INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Aplicação Android de tracking e recomendação automática de rotas

TIAGO MARQUES OLIVEIRA
Julho de 2018

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Aplicação Android de *tracking* e recomendação automática de rotas

Tiago Marques Oliveira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientadora: Elsa Ferreira Gomes

Porto, Julho 2018

«Those who can imagine anything,
can create the impossible.»

Alan Turing

Resumo

Um problema recorrente que os condutores enfrentam diariamente nas suas viagens de carro é o processo de planeamento de rotas. Uma má escolha dos trajetos nos percursos do dia a dia leva ao aumento do tempo despendido em deslocações.

Este projeto pretende ajudar a solucionar este problema, oferecendo um método que consegue sugerir a melhor rota para o utilizador dependendo do dia da semana e hora atual, com base em previsões feitas a partir dos seus trajetos previamente efetuados e registados.

Na solução desenvolvida, o utilizador tem a possibilidade de gerir os seus locais, que servem como identificadores do início e fim dos trajetos. Pode também efetuar o *tracking* das suas viagens, onde é armazenado um conjunto de coordenadas geográficas e a informação do dia da semana e hora atuais. Estas viagens podem depois ser visualizadas no histórico de viagens do utilizador. Ao solicitar a melhor rota, o sistema utiliza todas as rotas previamente gravadas para um dado par origem-destino para gerar um modelo de regressão capaz de prever a duração entre cada segmento do trajeto. Estas previsões das durações são posteriormente utilizadas pelo algoritmo de *Dijkstra* para obter a melhor rota.

Numa fase final, são apresentados os testes que serviram para avaliar a solução. Nesta fase foram gravados diversos trajetos através da aplicação desenvolvida, em vários dias da semana. Com os dados recolhidos são apresentadas as previsões obtidas da duração de cada troço e o respetivo erro associado. No final, verificou-se o correto funcionamento da solução com um erro baixo nas previsões efetuadas.

Palavras-chave: Trajetos, *Data mining*, GPS, Previsão, Recomendação de rotas

Abstract

A recurring problem that drivers face daily in their car journeys is the route planning process. A poor choice of routes in day-to-day journeys leads to an increase in time spent travelling.

This project aims to help solve this problem by offering a method that can suggest the best route for the user depending on the day of the week and current time, based on forecasts performed from their previously made and registered routes.

The developed solution allows the user to manage their places, which serve as identifiers of the beginning and end of routes. He can also track his journeys, where a set of geographic coordinates is stored, with the information of the current day of the week and time. These routes can then be viewed in the user's travel history. When requesting the best route, the system uses all routes previously recorded for a given origin-destination pair to create a regression model capable to predict the duration between each segment of the route. The predicted durations are then used by the Dijkstra algorithm to obtain the best route.

In a final phase, it is presented the tests used to evaluate the solution. In this phase several routes were recorded through the application developed, on several days of the week. The data collected shows the forecasts obtained for the duration of each section and the associated error. In the end, the proper operation of the solution was verified with a low error in the predictions made.

Keywords: Routes, Data mining, GPS, Forecasting, Route Recommendation

Agradecimentos

Começo por agradecer a todos os docentes do Departamento de Engenharia Informática do Instituto Superior de Engenharia do Porto que conheci durante o meu percurso académico e que contribuíram para o meu crescimento pessoal e das minhas habilidades.

Este projeto não teria sido possível sem a ajuda da minha orientadora Elsa Ferreira Gomes. Gostaria de agradecer pelo seu apoio, paciência e disponibilidade ao longo deste processo que foi essencial para o sucesso deste projeto.

Agradeço aos meus pais pelo seu apoio incondicional e pela força e incentivo que me prestaram e à minha irmã que sempre foi uma inspiração e um exemplo de dedicação e disciplina.

Por último, agradeço aos meus amigos pela ajuda e motivação e por me apoiarem ao longo de toda a minha vida académica.

A todos, um sincero obrigado!

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

SQL	*Structured Query Language*
API	*Application Programming Interface*
PHP	*Hypertext Preprocessor*
NoSQL	*Not Only Structured Query Language*
REST	*Representational state transfer*
GPS	*Global Positioning System*
CSV	*Comma-separated values*
JSON	*Javascript Object Notation*
JDBC	*Java Database Connectivity*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
ETL	*Extract, transform, load*
RDBMS	*Relational database management system*
SPP	*Shortest Path Problem*
SO	Sistema operativo
NCD	*New Concept Development*
FEE	*Fuzzy Front End*
CPU	*Central Processing Unit*

1 Introdução

Neste capítulo é apresentado o projeto inerente a este relatório, explicando a motivação que levou à sua realização, os seus objetivos e as suas funcionalidades. É também exposto o enquadramento do trabalho e por fim é apresentada a estrutura do presente documento.

1.1 Contexto

O veículo a motor é considerado o mais usado em todo o mundo pelas suas vantagens de ser rápido, confortável e conveniente para as viagens do dia-a-dia, seja para deslocação para o emprego, visitar amigos e família ou levar os filhos à escola. O desenvolvimento da economia e da tecnologia tem contribuído para o aumento do número de transportes nas últimas décadas com mais de 78 milhões de carros vendidos no ano de 2017 (Global car sales 1990-2017, 2017).

Com o exponencial crescimento da população e com uns estimados 54.5 por cento da população mundial a viver em áreas urbanas, segundo uma publicação das Nações Unidas relativo ao ano de 2016 (Nations, 2016), o aumento do congestionamento nas cidades torna-se inevitável.

Figura 1 – Média de horas despendidas no congestionamento (INRIX 2016 Global Traffic Scorecard, 2016) (imagem recortada)

De acordo com o Traffic Scorecard publicado pela INRIX (INRIX 2016 Global Traffic Scorecard, 2016) relativo ao ano de 2016, que pode ser visto na Figura 1, Portugal ocupa o 34º lugar no ranking das cidades europeias que mais horas passam no congestionamento com uma média de 17 horas. Com isto surge a necessidade de encontrar soluções no mercado que combatam este problema.

Com a previsão do uso de smartphones da população mundial ultrapassar os 36 por cento até 2018 (Number of smartphone users worldwide, s.d.) e com a integração de navegação GPS ser comum em todos os dispositivos recentes torna-se possível o desenvolvimento de aplicações de sugestão de trajetos. Assim, uma outra solução mais personalizada seria conseguir recomendar rotas otimizadas dependendo do dia e da hora e com base num histórico de rotas armazenadas por utilizador através de uma aplicação móvel.

1.2 Problema

A intenção de melhorar o processo de sugestão e planeamento de rotas está na génese deste projeto. Surgiu da observação de um problema recorrente nos atuais meios de encontrar rotas otimizadas e com trânsito reduzido entre um local de origem e um destino para o condutor.

De acordo com um estudo feito pela *Jacobs media strategies* as informações de trânsito em que os condutores mais dependem continua a ser as transmitidas pela rádio. Após uma investigação mais detalhada descobriu-se que o uso de aplicações de trânsito, com especial destaque do Google Maps, torna-se cada vez mais popular neste setor nas gerações mais recentes com uma diferença de apenas 6% dos questionados a preferirem as informações de trânsito através do rádio (Jacobs, 2016).

Embora estas aplicações móveis disponibilizem informações de trânsito em tempo real e inúmeras opções para a escolha do melhor caminho para o condutor, desde a preferência de

autoestradas ou estradas nacionais até à escolha de um caminho sem portagens, este caminho nunca é baseado num histórico de trajetos previamente efetuados, o que tornaria a sugestão de trajetos mais baseado nas características de cada condutor.

Novas aplicações como o Waze já disponibilizam mecanismos para encontrar o melhor caminho baseado nos trajetos do condutor ao fim de algumas viagens, mas a escolha deste trajeto não tem em consideração fatores como a hora ou dia da semana em que as condições de trânsito e o tempo de duração do trajeto podem variar significativamente.

1.3 Objetivos

A origem deste projeto surgiu da intenção de melhorar o processo de sugestão de rotas de modo a evitar congestionamentos, com base nas rotas do utilizador. Foram estabelecidos como objetivos a criação de uma aplicação cliente-servidor que permita recolher informação referente aos percursos do condutor para ser usada numa posterior otimização dos seus percursos.

A aplicação móvel contém as seguintes funcionalidades:

- Permitir fazer o *tracking* dos trajetos do utilizador durante os seus percursos no dia-a-dia;

- Armazenar um conjunto de coordenadas geográficas que correspondem ao percurso efetuado pelo condutor;

- Enviar as coordenadas geográficas no final da viagem para um servidor remoto contendo a informação da data e hora da viagem;

- Apresentar uma lista de pares origem-destino previamente armazenados e recomendar o melhor trajeto do par selecionado apresentando-o num mapa;

- Visualizar o histórico de viagens.

A aplicação servidora terá as seguintes funcionalidades:

- Armazenar as coordenadas geográficas referentes aos trajetos do condutor;

- Agrupar as coordenadas geográficas recebidas com as coordenadas já existentes que se encontrem na mesma localização;

- Armazenar as viagens do utilizador;

- Devolver os pares origem-destino do utilizador;

- Calcular o trajeto otimizado tendo por base o histórico de viagens do condutor;

- Devolver a previsão de um trajeto otimizado entre o par origem-destino com base nos trajetos previamente guardados do condutor;

- Devolver o histórico de trajetos gravados do condutor.

