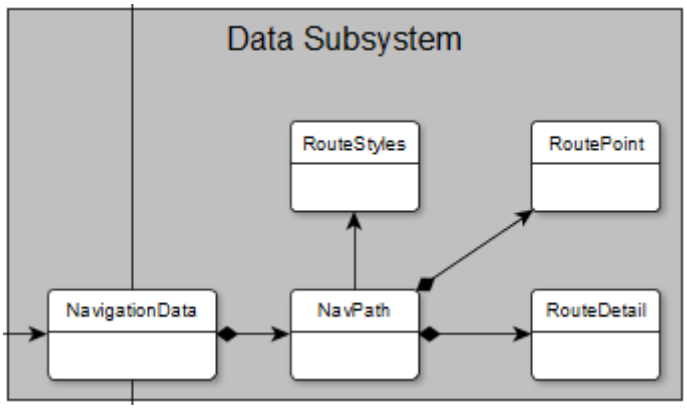


Figura 9: *Data Subsystem*

O subsistema de dados, ou *Data Subsystem*, é responsável por guardar todos os dados referentes à navegação do utilizador na memória do dispositivo. Como tal, inclui todos

os dados referentes à sua posição geográfica, à sua direção, ao nível de precisão do sinal de GPS, e ao seu atual percurso sob a forma de um objeto NavPath.

Cada objeto NavPath encapsula, por sua vez, a lista dos pontos que o compõe sob o formato de objetos RoutePoint. Estes objetos, que se referem a pontos geográficos com coordenadas latitude / longitude, incluem características como trânsito, obras na via, acidentes, entre outros.

Cada NavPath, ao ser criado a partir de uma lista de pontos, gera e guarda outros elementos como a lista de indicações a apresentar ao utilizador através do modelo RouteDetails e as camadas a desenhar no mapa com as linhas do percurso. Esta classe é também responsável por processar adições ou remoções dos seus pontos de percurso, o que deverá suceder de cada vez que o percurso é ajustado.

A classe RouteStyles serve apenas para encapsular os vários estilos de linhas possíveis a apresentar em cada percurso, tais como trechos de estrada sem transito ou com pouco, médio ou muito trânsito. O seu objetivo é, simplesmente, facilitar a alteração dos elementos visuais dos percursos caso desejado.

8.3.2.4. *Routing Subsystem*

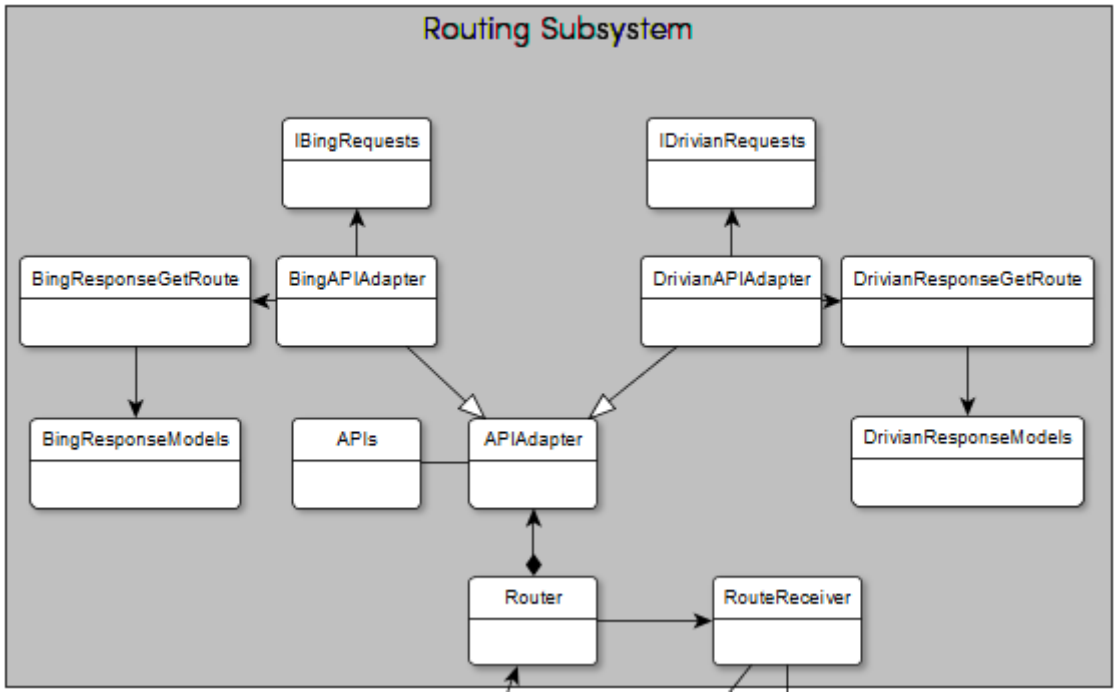


Figura 10: *Routing Subsystem*

Sempre que é necessário calcular rotas entre dois pontos geográficos, a navegação recorre ao subsistema de roteamento. Este subsistema é responsável pela criação de

rotas, fazendo preferencialmente pedidos à API externa de cálculo de rotas para calcular um novo percurso. Caso ocorra algum problema neste pedido ou a ligação à *Internet* não esteja disponível, o cálculo é feito de forma local, *offline*, no próprio dispositivo móvel. No entanto, por ser *offline*, a rota apresentada deste modo não inclui informação de trânsito ou de imprevistos na via tais como acidentes ou obras na estrada, por exemplo.

Este subsistema também oferece a possibilidade de efetuar um cálculo exclusivamente *offline*, o que é utilizado no caso de um ajuste da rota.

Funcionando como "porta de entrada" para o subsistema, um Router recebe pedidos de cálculo de rotas e, caso haja acesso à *Internet*, transmite esse pedido para o adaptador da API de cálculo de rotas. O Router é também responsável por efetuar cálculos de rotas *offline* recorrendo à biblioteca de roteamento Graphhopper.

Após receber ou calcular uma rota, e dada a natureza assíncrona dos pedidos de cálculo de rotas, o Router transmite o percurso a um objeto RouteReceiver, responsável por fazer o seu pós-processamento consoante este seja uma nova rota ou um ajuste de percurso, criando um novo objeto de percurso ou adicionando pontos ao percurso atual, respetivamente.

8.3.2.5. *Navigation Subsystem*

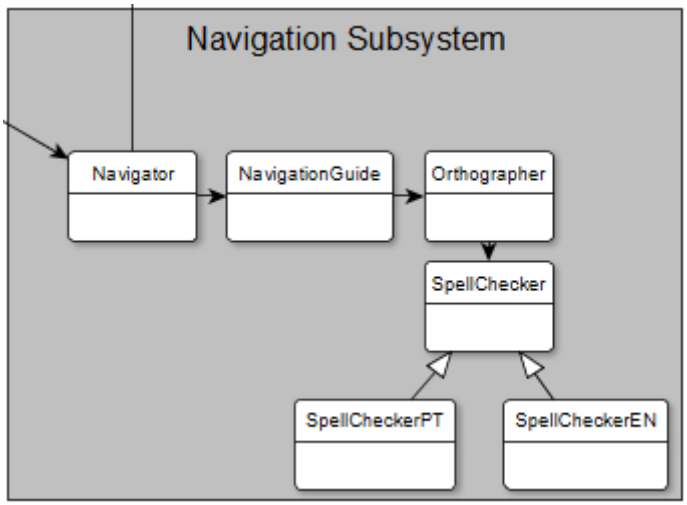


Figura 11: *Navigation Subsystem*

Sendo considerado o sistema central do módulo de navegação, o subsistema de navegação é responsável por interpretar se o utilizador se encontra a seguir o percurso,

pedir ajustes de rota, transmitir instruções relevantes, e construir instruções omissas consoante necessário.

De cada vez que o sistema atualiza os dados sobre a posição do utilizador no espaço geográfico, o Navigator processa esses dados através de um algoritmo que tenta interpretar se o utilizador está, de facto, a seguir o percurso: o algoritmo de acompanhamento de rotas. Se o utilizador estiver a seguir o percurso esperado, o Navigator informa o NavigationGuide, que, por sua vez, deverá apresentar uma instrução relevante se necessário.

Caso o utilizador não se encontre a seguir o percurso, o Navigator tem a responsabilidade de ajustar o percurso. A forma como este ajuste é calculado é relegada para o algoritmo de ajuste de rotas.

Os dois algoritmos mencionados, o algoritmo de acompanhamento de rotas e o algoritmo de ajuste de rotas, serão devidamente explicados em detalhe na próxima subsecção deste relatório.

O NavigationGuide, por sua vez, é responsável por transmitir ao utilizador instruções relevantes e pertinentes em áudio, recorrendo a um sistema de texto para voz.

De forma a garantir que as instruções transmitidas ao utilizador são sempre pertinentes, o NavigationGuide divide as instruções em dois tipos: instruções imediatas e instruções distantes. Esta interpretação é feita mediante a distância a que o utilizador se encontra de um ponto de instrução.

Instruções imediatas tomam sempre precedência sobre instruções distantes. Caso não haja instruções imediatas a transmitir, o sistema começará a transmitir instruções sobre pontos distantes a partir de uma lista ou "*queue*". Esta *queue* tem um máximo de três instruções, sendo que, ao ultrapassar este limite, as instruções mais antigas são retiradas por serem consideradas obsoletas.

O NavigationGuide utiliza preferencialmente instruções provenientes da API de cálculo de rotas. No entanto, caso a API não ofereça instruções, o sistema constrói instruções de raiz com base na tipologia do percurso. Este sistema de construção de instruções será também devidamente detalhado na próxima subsecção deste relatório.

Devido ao facto de que as instruções sobre o percurso são transmitidas ao utilizador através de áudio, torna-se essencial garantir que o seu texto seja construído de forma

clara, sem erros ortográficos, para que seja pronunciado corretamente. Um objeto Orthographer tem a responsabilidade de rever o texto a ser interpretado pelo sistema de texto para voz, seja este proveniente da API de cálculo de rotas ou do sistema de construção de instruções, pedindo as devidas correções ortográficas conforme necessário. Estas correções são feitas com recurso a um objeto SpellChecker, que é selecionado dependendo da linguagem escolhida para a aplicação.

8.3.2.6. *Map Management Subsystem*

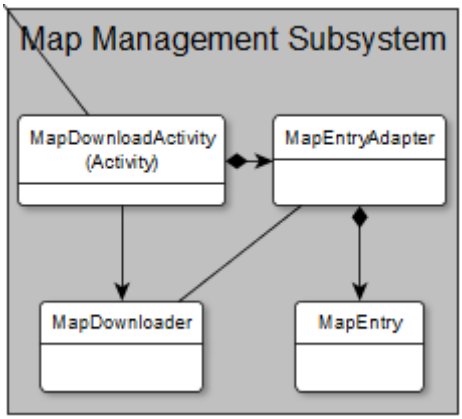


Figura 12: *Map Management Subsystem*

O subsistema de gestão de mapas gere tudo o que se refere aos ficheiros de mapas e grafos a serem utilizados pelo sistema. Como tal, verifica que ficheiros se encontram presentes no dispositivo, e se os ficheiros de mapas de cada país se encontram completos e em boas condições. É também responsável por gerir toda a aquisição de novos mapas, caso solicitada pelo utilizador.

Sendo uma classe Activity de Android, a MapDownloadActivity é responsável por apresentar uma lista de todos os mapas disponíveis no servidor, assim como verificar que mapas já se encontram presentes no dispositivo. Desse modo, consegue informar facilmente o utilizador que mapas tem e que mapas poderá ter, assim como contemplar quaisquer alterações ao ficheiros de mapas disponíveis no servidor. Esta classe é também responsável por interpretar as escolhas do utilizador no que se refere a mapas que pretende apagar ou transferir.

A classe MapDownloader implementa um sistema de transferência de mapas que acede ao servidor, transfere um ficheiro comprimido referente a um mapa, e extrai os seus vários ficheiros para a diretoria pertinente. Gere também uma lista, ou "*queue*", para o caso de o utilizador solicitar a transferência de múltiplos mapas.

8.3.2.7. *Visit Suggestions Subsystem*

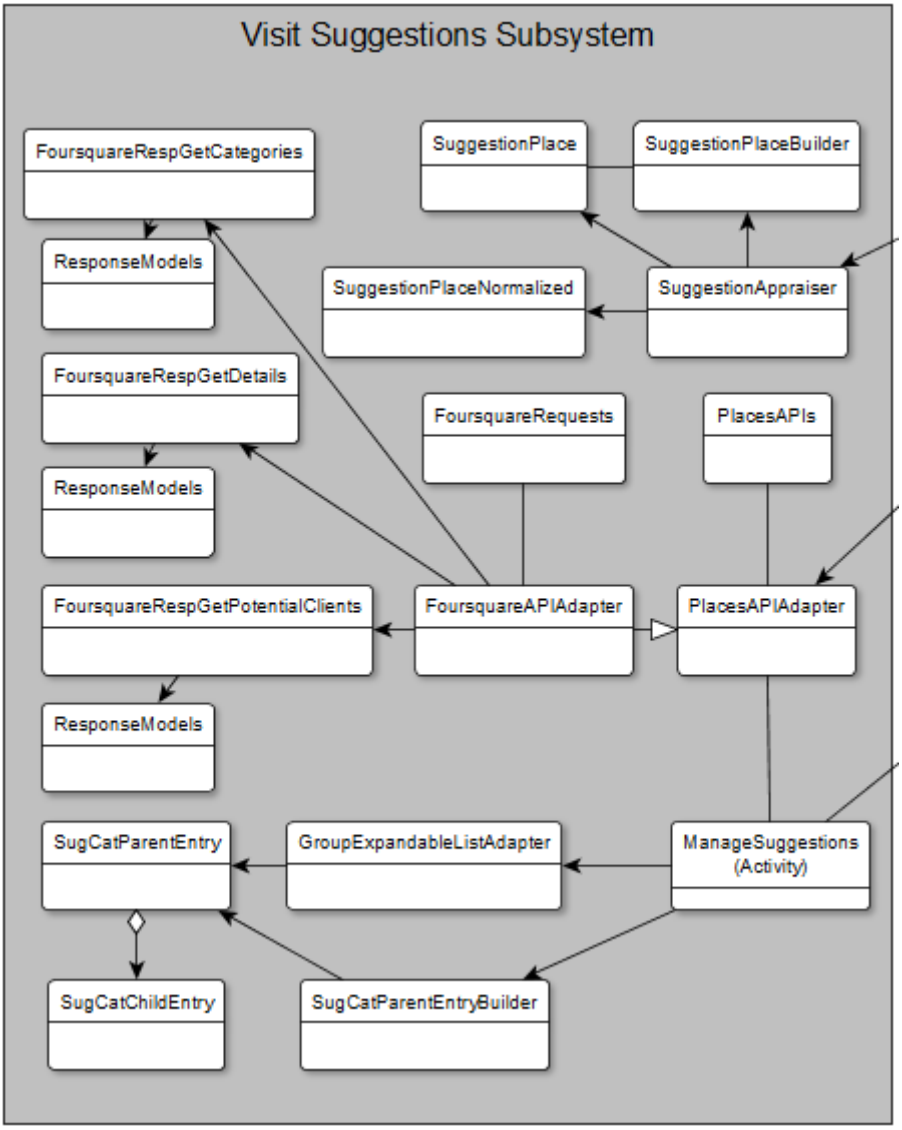


Figura 13: *Visit Suggestions Subsystem*

Para apresentar ao utilizador uma lista de potenciais clientes e sugestões de visita no mapa, o módulo de navegação recorre ao subsistema de sugestões de visita. O objetivo essencial deste subsistema é proporcionar uma lista de estabelecimentos ao utilizador, que correspondam a potenciais clientes, e que este poderá visitar para angariar novos clientes para a sua empresa.

Este subsistema tem como funções definir um conjunto de categorias de tipos de estabelecimentos, disponibilizando uma lista para que o utilizador selecione as categorias que considera mais pertinentes. Tem também a responsabilidade de realizar pedidos a uma API externa de Locais e Estabelecimentos de forma a saber que estabelecimentos correspondentes às categorias selecionadas se encontram perto da

localização do utilizador ou do seu destino, conforme desejado. Após adquirir a lista de estabelecimentos próximos, este subsistema ordena os resultados com base numa medida de relevância.

No final do processo, deverá ser oferecida ao utilizador uma lista de potenciais clientes ordenada por relevância para o seu negócio.

De forma similar aos outros subsistemas, este subsistema integra também um adaptador para a API de Locais e Estabelecimentos a utilizar. A intenção foi a de permitir a utilização de outra API caso necessário.