1.4 Estrutura do Documento

O presente documento está dividido pelos seguintes capítulos:

Introdução: Neste capítulo é apresentado o que se encontra descrito na dissertação incluindo o problema abordado, os objetivos que se esperam atingir e a abordagem preconizada.

Abordagem seguida: Este capítulo descreve a abordagem que foi seguida no desenvolvimento deste projeto, são estudadas técnicas e algoritmos de *data mining* e problemas de otimização.

Estado da Arte: Aqui é descrito o estado atual de outras abordagens semelhantes, são avaliadas tecnologias para o desenvolvimento da solução e são estudados problemas pertinentes para o contexto do projeto.

***Design*:** Este capítulo descreve todo o *design* da solução, identificação e detalhe de requisitos, *design* arquitetural com a avaliação de diferentes alternativas, modelo de domínio e modelo de componentes.

Desenvolvimento e Implementação: Este capítulo documenta o trabalho elaborado durante o desenvolvimento e implementação das aplicações. Aqui são apresentadas as ferramentas usadas, detalhes de código, opções tomadas e por fim é explicado o processo de implantação da aplicação servidora.

Testes e Resultados: Aqui são descritos os procedimentos de teste aplicados e é descrita a avaliação feita à solução desenvolvida, documentando-se os resultados obtidos.

Conclusão: Neste capítulo é apresentada uma síntese do projeto, os objetivos atingidos e o possível trabalho futuro.

2 Abordagem seguida

Esta seção tem como objetivo expor, de forma sucinta, a abordagem ao problema, referindo as diferentes fases da construção deste projeto.

A primeira fase deste processo focou-se essencialmente na discussão e refinamento do problema.

A fase seguinte do projeto consistiu na pesquisa do estado da arte. Este estudo envolveu a investigação de técnicas de *data mining* e o seu método de aplicação, assim como os diferentes tipos de algoritmos relacionados. Foram também encontradas soluções e abordagens semelhantes que foram estudadas e comparadas entre si usando métodos como o AHP (*Analytic Hierarchy Process*) e o QFD (*Quality Function Development*). Ainda nesta fase foram também estudadas e comparadas ferramentas relevantes para serem utilizadas na resolução do problema.

Posteriormente foi feita uma análise detalhada ao problema, identificando-se os requisitos funcionais e não funcionais, o *design* da arquitetura com a avaliação de diferentes alternativas, e a construção de outros modelos relevantes para o *design* da solução.

Após concluída toda a análise e *design* do problema procedeu-se ao desenvolvimento e implementação da solução. Ao longo deste processo foram também realizados testes de forma a garantir o correto funcionamento de todas as funcionalidades das aplicações.

Por fim, a solução desenvolvida foi devidamente avaliada e foram analisados os resultados obtidos.

Na conclusão é apresentada uma síntese do trabalho desenvolvido, referindo-se os objetivos atingidos e o possível trabalho futuro.

2.1 *Data Mining*

Segundo Han et. Al, o crescimento exponencial do volume de dados disponível é "resultado da informatização da sociedade e do rápido desenvolvimento de ferramentas de coleção e armazenamento de dados" (Han, Kamber, & Pei, 2012). As práticas científicas e de engenharia geram continuamente enormes conjuntos de dados em experiências científicas, observações da engenharia, entre outros. As redes de comunicações transportam diariamente dezenas de *petabytes*, e na indústria de medicina e de saúde existe também uma grande geração de dados em registos médicos, monitorização de pacientes e imagens médicas (Han, Kamber, & Pei, 2012).

Assim, a necessidade de poderosas e versáteis ferramentas para descobrir informação útil entre a enorme quantidade de dados e a transformar em conhecimento organizado levou ao aparecimento do *data mining* (Gama, Carvalho, Faceli, Lorena, & Oliveira, 2015).

"*Data mining* é o processo de inferir conhecimento a partir de um grande conjunto de dados" (Gupta, 2015).

Existem diversos algoritmos de *data mining*, que podem ser usados dependendo dos dados e da solução que se pretende. Os tipos de algoritmos disponíveis são:

- Algoritmos de classificação: usados na previsão de uma ou mais variáveis discretas, com base nos outros atributos no conjunto de dados;

- Algoritmos de regressão: preveem uma ou mais variáveis contínuas, tal como lucro ou pedra, baseados em outros atributos no conjunto de dados;

- Algoritmos de segmentação: usados na divisão dos dados em grupos (*clusters*) de dados com propriedades semelhantes;

- Algoritmos de associação: encontram correlações entre os diferentes atributos num conjunto de dados (Data Mining Algorithms, s.d.).

No âmbito deste trabalho, foram apenas utilizados algoritmos de regressão como se explica na secção seguinte.

2.1.1 Regressão

Segundo Gupta, a "regressão é uma técnica de *data mining* usada para aplicar uma equação a um conjunto de dados" (Gupta, 2015). Equação é entendida como uma igualdade envolvendo uma ou mais variáveis.

No contexto deste projeto, a regressão irá ser usada para prever os melhores trajetos para o utilizador. Assim, serão de seguida referidos os algoritmos de regressão linear, as árvores de

regressão, as máquinas de vetores de suporte (SVMs), as florestas aleatórias (*Random Forests*) e por fim o M5P.

A regressão linear é um modelo global, onde existe apenas uma fórmula preditiva aplicada a todo o conjunto de dados. Assim, quando os dados contêm muitas características com interações complexas e não-lineares, a construção de um modelo global único torna-se uma tarefa bastante difícil (Shalizi, 2006).

Uma alternativa a esta abordagem são as árvores de regressão, que consistem na subdivisão do espaço em regiões de menor dimensão, com uma mais fácil gestão das interações. Este espaço pode ser continuamente particionado até se obter um espaço em que é possível associar modelos simples, sendo este processo chamado de partição recursiva (Shalizi, 2006).

As árvores são usadas para representar a partição recursiva e cada uma das folhas (nós terminais) representa uma célula da partição e tem anexado um modelo simples, aplicado apenas a essa célula. Os nós interiores estão rotulados com perguntas e os ramos com as respostas às questões anteriores. Assim, um ponto x pertence a uma folha caso corresponda à célula da partição. Para descobrir essa célula é feita uma sequência de perguntas sobre as características, começando no nó da raiz (Shalizi, 2006).

A máquina de vetor de suporte (*Support Vector Machine*) é um algoritmo de aprendizagem supervisionada de *machine learning* que pode ser aplicado tanto na classificação como na regressão de dados.

A ideia base deste modelo baseia-se na descoberta de um hiperplano, uma linha que separa e classifica um conjunto de dados, que melhor divide esses dados em duas classes. Neste contexto, os vetores de suporte são os pontos mais próximos do hiperplano, críticos para o conjunto de dados. Embora as máquinas de vetor de suporte possuam uma alta precisão, a sua aplicação não é adequada em conjuntos de dados de grandes dimensões (Bambrick, 2016).

O M5P apresenta uma reconstrução do algoritmo M5 para gerar uma árvore de modelos de regressão a partir de dados empíricos. Neste modelo, em cada ramo a árvore armazena o modelo de regressão linear que prevê os valores da porção do *dataset* que alcança essa folha.

O conjunto de dados é divido em diferentes porções de acordo com certos atributos dos dados. O desvio padrão é geralmente usado como um critério que determina qual é o melhor atributo a dividir os dados em cada nó. Aqui o atributo usado é o que exibe a maior expectativa na redução do erro. O processo termina quando apenas algumas instâncias permanecem. A árvore será então podada de volta e um processo de suavização será realizado no final para compensar descontinuidades acentuadas entre os modelos lineares adjacentes (Quinlan, 2006).

2.2 Otimização

O contexto do problema que deu origem a este projeto tem associado um problema de otimização, onde é necessário identificar formas de otimizar as viagens do dia-a-dia do condutor. Assim, tornou-se importante efetuar um estudo de problemas já existentes nesta área.

2.2.1 Problema do caminho mais curto (*Shortest-path problem*)

O planeamento de rotas em redes de transporte é um dos tópicos mais pesquisados na área de engenharia de algoritmos que traz bastantes vantagens em aplicações do mundo real como sistemas de navegação como o Google Maps (Popa & Popescu, 2016).

O algoritmo do caminho mais curto é normalmente associado a esse planeamento de rotas pois tem um grande número de aplicações e tem sido "amplamente aplicado na computação de endereçamento da Internet, sistemas de transporte inteligentes, sistemas de informação geográfica urbana e sistemas de informação geográfica militar" (Yang, Liu, & Lin, 2014).

O problema do caminho mais curto está normalmente associado à teoria de grafos e tem inúmeros algoritmos associados para a sua resolução. Entre estes algoritmos, Dijkstra (Dijkstra's algorithm, s.d.) é geralmente considerado o melhor, sendo que muitos investigadores baseiam as suas otimizações neste algoritmo (Zhan & Noon, 1998).

O algoritmo de Dijkstra, concebido pelo cientista de computadores Edsger Dijkstra em 1956 (Misa, 2010), é um algoritmo que resolve o problema do caminho mais curto com arestas de peso não negativo. Para um dado nó inicial de um grafo, este algoritmo encontra o caminho mais curto entre esse nó e todos os outros embora possa ser usado como algoritmo de caminho mais curto de fonte única *(*SSSP) ao parar a sua execução uma vez encontrado o caminho mais curto até ao nó de destino determinado (Dijkstra's Algorithm for Single Source Shortest Paths in, 2016). Em termos da notação *Big-O* (Massachusetts Institute of Technology), a complexidade deste algoritmo é de $O(n^2)$ o que pode ser considerado eficiente em comparação com outros algoritmos semelhantes (Popa & Popescu, 2016). De seguida vai ser apresentada uma breve explicação do funcionamento do algoritmo de Dijkstra.

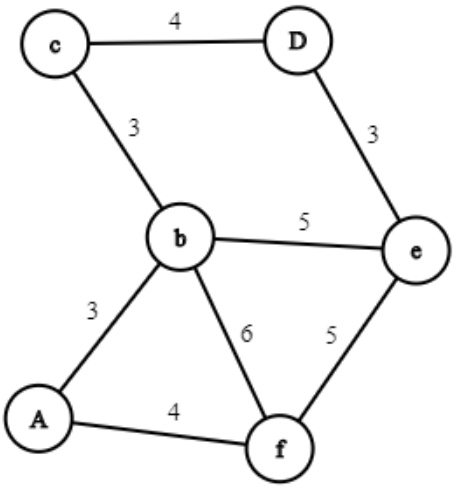

Figura 2 – Grafo exemplificativo

O grafo é representado por vértices ou nós, denotado no algoritmo por v ou u e arestas com um peso associado que conecta dois nós sendo o peso de um determinado nó denotado por $w(u, v)$. Na Figura 2 está presente um grafo exemplificativo onde é possível observar os vários nós interligados por arestas com um peso associado a cada uma. Ao comparar o peso de cada aresta, neste modelo, e considerando um nó inicial *A* e um nó final *D*, o algoritmo de Dijkstra encontraria como o caminho de custo mínimo *A-b-c-D*.

No início do algoritmo são inicializados três valores:

- Um *array* de distâncias ($dist$) do nó inicial s até todos os restantes nós, inicializadas a 0;

- Uma *queue* (Q) de todos os nós do grafo;

- Um *array* vazio (S) para indicação dos nós já visitados pelo algoritmo.

Após inicializadas as variáveis necessárias o algoritmo executa os seguintes passos:

1. Enquanto Q não estiver vazio, é retirado um nó v com a menor $dist(v)$ e que ainda não esteja presente no *array* S. No primeiro ciclo de execução do algoritmo o nó inicial é escolhido neste passo.

2. O nó v é adicionado a S indicando que o nó em questão já foi visitado;

3. Os valores da lista $dist$ relativos aos valores dos nós adjacentes do nó atual v são atualizados com novos valores caso seja encontrada uma distância menor da já existente com a condição $dist(v) + weight(u, v) < dist(u)$;

A execução do algoritmo de Dijkstra termina quando todos os nós do grafo forem visitados e for encontrada a menor distância entre cada nó onde $dist$ irá conter a árvore dos caminhos de custo mínimo desde o dado nó inicial (Abiy, Pang, Khim, Ross, & Williams, s.d.).

3 Estado da Arte

Este capítulo está dividido em várias secções. Na secção 3.1 é apresentado o desenvolvimento do módulo de Análise de Valor. Na secção 3.2 é apresentado o estudo e comparações feitas às soluções e abordagens semelhantes já existentes no mercado. Na secção 3.3 são investigadas as APIs disponibilizadas pela Google e por fim, na secção 3.4 são apresentadas as ferramentas relevantes para o projeto que foram investigadas de modo a escolher a que mais se adequa ao projeto.

3.1 Análise de Valor

A constante adoção de práticas de inovação traz vantagens competitivas, aumentando a possível participação de mercado e maior lucratividade, mas é uma tarefa complexa que necessita de um longo processo de planeamento.

O desenvolvimento de novos produtos contém um elevado risco económico para uma organização pois consome recursos e existe a possibilidade de o produto não ser bem-sucedido no mercado.

Com o objetivo de minimizar estes riscos são utilizadas técnicas para a melhoria do processo de inovação.

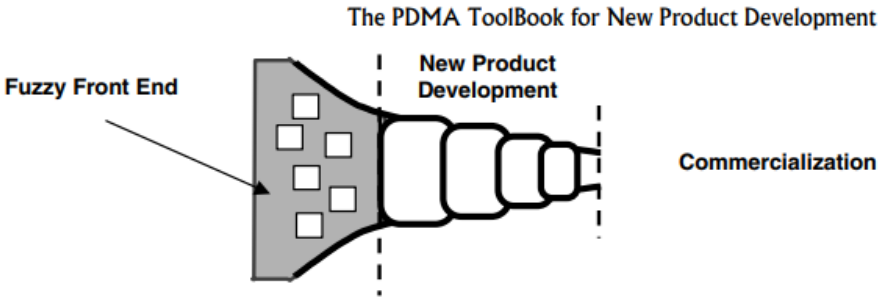

Figura 3 – Processo de inovação dividido em três partes (Koen, et al.)

Na Figura 3 está presente o modelo de inovação que pode ser dividido em três partes:

Fuzzy Front End **(FFE)** – fase onde ocorre a identificação de ideias e desenvolvimento de conceitos;

New Product Development **(NPD)** – esta fase abrange todo o processo de desenvolvimento do novo produto;

Comercialização – fase final onde o produto é introduzido no mercado e é feita a sua propaganda.

"A primeira parte – o FFE – é geralmente considerada como uma das maiores oportunidades para a melhoria do processo geral de inovação" (Cooper, 1993).

Como o FFE não possui termos e definições para os seus elementos chave irá ser usada a notação e terminologia do modelo *New Concept Development* (NCD) (Koen, et al.).

O modelo NCD divide a fase de inovação em três partes:

- O motor;
- Os elementos do FFE;
- Os fatores ambientais externos;

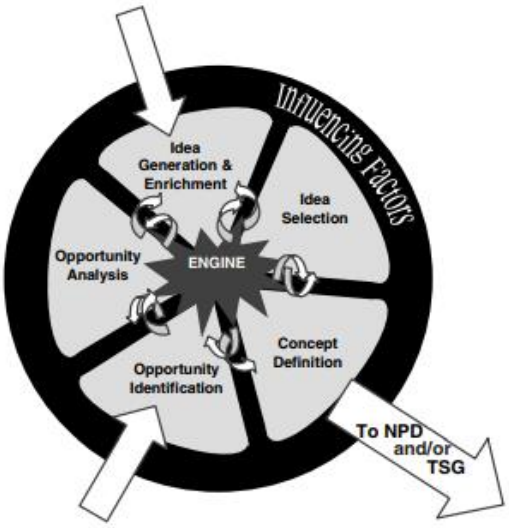

Figura 4 – Modelo NCD (Koen, et al.)

Estes componentes, que podem ser observados na Figura 4, vão ser de seguida detalhados.

3.1.1 Motor

O motor do modelo consiste na liderança, cultura e estratégia de negócio que controla os cinco elementos principais controlados pela organização.

3.1.2 Elementos do FFE

Existem cinco elementos identificados no modelo NCD que são os seguintes:

- A identificação da oportunidade;

- A análise da oportunidade;

- A geração e enriquecimento de ideias;

- A seleção de ideias;

- A definição do conceito.

Identificação da oportunidade

Para a criação deste projeto foram necessárias perceber as tendências do mercado atual de modo a ser possível chegar a uma solução inovadora.

O desenvolvimento desta aplicação surge da intenção de inovar o processo de sugestão e planeamento de rotas.

Após uma avaliação do mercado verificou-se que 56% dos condutores continuam a depender das informações de trânsito transmitidas pelas emissões da rádio. Embora este seja um número significativo, após um estudo mais detalhado observou-se que nas gerações mais recentes o uso de aplicações mobile para a receção destas informações de trânsito continua a ser cada vez mais popular (Jacobs, 2016).

Foi também verificado que o uso de *smartphones* pela população mundial está previsto atingir 36 por cento até 2018 (Number of smartphone users worldwide, s.d.), sendo que a integração de navegação GPS é já comum nos mais recentes dispositivos, tornando-se assim possível o desenvolvimento de aplicação para a sugestão de trajetos.

Análise da Oportunidade

Com o número crescente de condutores a usar os *smartphones* para obter informações de trânsito foi feita uma análise ao mercado para perceber quais as aplicações que mais se destacam nesta área como potenciais competidores, verificar necessidades dos clientes e determinar o enquadramento das capacidades e fraquezas da aplicação a ser desenvolvida no mercado.

De modo a sustentar a análise da oportunidade feita consultaram-se o número de transferências das aplicações de *tracking* e sugestão de rotas mais populares, identificando-se o Waze (Waze - GPS, Maps, Traffic Alerts & Live Navigation, s.d.) com mais de 7 milhões de transferências efetuadas na loja *Android* e o Google Maps (Maps - Navigation & Transit, s.d.) com perto de 9 milhões de transferências.

Posteriormente foi feito um estudo da utilização de cada aplicação para perceber vantagens e desvantagens de cada uma. Embora estas aplicações apresentem informações de trânsito em tempo real e múltiplas opções para uma escolha personalizada do melhor caminho, o trajeto sugerido nunca está enquadrado com as rotas previamente feitas pelo condutor. O Waze tenta combater este problema ao guardar os trajetos entre uma origem e um destino do condutor, conseguindo assim encontrar o melhor caminho, mas não existe neste caso a consideração de fatores como a hora ou dia da semana em que as condições de trânsito podem variar e com isto o tempo do trajeto.