A classe ManageSuggestionsActivity, também uma Activity de Android, apresenta a lista de categorias a selecionar para as sugestões de visita. O utilizador poderá selecionar as categorias percorrendo a lista ou utilizando uma barra de pesquisa.

Ao pedir para serem apresentadas sugestões de visita no mapa, é efetuado um pedido à API de Locais e Estabelecimentos cujos dados, recebidos em formato JSON, são posteriormente traduzidos em modelos SuggestionPlace. Cada objeto SuggestionPlace refere-se a um estabelecimento de uma potencial sugestão de visita, e contém dados necessários ao cálculo da sua relevância. Alguns destes dados incluem a reputação do estabelecimento, o número de visitas de clientes que costuma ter, a sua distância ao utilizador, entre outros. Os vários indicadores serão devidamente detalhados na próxima subsecção deste relatório, durante a explicação do algoritmo de ordenação de sugestões de visita.

Após receção e processamento dos dados dos vários estabelecimentos, a classe SuggestionAppraiser classifica as várias sugestões de visita por ordem de relevância para o utilizador. O funcionamento do algoritmo de classificação de sugestões de visita também será devidamente detalhado na próxima subsecção deste relatório.

Após construída a lista de estabelecimentos ordenada por relevância, esta é apresentada ao utilizador de forma visual, através de marcadores no mapa. O utilizador, por sua vez, pode iniciar uma nova viagem para o estabelecimento desejado.

8.4. Elementos Relevantes

Nesta secção serão expostos os elementos mais valorizadores ou essenciais para o projeto, por serem considerados fundamentais para a sua conceção. Estes elementos incluem os vários algoritmos desenvolvidos de raiz para o módulo de navegação do Drivian Tasks, assim como um *script* criado para facilitar a preparação de mapas e grafos. Será feita uma breve descrição do objetivo e dos requisitos de cada elemento, seguindo da sua descrição.

8.4.1. Algoritmo de Acompanhamento de Rotas

Para conseguir inferir se o utilizador se encontra a seguir a rota definida, foi criado um algoritmo de acompanhamento de rotas. O objetivo deste algoritmo é perceber em que ponto da rota se encontra o utilizador para, em última instância, apresentar instruções pertinentes, representar visualmente no ecrã do dispositivo a posição do utilizador em relação à rota ou, caso necessário, ajustar o percurso. Como já descrito anteriormente neste relatório, embora se trate de um problema comum com soluções já implementadas, não é do nosso conhecimento a existência de um algoritmo bem estabelecido e disponível ao público centrado nesta problemática. Atendendo ao facto de que o sistema de navegação foi desenvolvido de raiz, sem recurso a nenhum SDK ou biblioteca que implemente funcionalidades de navegação, tornou-se necessário desenvolver um algoritmo com esta finalidade.

Antes de desenvolver o algoritmo em questão, foram definidos alguns requisitos gerais para a sua implementação:

Visto que, para aplicação Drivian Tasks, foi decidido que o dispositivo fará uma leitura do sinal de GPS a cada segundo, o algoritmo também deverá ser executado a cada segundo, ou seja, de cada vez que o dispositivo interpretar as suas coordenadas no espaço geográfico. Isto é essencial para que haja concordância entre a realidade e o que é apresentado ao utilizador de forma audiovisual. Esta particularidade, em acréscimo ao facto de os dispositivos móveis terem bateria limitada, tornam necessário que o algoritmo seja extremamente leve e efetue um número limitado de cálculos de cada vez que é executado. É imperativo que o algoritmo possa ser executado todos os segundos e, ao mesmo tempo, consiga garantir a longevidade da bateria do dispositivo. Como tal, deve ser evitado que o algoritmo percorra exaustivamente, uma vez por segundo, a lista de pontos de um percurso na sua totalidade. Estas listas podem chegar a conter milhares

de pontos, o que, pelo facto de a implementação deste algoritmo ser em Java, não é ideal dada a ineficiência de Java a processar listas de objetos (Sestoft, 2010).

De igual modo, para que seja verdadeiramente útil, o algoritmo deverá ter um elevado nível de precisão. Isto é necessário para que a rota não seja ajustada desnecessariamente. Atendendo às características da tecnologia GPS já descritas neste relatório, em particular o seu nível de precisão volátil, é extremamente importante que o algoritmo consiga contemplar um nível de precisão variável e inconstante.

Por fim, é importante frisar que o problema em questão é mais complexo do que simplesmente encontrar o ponto do percurso que está mais próximo do utilizador e ver se este se encontra dentro de uma distância mínima. Com efeito, existem diversos casos que necessitam de uma lógica mais complexa. Alguns destes casos que exigiram uma implementação mais intrincada do algoritmo incluem:

O facto de o utilizador se poder encontrar numa reta entre dois pontos demaiado distantes e não estar dentro da distância mínima de nenhum para ser considerado como estando a seguir o percurso (figura 14). Se o algoritmo não contemplar este cenário, o sistema reajustará o percurso do utilizador de cada vez que este conduzir numa reta.

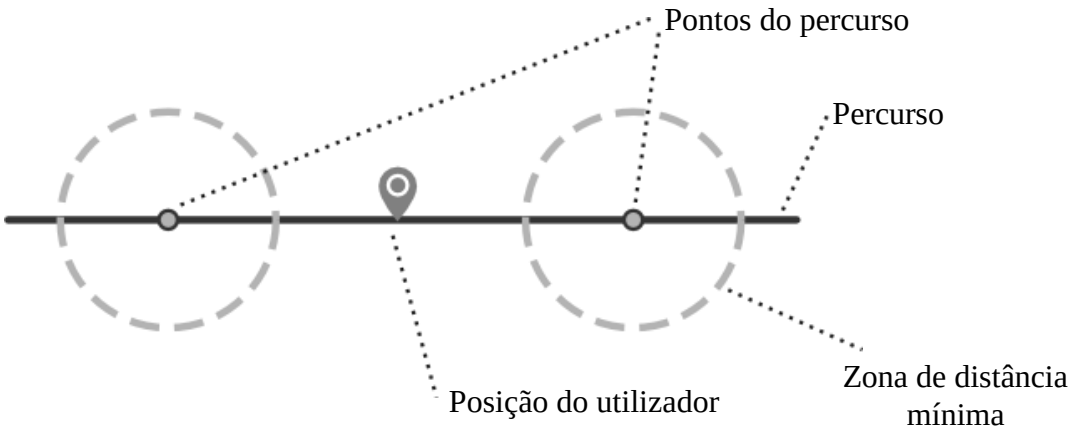


Figura 14: Acompanhamento do Utilizador em Estradas Retas

Do mesmo modo, existem várias situações nas quais o ponto mais próximo não é o ponto desejado, como se sucede nos três cenários apresentados na figura 15.

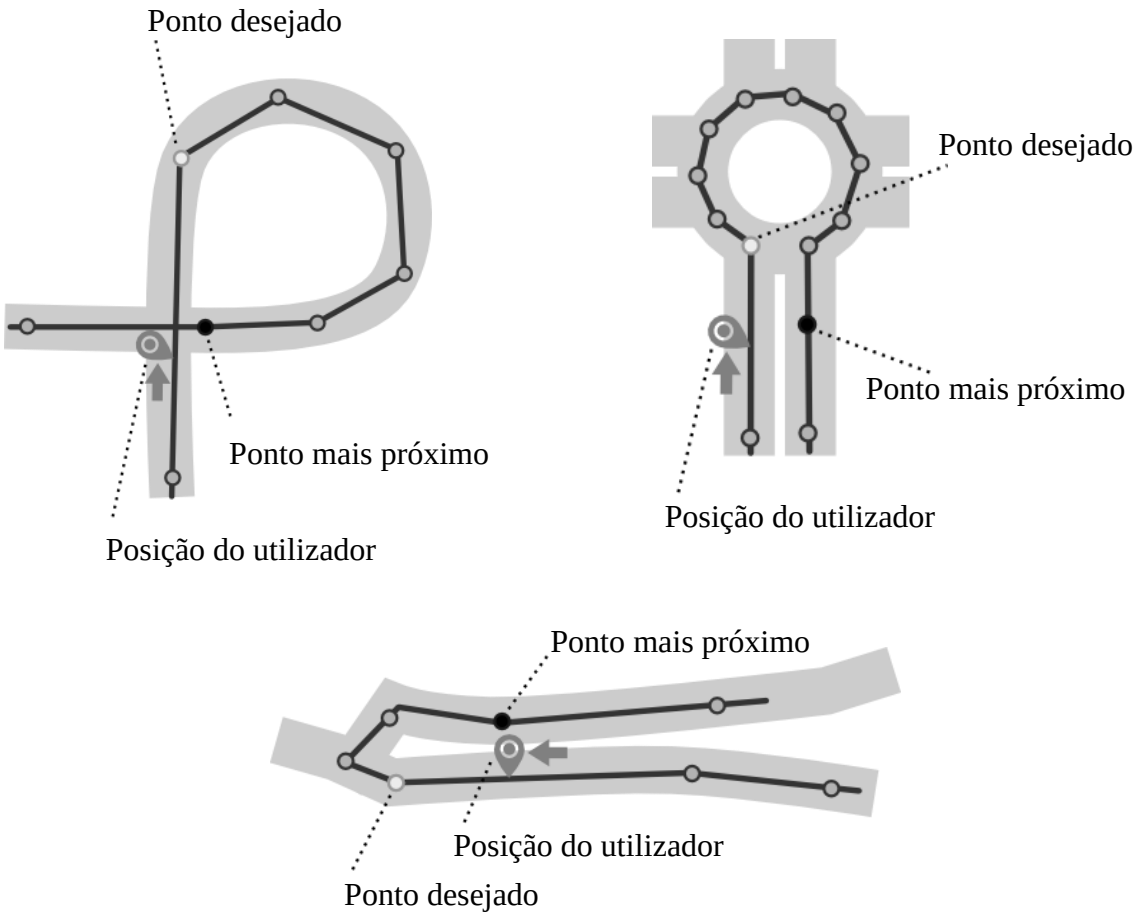


Figura 15: Cenários Problemáticos para o Acompanhamento de Rotas

Para englobar estes cenários, o sistema foi estruturado de forma a iterar pela lista de pontos do percurso, e parar assim que a distância dos pontos ao utilizador começar a aumentar. Como resultado, o ponto desejado é considerado o próximo ponto do percurso.

Outro dos cenários mais comuns, proveniente especificamente da Bing Maps API, revelava uma disposição imprecisa dos pontos de percurso em rotundas, como é possível observar na figura 16.

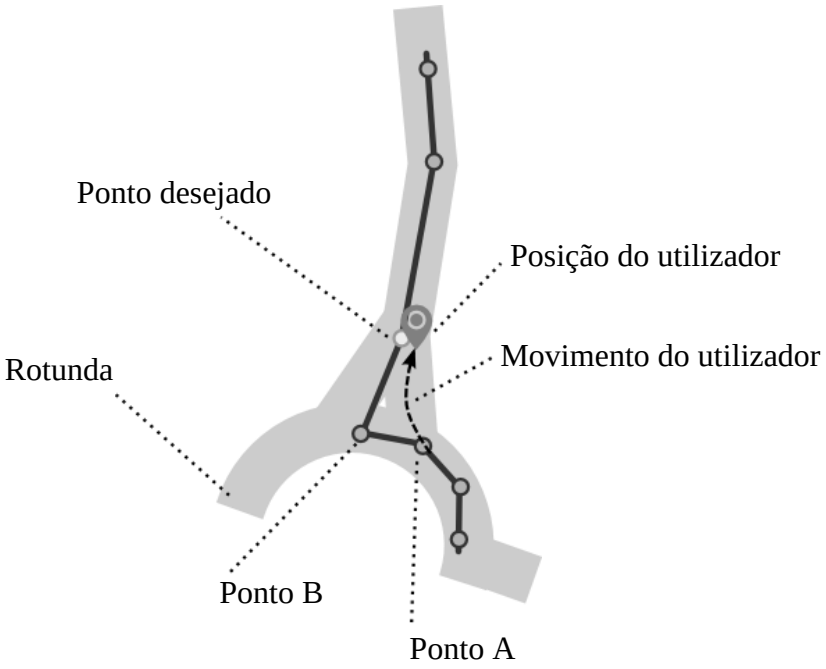


Figura 16: Disposição Imprecisa dos Pontos em Rotundas

No cenário apresentado, o algoritmo assumiria o ponto A como sendo o mais próximo do utilizador ao iterar pela lista de pontos e verificar que a distância do utilizador começa a aumentar, sendo a distância do utilizador ao ponto B superior à distância do utilizador ao ponto A.

A solução encontrada, que revelou ser útil em outros casos similares, refere-se a fazer com que o algoritmo percorra um número limitado de pontos depois de "perder" o utilizador, podendo voltar a localizá-lo perto de um ponto posterior. É importante que este cálculo seja feito apenas depois de “perder” o utilizador, para não entrar em conflito com os cenários apresentados na figura 15. Ou seja, este cálculo só ocorre caso seja garantido que o utilizador não esteja realmente a seguir o percurso, ou caso se encontre numa situação similar à descrita.

O algoritmo desenvolvido contempla todas estas particularidades, o que o torna mais robusto perante outras situações que possam surgir. O seu código Java encontra-se no anexo E.

8.4.2. Algoritmo para Ajuste de Rotas

Com o objetivo de limitar o número de pedidos feitos à API de cálculo de rotas, foi essencial desenvolver um algoritmo que defina de que forma os ajustes de percurso são

efetuados. Com efeito, seria simples determinar que, de cada vez que o utilizador se desviasse do percurso, fosse enviado um novo pedido de percurso à API de cálculo de rotas. No entanto, esta decisão teria como consequência efetuar um grande número de pedidos, o que traz custos que influenciariam o preço final do produto. Do mesmo modo, optar por fazer ajustes de rota exclusivamente *offline*, recalculando a rota localmente, no dispositivo, necessitaria de bastante capacidade de processamento, teria impacto na duração da bateria, e perder-se-iam informações de trânsito e de eventos na via.

Torna-se essencial delinear uma solução que limite o número de pedidos feitos à API de cálculo de rotas, ao mesmo tempo que mantém o máximo de informação possível proveniente da rota original.

Para esse fim, foi definida uma solução que faz ajustes parciais da rota. Quando o utilizador estiver demasiado longe da rota original, o percurso será cortado a uma certa distância do utilizador, e será feito um cálculo local, *offline*, da rota desde a posição do utilizador até ao ponto de corte, como representado na figura 17. Deste modo, limitam-se o número de pedidos enviados à API de cálculo de rotas, ao mesmo tempo que se mantém a grande maioria da informação de trânsito e eventos na estrada.

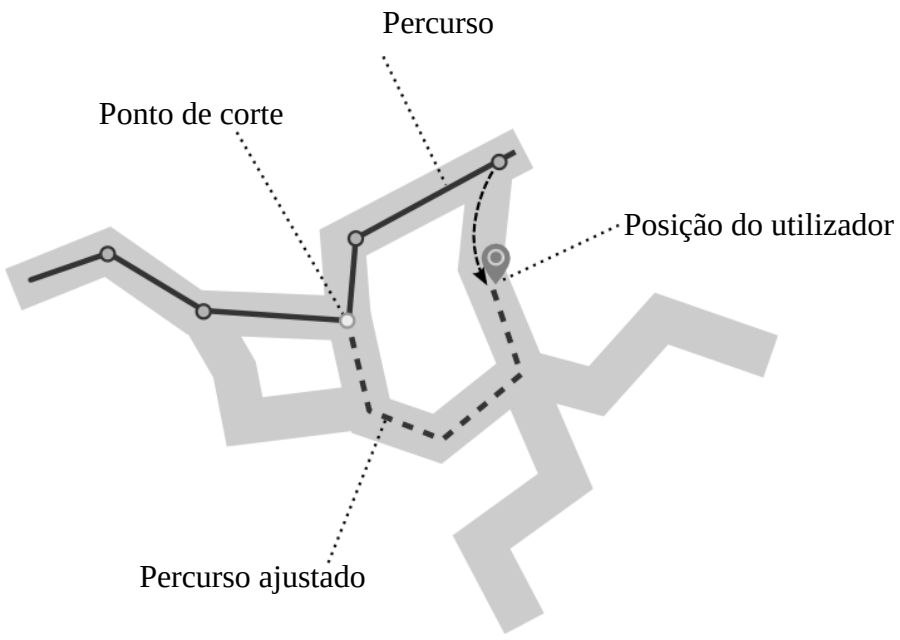


Figura 17: Ajuste Inicial de Percurso

Após feito este primeiro ajuste de rota, é determinada uma "zona verde" à volta do utilizador, representada na figura 18.