Verificou-se assim uma margem para o desenvolvimento de uma nova solução para abordar o problema em questão.

Geração de Ideias e o seu Enriquecimento

Após a análise do problema identificou-se a necessidade de abordar aspetos importantes da aplicação. Numa primeira fase foi debatida a estrutura da aplicação e como seria feito o *tracking* dos trajetos do condutor. Posteriormente foram também abordadas questão mais técnicas de como seriam persistidos os trajetos do condutor e em que fase da execução da aplicação e como seria alcançada a melhor rota.

Seleção das Melhores Ideias

Ficou definido o desenvolvimento de uma aplicação com uma estrutura cliente/servidor onde o *tracking* seria feito através do GPS do dispositivo e com o auxílio de tecnologias Google. Os trajetos do condutor irão ser persistidos numa base de dados remota através da comunicação com o servidor, sendo que a persistência dos dados só deverá ocorrer no final de cada viagem. Por fim, de modo a alcançar a melhor rota será simulado um problema de grafos onde será aplicado o algoritmo de Dijkstra para descobrir o melhor caminho.

Definição de Conceito

Com a aplicação de *tracking* e sugestão automática de rotas pretende-se que os condutores possam guardar um histórico de viagens no seu dia-a-dia e possam posteriormente pedir sugestões da melhor rota baseada nesses mesmos trajetos.

Valor

Uma proposta de valor bem estruturada permite identificar facilmente e inequivocamente o que se irá fornecer ao cliente, devendo ser associado a benefícios.

A solução proposta pela aplicação de *tracking* e sugestão de rotas aos clientes é simples e objetiva, apresentando uma alternativa mais personalizada e baseada nas características de cada condutor no planeamento de rotas do dia-a-dia. Com esta aplicação os clientes conseguem saber o tempo exato de cada trajeto entre uma origem e um destino, sendo possível sugerir qual o trajeto ideal baseado em fatores como o dia e a hora, sendo este o principal benefício da solução.

Valor para o Cliente

O valor para o cliente pode ser visto como a perceção da vantagem que se obtém entre os benefícios e os sacrifícios de uma determinada oferta de valor.

Esta solução apresenta-se como uma alternativa de grande valor para o mercado, no âmbito das aplicações de *tracking*, sugestão e planeamento de rotas. Embora seja necessário um esforço inicial por parte dos clientes numa primeira fase do uso da aplicação, os benefícios tornam-se evidentes com o uso a longo prazo, quando o número de rotas guardadas em histórico torna-se significativo.

Na Tabela 1 efetua-se uma comparação entre os benefícios e sacrifícios prestados pela aplicação e também os benefícios da relação comercial entre os clientes e o projeto.

Tabela 1 - Benefícios e Sacrifícios

Domínio/Âmbito	Serviço	Relacionamento
Benefícios	Pesquisa de origens e destinos inteligente *Tracking* automático de trajetos Sugestão das melhores rotas Usabilidade Fiabilidade	
Sacrifícios	Necessidade de efetuar *tracking* numa fase inicial para ser posteriormente possível sugerir trajetos	

Valor Percecionado

Segundo Zeithmal, "o valor percecionado é a avaliação geral do cliente da utilidade de um produto com base nas perceções do que é recebido" (Zeithaml, 1988).

A solução apresenta-se como uma alternativa aos atuais métodos usados para planeamento e sugestão de rotas apresentando um modo de funcionamento inovador, baseado nas características de cada condutor. Com o uso a longo prazo desta aplicação, o tempo que os clientes despendem em viagem pode ser significativamente reduzido no dia-a-dia.

Proposta de Valor

A aplicação de *tracking* e sugestão de rotas visa inovar os atuais processos usados pelos condutores no planeamento de trajetos entre o seu local atual e um destino. Isto é conseguido através da criação de um histórico de viagens ao longo do tempo, com a informação adicional do dia e hora da viagem, de modo a ser possível posteriormente encontrar o melhor caminho dependendo destes fatores. Esta solução difere das atuais por usar um histórico de rotas do condutor, tornando assim a sugestão de rotas mais eficaz baseada nas características de cada condutor.

3.1.3 QFD (Quality Function Deployment)

O QFD é um sistema usado para "planear um produto ou serviço baseado nas exigências do cliente que envolve todos os membros do fabricante ou da organização" (Warwick Manufacturing Group, 2007).

O princípio deste sistema consiste em analisar os requisitos propostos pelo cliente e dividi-los em segmentos para ser possível identificar os meios de atingir os objetivos de cada um deles.

O processo do QFD é o seguinte:

- Requisitos do cliente

- Requisitos de *design*

- Características dos componentes

- Requisitos das operações

- Procedimentos do trabalho

De modo a que uma organização consiga vender os seus produtos é necessário compreender as necessidades do cliente para ser possível cumprir os seus requisitos. Assim, a coleção e uso dos requisitos do cliente é a base do QFD (Warwick Manufacturing Group, 2007).

O primeiro passo consiste em definir a cadeia de clientes que será o público-alvo do serviço ou produto que vai ser criado. Com o objetivo de abranger o máximo número de clientes são também considerados as seguintes pessoas:

- As que compram os produtos atuais da organização;

- As que compram os produtos das organizações concorrentes;

- As que mudaram para a organização concorrente;

- As que estão satisfeitas;

- As que estão insatisfeitas;

O passo seguinte consiste em definir os requisitos recolhidos dos clientes que devem estar na própria linguagem do cliente.

De seguida são feitos questionários ao cliente para se obter a importância de cada requisito. Esta informação é depois inserida no gráfico QFD e pode ser usada para efeitos de melhorar rapidamente as áreas de maior interesse para o cliente.

Depois dos requisitos do cliente estarem definidos com a respetiva importância associada é necessário determinar as características de engenharia que devem ser otimizadas para assegurar a satisfação do cliente (Warwick Manufacturing Group, 2007).

“Estas características devem descrever o produto em termos mensuráveis e deve afetar diretamente as perceções do cliente” (Warwick Manufacturing Group, 2007).

O passo subsequente é avaliar como é o desempenho da organização em relação à competição direta. Esta comparação deve ter como base as características de engenharia já identificadas de modo a atribuir um ranking competitivo a esses pontos específicos (Warwick Manufacturing Group, 2007).

Após definidos os requisitos do cliente e as características de engenharia são de seguida preenchidas as matrizes de relações e de conflitos. A ideia por trás do preenchimento destas matrizes consiste em realçar as relações entre os requisitos do cliente e as características de engenharia, e as associações de suporte/conflito entre as características de engenharia

Na matriz das relações podem ser usados três símbolos para distinguir os vários níveis de força na associação entre os componentes.

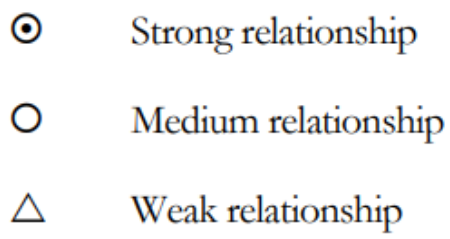

Figura 5 – Símbolos da matriz das relações

Como pode ser visto na Figura 5, o primeiro símbolo representa uma forte relação entre os dois componentes, o segundo uma relação mediana e por fim o último representa uma relação fraca.

Na matriz de conflitos triangular que se encontra no topo do sistema QFD podem ser usados quatro símbolos para descrever as relações entre as características de engenharia.

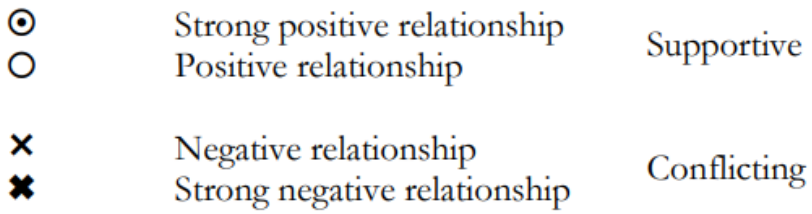

Figura 6 – Símbolos da matriz de conflitos

A Figura 6 apresenta os símbolos usados no sistema QFD, onde os dois primeiros representam uma relação de suporte fortemente positiva e positiva respetivamente e os dois símbolos seguintes servem para representar relações de conflito negativas e fortemente negativas respetivamente.

Uma vez estabelecidas as relações o próximo passo consiste em determinar os valores alvo das características de engenharia, uma estimativa da dificuldade para atingir esse valor e um nível de importância para as características de engenharia (Warwick Manufacturing Group, 2007).

Os valores alvo das características de engenharia, ao contrário das especificações de *design* apenas refletem o que é necessário para alcançar a satisfação do cliente e não têm relação com o que pode atualmente ser realmente atingido.

A estimativa da dificuldade feita de seguida pode ter um grande impacto na fase de análise e variam muito entre sistemas QFD porque cada estudo representa uma situação única.

O nível de importância das características de engenharia representam uma combinação entre a importância dos requisitos para o cliente e a força entre as relações entre os requisitos do cliente e as características de engenharia.

No contexto deste projeto a aplicação do QFD é feita para avaliar os requisitos do cliente e as características de engenharia, efetuando-se também uma comparação das abordagens existentes no mercado com a solução proposta.