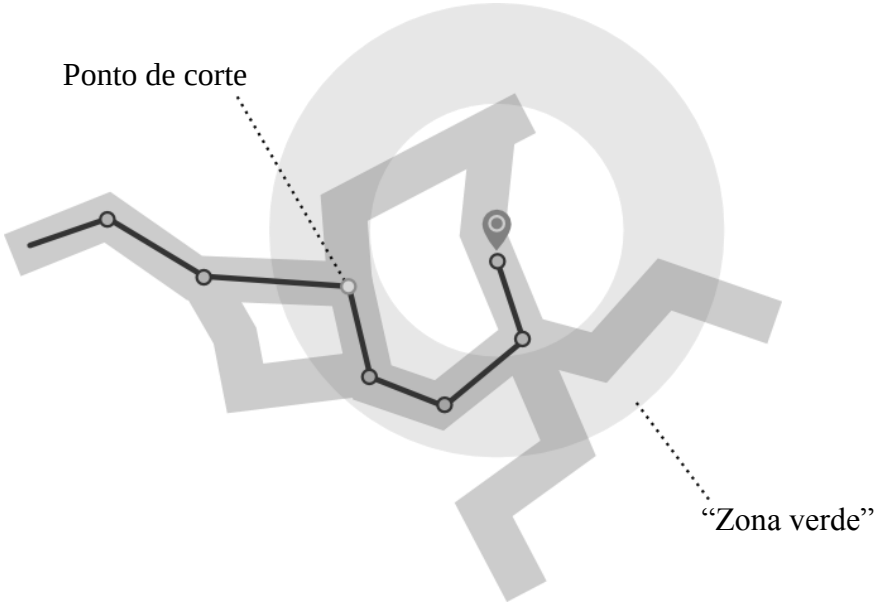


Figura 18: Raio de Ajuste

Se, após o primeiro ajuste, o utilizador se voltar a desviar da rota, ocorrerá um dos seguintes cenários:

Se o ponto de corte inicial estiver dentro da "zona verde", será recalculado um ajuste de rota local, *offline*, desde a posição do utilizador até esse mesmo ponto. Deste modo, é preservado o máximo de informação possível da rota original, e o reajuste de percurso é sempre feito em relação ao ponto da rota original, evitando que o utilizador vá construindo uma rota "trecho a trecho" à medida que se desvia do caminho, como apresentado na figura 19.

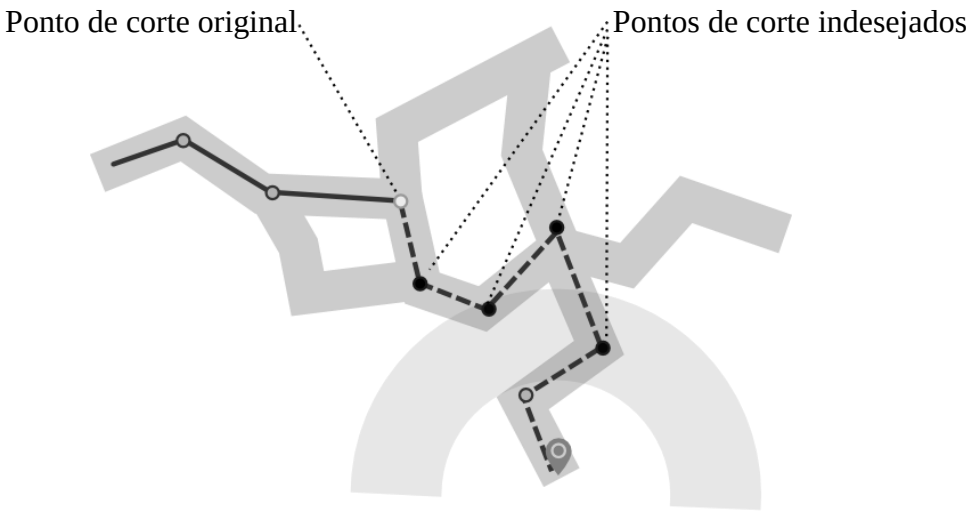


Figura 19: Construção de Rota Trecho a Trecho

Se o ponto de corte estiver demasiado próximo ou demasiado distante, será procurado um novo ponto de corte a uma certa distância do utilizador, como representado na figura 20.

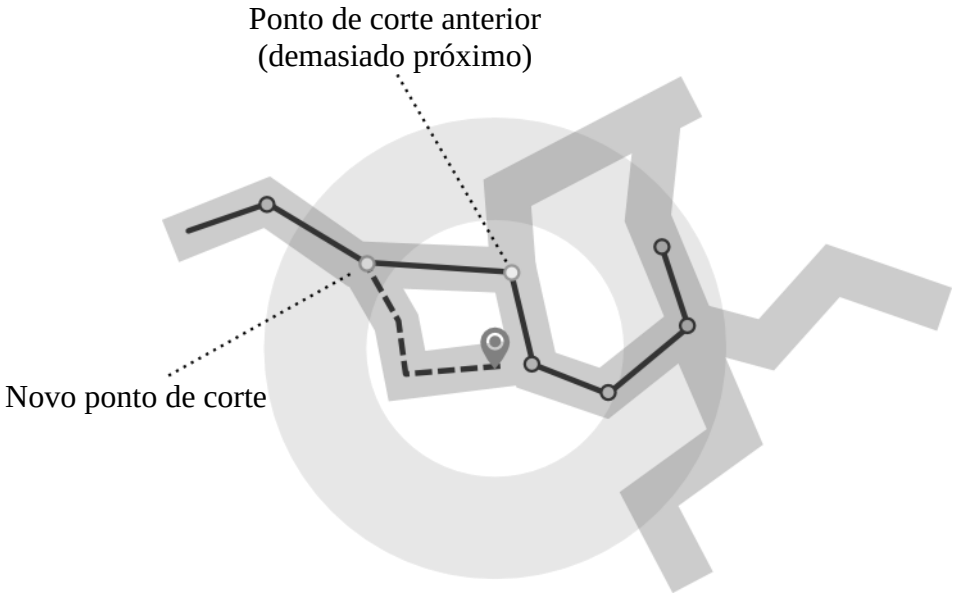


Figura 20: Ponto de Corte Demasiado Próximo

É importante que seja determinado um novo ponto de corte se o utilizador estiver demasiado próximo do ponto de corte original, caso contrário é muito provável que um ajuste resulte num cálculo de rota ineficiente (figura 21).

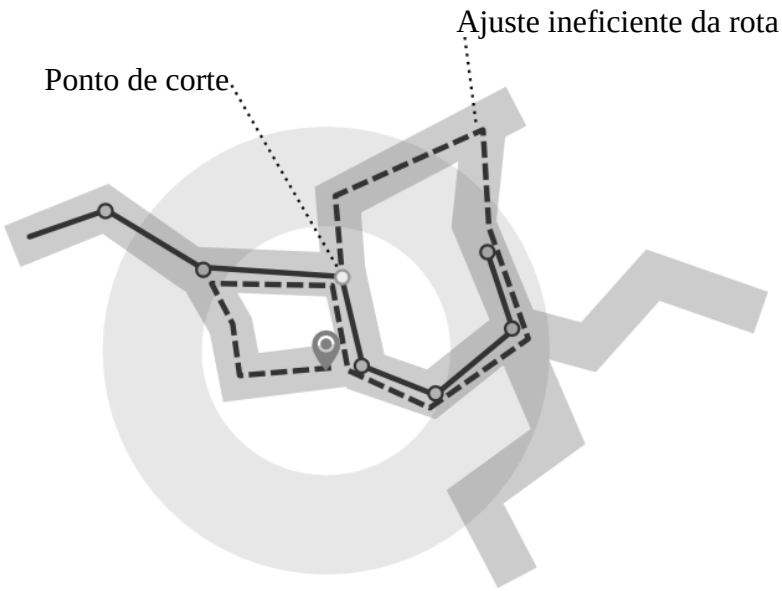


Figura 21: Ajuste Ineficiente da Rota

Este novo ponto de corte tem, necessariamente, de ser sempre posterior ao ponto de corte inicial, e pertencer sempre à rota original.

Após definido o novo ponto de corte, será então recalculado um ajuste de rota local, *offline*, até ele.

Em qualquer dos cenários, se não encontrar um possível ponto de corte no percurso original, muito provavelmente porque a rota original já demasiado distante do utilizador, então será efetuado um novo pedido de percurso à API de cálculo de rotas desde a posição atual do utilizador até ao destino.

A solução apresentada cumpre os objetivos de minimização de número de pedidos e mantém o máximo de informação original possível, ao mesmo tempo que apresenta um ajuste de percurso eficaz. O seu código Java encontra-se exposto no anexo F.

8.4.3. Algoritmo de Construção de Instruções

Embora o sistema desenvolvido para a apresentação de instruções dê preferência à utilização de instruções provenientes diretamente da API de cálculo de rotas, estas nem sempre se encontram presentes. É muito comum existirem particularidades no percurso sem uma instrução associada. Para garantir que o utilizador recebe uma nova instrução de cada vez que tem de efetuar uma manobra, torna-se essencial desenvolver um

algoritmo que, na ausência de instruções, consiga interpretar a tipologia do percurso e gerar instruções pertinentes.

O algoritmo desenvolvido encontra-se no anexo G.

8.4.4. Algoritmo de Ordenação de Sugestões de Visita

Para ordenar por relevância os estabelecimentos de potenciais clientes e sugerir visitas que possam trazer novos clientes à empresa ao utilizador, foi desenvolvido um algoritmo de ordenação de sugestões de visita.

O objetivo deste algoritmo é atribuir uma medida de relevância a cada estabelecimento de um potencial cliente num determinado raio. Esta medida deverá ser descritiva do seu valor para o negócio do utilizador, resultando numa lista de estabelecimentos de potenciais clientes numa área. Esta lista, por sua vez, deverá ser ordenada pelo valor de relevância de cada estabelecimento.

Este algoritmo pesa diversas características de cada estabelecimento como a sua distância ao utilizador, o número de categorias que partilha com as categorias selecionadas pelo utilizador, a sua reputação, o número de *likes* na sua página de Facebook, e o número médio de visitas, e *checkins* e comentários feitos por clientes. No entanto, a existência destas características não é garantida em cada pedido, pelo que o algoritmo deverá contar com a potencial omissão de valores.

Alguns dos dados recolhidos sobre os estabelecimentos foram cruzados, resultando em novas métricas úteis para o algoritmo. Nesse sentido, considera-se que um estabelecimento com poucas visitas e muitos *likes*, por exemplo, é uma proposta de maior valor que um estabelecimento com muitas visitas e igual número de *likes*. Entende-se que o número de *likes* é diluído pelo número de clientes. De igual forma, foi desenvolvida uma nova métrica através do cruzamento da distância do estabelecimento relativamente ao utilizador e o número de categorias que partilha com as categorias selecionadas. Um estabelecimento que corresponde a poucas das categorias selecionadas mas que esteja mais próximo tem menor valor que um estabelecimento mais distante que partilhe um maior número de categorias. Espera-se, deste modo, conseguir inferir uma medida de relevância mais pertinente do que a resultante da simples análise dos dados originais.

De igual modo, considera-se que não é pertinente apresentar estabelecimentos que não estejam abertos no momento em que o utilizador faz a pesquisa, ou estabelecimentos que o utilizador já tenha visitado. Estes casos são simplesmente ignorados.

A construção do algoritmo resultou na implementação presente no anexo H.

8.4.5. Adaptadores

Com o objetivo de garantir a longevidade e adaptabilidade de todo o módulo de navegação, toda a comunicação com as diversas APIs foi desenvolvida com recurso a adaptadores. Deste modo, é possível utilizar qualquer API de cálculo de rotas ou de locais e estabelecimentos que seja considerada pertinente. Esta decisão permite, eventualmente, utilizar as APIs e bases de dados da própria empresa. Em última instância, espera-se desenvolver uma aplicação verdadeiramente independente.

8.4.6. *Script* de Preparação de Mapas e Grafos

Para facilitar a disponibilização de novos mapas e respetivos grafos ou a atualização de mapas na aplicação, e atendendo a que os ficheiros de mapas e grafos têm que ser preparados de uma forma algo complexa e morosa antes de serem disponibilizados ao cliente, foi desenvolvido um *Shell Script* para Linux. Este *script* permite especificar o nome de um país e percorre automaticamente todos os passos necessários à preparação do seu mapa, gerando um único ficheiro pronto a colocar no servidor. Após colocado o ficheiro no servidor, o respetivo mapa estará pronto a ser transferido e utilizado pela aplicação móvel. Espera-se poupar o tempo que seria utilizado na elaboração dos vários mapas, permitindo assim a sua preparação de forma absolutamente automática.

Este *script* foi escrito e testado numa máquina virtual com o sistema operativo Xubuntu v.16.04 (Xubuntu, s.d.), com recurso à aplicação Virtual Box 5.2 (Virtual Box, s.d.).

Como complemento a este *script*, foi também escrito um pequeno manual de instruções para o caso de ocorrer algum problema e a preparação dos mapas necessitar de uma abordagem manual.

O funcionamento do *script* de preparação de mapas está presente no anexo I.

8.5. Dificuldades Encontradas

Durante o desenvolvimento do módulo de navegação para o Drivian Tasks, surgiram diversas dificuldades que não estavam, necessariamente, relacionadas com o desenvolvimento do código em si. Embora a grande maioria destas dificuldades tenha sido superada, algumas necessitaram de uma abordagem alternativa ou foram, simplesmente, abandonadas por não serem consideradas essenciais para o projeto.

As dificuldades mais proeminentes durante o projeto foram:

8.5.1. Preparação de Ficheiros de Mapas

Pelo facto de a apresentação dos mapas na aplicação estar relegada a uma biblioteca Java, a biblioteca de mapas V™, ou OpenScienceMaps, em conjunto com a sua integração com os respetivos grafos para cálculo de rotas, através da biblioteca Graphhopper, tornou-se necessário preparar os ficheiros de mapas de uma forma específica. Com efeito, encontrar um bom conjunto de procedimentos para preparar os vários ficheiros tornou-se uma tarefa complexa, que requereu bastante experimentação.

Contudo, e após definidos os procedimentos, esta dificuldade originou a ideia de documentar e automatizar todo o processo. Por esse motivo foi escrita a documentação para a preparação de mapas. Do mesmo modo, deu também origem ao desenvolvimento do *script* de preparação de mapas já mencionado. No final, esta dificuldade revelou ser uma mais-valia para o projeto.

8.5.2. Inicialização e Resumo da Aplicação

Um dos problemas mais complexos de resolver durante o desenvolvimento da aplicação foi a sua inicialização. Com efeito, a inicialização da aplicação implica um conjunto de tarefas assíncronas, cujo momento de finalização é imprevisível, e que necessitam de estar bem coordenadas entre si. Ademais, o desenvolvimento da aplicação a partir de um Fragmento de Android implicou a alocação de outros Fragmentos secundários, instanciados a partir do primeiro, o que gerou conflitos com a coordenação do seu ciclo de vida e com a coordenação de alguns elementos de *layout*.

Em acréscimo a este problema inicial, somavam-se algumas particularidades relacionadas com a colocação da aplicação em segundo plano. De cada vez que o utilizador colocava a aplicação em pausa, poderia apagar ficheiros de mapas em uso,

retirar permissões, ou desligar o serviço de localização por GPS. Tornou-se imperativo verificar a condição de cada um destes elementos, sem recomeçar todo o processo de inicialização. Caso contrário, os vários elementos presentes no mapa e guardados na memória do dispositivo, como o percurso atual ou os marcadores de potenciais clientes, seriam perdidos, pois o mapa em si seria reiniciado.

A solução implementada executa verificações ao *layout*, às permissões, ao serviço de GPS, e aos ficheiros de mapas, que, por sua vez, se coordenam com uma sequência de metas, ou booleanos. Estas metas garantem que apenas sejam reiniciados os elementos cujas condições não se verifiquem intactas, assim como os outros elementos que delas dependam.

8.5.3. Algoritmo de Acompanhamento de Rotas

Uma das dificuldades mais proeminentes deste estágio foi a implementação de um algoritmo de acompanhamento de rotas.

Como descrito anteriormente, o algoritmo em questão deveria ser leve do ponto de vista computacional e ter um elevado nível de precisão, mesmo recebendo dados de um sistema que, por natureza, poderia ter pouca precisão.

Após várias implementações, foi desenvolvido um algoritmo que tem demonstrado ótimos resultados em viagens no terreno, cumpre todos os requisitos e passa com sucesso os testes, pelo que é considerado como uma implementação viável para o projeto.

8.5.4. Texto para Voz em Português Europeu

Uma das dificuldades para a qual não foi encontrada uma solução viável foi a inclusão de um motor de texto para voz em Português Europeu. Atualmente a biblioteca nativa de Android não inclui esta opção, pelo que as instruções de navegação em áudio são apresentadas em Português do Brasil.

No entanto, foram exploradas várias alternativas para esta dificuldade. A melhor solução encontrada implica a instalação de um motor de texto para voz externo, e a posterior instalação manual de vários ficheiros de voz em Português Europeu. Uma solução que, embora funcione, é extremamente inconveniente e complexa para o utilizador final. Por esse motivo foi considerada inviável.

Estando exaustas as alternativas, e não sendo considerada uma funcionalidade de valor pela empresa Sentilant, esta funcionalidade foi abandonada. O argumento principal defende que os utilizadores de Android estão já habituados ao motor de texto para voz em Português do Brasil, pelo que a implementação de um motor para Português Europeu não traria grande acréscimo de valor à aplicação.

8.5.5. Seleção Automática das Categorias de Sugestões de Visita

Para facilitar a escolha das categorias para as sugestões de visita a potenciais clientes, foi explorada a possibilidade de desenvolver um algoritmo de escolha automática. O objetivo deste algoritmo seria selecionar as categorias que considerasse mais relevantes para o negócio do utilizador, sem que este precisasse de percorrer a lista de categorias de forma manual. O sistema de apresentação de potenciais clientes utilizaria então estas categorias para exibir sugestões de visita no mapa.

Os pontos de partida deste algoritmo seriam os códigos CAE (Códigos de Classificação de Atividades de Empresas), referentes às áreas de atividade económica de cada empresa. Nesse sentido, foi desenvolvido um sistema que acedia ao *website* do SICAE (Sistema de Informação da Classificação Portuguesa de Atividades Económicas) (SICAE, s.d.), o site de acesso público para consulta dos códigos CAE e, a partir do nome da empresa do utilizador registado no Drivian Tasks, simulava o preenchimento do formulário na página *web* e recolhia os códigos CAE resultantes.

Após todo o processo, era recuperada uma lista dos códigos de atividade económica da empresa do utilizador. Seria necessário triangular esses códigos com as várias categorias de sugestões de visita, e assim escolher as categorias consideradas mais relevantes para cada atividade económica.

Infelizmente, após o desenvolvimento desta primeira parte do sistema, não foi possível construir um algoritmo que fizesse a triangulação entre códigos CAE e as categorias de forma automatizada. Mesmo depois da discussão do assunto com os colegas da empresa, a única abordagem verdadeiramente exequível resumia-se a implementar manualmente uma tabela que cruzasse todos os possíveis códigos CAE com as mais de duzentas categorias de sugestões de visita. Foram exploradas diversas outras abordagens, das quais a mais interessante procurava decompor os nomes associados a cada CAE, como por exemplo "62010 - Atividades de Programação Informática", nas

suas sílabas essenciais. Este processo foi implementado com recurso a bibliotecas de *Stemming* e de *Lemmanization*. Seriam depois selecionadas as categorias de sugestões de visita que partilhassem alguma destas sílabas. Esta abordagem não obteve quaisquer resultados práticos.

Dada a pouca importância desta funcionalidade para o projeto, por não ter sido considerada como necessária pelos responsáveis da empresa, e pelo facto de a sua solução ser considerada pouco relevante para os objetivos de inovação deste estágio, a implementação deste algoritmo foi abandonada. No entanto, é importante referir que o sistema apresenta as sugestões de visita com base nas categorias selecionadas. Simplesmente, o utilizador terá que selecionar estas categorias manualmente.

8.5.6. Requisitos Não Implementados

Chegando à data de término do projeto, houve algumas funcionalidades que não foram implementadas na aplicação móvel do Drivian Tasks. Dada a falta de tempo dos elementos responsáveis pelo servidor da empresa, não foi possível preparar a API do Drivian para algumas particularidades previstas exclusivamente para a aplicação móvel. Como consequência, as funcionalidades de gestão de permissões para os operacionais no terreno, escolher opções de percurso, incluir pontos de abastecimento no percurso atual, reportar imprevistos e procurar parques de estacionamento, não foram implementadas.

Tratando-se de funcionalidades menos cruciais, correspondem a elementos *nice to have*, e, como tal, não tiveram qualquer impacto na qualidade do produto final. No entanto, prevendo a sua implementação futura, o sistema ficou preparado para a fácil implementação destas funcionalidades, ficando inclusivamente comentado algum código referente aos mesmos e os seus elementos de *layout* já preparados. Bastará apenas desenvolver as funcionalidades do lado do servidor.

8.6. Resumo

No atual capítulo foi explorada toda a fase de implementação do projeto. Foi feita uma descrição de todas as decisões de desenvolvimento tomadas antes da sua implementação, apresentada a lista dos vários elementos utilizados, e elaborada uma descrição detalhada de todo o sistema. Esta descrição foi feita com recurso ao modelo

C4 e à divisão da implementação em subsistemas para corroborar a explicação. Foram também destacados os elementos considerados mais relevantes para o módulo de navegação, assim como as várias dificuldades encontradas durante a implementação de todo o projeto.

Espera-se que este capítulo apresente uma imagem concreta de todos os elementos incluídos no módulo de navegação do Drivian Tasks.

9. Validação da Solução Implementada

Sendo uma aplicação de natureza geográfica, o módulo de navegação do Drivian Tasks deparar-se-á com um sem número de fatores imprevisíveis que poderão ativar falhas ocultas no sistema e, consequentemente, provocar comportamentos errados. Como tal, torna-se essencial construir um conjunto de testes que permitam adquirir um bom nível de confiança sobre as capacidades do sistema. Sendo impraticável testar todos os casos possíveis, pois dada a natureza geográfica da aplicação existem imensos casos difíceis de testar em laboratório, foram delineados testes que asseguram que o sistema consegue resolver de forma satisfatória, pelo menos, a grande maioria das situações de navegação.

9.1. Seleção de Elementos a Testar

Visto que não é possível, nem considerado vantajoso, testar o sistema completo em tempo útil, é essencial fazer uma boa seleção dos vários elementos a testar. Testes de *software* deverão sempre ser considerados numa perspetiva de análise do seu custo em relação ao seu benefício. No caso dos testes de *software*, o custo refere-se ao tempo de implementação. De igual forma, para que os testes sejam considerados fiáveis, deverão ser pequenos e com um foco reduzido. Os elementos a testar deverão ser, de preferência, os componentes cruciais do sistema, identificados a partir dos seus requisitos funcionais e não funcionais.

Os elementos testados foram:

- Algoritmo de acompanhamento de rotas;
- Algoritmo de ajuste de rotas;
- Sistema de construção automática de instruções de navegação;
- Sistema de cálculo de rotas;
- Construção e ajuste de rotas;
- Algoritmo de classificação de sugestões de visita;
- Interface de utilizador;
- Inicialização do sistema.

Note-se que os testes feitos ao algoritmo de ajuste de rotas referem-se aos testes feitos à forma como o sistema decide ajustar uma rota quando o utilizador se desvia do seu percurso. Ou seja, testa-se se o sistema faz um ajuste total ou apenas parcial da rota, calculado localmente, e se este funcionamento vai ao encontro do esperado. Isto distingue-se dos testes feitos ao sistema de construção e ajuste de rotas, que se referem ao sistema que, ao receber os dados de uma rota calculada ou ajustada, os transforma em modelos a serem consumidos pelo sistema.

9.2. Ferramentas Escolhidas

Para implementar os testes feitos sobre vários os elementos do sistema, foi necessário recorrer a um conjunto de ferramentas:

- JUnit4: Biblioteca *standard* para desenvolver testes de software para Java;
- Android Espresso: Biblioteca para implementar testes de interface de utilizador em Android;
- Roboelectric: Biblioteca para implementar testes que recorram a recursos e *assets* específicos de uma aplicação Android.

9.3. Estratégia de Testes

Para cada um dos elementos a testar, foi selecionado o critério de teste considerado mais apropriado que permita obter um elevado nível de fiabilidade no código.

Durante a formulação dos vários testes, foram utilizados critérios do domínio estrutural, lógico e de *input*.

Os critérios utilizados para testar cada um dos elementos encontram-se expostos na tabela 9.

Elemento a Testar	Critério de Teste
Algoritmo de acompanhamento de rotas	Base Choice Coverage (domínio de *input*)
Algoritmo de ajuste de rotas	Combinatorial Coverage (domínio de lógica)
Sistema de construção automática de instruções de navegação	Edge Coverage (domínio estrutural)
Sistema de cálculo de rotas	All Combinations Coverage (domínio de *input*)
Construção e ajuste de rotas	All Combinations Coverage (domínio de *input*)
Algoritmo de classificação de sugestões de visita	Base Choice Coverage (domínio de *input*)
Inicialização do sistema	Sem critério
Interface de utilizador	Sem critério

Tabela 9: Critérios de Cobertura

Relativamente à inicialização do sistema, foi feito um teste linear que simplesmente testa a sequência de inicialização da aplicação. O objetivo deste teste é garantir que futuras alterações feitas à aplicação não põem em causa a sequência inicial esperada pelos seus vários subsistemas.

No caso específico dos testes feitos à interface de utilizador, não foi possível selecionar um critério que permitisse corresponder a todas as necessidades deste sistema. Com efeito, o comportamento de um utilizador é algo imprevisível. O utilizador poderá selecionar quaisquer opções na aplicação em qualquer ordem. Sendo muitos dos elementos de interface dependentes de ações prévias do utilizador, o número de casos de teste possíveis é quase ilimitado.

Por esse motivo, foram delineadas quatro sequências de teste que visam testar todos os elementos de *layout*, desde a inicialização da aplicação, passando pelos vários menus, cálculos e ajustes de rotas e respetiva apresentação no mapa, apresentação de opções de tarefas, presença de marcadores no mapa, entre outros. Para desenhar o conteúdo destas sequências de teste, foram tomados em conta os requisitos não-funcionais, nomeadamente o requisito de preparar o mapa e os grafos no dispositivo dentro de um período de tempo aceitável, assim como as restrições de implementação e *design* que determinam que cada opção deve estar acessível em menos de três toques.

No total foi implementado um conjunto de noventa e cinco testes.

Em acréscimo às várias sequências de teste feitas sobre a interface do utilizador, foi igualmente testado o seu funcionamento em dispositivos com ecrãs de diferentes dimensões. Estes testes, feitos manualmente, resumiram-se à verificação exaustiva de todos os elementos de *layout* nos vários ecrãs, nas suas disposições horizontal e vertical, e resultaram na reformulação de alguns dos seus elementos.

Em adição aos testes programáticos feitos ao algoritmo de acompanhamento de rotas, este foi também validado com várias viagens feitas pelo próprio estagiário, sozinho e acompanhado por colegas da empresa, assim como pelo próprio responsável da Sentilant, o Professor Bruno Cabral.

Durante a formulação dos vários testes, foram, por várias vezes, identificadas algumas inconsistências no código, assim como alguns pontos que poderiam dar origem a erros. De igual modo, o desenvolvimento de testes permitiu identificar pontos de otimização. A simples formulação e implementação de testes ajudou a melhorar substancialmente a qualidade da aplicação em si, especialmente no que se refere ao algoritmo de acompanhamento de rotas e ao algoritmo de reajuste de rotas, dois elementos fundamentais do sistema.

9.4. Análise de Cobertura

Embora o nível de cobertura dos testes não seja descritivo da sua qualidade nem da sua relevância, não deixa de ser importante ter uma noção do quão completos estão os testes feitos ao sistema.

Note-se que, apesar de cobertura total não ser necessariamente um bom objetivo de teste, para o caso específico do sistema de construção automática de instruções, foi considerado pertinente apontar para cem porcento de cobertura. A razão principal prende-se com assegurar que as instruções apresentadas ao utilizador são sempre pertinentes, e nunca o induzem em erro. Sendo um sistema bastante linear, alcançar perto de cem porcento de cobertura permite ter um bom nível de confiança sobre a sua implementação.

Para todos os restantes casos, o nível de cobertura não teve qualquer impacto no desenvolvimento dos testes, sendo este simplesmente consequência da utilização dos vários critérios.

O nível de cobertura total atingido para todo o módulo de navegação foi de 58%.

A cobertura atingida para cada um dos vários elementos testados encontra-se exposta na tabela 10.

Elemento a Testar	Cobertura Inclusiva
Algoritmo de acompanhamento de rotas	89%
Algoritmo de ajuste de rotas	85%
Sistema de construção automática de instruções de navegação	99%
Sistema de cálculo de rotas	75%
Construção e ajuste de rotas	89%
Algoritmo de classificação de sugestões de visita	95%
Inicialização do sistema	66%
Interface de utilizador	-

Tabela 10: Relatório de Cobertura

Apesar de os testes delineados para a inicialização do sistema apenas alcançarem um nível de 66% de cobertura, o código não testado refere-se aos procedimentos que devem ocorrer caso se suceda algum problema durante a inicialização do sistema. Estes problemas incluem a ausência de ficheiros de mapas no dispositivo móvel, a ausência de sinal GPS, o facto de o utilizador não dar as permissões necessárias ao sistema para o seu bom funcionamento, entre outros. Cada uma destas possibilidades, embora não testadas através de código, foram testadas manualmente para assegurar que o sistema respondia da forma pretendida.

9.5. Resumo

Durante a fase de desenvolvimento de teste para o módulo de navegação do Drivian Tasks, foi implementado um total de noventa e cinco casos de teste.

Foi feita uma seleção inicial dos vários elementos a testar, efetuada a partir dos requisitos funcionais e não funcionais previstos para o sistema. Foram definidos os vários critérios de teste para cada componente e reunidas as ferramentas necessárias para a sua implementação. Por fim, foram implementados os vários testes resultando num total de 58% de cobertura. A simples implementação destes testes permitiu melhorar a funcionalidade de alguns elementos da aplicação.

Espera-se que os testes implementados permitam aumentar o nível de confiança na fiabilidade e funcionalidade do código, assim como testar o impacto de possíveis alterações feitas futuramente à aplicação.

10. Conclusões e Trabalho Futuro

O presente documento descreve o trabalho realizado durante dois semestres. Embora a proposta inicial tenha sido ligeiramente alterada, o facto de abandonar a implementação do sistema em iOS a troco do acréscimo da construção de um sistema inteiramente proprietário e desenvolvido de raiz apresentou vários desafios interessantes.

Este capítulo pretende concluir a descrição do trabalho desenvolvido no estágio. Como tal, descreverá brevemente a apresentação final feita na empresa Sentilant, apresentará uma avaliação pessoal sobre a experiência na entidade acolhedora, assim como um pequeno comentário sobre melhorias que, eventualmente, poderão ser feitas ao sistema, caso desejado.

10.1. Apresentação Final

No final do período de estágio, foi feita uma apresentação a toda a equipa da empresa durante a qual foi sumariado todo o processo de desenvolvimento do módulo de navegação do Drivian Tasks (Anexo I). Foi exposto todo o processo apresentado neste relatório, incluindo a exploração de componentes e o desenvolvimento de testes. No final, foi demonstrada a aplicação e todas as suas funcionalidades em execução. Os comentários feitos à mesma foram muito positivos, e alguns colegas demonstraram até ficar algo impressionados com a responsividade e fluidez da aplicação, assim como a forma como foram implementadas algumas das funcionalidades.

No anexo K encontram-se algumas capturas de ecrã da aplicação final, apresentada à empresa.

Atendendo ao feedback dado pelos elementos da empresa, assim como os comentários feitos pelos seus responsáveis, considero que o projeto correspondeu às expectativas e às necessidades da empresa.