3.2 Soluções e abordagens semelhantes

Nesta secção, descrevem-se algumas soluções e abordagens semelhantes já presentes no mercado. Posteriormente, são comparadas as características de cada aplicação através do método AHP e do sistema QFD.

3.2.1 Google Maps

Google Maps[1] é uma aplicação que disponibiliza navegação GPS em tempo real, trânsito e detalhes sobre diversas localizações. O reencaminhamento automático de trajetos com base na monitorização de trânsito em tempo real, obras e acidentes rodoviários permite encontrar o melhor trajeto para um determinado destino. O Google Maps permite também o uso dos mapas sem ligação à internet para obter direções e usar a navegação.

Esta aplicação é grátis e encontra-se disponível para iOS e Android.

3.2.2 Waze

Waze[2] é uma aplicação quer disponibiliza navegação GPS com atualizações em tempo real de trânsito. Estas atualizações em tempo real são efetuadas pelos próprios utilizadores que podem inserir informações desde trânsito, operações stop, obras, entre outros e que ficam visíveis para os restantes utilizadores.

Esta aplicação recolhe também informações sobre as viagens entre casa e trabalho do condutor e permite ao fim de três a quatro viagens prever qual o trajeto mais rápido.

[1] https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=en
[2] https://www.waze.com/

O uso do Waze consiste em inserir o endereço de destino e efetuar a viagem com a aplicação aberta no dispositivo móvel para contribuir passivamente para obtenção de informações de trânsito.

O Waze tem também integração com redes sociais como o Foursquare, Twitter e o Facebook que trazem funcionalidades adicionais à aplicação quer permitem ao condutor ver os seus amigos no mapa, partilhar os seus trajetos e saber quem está perto da sua localização.

Esta aplicação é grátis e está disponível para iOS e Android com versões superiores a 4.1 (Jelly Bean).

3.2.3 MapFactor GPS Navigation Maps

MapFactor Navigator[3] é uma aplicação de navegação GPS que utiliza mapas *offline* e que não necessita de uma ligação à internet. Inclui funcionalidades como navegação por voz em vários idiomas, planeamento de trajetos, pontos de interesse e apresenta indicações sobre a velocidade máxima no local do condutor.

Esta aplicação é grátis e está disponível para iOS e Android com versões superiores a 2.3 (Gingerbread).

3.2.4 Comparação das soluções

Com o objetivo de comparar as aplicações estudadas com a solução proposta vai ser usado o método AHP (**Erro! A origem da referência não foi encontrada.**) e será posteriormente apresentado o sistema QFD (3.1.3).

3.2.4.1 AHP

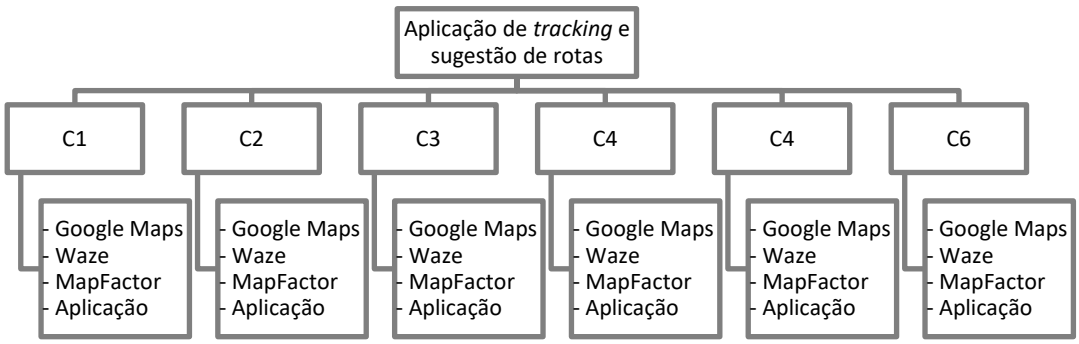

Figura 7 – Estrutura hierárquica

[3] http://navigatorfree.mapfactor.com/en/download/

Na aplicação deste método, o primeiro passo consiste em representar o problema através de uma estrutura hierárquica, apresentada na Figura 7, onde é apresentado o enunciado do objetivo geral de decisão, os critérios associados ao problema de decisão e as alternativas disponíveis e mais adequadas que serão neste caso as soluções semelhantes encontradas e a solução proposta. Os seis critérios apresentados no diagrama são os seguintes:

- C1: Suporte visual do mapa;

- C2: Qualidade da pesquisa de origens/destinos;

- C3: Tempo de execução a encontrar rotas;

- C4: Rotas baseadas no histórico de viagens do utilizador;

- C6: Facilidade de uso.

A segunda fase do método foi comparar os critérios da estrutura hierárquica criando-se assim uma matriz de comparação que pode ser observada na tabela 2.

Tabela 2 – Comparação dos pesos dos critérios

	C1	**C2**	**C3**	**C4**	**C5**	**C6**	**Prioridade relativa**
C1	1	4	2	½	½	2	0,18629
C2	¼	1	⅓	⅓	¼	2	0,069692
C3	½	3	1	½	2	3	0,181582
C4	2	3	2	1	3	4	0,311821
C5	2	4	½	⅓	1	3	0,185768
C6	⅓	½	⅓	¼	⅓	1	0,060959

Após a construção da tabela, foi efetuada a normalização dos seus valores de modo a ser possível obter o vetor de prioridades, presente na última coluna da tabela. Este vetor tem a função de identificar a ordem de importância de cada critério. Neste caso o critério referente à precisão das rotas é o mais importante, seguido do critério das rotas baseadas num histórico do utilizador, no suporte visual do mapa na aplicação e no tempo de execução a encontrar rotas. Os últimos dois critérios têm aproximadamente a mesma prioridade sendo os de menor importância.

Na fase seguinte foi avaliada a consistência das prioridades relativas, ao calcular a Razão de Consistência (RC), atingindo-se um valor 0.07112 que sendo menor que 0.1 confirma-se a consistência dos valores.

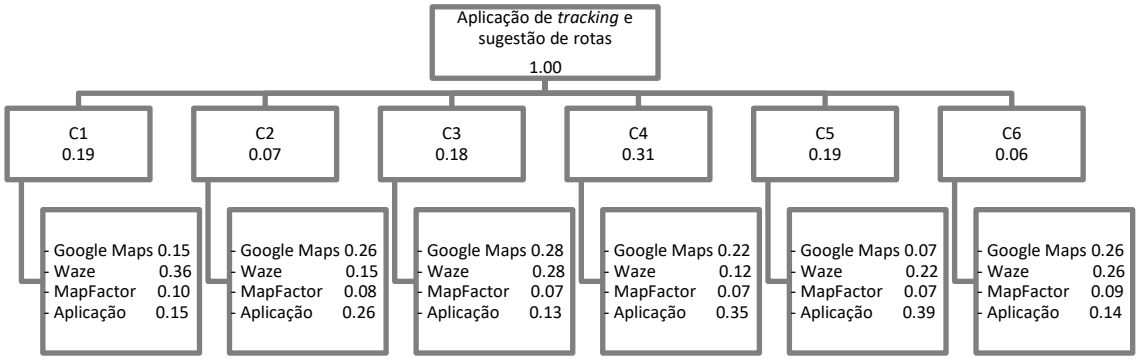

Figura 8 – Comparação paritária para cada critério

De modo a determinar a importância relativa de cada uma das alternativas do problema em questão foi construída uma matriz de comparação paritária para cada critério, calculando também o vetor de prioridade para cada um, atingindo-se os resultados que podem ser observados na Figura 8.

Por fim, na última etapa do método AHP e de modo a obter as prioridades compostas das alternativas, multiplicou-se os valores anteriormente obtidos pelas prioridades relativas.

Como a precisão dos trajetos e as rotas baseadas no histórico de viagens do utilizador são os critérios com maior peso, e a aplicação a ser desenvolvida foca-se bastante nestas funcionalidades, verificou-se que a solução proposta seria a mais adequada atingindo um valor de 0.26. De seguida ficou o Waze com 0.22, dado que esta aplicação também apresenta mecanismos baseados no histórico, e por fim o Google Maps e o MapFactor com 0.19 e 0.07 respetivamente.

3.2.4.2 QFD

No contexto da aplicação de *tracking* e sugestão de caminhos do presente relatório foram identificados os seguintes requisitos:

- Apresentação de trajetos no mapa;

- Procura de origens e destinos inteligente;

- Rápida capacidade de resposta;

- Precisão dos trajetos sugeridos;

- Baixo gasto de bateria;

- Interface gráfica de fácil uso.

Durante a fase de identificação das características de engenharia foram identificadas as seguintes:

- Taxa de erro de integridade dos trajetos guardados;

- Eficácia na otimização da rota;

- Taxa de erros nas rotas sugeridas;

- Intervalo entre coordenadas guardadas;

- Consumo de energia;

- Desempenho da aplicação;

- Interface gráfica intuitiva.