10.2. Avaliação Pessoal

A avaliação que faço do meu tempo de estágio na empresa Sentilant é extremamente positiva. Tive a oportunidade de desenvolver um projeto que, na sua fase inicial de conceção, não sabia como abordar. Sob orientação do Professor Bruno Cabral, na

empresa Sentilant, em conjunto com a orientação dada pelo Professor João Durães, do ISEC, consegui superar com sucesso os vários desafios colocados por um projeto desta natureza. É com imensa satisfação que reconheço que desenvolvi um projeto robusto, completo, e, acima de tudo, relevante para os seus vários utilizadores. Aprendi imenso enquanto profissional e tenho noção que esta experiência me tornou mais apto a resolver problemas sem uma resposta linear, cuja solução ainda necessita de ser investigada e desenvolvida. As aprendizagens que considero mais relevantes durante a minha experiência na Sentilant foram:

- O desenvolvimento de aplicações de natureza geográfica. Embora já tenha adquirido alguma experiência no desenvolvimento de aplicações ubíquas na Unidade Curricular de Computação Ubíqua, o facto de desenvolver uma aplicação que não é apenas para fins académicos traz responsabilidades associadas que implicam a estruturação de uma aplicação robusta, com um nível de qualidade que nunca antes fora uma preocupação para mim;
- A consolidação dos processos de pesquisa e estruturação do Estado da Arte, com um objetivo concreto em mente;
- A consolidação de competências de desenvolvimento de testes de *software*;
- A escrita de testes de *layout* e de interface de utilizador, que nunca antes tive a oportunidade de desenvolver;
- A consolidação de conhecimentos de APIs RESTful e de comunicação remota;
- A consolidação de conhecimentos de arquitetura de *software*. Como já referido, o facto de desenvolver uma aplicação com uma verdadeira finalidade prática implica que a arquitetura do programa esteja muito bem estruturada;
- A aplicação de padrões de código, como adaptadores e *builders*, por exemplo, que tive a oportunidade de utilizar durante o desenvolvimento do módulo de navegação;
- A programação de *Shell Scripts* para Linux, que nunca antes tive a oportunidade de escrever;
- O desenvolvimento de *web crawlers*. Embora não tenha sido utilizado na versão final da aplicação, o facto de construir um script que procura informação numa página *web*, neste caso a procura dos códigos CAE de uma empresa, constituiu uma aprendizagem muito interessante para mim;

- A aprendizagem de novos elementos Java, especialmente os direcionados para Android. Embora já tenha bastante experiência com esta linguagem de programação, é sempre possível assimilar novos conceitos e aprender a utilizar ferramentas de novas maneiras com novos objetivos em mente.

10.3. Trabalho Futuro

Desde a conceção inicial do projeto descrito neste documento, houve a preocupação de delinear e desenvolver uma aplicação extremamente completa e igualmente pertinente para um conjunto variado de utilizadores. Apesar de tudo, é sempre possível melhorar um projeto e, não fugindo este à regra, as possíveis melhorias que, na minha opinião, seriam pertinentes incluem:

- A inclusão de um subsistema de apoio à navegação baseado em sensores, tais como giroscópio ou acelerómetro. Este subsistema teria o propósito de assegurar a qualidade da navegação durante os raros momentos em que a receção do sinal GPS estivesse instável.
- O desenvolvimento de mais testes de *layout*. Com efeito, o número de elementos de *layout* neste projeto, assim como o número de possibilidades para a sua disposição, é imenso. Embora os testes considerados essenciais tenham sido implementados por completo, aumentando assim confiança no seu código, seria interessante explorar um leque maior de testes, que contemplasse a grande maioria das possibilidades.
- O desenvolvimento das funcionalidades abandonadas. Dado que alguns dos requisitos classificados como *Nice to Have* foram abandonados pelas mais variadas razões, estes não deixam de ser requisitos pertinentes, e seria interessante acrescentá-los à aplicação num momento posterior.
- A implementação da aplicação em iOS, assim que o número de utilizadores com este sistema operativo justifique o investimento.
- O acréscimo de algumas funcionalidades na aplicação web do Drivian Tasks, tais como ativar ou desativar permissões e funcionalidades de gestão de tarefas na aplicação móvel. Estas funcionalidades terão o propósito de limitar as opções da aplicação móvel apresentadas aos trabalhadores no terreno, para que estes não tenham acesso a um conjunto imenso de funcionalidades que, em alguns casos, não são pertinentes.

Bibliografia

About the Waze Transport SDK (2018). Disponível em: https://developers.google.com/waze/intro-transport

Ahuja, R. K., Mehlhorn, K., Orlin, J. B., & Tarjan, R. E. (2011). *Faster algorithms for the shortest path problem*. Saarbrücken: Saarländische Universitäts, Landesbibliothek.

Bing Routes API (2018). Disponível em: https://msdn.microsoft.com/en-us/library/ff701705.aspx

Breakthrough Technologies for National Security (2015). Disponível em: https://www.darpa.mil/attachments/DARPA2015.pdf

Bruno, D.R. & Osorio, F. S. (2017). *Image Classification System Based on Deep Learning Applied to the Recognition of Traffic Signs for Intelligent Robotic Vehicle Navigation Purposes*. 2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR). Curitiba, pp. 1-6.

C4 Model. Disponível em: https://c4model.com/

Cherkassy, B. V., Goldberg, A. V., & Radzik, T. (1993*). Shortest paths algorithms: Theory and experimental evaluation.* Stanford, Calif: Stanford University, Dept. of Computer Science.

Cline, A. (2015). *Agile development in the real world.*Tsui, F. F., Karam, O., & Bernal, B. (2014). *Essentials of software engineering, third edition.* Burlington, Mass: Jones & Bartlett Learning.

Galileo (2018). Disponível em: https://en.wikipedia.org/wiki/Galileo_(satellite_navigation)

GitHub. Disponível em: www.github.com

Google Maps Platform. Disponível em: https://developers.google.com/maps/

GPS (2017). Disponível em: https://www.gps.gov/systems/gps/

Graphhopper (2018). Disponível em: https://github.com/graphhopper/graphhopper

Here Maps SDK (2018). Disponível em: https://developer.here.com/develop/mobile-sdks

Inertial Navigation System. Disponível em: https://en.wikipedia.org/wiki/Inertial_navigation_system

InkScape. Disponível em: https://inkscape.org/en/

ISEC. Disponível em: https://www.isec.pt/PT/Instituto/

Koszalka, I. P., Koszalka, L., & Kasprzak, A. (2015). *Comparative Analysis of the Algorithms for Pathfinding in GPS Systems.* ICN 2015 : The Fourteenth International Conference on Networks. Barcelona, Spain.

Lendino, J. (2015). *DARPA to Re-invent GPS Navigation Without the Use of Satellites*. Disponível em: https://www.extremetech.com/extreme/202111-darpa-to-re-invent-gps-navigation-without-satellites

MapBox SDK (2018). Disponível em: https://www.mapbox.com/navigation-sdk/

Mapsforge (2018). Disponível em: https://github.com/mapsforge/mapsforge

Mapsforge Download Server. Disponível em: http://download.mapsforge.org/

Morales, J. J., Roysdon, P. F., & Kassas, Z. M. (2016). *Signals of Opportunity Aided Inertial Navigation*. Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION). Portland, Oregon, pp. 1492 - 1501.

Murase, H. (2017). *Image Recognition for Driver Assistance in Intelligent Vehicles.* 2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA), Nagoya, pp. 406-411.

Navit (2018). Disponível em: https://github.com/navit-gps/navit

OpenStreetMap Data Extracts. Disponível em: http://download.geofabrik.de/

Osmdroid (2018). Disponível em: https://github.com/osmdroid/osmdroid

OSRM Backend (2018). Disponível em: https://github.com/Project-OSRM/osrm-backend

Places API Policies (2018). Disponível em: https://developers.google.com/places/web-service/policies#terms_of_use_and_privacy_policy_requirements

Rubin, K. S. (2017). Essential Scrum: *A Practical Guide to the Most Popular Agile Process*. Upper Saddle River, NJ: Addison-Wesley.

Sentilant. Disponível em: www.sentilant.com/

Sestoft, P. (2010) *Numeric performance in C, C# and Java.* IT University of Copenhagen: Denmark. Retrieved from: http://www.itu.dk/~sestoft/papers/numericperformance.pdf

Shamaei, K., Khalife, J., Kassas, Z. M. (2016). *Performance Characterization of Positioning in LTE Systems*. Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION). Portland, Oregon, pp. 2262 - 2270.

SICAE. Disponível em: http://www.sicae.pt/Consulta.aspx

Skog, I., & Händel, P. (2005). *A Low-Cost GPS Aided Inertial Navigation System for Vehicle Applications*. 13th European Signal Processing Conference, Antalya, pp. 1-4.

TallyGo SDK. Disponível em: https://tallygo.com/

Virtual Box. Disponível em: https://www.virtualbox.org/

V™ (2018). Disponível em: https://github.com/mapsforge/vtm

Xubuntu. Disponível em: https://xubuntu.org/

yEd. Disponível em: https://www.yworks.com/products/yed

Anexos

Anexo A: Proposta de Estágio

PROPOSTA DE ESTÁGIO

Ano Letivo de 2017/2018

em Mestrado em Informática e Sistemas (Desenvolvimento de Software)

TEMA

Navegação *turn-by-turn* em iOS e Android

Os sistemas de gestão de tarefas (*field service management*) têm por objetivo aumentar a eficiência operacional das organizações, promovendo simultaneamente uma redução de custos com a operação. Nesse sentido, muito sistemas permitem não só identificar os recursos aos quais serão atribuídas as tarefas e a ordem pela qual serão executadas, mas também planear as rotas mais económicas entre os locais onde se realizam essas tarefas. Infelizmente, os condutores nem sempre têm acesso às rotas planeadas pelo sistema, tendo de recorrer a serviços de navegação de terceiros para "navegar" até aos locais predefinidos. Muitas vezes, as rotas propostas por estes sistemas externos não são as previstas e, quase nunca, as mais económicas. O objetivo deste estágio é desenvolver um componente de navegação *turn-by-turn* para as aplicações móveis de um sistema de *field service management* desenvolvido pela Sentilant.

Âmbito

Atualmente sediada na Incubadora de empresas do Instituto Pedro Nunes, a empresa Sentilant é uma empresa spin-off da Universidade de Coimbra, dedica-se à investigação e desenvolvimento de soluções inteligentes multiplataforma. Um dos produtos atuais da Sentilant suporta, para além de outras funcionalidades, o planeamento e a execução operacional de tarefas no terreno em tempo real, com recurso a uma aplicação para dispositivos móveis. Com o presente estágio, a empresa pretende estender as funcionalidades desta aplicação com a incorporação de capacidade de navegação *turn-by-turn* que permitam aos condutores seguir uma rota pré-definida e, simultaneamente, atualizada em tempo real de forma a reagir a variações no trânsito e outras condições anormais.

Objectivos

O presente projecto pretende atingir os seguintes objetivos genéricos:

- Desenvolver uma componente de navegação *turn-by-turn* para aplicações Android e IOS
- A componente de navegação deve permitir ao seu utilizador seguir uma rota proposta pelo componente de otimização de rotas do sistema

- A componente de navegação deve ser capaz de comunicar com serviços externos (trânsito, meteorologia, etc) e com outras aplicações (e.g., Waze) de forma a sugerir alterações a rotas sempre que não seja possível cumprir a rota prevista em condições favoráveis

Programa de trabalhos

O estágio consistirá nas seguintes atividades e respetivas tarefas:

- ***T1*** – *Definição de requisitos* – Definição de requisitos funcionais e não-funcionais.
- ***T2*** – *Estado da arte* – Estudo das tecnologias e técnicas para desenvolver aplicações de navegação *turn-by-turn* em Android e iOS.
- ***T3*** – *Programação* – Desenvolvimento, validação e verificação do código fonte para a componente de navegação.
- ***T4*** – *Integração* – Integração da componente desenvolvida com as aplicações de iOS e Android do sistema de gestão operacional.
- ***T5*** – *Validação* – Avaliação e validação da implementação.
- ***T6*** – *Escrita do relatório*– Escrita do relatório final do Estágio.

Calendarização das Tarefas

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

O plano de escalonamento dos trabalhos é apresentado em seguida:

	Meses									
Tarefas	**Oct-17**	**Nov-17**	**Dec-17**	**Jan-18**	**Feb-18**	**Mar-18**	**Apr-18**	**May-18**	**Jun-18**	**Jul-18**
T1										
T2										
T3										
T4										
T5										
T6										
Metas		**M1**		**M2**		**M3**		**M4**	**M5**	**M6**

M1 Tarefa T1 terminada

M2 Tarefa T2 terminada

M3 Versão 0.1 da componente de navegação em Android

M4 Versão 0.1 da componente de navegação em iOS e integração

M5 Tarefa T5 terminada

M6 Tarefa T6 terminada

Resultados

Os resultados do estágio serão consubstanciados num conjunto de documentos a elaborar pelo estagiário de acordo com o seguinte plano:

M1

R1.1: Documento de requisitos

M2:

R2.1: Documento do estado da arte

M3:

R3.1: Código fonte para componente em Android

M4:

R4.1: Código fonte para componente em iOS

R4.2: Plano de testes

M5:

R5.1: Documento de validação assinado pela Sentilant

M6:

R6.1: Relatório de estágio

Local de Trabalho

O trabalho decorrerá nas instalações da Sentilant na Incubadora de empresas do Instituto Pedro Nunes, Rua Pedro Nunes, 3030-199 Coimbra. Espera-se que o Estagiário cumpra um horário em regime de full-time na empresa.

Metodologia

A metodologia de trabalho a seguir terá em conta os seguintes aspetos:

- Integração do estagiário no processo de desenvolvimento de software interno;
- Utilização de uma metodologia ágil para gestão do desenvolvimento baseada em SCRUM;
- Recurso a técnicas de estimação para planeamento detalhado de *sprints*;
- Reuniões de início e fim de *sprint* para definição e avaliação de objetivos;
- Escrita de documentação inerente às funcionalidades desenvolvidas, de acordo com os requisitos da empresa.

Orientação

Empresa:

Pedro Costa (pacosta@sentilant.com)

Bruno Cabral (bruno.cabral@sentilant.com)

DEIS-ISEC:

Jorge Bernardino (jorge@isec.pt)

Professor Coordenador

Caracterização e Remuneração

- Data de início: 16 de Outubro 2017 (flexível)
- Data de fim: 15 de Junho 2018 (flexível)
- Horário: 50% entre 16/10/2017 e 25/02/2018; 100% entre 26/02/2018 e 15/06/2018
- Tipo de formação oferecida aos estagiários: formação em trabalho e acompanhamento diário.
- Local: Sentilant, Incubadora do IPN, Coimbra

Anexo B: Mockups do Módulo de Navegação

Visão de Navegação

Visão Geral de Percurso

Menu Principal

Instruções de Percurso

Opções de Tarefa em Viagem

Visualização de Estatísticas de Navegação

Opções de Tarefa em Execução

Opções de Sugestão de Visita

Anexo C: Apresentação Sobre Requisitos

Drivian: Componente Turn-by-Turn

Luís Henriques
a21260884

Mestrado em Informática e Sistemas, 2017

Componente Turn-by-Turn

Aplicações Analisadas

1. Características Pertinentes: Turn-by-Turn
2. Características Pertinentes: Apoio à Condução

Visão Geral

1. Navegação
2. Menu Principal
3. Lista de Percurso
4. Opções de Tarefa
5. Opções no Mapa
6. Pop-ups

Lista de Funcionalidades

Aplicações Analisadas

Aplicações Analisadas

15 Aplicações de navegação turn-by-turn

14 Aplicações de apoio à condução

Aplicações Analisadas

Características Pertinentes: Turn-by-Turn

1. Apresenta previsões de chegada e condições de trânsito em tempo real
2. Recalcula rotas baseado nas condições de trânsito
3. Assiste o condutor através de indicações com voz
4. Alerta se o condutor conduzir acima da velocidade permitida
5. Apresenta pontos de interesse pertinentes
6. Permite navegação em modo offline
7. Disponibiliza opções de rota (evitar autoestradas, caminhos locais, ...)
8. Permite procurar parques de estacionamento próximos do destino
9. Permite procurar pontos de abastecimento no percurso
10. Disponibiliza uma opção para reportar imprevistos

Aplicações Analisadas

Características Pertinentes: Apoio à Condução

1. Permite desativar notificações durante a viagem
2. Permite a leitura e envio de mensagens de texto com comandos de voz
3. Memoriza o lugar de estacionamento
4. Estima as despesas com combustível
5. Permite aplicar filtros às estimativas de combustível como gasóleo, gás, gasolina
6. Permite calcular as despesas com portagens desde o início ao fim da viagem
7. Permite criar perfis de veículos, com impacto nas estimativas
8. Em caso de acidente, faz automaticamente uma chamada de emergência