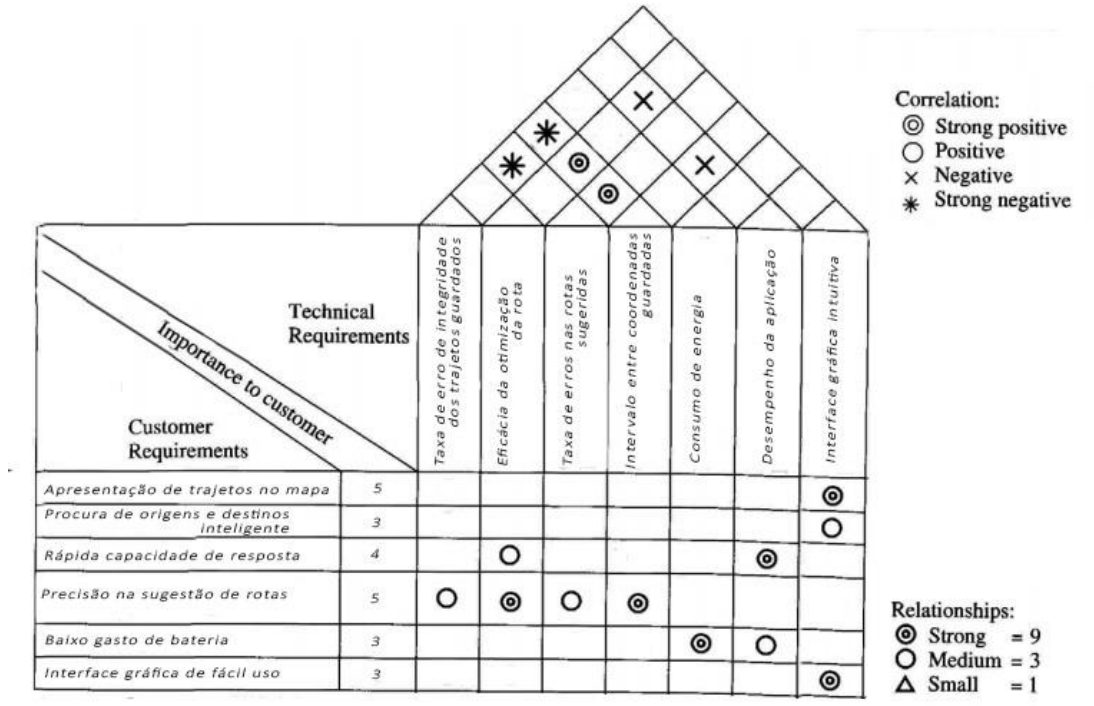

Figura 9 – Matriz de relações conflitos

Na Figura 9 podem ser observadas as relações entre os requisitos e as características de engenharia da solução, como por exemplo a apresentação de trajetos no mapa e a procura de locais inteligente que tem uma relação forte e média respetivamente com a intuitividade da

interface gráfica e a precisão na sugestão de rotas que está relacionado principalmente com a eficácia da otimização da rota e com o intervalo entre coordenadas guardadas.

Em relação à matriz de conflitos é possível ver por exemplo uma relação bastante positiva entre a eficácia da otimização da rota e o intervalo de coordenadas guardadas visto que um maior número de coordenadas guardadas numa viagem melhorará a eficácia na procura da melhor rota.

De seguida é apresentada, na Figura 10, o resultado final do sistema QFD do contexto deste projeto. Aqui já é observável a comparação dos requisitos do cliente e das características de engenharia da solução proposta e de duas abordagens semelhantes (Google Maps e Waze).

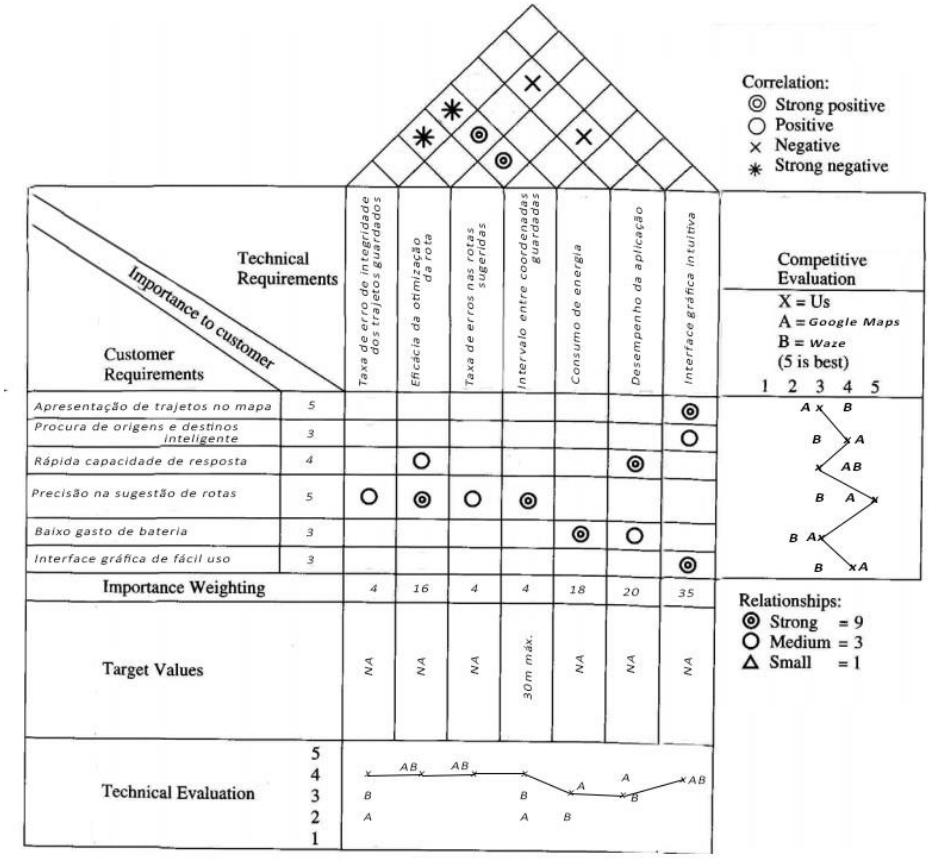

Figura 10 – QFD

3.3 Google APIs

Nesta secção, referem-se algumas das APIs disponibilizadas pela Google que se inserem no âmbito do trabalho descrito.

3.3.1 Google Maps

A Google Maps Android API (Google Maps Android API, s.d.) possibilita a adição de mapas a aplicações Android. Esta API gere automaticamente o acesso aos servidores do Google Maps, *download* de dados, a exibição do mapa na aplicação e a resposta à interação dos utilizadores com o mapa. Existem também chamadas à API que permitem modificar a vista do utilizador de um mapa em particular e ainda adicionar os seguintes gráficos:

- Ícones em posições específicas no mapa

- Conjuntos de segmentos de reta

- Segmentos fechados

- Gráficos de *bitmap* ancorados em posições específicas no mapa

- Conjunto de imagens que podem ser sobrepostas ao mapa base.

3.3.2 Google Maps Roads

A Google Maps Roads API (Introduction to the Google Maps Roads API, s.d.) fornece serviços que permitem identificar as estradas onde o condutor está a viajar e apresenta também informação adicional sobre esse local, tal como limites de velocidade. Os serviços disponibilizados são os seguintes:

- Mover para as estradas: este serviço retorna a geometria da estrada de melhor ajuste para um dado conjunto de coordenadas GPS. A chamada ao serviço tem um limite de cem pontos de GPS recolhidos ao longo de um percurso e retorna um conjunto de dados semelhantes, mas com os pontos colocados nas estradas mais prováveis onde o veículo viajava. Fornece ainda a opção de se obter a resposta com os dados interpolados o que resulta num caminho que segue a geometria da estrada mais suavemente.

- Estradas mais próximas: Este serviço retorna segmentos de estradas individuais para um determinado conjunto de coordenadas GPS que não precisam de fazer parte de um caminho contínuo.

- Limites de velocidade: Este serviço retorna o limite de velocidade de um determinado segmento de estrada. Este serviço encontra-se disponível apenas para clientes do Plano Premium da API do Google Maps.

3.3.3 Google Maps Directions

A Google Maps Directions API (Google Maps Directions API, s.d.) consiste num serviço que calcula as direções entre locais mediante o meio de transporte. Este serviço permite também

especificar um conjunto de waypoints para calcular as rotas através de locais adicionais sendo que neste caso o trajeto retornado inclui escalas em cada um dos waypoints fornecidos. Por fim, esta API permite ainda prever o tráfego do trajeto devolvido com base em médias anteriormente registadas.

3.3.4 Google Places

A Google Places API for Android (Google Places API, s.d.) disponibiliza serviços que retornam a mesma informação usada pelo Google Maps. Estes serviços podem ser usado nas aplicações usando os seguintes componentes:

- Selecionador de região: este *widget* apresenta um mapa interativo e uma lista de locais próximos, endereços geográficos e empresas locais. Os utilizadores podem assim selecionar uma região e irão ser apresentados os detalhes do local selecionado.

- Preenchimento automático: este serviço apresenta previsões de locais em resposta às pesquisas do utilizador. Enquanto o utilizador digita, o serviço retorna sugestões de locais tais como empresas, endereços e pontos de interesse.

- Detalhes do local: este serviço fornece informações detalhadas sobre os locais introduzidos tais como o nome, morada, coordenadas geográficas, tipo de local (museu, bar, entre outros) que podem ser acedidas através do identificador único do local.

3.3.5 Conclusão

O estudo das várias APIs disponibilizadas pela Google permitiu obter uma visão geral das funcionalidades que podem ser implementadas com recurso a estes serviços. Com especial destaque da Google Maps API que possibilita a integração de mapas nas aplicações Android e da Google Maps Roads API que permite melhorar a precisão dos conjuntos de coordenadas obtidos pela aplicação móvel.

3.4 Ferramentas Relevantes

Nesta secção, descrevem-se algumas das ferramentas relevantes para o desenvolvimento deste projeto. São justificadas as escolhas das ferramentas adotadas com base na análise e comparações efetuadas.

3.4.1 Bases de Dados

Nesta secção, apresentam-se os vários sistemas de bases de dados que foram analisados de modo a escolher a melhor opção para o atual projeto.

3.4.1.1 MySQL

O MySQL (MySQL, s.d.) é um sistema *open source* de gestão de bases de dados relacionais desenvolvido pela Oracle. O servidor de base de dados MySQL é rápido, confiável, escalável e de fácil uso.

3.4.1.2 Firebase Realtime Database (Base de dados em tempo real Firebase)

A *Firebase Realtime Database* (Firebase Realtime Database, s.d.) consiste numa base de dados NoSQL hospedada numa *cloud*. Os dados são armazenados em JSON e sincronizados em tempo real entre todos os clientes conectados. Esta base de dados é útil no desenvolvimento de aplicações *cross-platform* em que todos os clientes partilham uma mesma instância da base de dados e recebem automaticamente atualizações com os dados mais recentes.

3.4.1.3 Conclusão

Após uma avaliação das bases de dados estudadas foi escolhida integração com o MySQL. Embora as bases de dados NoSQL apresentem vantagens na sua utilização, no contexto deste projeto ficou decidido usar uma base de dados relacional.

3.4.2 Hosting

Nesta seção, é apresentado o estudo efetuado das várias plataformas de hospedagem de aplicações que permitiu escolher a melhor opção para o projeto atual.

3.4.2.1 Openshift Online

Openshift Online (OpenShift: Container Application Platform by RedHat, s.d.) é uma plataforma de desenvolvimento e hospedagem de aplicações na nuvem pública da *Red Hat* que torna automático o processo de provisionamento, gestão e escalamento de aplicações.

Suporta uma grande variedade de linguagens e serviços que podem ser adicionadas às aplicações através de cartuchos que podem ser *frameworks* web, bases de dados, serviços de monitorização e conetores a *backends* externos. As tecnologias suportadas incluem Java, PHP, Node.js, MySQL, PostgreSQL, entre outros.

O *OpenShift Onlin*e disponibiliza uma versão gratuita que está limitada a apenas um projeto com a possibilidade de armazenar até um *gigabyte*.

3.4.2.2 **Microsoft Azure**

Azure (Microsoft Azure Cloud Computing Platform & Services, s.d.) consiste num conjunto de serviços *cloud* que permite a *developers* e a profissionais de tecnologia da informação (TI) criar, implantar e gerir aplicações através de uma rede global de centros de processamento de dados (CPD).

O Azure disponibiliza vários modelos de computação, cada um com a sua função específica, que podem ser usados separadamente ou em conjunto dependendo da solução que se pretende. Os modelos de computação disponibilizados são os seguintes:

- Azure *Virtual Machines* (Máquinas Virtuais Azure)

Esta funcionalidade permite criar uma máquina virtual quando necessário, tanto de uma imagem padrão ou uma fornecida.

- *Web Apps* (Aplicações Web)

O Azure permite implantar aplicações web sem acesso administrativo. Possibilita a implantação de uma aplicação web já existente ou a criação de uma nova diretamente na *cloud*. Posteriormente às aplicações estarem a executar é possível adicionar ou remover instâncias dinamicamente, dependendo do Azure Web Apps para balancear a carga entre elas. O Azure Web Apps suporta .NET, PHP, Node.js, Java, Python juntamente com base de dados SQL e Base de Dados Azure para MySQL para armazenamento relacional.

- Cloud Services (Serviços em cloud)

O Azure Cloud Services permite a criação de aplicações *web* escaláveis, confiáveis e com poucas necessidades de administração. Suporta as mesmas tecnologias do Azure Web Apps e o código é executado em máquinas virtuais que executam uma versão *Windows Server*.

O Azure disponibiliza diversas maneiras para armazenar e gerir dados e em todas elas existem sempre três cópias dos dados sincronizados com o centro de processamento de dados Azure. As opções de armazenamento são as seguintes:

- Azure SQL Database (Base de Dados SQL Azure)

Para armazenamento relacional o Azure disponibiliza a Base de Dados SQL. A Azure SQL Database disponibiliza todas as funcionalidades de um sistema de gestão de uma base de dados relacional, incluindo transações atómicas, acesso concorrente entre vários utilizadores garantindo a integridade dos dados, *queries* SQL e um modelo de programação familiar. Esta base de dados pode ser acedida através do *Entity Framework*, JDBC ou outra tecnologia similar. A parte de administração da base de dados é gerida pelo Azure havendo apenas a possibilidade de gerir os dados e a autorização de acesso a estes.

- Tables

A Azure Tables é uma abordagem de NoSQL chamada base de dados chave/valor. Esta funcionalidade permite armazenar vários tipos de dados e é capaz de retornar estes dados ao disponibilizar uma chave única desse grupo. O acesso a estes dados é rápido e estes podem ser escaláveis onde uma tabela pode guardar até um *terabyte* de dados.

- Preço

Através do programa DreamSpark disponível para estudantes do ISEP é possível ter acesso a uma versão grátis do Microsoft Azure.

3.4.2.3 Conclusão

Comparando as duas plataformas de *hosting* em cima abordadas, decidiu-se optar pelo uso do Microsoft Azure. Embora a versão grátis do Azure limite muito as suas funcionalidades, através do programa DreamSpark é possível obter uma versão mais completa sem custos o que torna a escolha desta plataforma a melhor opção.

3.4.3 Data Mining

Nesta secção, apresenta-se o estudo efetuado a diversas ferramentas de *data mining* disponíveis que permitiu escolher a mais apropriada ao projeto atual.

3.4.3.1 **Weka *(Waikato Environment for Knowledge Analysis)***

"Weka é uma coleção de algoritmos de *machine learning* para tarefas de *data mining*" (The University of Waikato, s.d.). Desenvolvido pela Universidade Waikato na Nova Zelândia, esta ferramenta contém algoritmos que podem ser aplicados diretamente a um dataset ou invocados através de uma aplicação Java.

Algumas das funcionalidades principais do Weka são as seguintes:

- Pré-processamento de dados: O Weka suporta extensões de ficheiros de texto populares como o CSV e JSON para importação assim como ARFF (uma extensão própria do Weka). Existe também suporte para importação de dados de bases de dados JDBC incluindo filtros para facilitar a análise dos mesmos (Witten & Frank, 2016);

- Classificação de dados: Um grande número de algoritmos de classificação, incluindo funções matemáticas, classificadores Bayesian, classificadores *lazy* com a implementação de cálculos do "vizinho mais próximo" e classificadores baseados em árvore (Witten & Frank, 2016);

- Agrupamento de dados: Existem algoritmos de agrupamento de dados como variações do método "*k-mean*" assim como a densidade e algoritmos de agrupamento com hierarquias (Witten & Frank, 2016);

- Seleção de atributos: Métodos para avaliar o atributo que mais contribui na previsão de um certo resultado (Witten & Frank, 2016);

- Visualização de dados: É possível visualizar os dados usando vários métodos, sendo possível compará-los com variáveis específicas estando também disponíveis algumas ferramentas para uma análise mais detalhada (Witten & Frank, 2016).

3.4.3.2 RServe

O RServe é um servidor TCP/IP que permite a outros programas a utilização do R (R Foundation, s.d.) usando várias linguagens sem a necessidade da inicialização do R nem a ligação com a sua biblioteca (RForge, s.d.).

As implementações da parte do cliente podem ser implementadas em C/C++, PHP ou JAVA, existe também suporte de conexões remotas, autenticação e transferência de ficheiros (RForge, s.d.).

Uma opção desta ferramenta é o Java/R Interface (JRI) que permite a execução do R em aplicações Java numa única *thread*, sem o conceito do cliente/servidor. Neste caso a biblioteca do R é carregada dinamicamente no Java disponibilizando uma API Java para a funcionalidade do R (RForge, s.d.).

3.4.3.3 RapidMiner

RapidMiner é uma ferramenta *open-source* escrita em Java, disponível para Windows e Linux, que foi desenvolvida pela Universidade de Dortmund em 2001 (Land & Fischer, 2012). Usa um modelo cliente/servidor em que os utilizadores podem introduzir os seus *scripts* usando a linguagem de programação R. A principal característica do RapidMiner é a sua capacidade de fornecer a solução de uma análise avançada sem a necessidade de codificar (Norris, 2013).

Algumas das ferramentas disponíveis são o carregamento e transformação de dados (ETL), pré-processamento de dados e a sua visualização, análise preditiva e modelação estatística. São também fornecidos esquemas de aprendizagem, algoritmos e modelos que são extensíveis usando *scripts* em Python ou R (Norris, 2013).

3.4.3.4 Conclusão

Comparando as diferentes opções das tecnologias de *data mining*, encontraram-se vantagens e desvantagens na utilização de cada uma. Assim, decidiu-se usar o Weka na integração da

solução pela sua fácil integração com o Java e a simples integração com uma base de dados MySQL. O Weka foi integrado através de uma dependência Maven que permite a utilização das suas funcionalidades no Java.