7

Visão Geral

8

Visão Geral
Navegação

9

Visão Geral
Navegação

10

Visão Geral
Navegação

11

Visão Geral
Menu Principal

12

Visão Geral
Menu Principal

13

Visão Geral
Menu Principal

14

Visão Geral
Menu Principal

15

Visão Geral
Lista de Percurso

16

Visão Geral
Lista de Percurso

17

Visão Geral
Opções de Tarefa

18

Visão Geral
Opções de Tarefa

19

Visão Geral
Opções de Tarefa

20

Visão Geral
Opções de Tarefa

21

Visão Geral
Opções no Mapa

22

Visão Geral
Opções no Mapa

23

Visão Geral
Opções no Mapa

24

Visão Geral
Pop-ups

25

Visão Geral
Pop-ups

26

Visão Geral
Pop-ups

27

Visão Geral
Pop-ups

28

Lista de Funcionalidades

29

Lista de Funcionalidades

RFM-IN-001	Conhecer o percurso para o dia	*Must*
RFM-IN-002	Conduzir para o próximo destino	*Must*
RFM-IN-003	Ver a lista de indicações do percurso atual	*Should*
RFM-IN-004	Ouvir alerta para condução acima da velocidade permitida	*Should*
RFM-IN-005	Selecionar percurso de volta	*Must*
RFM-IN-006	Escolher opções de percurso	*Nice to Have*
RFM-IN-007	Corrigir destino	*Should*
RFM-IN-008	Ver informações sobre tarefas	*Must*
RFM-IN-009	Ver opções de tarefa	*Must*
RFM-IN-010	Adicionar complementos de tarefa	*Nice to Have*
RFM-IN-011	Receber novas tarefas	*Must*

30

Lista de Funcionalidades

RFM-IN-012	Fazer chamadas para o cliente atual	*Must*
RFM-IN-013	Enviar mensagens automáticas para o cliente atual	*Should*
RFM-IN-014	Fazer chamadas para a central	*Must*
RFM-IN-015	Receber sugestões de visita espontânea	*Should*
RFM-IN-016	Telefonar a cliente de visita espontânea	*Should*
RFM-IN-017	Pesquisar clientes	*Nice to Have*
RFM-IN-018	Ver pontos de abastecimento no percurso	*Should*
RFM-IN-019	Incluir um ponto de abastecimento no percurso atual	*Should*
RFM-IN-020	Ver estatísticas de condução	*Nice to Have*
RFM-IN-021	Reportar imprevistos	*Should*
RFM-IN-022	Procurar parques de estacionamento	*Nice to Have*

Questões

Anexo D: Apresentação Sobre Alternativas de Implementação

Drivian: Componente Turn-by-Turn

Luís Henriques
a21260884

Mestrado em Informática e Sistemas, 2017

isec

2

Drivian: Componente Turn-by-Turn

- Navegação com SDKs
- Navegação a partir de APIs
- Grelha comparativa

3

Drivian: Componente Turn-by-Turn

Visão Geral de Percurso

Navegação

4

Navegação com SDKs

5

Here SDK

- ✓Rotas cientes de trânsito
- ✓Navegação com voz em Português
- ✓Suporte para mapas offline
- ✓Opções de rota (evitar autoestradas, ...)
- ✓Custos de portagens
- ✓Localização de radares
- ✓Rotas para veículos pesados
- ✓Opções para perfis de veículos pesados
- ✓Android + iOS

Preços

- €21 p/ ano para ligeiros (€1.75 por mês)
- €22.80 p/ ano para pesados (€1.90 por mês)
- + Back office route planning:
  - Car Navigation €20.30 p/ ano
  - Truck Navigation €29 p/ano

6

MapBox

- ✓Rotas cientes de trânsito
- ✓Navegação com voz em Português (Amazon Polly)
- ✓Instruções de via (não apenas de estrada)
- ✓Suporte para mapas offline
- ✓Localização de radares (para finais de 2017)
- ✓Android + iOS
- ✓Extremamente customizável (até mapas)
- ✓Preços
  - €0.50 por cada 500 users mensais (após os 50.000 iniciais)
  - + Amazon Polly
    - Gratuito nos primeiros 12 meses (com limite de 5 milhões de caracteres p/mês)
    - 4USD por cada 1 milhão de caracteres

7

Navit

✓Suporte para mapas offline
✓Opções de rota (evitar autoestradas, …)
✓Android
✓Preços
Gratuito
Licença GPL e LGPL

8

Outros SDKs

Skobbler
- Não disponível

Navmii
- À espera de resposta para obter chave

Sygic SDK
- À espera de resposta para obter chave

Carto
- À espera de resposta para obter chave

MapQuest
- Não calcula rotas cientes de trânsito em Portugal

MapZen
- Não calcula rotas cientes de trânsito em Portugal

TallyGo
- Sem cobertura em Portugal

AntaresNav
- Sem cobertura em Portugal

9

Navegação a partir de APIs

10

Navegação a partir de APIs

Mapas online	Mapas offline
Caching de mapas conforme necessário	275 MB
Impossível calcular rotas offline (incluindo túneis)	Rotas offline
	Download de mapas
	Update de mapas integrado
	Manutenção dos mapas
	Mapas no nosso servidor
	Aplicação independente de terceiros

Rotas online otimizadas
Direções com voz
Library de voz em Português Europeu – a testar
Integração de qualquer API
Rotas cientes de trânsito (apenas online)
Rotas para pesados (apenas online)

11

Google APIs

✓Cobertura em Portugal
✓Rotas cientes de trânsito (apenas em versão premium)
✓Opções de rota (evitar autoestradas, por exemplo)
✓Preços:
€0.50 por cada 1000 pedidos

12

Graphhopper APIs

✓Cobertura em Portugal
✓Rotas cientes de trânsito
✓Opções de rota (evitar autoestradas, por exemplo)
✓Cálculo de custos de portagens
✓Rotas para veículos pesados
✓Preços:
€39 / mês por 5.000 pedidos diários
€109 por 15.000
€239 por 50.000

13

Openroute APIs

✓Cobertura em Portugal
? Rotas cientes de trânsito
✓Opções de rota (evitar autoestradas, por exemplo)
✓Cálculo de custos de portagens
✓Rotas para veículos pesados
✓Perfis de veículos pesados
✓Preços:
€55 / mês por <10.000 pedidos diários
€105 por <20.000

14

Bing Maps APIs

✓Cobertura em Portugal
✓Rotas cientes de trânsito
✓Opções de rota (evitar autoestradas, por exemplo)
? Cálculo de custos de portagens
✓Rotas para veículos pesados
✓Perfis de veículos pesados
✓Preços:
Por asset + backoffice

15

Via Michelin API

- ✓Cobertura em Portugal
- ✓Rotas cientes de trânsito
- ✓Opções de rota (evitar autoestradas, por exemplo)
- ✓Cálculo de custos de portagens
- ✓Rotas para veículos pesados
- ✓Perfis de veículos pesados
- ✓Preços:
  - €5000 p/ ano por 0 a 500.000 pedidos anuais
  - €8000 p/ ano por 500.000 a 1.000.000 pedidos anuais
  - ...

16

Questões

Anexo E: Código do Algoritmo de Acompanhamento de Rotas

```
public GeoPoint findUser(){

    NavPath path = NavigationData.getCurrentPath();

    // Route tracking algorithm
    if (path != null && !path.getPathPoints().isEmpty() && !NavigationData.isCalculatingPath())
{
        double freeWalkRadius = 100; // the user may walk freely on a 100 m radius around the
first point (point 0)

        // if the the user didn't begin following the path yet, we allow him to move freely
        if (path.getLastPassedPointIndex() == 0 // if the user is still within the
freeWalkRadius of point 0
                &&
path.getPathPoints().get(0).getGeoPoint().sphericalDistance(NavigationData.getUserPosition(conte
xt)) < freeWalkRadius) {
            userIsOnPath = true; // we declare that the user was found on the path
            NavigationData.setCameraBearing(NavigationData.getUserBearingInDegrees()); // we
update the turn-by-turn camera direction to be the user's natural direction
            return NavigationData.getUserPosition(context); // and simply return the detected
user position
        }

        if (NavigationData.getArrivalTime() == 0
                || (System.currentTimeMillis() < NavigationData.getArrivalTime())) { // if the
user is still on schedule (or if there is no calculated schedule at all)

            GeoPoint userPos = NavigationData.getUserPosition(context); // we keep the user's
position to improve code legibility

            // If everything goes well, we'll just perform this calculation once or twice, as
the user is following the path
            for (int i = path.getLastPassedPointIndex(); i < path.getLastPassedPointIndex() +
50; i ++ ) { // NOTE: we do this for the next 50 points? Should we check against coldRadius
instead?

                // if there is, at least, one more point in our path
                if (path.getPathPoints().size() > i + 1) {

                    // we try to identify which points in our path are behind and in front of
the user
                    RoutePoint possiblePointBehind = path.getPathPoints().get(i);
                    RoutePoint possiblePointInFront = path.getPathPoints().get(i + 1);

                    // we measure the distance from the point behind the user to the point in
front of him (point A -> point B)
                    double distAtoB =
possiblePointBehind.getGeoPoint().sphericalDistance(possiblePointInFront.getGeoPoint()); //
NOTE: We can't use possiblePointInFront.getDistanceFromPreviousPoint(), since it may return 0

                    // we also retrieve the distance from the point behind the user to the
user's position (point A -> user)
                    double distUserToPointBehind =
userPos.sphericalDistance(possiblePointBehind.getGeoPoint());

                    // and the distance from the user to the point in front of him as well (user
-> point B)
                    double distUserToPointInFront =
userPos.sphericalDistance(possiblePointBehind.getGeoPoint());

                    // if we pass by the lastOriginalRoutePoint, we reset it
                    if (lastOriginalRoutePoint != null &&
lastOriginalRoutePoint.equals(possiblePointBehind)) {
                        Log.w(TAG, "Passing by last original RP");
                        lastOriginalRoutePoint = null;
                    }
```

```
                    /*if the user is closer to the point behind him than the point in front of
him is, and the same is true for the point in front as well, it means the user is between both
points */

                    if ((distUserToPointBehind - NavigationData.getGpsAccuracy()) < distAtoB
                            && (distUserToPointInFront - NavigationData.getGpsAccuracy()) <
distAtoB) { // we also add the gps accuracy

                        // then we retrieve a point between possiblePointBehind and
possiblePointInFront that is the closest to the user's position
                        GeoPoint connectionPoint = findConnectionPoint(
                                possiblePointBehind.getGeoPoint(),
                                possiblePointInFront.getGeoPoint(),
                                userPos);

                        // then we check the distance between the user and this connection
point. If the user is close enough, we know the user is on the path
                        if (connectionPoint.sphericalDistance(userPos) -
NavigationData.getGpsAccuracy() < warmRadius) {

                            // we update the turn-by-turn camera direction

NavigationData.setCameraBearing(possiblePointBehind.getGeoPoint().bearingTo(possiblePointInFront
.getGeoPoint()));

                            userIsOnPath = true;

                            // We don't let the user's position move backwards
                            if (path.followPath(possiblePointBehind)) { // to do so, if the
point index was updated
                                previousDistanceToPointBehind = 0; // we reset the distance to
the previous point, as it is no longer relevant. After all, we have a new point behind the user
                            }

                            if (distUserToPointBehind > previousDistanceToPointBehind) { // if
the user moved forward
                                previousDistanceToPointBehind = distUserToPointBehind; // we
update the distance to the point behind
                                return connectionPoint; // and return the user's position on the
path line (the connection point's position)

                            } else {
                            /* but if the user didn't move forward, we want to return a point on
the same position as before. This way, if the user moves backwards, the marker won't move
backwards as well. It stays on the same position as before */
                                float bearing = (float)
possiblePointBehind.getGeoPoint().bearingTo(possiblePointInFront.getGeoPoint());
                                return
possiblePointBehind.getGeoPoint().destinationPoint(previousDistanceToPointBehind, bearing);
                            }
                        }
                    }

                }
            }
            // if we arrive at this point without returning a valid user position on our path,
we adjust the route
            adjustCurrentPath();

        } else {
            // if we already passed the expected arrival time, we simply calculate the route
from scratch (and recalculate ETA in the process)
            if (!NavigationData.isCalculatingPath())
                calculateCurrentPathFromScratch(); // we calculate the path from scratch
        }
    }

    // if we can't find the user on our path, or if there is no path at all
    userIsOnPath = false; // we declare that the user is not following the path
    NavigationData.setCameraBearing(NavigationData.getUserBearingInDegrees()); // we update the
turn-by-turn camera direction to be the user's natural direction
    return NavigationData.getUserPosition(context); // and simply return the detected user
position
```

```
}


/**
 * Finds a point on the line between pointA and pointB that is the closest to the specified
position
 * @param pointA the starting point of a line
 * @param pointB the end point of a line
 * @param position a position that is, supposedly, somewhere between point A and point B
 * @return a point on the line between A and B that is the closest to the given position
 */
private GeoPoint findConnectionPoint(GeoPoint pointA, GeoPoint pointB, GeoPoint position){

    double behindX = pointA.getLongitude();
    double behindY = pointA.getLatitude();
    double inFrontX = pointB.getLongitude();
    double inFrontY = pointB.getLatitude();
    double posX = position.getLongitude();
    double posY = position.getLatitude();

    // so we calculate the new point in our straight road segment that is the closest to the
user
    Point p = GeoPointUtils.nearestSegmentPoint(behindX, behindY, inFrontX, inFrontY, posX,
posY);

    // and we translate it to a GeoPoint
    return new GeoPoint(p.getY(), p.getX());
}
```

Anexo F: Código do Algoritmo de Ajuste de Rotas

```
private void adjustCurrentPath(){
    if (!NavigationData.isCalculatingPath()) {
        // if we couldn't return a valid user position on the path, we adjust the current path
        NavPath currentPath = NavigationData.getCurrentPath();

        // if we don't have a router or a path, we can't navigate
        if (router == null || currentPath == null) {
            return;
        }

        // finding the lastOriginalRoutePoint (can be the same one as before)
        lastOriginalRoutePoint = findEdgePointOnPath(currentPath);
        if (lastOriginalRoutePoint != null) {

            calculateCurrentPathUpTo(lastOriginalRoutePoint); // if we find it, we calculate a
path to it

        } else {
            calculateCurrentPathFromScratch();
        }

    }
}


/**
 * Finds the last valid point at the edge of the cold radius
 * @param path the path
 * @return Returns a valid point at the edge of the cold radius
 */
@Nullable
private RoutePoint findEdgePointOnPath(NavPath path){

    GeoPoint userPos = NavigationData.getUserPosition(context);

    // first of all, if we are too close to the waypoint, we simply return it
    if (userPos.sphericalDistance(path.getWaypoint().getGeoPoint()) < greenZoneFarEdge) {
        return path.getWaypoint();
    }

    // if lastOriginalRoutePoint  is within the green zone, we return it
    if (lastOriginalRoutePoint != null) {
        double distanceToLastOriginalRP =
userPos.sphericalDistance(lastOriginalRoutePoint.getGeoPoint());

        if (distanceToLastOriginalRP > greenZoneCloseEdge
                && distanceToLastOriginalRP < greenZoneFarEdge) {
            return lastOriginalRoutePoint;
        }
    }

    // else, we'll try to find a valid connection point in our path (a new
lastOriginalRoutePoint)
    List<RoutePoint> pathPoints = path.getPathPoints();
    int beginSearchIndex;


    // if we have a valid lastOriginalRoutePoint
    if (lastOriginalRoutePoint != null // it exists
            && pathPoints.contains(lastOriginalRoutePoint) // it is in our path
            && pathPoints.indexOf(lastOriginalRoutePoint) > path.getLastPassedPointIndex()) { //
and it has a higher index than the last point passed by the user
        beginSearchIndex = pathPoints.indexOf(lastOriginalRoutePoint); // we begin our search
from its index
    } else { // if we don't have a valid lastOriginalRoutePoint
```

```
        beginSearchIndex = path.getLastPassedPointIndex(); // we begin searching from the index
of the last point we passed by
    }

    // the point we are going to return
    RoutePoint edgePoint = null;
    // preventing bugs
    if (beginSearchIndex >= pathPoints.size()) {
        beginSearchIndex = 0;
    }

    for (int i = beginSearchIndex; i < pathPoints.size(); i ++) {
        // the point behind the user is ignored. We want to begin our search from the point in
front of him

        if (i == path.getLastPassedPointIndex() || i == 0) {
            continue; // we do this just to simplify. It would be more complicated to check
pathPoints.size() against beginSearchIndex + 1
        }

        RoutePoint rp = path.getPathPoints().get(i);
        // if we reach the waypoint, we stop looking
        if (rp.isWaypoint()) {
            edgePoint = rp;
            break;
        }

        // if we are looking too far away already, we simply stop looking
        if (userPos.sphericalDistance(rp.getGeoPoint()) > greenZoneFarEdge) {
            break;
        }

        // if the point is within the green zone (closeEdge to midPoint)
        if (userPos.sphericalDistance(rp.getGeoPoint()) > greenZoneCloseEdge
                && userPos.sphericalDistance(rp.getGeoPoint()) < greenZoneFarEdge) {
            edgePoint = rp; // we'll assign it as being our best option so far

        } else if (edgePoint != null) { // if we leave the green zone after finding a point
            break; // we stop searching. We already found a valid point
        }
    }
    return edgePoint;
}
```

Anexo G: Código do Sistema de Construção de Instruções

```
public String getInstructionSpeech(RoutePoint instructionPoint, InstructionType type, double
distance) {

    // now we'll build the text message depending on what we find
    String voiceMessage = "";
    String justDistanceStr = ""; // a string containing distance information only

    /* Non-standard conditions, in which we do not build instructions */
    if (type == InstructionType.NEARBY
            && instructionPoint.getIcon() == RoutePoint.DirIcon.DESTINATION) { // or if we just
arrived at our destination

        voiceMessage = translateSign(RoutePoint.DirIcon.DESTINATION, InstructionType.NEARBY);

    }     else     if     (instructionPoint.getIcon()     !=     RoutePoint.DirIcon.START     &&
instructionPoint.getIcon() != RoutePoint.DirIcon.READJUSTED) {
        // we only talk about distances if we are referring to a far away instruction
        if (type == InstructionType.FAR_AWAY) {

            // if the next instruction is very near, we won't add the distance
            if (distance > 50) { // so, if the instruction is NOT too close (under 50 meters)

                // we add the prefix
                String  distPrefix  =  context.getString(R.string.voice_distance_prefix).concat("
");

                String[] distStr = String.valueOf(distance).split("\\.");
                String baseDist = distStr[0];

                // final string elements
                String finalDist;
                String metric;
                String  andAHalf  =  "";  //  literally  "and  a  half",  if  needed  (in  case  of
kilometers)

                if (baseDist.length() > 3) { // kilometers

                    try {
                        // trying to find if the remaining distance is >= 500m
                        double  meters  =  Double.valueOf(baseDist.substring(baseDist.length()  -
3));

                        if (meters >= 500) { // if so, we add the "and a half" text
                            andAHalf                                                              =
context.getString(R.string.voice_distance_metric_and_a_half);
                        }


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

                    finalDist = baseDist.substring(0, baseDist.length() - 3); // 1500 to 15

                    metric  =  context.getString(R.string.voice_distance_metric_kilometers);  //
metric is always "kilometers" (plural) in case parsing fails

                    try {

                        double km = Double.valueOf(finalDist);

                        if (km < 2) { // single kilometer
                            metric                                                                =
context.getString(R.string.voice_distance_metric_single_kilometer);
                        }
```

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

                } else { // meters
                    finalDist = baseDist; // 366 = 366 meters

                    // we clear the last number of the distance, so the voice isn't too
specific. E.g: 231 = 230 & 23 = 20
                    finalDist = finalDist.substring(0, finalDist.length() - 1);
                    finalDist = finalDist.concat("0");

                    metric = context.getString(R.string.voice_distance_metric_meters);

                }

                // we take out any 0s before
                finalDist = finalDist.replaceFirst("^0+(?!$)", "");

                // building the distance string. Note: the last ", " must be here, so we can
compare string lengths by the end
                voiceMessage += distPrefix
                        .concat(" ")
                        .concat(finalDist)
                        .concat(" ")
                        .concat(metric)
                        .concat(" ")
                        .concat(andAHalf)
                        .concat(", ");

                // we keep the string containing just distance, so we ignore the whole message
if no further content is added
                justDistanceStr = voiceMessage;
            }

            // up to this point, if we are not referring to an immediate point, we have a valid
distance sentence
        }

        /* now we'll building the instruction itself according to the information we have
        we'll prioritize the already built-in instruction, but most times there is none */
        if                    (!instructionPoint.getInstruction().isEmpty()                    ||
!instructionPoint.getInstruction().equals("")) { // if we have a voice instruction
            voiceMessage += instructionPoint.getInstruction(); // we simply use it

        } else { // if we do not have a built in instruction, we'll use the sign as a guide for
our instruction
            if(instructionPoint.getIcon() != RoutePoint.DirIcon.ROUNDABOUT) {
                voiceMessage += translateSign(instructionPoint.getIcon(), type); // we translate
the sign as appropriate
            } else {
                // as for roundabouts, we want to treat them differently
                if (instructionPoint.getRoundaboutExit() > 0) { // if there is a set roundabout
exit
                    if (type.equals(InstructionType.FAR_AWAY)) {
                        voiceMessage += " "
                                .concat(context.getString(R.string.voice_enter_roundabout))
                                .concat(", ");
                    }
                    // if it is a nearby instruction, we only want the text "take the 2nd exit"
without the "at the roundabout" part
                    voiceMessage += " "

.concat(context.getString(R.string.voice_enter_roundabout_with_exit))
                            .concat(" ")
                            .concat(String.valueOf(instructionPoint.getRoundaboutExit()))
                            .concat(" ")

.concat(context.getString(R.string.voice_enter_roundabout_exit_suffix))
                            .concat(". ");
                }

                // NOTE: We ignore roundabout instructions without exit
```

```
            }
        }

        /* up to this point, we have a full instruction that may or may not have a distance.
Some examples:
        e.g: in about 500 meters, at the roundabout, take the second exit <- distant instruction
with built-in instruction text
        e.g: in about 230, turn right <- distant instruction with translated sign
        e.g: turn left <- nearby instruction
        e.g: in about 1 and a half kilometers, you will enter <- distant instruction with road
name change
        e.g: you just entered <- nearby instruction with road name change
        e.g: in about 200 meters, <- distant instruction with UNKNOWN type (that's why we keep
justDistanceStr) */

        /* finally, we compare the street name of our instruction point with the previously
detected instruction point
        Note: we haven't updated nextInstructionPoint yet. We can still compare it with the
current one */
        if (!instructionPoint.getStreetName().isEmpty()) { // so, if we have a street name

            // if the street name wasn't already mentioned in the built-in instruction
            if (!instructionPoint.getInstruction().contains(instructionPoint.getStreetName())

                // nor in the previous nearby instructions
                && (lastPlayedNearbyInstruction == null
                ||
!instructionPoint.getStreetName().equals(lastPlayedNearbyInstruction.getStreetName()))

                // and this is a nearby instruction
                && (type == InstructionType.NEARBY

                // or, in case it is a distant instruction, if we don't repeat the street name
                || (type == InstructionType.FAR_AWAY
                && (lastPlayedFarAwayInstruction == null
                ||
!instructionPoint.getStreetName().equals(lastPlayedFarAwayInstruction.getStreetName())))) {

                    // we add the prefix, if the message doesn't have one yet
                    if
(!voiceMessage.contains(context.getString(R.string.voice_road_name_change_distant))
                            &&
!voiceMessage.contains(context.getString(R.string.voice_road_name_change_immediate))) {
                        voiceMessage += ", "

.concat(context.getString(R.string.voice_road_name_change_distant));
                    }
                    voiceMessage  +=  "  ".concat(instructionPoint.getStreetName());  //  and  we
finally add the street name
            }
        }
    }
    // finally, we simply correct the instruction
    Orthographer orthographer = new Orthographer(NavSettings.getLocale());
    voiceMessage = orthographer.correctText(voiceMessage);
    voiceMessage = voiceMessage.replaceAll(" , ", ", ");

    /* let's do some checking so we don't return unwanted messages. We'll ignore messages that:
    - contain only distance information
    - are exactly the same as the last played message */
    if (voiceMessage.equals(justDistanceStr) || voiceMessage.equals(lasPlayedText)) {
        Log.w(TAG, "Ignored instruction -> " + voiceMessage );
        return "";
    }
    lasPlayedText = voiceMessage;
    return voiceMessage;
}
```

```
/**
 * Translates an icon to text to be read by the text to speech
 * @param icon the icon from RoutePoint.DirIcon
 * @return the text to be read
 */
@NonNull
private String translateSign(RoutePoint.DirIcon icon, InstructionType type) {
    switch (icon) {
        case START: return context.getString(R.string.voice_start);
        case READJUSTED: return context.getString(R.string.voice_readjusted);
        case CONTINUE: return context.getString(R.string.voice_continue);
        case DESTINATION:
            if (type == InstructionType.NEARBY) {
                return ""; // this instruction is ignored, since we will force it from the
MapMaster instance
            } else {
                return context.getString(R.string.voice_destination_distant);
            }
        case TURN_SLIGHT_LEFT: return context.getString(R.string.voice_turn_slight_left);
        case TURN_SLIGHT_RIGHT: return context.getString(R.string.voice_turn_slight_right);
        case TURN_LEFT: return context.getString(R.string.voice_turn_left);
        case TURN_RIGHT: return context.getString(R.string.voice_turn_right);
        case TURN_SHARP_LEFT: return context.getString(R.string.voice_turn_sharp_left);
        case TURN_SHARP_RIGHT: return context.getString(R.string.voice_turn_sharp_right);
        case KEEP_LEFT: return context.getString(R.string.voice_keep_left);
        case KEEP_RIGHT: return context.getString(R.string.voice_keep_right);
        case ON_RAMP_LEFT:return context.getString(R.string.voice_enter_ramp_left);
        case ON_RAMP_RIGHT:return context.getString(R.string.voice_enter_ramp_right);
        case OFF_RAMP_LEFT:return context.getString(R.string.voice_exit_ramp_left);
        case OFF_RAMP_RIGHT:return context.getString(R.string.voice_exit_ramp_right);
        case ROUNDABOUT: return context.getString(R.string.voice_enter_roundabout);
        case LEAVE_ROUNDABOUT: return context.getString(R.string.voice_leave_roundabout);
        case ROAD_NAME_CHANGE: return ""; // we decided to ignore road name change instructions
        case U_TURN: return context.getString(R.string.voice_u_turn);
        default: // UNKNOWN and others
            return "";
    }
}
```

Anexo H: Código do Algoritmo de Ordenação de Sugestões de Visita

```
public ArrayList<SuggestionPlace> rankSuggestions(List<SuggestionPlace> entries) {

    /* NOTES:
    The value of each like depends on the number of visitors. If we have lots of visitors but
few likes, it is a bad scenario.
    Conversely, is we have few visitors but almost every single one of them liked the
establishment, that is very good
    The parameter "likeScore" will reflect this

    Of course the number of visitors itself is also considered

    The value of the distance is also affected by the number of matching categories. A somewhat
distant establishment
    that matches many of the categories chosen by the user should be preferred above a closer
establishment that only
    matches a single category

    All other factors are a matter of simple scoring */

    // retrieving normalized entries
    Map<SuggestionPlaceNormalized, SuggestionPlace> normalizedEntries =
normalizeEntries(entries);

    // scoring each entry
    for (SuggestionPlaceNormalized s : normalizedEntries.keySet()) {
        scoreEntry(s);
    }

    // sorting all entries by score
    Map<SuggestionPlaceNormalized, SuggestionPlace> sortedEntries =
shellSort(normalizedEntries);
    Set<SuggestionPlaceNormalized> sortedKeys = sortedEntries.keySet();

    // creating a new list that inserts all ordered elements
    ArrayList<SuggestionPlace> returnList = new ArrayList<>();
    for (SuggestionPlaceNormalized key : sortedKeys) {
        returnList.add(sortedEntries.get(key));
    }

    return returnList;
}

/**
 * Normalizes all data from each entry
 * @param entries the entries to be normalized
 * @return a map with a fully normalized entry, and its matching regular entry
 */
@NonNull
private Map<SuggestionPlaceNormalized, SuggestionPlace> normalizeEntries(List<SuggestionPlace>
entries) {
    // let's find the maximum values so we can normalize all data
    double maxDistance = 0;
    int mostSharedCats = 0;
    double bestReputation = 0;
    int mostLikes = 0;

    int mostClientVisits = 0;
    int mostUsers = 0;
    int mostCheckins = 0;
    int mostTips = 0;

    /* most times, not every entry has the required data, in which case, we'll use the best
found one. This is in order of preference */
    boolean allEntriesHaveVisits = true;
    boolean allEntriesHaveUsers = true;
    boolean allEntriesHaveCheckins = true;
```

```
    boolean allEntriesHaveTips = true;

    List<SuggestionPlace> validEntries = new ArrayList<>();

    // retrieving max to normalize all values
    if (entries != null) {
        for (SuggestionPlace entry : entries) {

        /* first we check for the "deal breakers". Is the user already visited the place
        or the place is not open, we won't even consider it */
            if (entry.isOpenNow() && !entry.userAlreadyVisited()) {

                // retrieving max distance
                double distanceToUser = entry.getDistanceToUser();
                if (distanceToUser > maxDistance) {
                    maxDistance = distanceToUser;
                }

                // retrieving max shared categories
                int numberSharedCats = entry.getNumberOfMatchingCategories();
                if (numberSharedCats > mostSharedCats) {
                    mostSharedCats = numberSharedCats;
                }

                // retrieving max reputation
                double rep = entry.getReputation();
                if (rep > bestReputation) {
                    bestReputation = rep;
                }

                // retrieving max likes
                int likes = entry.getNumberOfLikes();
                if (likes > mostLikes) {
                    mostLikes = likes;
                }

                // retrieving max number of tips and checking if every entry has tips
                int tips = entry.getNumberOfTips();
                if (tips > 0) {
                    if (tips > mostTips) {
                        mostTips = tips;
                    }
                } else {
                    allEntriesHaveTips = false;
                }

                // retrieving max number of visits and checking if every entry has visits. We
also check for the validity of this data
                int visits = entry.getNumberOfVisits();
                if (visits > 0 && entry.getNumberOfLikes() <= visits) {
                    if (visits > mostClientVisits) {
                        mostClientVisits = visits;
                    }
                } else {
                    allEntriesHaveVisits = false;
                }

                // retrieving max number of users and checking if every entry has users.  We
also check for the validity of this data
                int users = entry.getNumberOfUsers();
                if (users > 0 && entry.getNumberOfLikes() <= users) {
                    if (users > mostUsers) {
                        mostUsers = users;
                    }
                } else {
                    allEntriesHaveUsers = false;
                }

                // retrieving max number of checkins and checking if every entry has checkins.
We also check for the validity of this data
                int checkins = entry.getNumberOfCheckins();
                if (checkins > 0 && entry.getNumberOfLikes() <= checkins) {
                    if (checkins > mostCheckins) {
                        mostCheckins = checkins;
```

```
                }
            } else {
                allEntriesHaveCheckins = false;
            }

            validEntries.add(entry);

        } else {
            Log.e(TAG, entry.getName() + " won't be considered. Is it open? " +
entry.isOpenNow() + ". Did the user visit it already? " + entry.userAlreadyVisited());
        }
    }
}

    boolean useWeightedScores = true; // should we score the entries based on a weight? That
depends if there is one available

    // picking the best weight factor (by order of preference)
    if (!allEntriesHaveVisits && !allEntriesHaveUsers && !allEntriesHaveCheckins) {
        Log.w(TAG, "Not enough data to use weights. Weights won't be considered" );
        useWeightedScores = false; // if we do not have any weight available, we can't calculate
relevance based on it
    }

    // Now that we have all our maximum values, we'll create new entries with normalized values
    Map<SuggestionPlaceNormalized, SuggestionPlace> normalizedMap = new HashMap<>(); // we'll
also store our entries on a map

    for (SuggestionPlace entry : validEntries) {

        double likeScore = 0; // if we are not considering like score, it is simply 0 for every
entry
        // choosing the proper weight depending on the data we have available
        if (useWeightedScores) {
            int likeWeight;
            if (allEntriesHaveVisits) {
                likeWeight = entry.getNumberOfVisits();

            } else if (allEntriesHaveUsers) {
                likeWeight = entry.getNumberOfUsers();

            } else {
                likeWeight = entry.getNumberOfCheckins();
            }

            likeScore = (double) entry.getNumberOfLikes() / (double) likeWeight; // by its very
nature, like score is already normalized

        }

        // normalizing all values and creating a new entry with normalized data
        SuggestionPlaceNormalized normalizedEntry = new SuggestionPlaceNormalized(
                normalizeDist(entry.getDistanceToUser(), maxDistance),  // distance to user is
inverted. Higher values mean it is closer
                normalize(entry.getNumberOfMatchingCategories(), mostSharedCats),
                normalize(entry.getReputation(), bestReputation),
                normalize(entry.getNumberOfLikes(), mostLikes),
                allEntriesHaveTips? normalize(entry.getNumberOfTips(), mostTips) : 0, // we only
add number of tips if we have such data
                allEntriesHaveVisits? normalize(entry.getNumberOfVisits(), mostClientVisits) :
0,
                allEntriesHaveUsers? normalize(entry.getNumberOfUsers(), mostUsers) : 0,
                allEntriesHaveCheckins? normalize(entry.getNumberOfCheckins(), mostCheckins) :
0,
                likeScore);

        // adding the normalized entry, as well as the matching entry, to the map
        normalizedMap.put(normalizedEntry, entry);
    }
    return normalizedMap;
}


private void scoreEntry(SuggestionPlaceNormalized entry) {
```

```
    // distance is already inverted!
    double distanceScore = entry.getNormalizedNumberOfMatchingCategories() *
entry.getNormalizedDistanceToUser();

    // pay attention to the weights we are setting
    double totalScore = (2 * distanceScore)
            + (2 * entry.getLikeScore())
            + entry.getNormalizedReputation()
            + entry.getNormalizedNumberOfMatchingCategories()
            + entry.getNormalizedNumberOfLikes()
            + entry.getNormalizedNumberOfUserVisits()
            + entry.getNormalizedNumberOfUsers()
            + entry.getNormalizedNumberOfCheckins()
            + (0.5 * entry.getNormalizedNumberOfTips());

    // relevance is in percentage
    entry.setRelevance( (100 * totalScore) / 10.5 );
}
```

Anexo I: *Script* de Preparação de Mapas e Grafos

```
echo "Which continent do you want to download maps from?"
read CONTINENT
echo "Which country do you want to download maps from?"
read COUNTRY

# adjusting continent names to match urls
case $CONTINENT in
    "australia" | "oceania" | "australia and oceania" | "australia-oceania")
        GEOFABRIK_CONTINENT="australia-oceania"
        MAPSFORGE_CONTINENT="australia-oceania"
        break
        ;;

    "north america")
        GEOFABRIK_CONTINENT="north-america"
        MAPSFORGE_CONTINENT="north-america"
        break
        ;;

    "central america")
        GEOFABRIK_CONTINENT="central-america"
        MAPSFORGE_CONTINENT="central-america"
        break
        ;;

    "south america")
        GEOFABRIK_CONTINENT="south-america"
        MAPSFORGE_CONTINENT="south-america"
        break
        ;;

    *)
        GEOFABRIK_CONTINENT=$CONTINENT
        MAPSFORGE_CONTINENT=$CONTINENT
        ;;
esac

echo "Downloading maps from $CONTINENT: $COUNTRY"
echo "Downloading graph file from http://download.geofabrik.de/$GEOFABRIK_CONTINENT/$COUNTRY"
wget http://download.geofabrik.de/$GEOFABRIK_CONTINENT/$COUNTRY-latest.osm.pbf
echo "Downloading map tiles from
http://download.mapsforge.org/maps/v4/$MAPSFORGE_CONTINENT/$COUNTRY"
wget http://download.mapsforge.org/maps/v4/$MAPSFORGE_CONTINENT/$COUNTRY.map
echo "-------------------- Download complete --------------------"
echo ".......Renaming files"
mv $COUNTRY-latest.osm.pbf $COUNTRY.osm
echo ".......Processing maps"
./graphhopper.sh import $COUNTRY.osm
mv $COUNTRY.map ./$COUNTRY-gh/$COUNTRY.map
cd $COUNTRY-gh
sudo zip -r ./$COUNTRY.ghz ./* #zipping all files
mv $COUNTRY.ghz ../
cd ..
echo ".......File $COUNTRY-ghz is ready. Would you like to remove the other files? (Y or N)"
read DELETEOPTION
if [ $DELETEOPTION = "Y" ] || [ $DELETEOPTION = "y" ] ; then
    echo "Folder $COUNTRY-gh was deleted"
    sudo rm -rf $COUNTRY-gh
    sudo rm -rf $COUNTRY.osm
else
    echo "Folder $COUNTRY-gh was not deleted"
fi
echo "-------------------- Map processing complete --------------------"
```

Anexo J: Apresentação Final

Módulo de Navegação para a Aplicação Móvel do Drivian Tasks

Luís Henriques
a21260884

Estágio de Mestrado em Informática e Sistemas
2017 / 2018

Sentilant

2

Módulo de Navegação para a Aplicação Móvel do Drivian Tasks

- Objetivo
- Estado da Arte
- Levantamento de Requisitos e Elaboração de Mockups
- Levantamento de Alternativas de Implementação
- Desenvolvimento
- Testes
- Apresentação da Aplicação

3

Objetivo

4

Objetivo

Desenvolver um módulo de navegação para a aplicação móvel do Drivian Tasks que:

- Corresponda à qualidade dos sistemas de navegação atuais
- Expanda as funcionalidades de gestão de tarefas já presentes no Drivian Tasks

5

Estado da Arte

6

Estudo de Aplicações de Navegação

	GPS Navigation	GPS Tracker	Route Finder	Karta GPS	MotionX GPS Drive	MapQuest GPS	RoadMate On-the-Go	Waze	Google Maps	TomTom GPS	Sygic	INRIX Traffic	HERE WeGo	CoPilot GPS	ViaMichelin
Navegação 3D	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
ETAs e condições de trânsito em tempo real	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Participação e alertas da comunidade								✓		✓	✓	✓			✓
Recálculo de rotas baseado nas condições de trânsito	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Assistência de voz	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sugestão de escolha de vias	✓			✓			✓		✓	✓	✓			✓	
Pontos de interesse		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Localização de polícia, operações stop, câmaras de velocidade	✓			✓			✓	✓		✓	✓				
Mapas offline	✓			✓	✓		✓			✓	✓		✓	✓	
Estimativas de custos de viagem (combustível, portagens, ...)						✓					✓		✓		✓
Opções de rota (evitar auto-estradas, ...)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Liga ao painel do carro							✓	✓			✓				
Rotas para contactos da lista telefónica			✓	✓						✓					
Partilha de informação (ETA, posição, ...) com social media	✓		✓	✓	✓	✓	✓	✓		✓	✓	✓			
Apresentação de preços de combustível	✓					✓	✓	✓			✓				
Integração com calendário												✓			

7

Estudo de Aplicações de Navegação

- Apresenta previsões de chegada e condições de trânsito em tempo real
- Recalcula rotas baseado nas condições de trânsito
- Assiste o condutor através de indicações com voz
- Alerta se o condutor conduzir acima da velocidade permitida
- Apresenta pontos de interesse pertinentes
- Permite navegação em modo offline
- Disponibiliza opções de percurso (evitar autoestradas, caminhos locais, ...)
- Permite procurar parques de estacionamento próximos do destino
- Permite procurar pontos de abastecimento no percurso
- Disponibiliza uma opção para reportar imprevistos

8

Estudo de Aplicações de Apoio à Condução

1. Car Dashdroid
2. Drivemode
3. iOnRoad
4. Car Mode
5. Roadtrippers
6. TollSmart
7. Glympse
8. Taxmileage
9. Fuel Buddy
10. GasBuddy
11. Max Speed
12. MotionX GPS (iPhone)
13. Automatic Classic
14. Carcorder (iPhone)

9

Estudo de Aplicações de Apoio à Condução

- Permite desativar notificações durante a viagem
- Permite a leitura e envio de mensagens de texto com comandos de voz
- Memoriza do lugar de estacionamento
- Estima as despesas com combustível
- Permite aplicar filtros às estimativas de combustível como gasóleo, gás, gasolina
- Permite calcular as despesas com portagens desde o início ao fim da viagem
- Permite criar perfis de veículos, com impacto nas estimativas
- Em caso de acidente, faz automaticamente uma chamada de emergência

10

Levantamento de Requisitos e Elaboração de Mockups

11

Levantamento de Requisitos

Referência	Requisito	Prioridade
RF-AM-001	Conhecer o percurso para o dia	*Must*
RF-AM-002	Conduzir para o próximo destino	*Must*
RF-AM-003	Ver a lista de indicações do percurso atual	*Should*
RF-AM-004	Ouvir alerta para condução acima da velocidade permitida	*Should*
RF-AM-005	Selecionar percurso de volta	*Must*
RF-AM-006	Escolher opções de percurso	*Nice to Have*
RF-AM-007	Corrigir destino	*Should*
RF-AM-008	Ver informações sobre tarefas	*Must*
RF-AM-009	Ver opções de tarefa	*Must*
RF-AM-010	Adicionar complementos de tarefa	*Nice to Have*
RF-AM-011	Receber novas tarefas	*Must*
RF-AM-012	Fazer chamadas para o cliente atual	*Must*
RF-AM-013	Enviar mensagens automáticas para o cliente atual	*Should*
RF-AM-014	Fazer chamadas para a central	*Must*

12

Levantamento de Requisitos

Referência	Requisito	Prioridade
RF-AM-015	Receber sugestões de visita espontânea	*Should*
RF-AM-016	Telefonar a cliente de visita espontânea	*Should*
RF-AM-017	Pesquisar clientes	*Nice to Have*
RF-AM-018	Criar novas tarefas	*Should*
RF-AM-019	Incluir um ponto de abastecimento no percurso atual	*Nice to Have*
RF-AM-020	Ver estatísticas de condução	*Nice to Have*
RF-AM-021	Reportar imprevistos	*Nice To Have*
RF-AM-022	Procurar parques de estacionamento	*Nice to Have*

13

Elaboração de Mockups

14

Levantamento de Alternativas de Implementação

15

SDKs

	MapBox	Here	Navit	Nokia	Skobbler	SygicAut	Carto	MapQuest	MapCon	TALLYGO	Antavaglav
Cobertura em Portugal	✓	✓	✓	✓	✓	✓	?	✓	✓		? (Hungary Only?)
Rotas clientes de trânsito	✓	✓		✓	?	✓	✓	***	(Através de TransitLand)	✓	?
Navegação com voz em Portugal	✓ (Suporte para Amazon Polly)	✓		?	?	✓	?	?	?		
Navegação 3D	✓	✓	✓	✓	✓	✓	✓	✓	✓		?
Instruções de via (não apenas de estrada)	✓					✓					
Suporte para mapas offline	✓	✓	✓	✓	✓	✓	✓		✓**	✓	?
Opções de rota (evitar autoestradas, ...)		✓	✓	?	?	✓	?	✓	✓	✓	?
Custos de portagens		✓		?	?		?				?
Localização de radares	✓ (Previsto para finais de 2017)	✓		?	?	✓	?	?			?
Rotas para veículos pesados	?	✓		✓		✓		?		✓	
Opções para perfis de veículos pesados	?	✓		?		✓		?			
Android	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
iOS*	✓	✓		✓	✓	Entre7 a14 dias, dependendo do SDK	✓	✓	✓ (Swift)	✓ (Swift)	✓
Versão de avaliação	< 50.000 utilizadores mensais ativos	90 dias 100K transações mensais	-	?	Não disponível		30 dias	< 15.000 transações mensais	< 100 / 25.000 pedidos (dependendo do tipo de pedido)	20.000 viagens	?

APIs

	Google APIs	GraphHopper APIs	Openroute Service APIs	BingMaps API	Via Michelin	TomTom	Here	MapBox	Sygic Maps	MapQuest	RouteXL	Routific
Cobertura em Portugal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	?	✓
Rotas clientes de trânsito	✓	✓	?	✓	✓	✓	✓	✓	✓			
Opções de rota (evitar autoestradas, ...)	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Resposta com texto das indicações do percurso	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Calculo de custos de portagens		✓			✓		✓					
Localização de radares									✓			
Rotas para veículos pesados		✓	✓	✓	✓		✓		✓			
Opções para perfis de veículos pesados		✓	✓		✓		✓		✓			
Permite fazer pedidos de rota com vários pontos obrigatórios?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Disponibiliza mapas isolados				✓	✓	✓	✓	✓				
Permite pedir dados de trânsito isolados				✓		✓	✓	✓				

17

Projetos para Referência

1. AndRoad
2. Google Map Direction Turn by Turn
3. MyRoute
4. TurnByTurnNavigation
5. Turn-By-Turn-Navigation
6. Turn-by-turn-navigation
7. TurnByTurnNavi
8. PocketMaps
9. RouteOffline
10. OfflineMap
11. OfflineRouteMatcher
12. OfflineMapAndroidDemo
13. OfflineMap
14. Cunavigator
15. GarphHopper
16. Offline Routing Java
17. OSM Routing App
18. Osmand
19. OsmDroid
20. OfflineRoutingSample
21. Graphhopper
22. Mapsforge
23. Hellomap3d-android
24. Mobile-android-samples

18

Elaboração de Protótipos

1. TestGoogleMaps
2. TestBingMaps
3. TestHereMaps
4. TestMapBox
5. TestNavit
6. TestGraphhopper
7. TestMapsForge
8. TestTallyGo

19

Planos de Previsão de Implementação

Implementação com recurso a SDK: 486 horas

Elementos necessários
- SDK para Android

Vantagens
- Implementação simples e rápida
- Qualidade da navegação não é da responsabilidade da equipa de desenvolvimento

Desvantagens
- Preço compromete a escalabilidade da aplicação
- Impossível escolher a API de cálculo de rotas a utilizar
- Impossível gerir o número de pedidos feitos à API
- Impossível adicionar outros elementos de navegação não incluídos no SDK
- Qualidade da navegação não é da responsabilidade da equipa de desenvolvimento

20

Planos de Previsão de Implementação

Implementação de raiz: 746 horas

Elementos necessários
- API para cálculo de rotas
- Bibliotecas de mapas para Android
- Ficheiros de mapas
- Grafos para cálculo de rotas

Vantagens
- Controlo de custos
- Possibilidade de gerir o número de pedidos feitos à API
- Possibilidade de escolher a API de cálculo de rotas a utilizar
- Possibilidade de desenvolver um sistema que corresponda por completo a todas as necessidades
- Qualidade da navegação é da responsabilidade da equipa de desenvolvimento
- Desenvolvimento de um módulo proprietário e adaptável a outros projetos

Desvantagens
- Implementação complexa e demorada
- Qualidade da navegação é da responsabilidade da equipa de desenvolvimento

21

Desenvolvimento

22

Decisões Gerais

- Desenvolvimento exclusivo para Android
- Implementação de raiz
- Desenvolvimento ágil
- Desenvolvimento do módulo de navegação de forma isolada

23

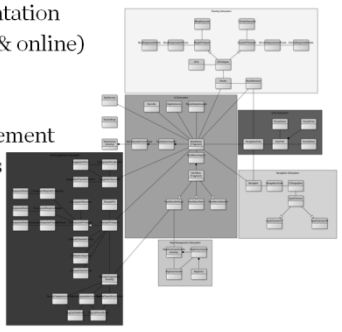
Ferramentas a Utilizar

- Android Studio
- Biblioteca de Mapas V™
- Mapas Mapsforge
- Bing Routing API
- Drivian Routing API
- Motor de Cálculo de Rotas Graphhopper
- Grafos para Cálculo de Rotas do OpenStreetMaps

24

Subsistemas

- UI & Map Presentation
- Routing (offline & online)
- Navigation
- Data
- Map File Management
- Visit Suggestions



25

Aspetos Relevantes

Algoritmo de acompanhamento de rotas
- Problema mais complexo do que simplesmente encontrar o ponto mais próximo na rota

Algoritmo para reajuste de rotas
- Define quando e de que forma uma rota deve ser reajustada e minimiza o número de pedidos feitos à API de cálculo de rotas

Algoritmo de classificação de sugestões de visita
- Para apresentar ao utilizador sugestões de visita por ordem de relevância

26

Aspetos Relevantes

Adaptadores para as diversas APIs
- Para poder alterar qualquer uma das APIs utilizadas

Sistema de apresentação e construção de instruções
- Constrói instruções a partir dos elementos da rota, caso não haja instruções provenientes da API
- Gere as instruções apresentadas ao utilizador, assegurando de que são sempre pertinentes

Script de preparação de mapas e grafos
- Para facilitar a disponibilização e atualização dos ficheiros de mapas e respetivos grafos

27

Testes

28

Testes

Total: 95 casos de teste

Algoritmo de acompanhamento de rotas
Domínio de Input
Critério: Base Choice Coverage
13 testes

Sistema de construção de rotas
Domínio e Input
Critério: All Combinations Coverage
21 testes

Algoritmo de decisão de ajuste de percurso
Domínio de Lógica
Critério: Combinatorial Coverage
12 testes

Sistema de Construção de Instruções
Domínio de Estrutura
Critério: Edge Coverage
8 testes
100% de cobertura (Importante para garantir qualidade e pertinência das instruções)

Sistema de construção de dados de percurso
Domínio de Input
Critério: All Combinations Coverage
17 testes

Algoritmo de Classificação de Sugestões de Visita
Domínio de Input
Critério: Base Choice Coverage
20 testes

UI
4 Sequências de teste
Sem critério

29

Questões

Luís Henriques
a21260884@alunos.isec.pt

Anexo K: Imagens do Módulo de Navegação