3.4.4 SQL (Structured Query Languages) vs NoSQL (Not Only Structured Query Languages)

O rápido crescimento da internet e o aumento da complexidade do *software* atualmente criado, tem levado a uma procura de métodos mais eficientes de armazenar e transacionar dados. Grandes quantidades de dados *online* são transacionadas diariamente o que implica o uso uma solução de armazenamento organizada, que assegure consistência e deteção de erros. No contexto das tecnologias das bases de dados existem dois tipos principais: bases de dados relacionais (SQL) e bases de dados não relacionais (NoSQL) (Pore & Pawar, 2015).

O principal conceito de uma base de dados SQL baseia-se nas relações (tabelas) como modo de armazenar os dados. As tabelas podem estar relacionadas entre si e podem estar interligadas através de chaves estrangeiras ou colunas em comum. Cada tabela está dividida em colunas, representativas do campo, e linhas identificando o registo (Pore & Pawar, 2015).

As bases de dados NoSQL apresentam-se como não relacionais, distribuídas e escaláveis horizontalmente. Desvantagens do uso de bases de dados não relacionais é a falta de suporte RDBMS (*relational database management system*), limitações a nível da integridade de *constraints* como chaves estrangeiras, e um suporte limitado no processamento de transações (Hinai, 2016).

As principais diferenças entre as bases de dados relacionais e não relacionais podem ser sumarizadas em três pontos:

Escalabilidade

As bases de dados relacionais foram projetadas para serem escaláveis verticalmente, o que implica uma maior capacidade de *hardware* à medida que o tamanho dos dados aumenta, enquanto as bases de dados NoSQL escalam horizontalmente, em que novas partições são adicionadas quando necessário.

Estrutura

As bases de dados SQL lidam com tabelas estruturadas para armazenamento dos dados, que podem estar relacionadas entre si, enquanto as bases de dados não relacionais gerem dados não estruturados. Alguns estudos afirmam que as bases de dados NoSQL possuem um melhor desempenho em comparação com as bases de dados relacionais que mantêm os dados num estado normalizado o que provoca uma diminuição no seu desempenho (Hammes, Medero, & Mitchell, 2014).

Consistência

A consistência de uma base de dados é a propriedade que assegura que os dados apenas são armazenados caso passem um conjunto de regras e restrições que confirmem que o seu estado se mantém válido. Estas restrições podem ser especificadas como uma fórmula, que caso seja violada a transação deve ser abortada (Geppert & Wimmers).

As bases de dados relacionais baseiam-se nas propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) durante as operações de gestão de dados, que asseguram a sua consistência.

As bases de dados NoSQL, por outro lado, baseiam-se nas propriedades BASE (*Basically Available, Soft State, Eventual Consistency*) que valoriza mais a disponibilidade, mas não oferece uma consistência garantida dos dados enquanto uma operação de escrita está em execução (Pore & Pawar, 2015).

Embora ambas as bases de dados relacionais e não relacionais possuam vantagens e desvantagens do seu uso, no contexto deste projeto as bases de dados SQL aparentam estar mais adequadas por possuírem uma estrutura de dados mais adequada e por uma melhor gestão dos dados armazenados.

4 Design

Este capítulo descreve o *design* da solução proposta, a identificação de requisitos funcionais e dos não-funcionais e respetiva descrição detalhada, o *design* da arquitetura com a avaliação de diferentes alternativas, o modelo de domínio, o modelo de dados, o modelo de componentes e de implantação.

4.1 Requisitos funcionais

Nesta seção são descritas as funcionalidades que vão estar disponíveis aos utilizadores. Vai ser apresentado um diagrama de casos de uso para expor uma visão geral das funcionalidades presentes na solução e de seguida vai ser apresentada uma descrição em detalhe de cada um deles.

4.1.1 Diagrama de casos de uso

Na Figura 11, é apresentado o diagrama de casos de uso da solução. O 'Utilizador Não Registado' representa uma pessoa que utiliza a aplicação pela primeira vez e necessita de efetuar o seu registo para ter acesso às restantes funcionalidades. O 'Utilizador' representa o ator que utiliza o sistema.

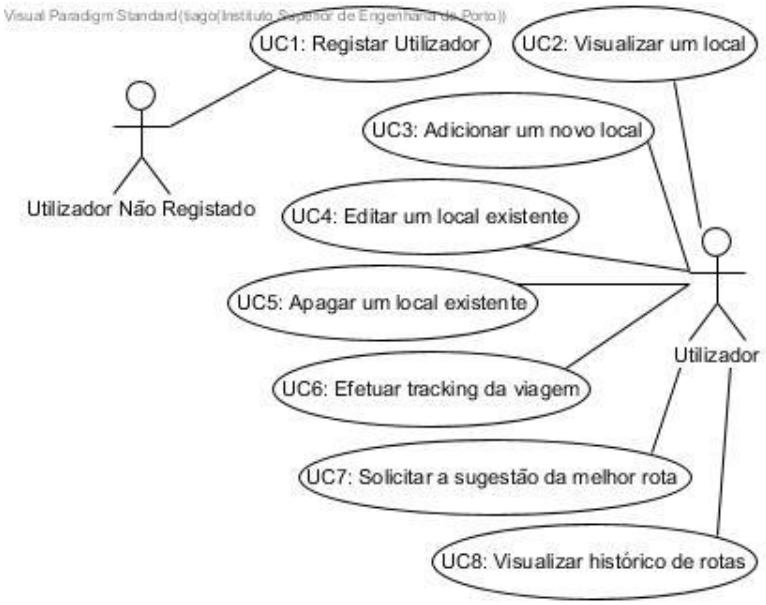

Figura 11 – Diagrama de casos de uso

4.1.2 UC1: Registar Utilizador

4.1.2.1 Formato breve

O utilizador não registado inicia o seu registo no sistema. O sistema solicita os dados necessários (email e password). O utilizador insere os dados solicitados. O sistema valida os dados, cria o utilizador e informa o utilizador do sucesso da operação.

4.1.2.2 SSD

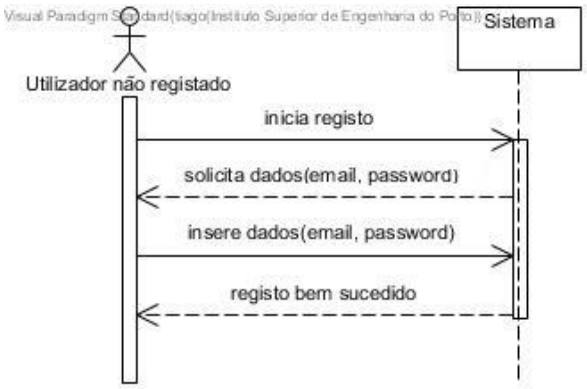

Figura 12 – UC1 SSD

4.1.2.3 Formato completo

Ator principal

Utilizador não registado

Partes interessadas e seus interesses

Utilizador não registado: Pretende que o registo seja rápido e seguro para garantir o seu acesso posterior ao sistema.

Pré-condições

O utilizador não está autenticado.
O dispositivo móvel deverá estar ligado à rede.

Pós-condições

A informação do utilizador é registada no sistema.

Cenário de sucesso principal

1. O utilizador (não registado) inicia o processo de registo.
2. O sistema solicita os dados necessários (*email* e *password*).
3. O utilizador (não registado) insere os dados solicitados.
4. O sistema valida os dados, cria o utilizador e informa o utilizador do sucesso da operação.

Extensões

***a.** O utilizador solicita o cancelamento da operação.

O caso de uso termina.

4a. Dados mínimos obrigatórios em falta.

1. O sistema informa quais os dados em falta.
2. O sistema permite a introdução dos dados em falta (passo 2)

4b. O sistema deteta que os dados (ou algum subconjunto dos dados) já existem no sistema.
1. O sistema alerta o utilizador para o facto.
2. O sistema permite a reintrodução dos dados (passo 2)

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

-

4.1.3 UC2: Visualizar um local

4.1.3.1 Formato breve

O utilizador inicia o processo de visualização de um novo local. O sistema apresenta a lista dos locais do utilizador. O utilizador seleciona o local que pretende visualizar. O sistema apresenta todas as informações do local.

4.1.3.2 SSD

Figura 13 – UC2 SSD

4.1.3.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: Apresentadas as informações corretas do local.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.

Pós-condições

O utilizador visualiza o local pretendido.

Cenário de sucesso principal

1. O utilizador inicia o processo de visualização de um local.
2. O sistema apresenta a lista de locais do utilizador.
3. O utilizador seleciona um local.
4. O sistema apresenta todas as informações do local (nome, endereço e categoria).

Extensões

***a.** O utilizador solicita o cancelamento da operação.

O caso de uso termina.

2a. Não existem locais no sistema. A aplicação informa o utilizador do facto.

O caso de uso termina.

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

-

4.1.4 UC3: Adicionar um novo local

4.1.4.1 Formato breve

O utilizador inicia o processo de inserção de um novo local. O sistema apresenta uma lista de campos obrigatórios para a criação de um novo local (nome do local, endereço, categoria). O utilizador insere os dados solicitados. O sistema valida os dados, cria o novo local e informa o utilizador do sucesso da operação.

4.1.4.2 SSD

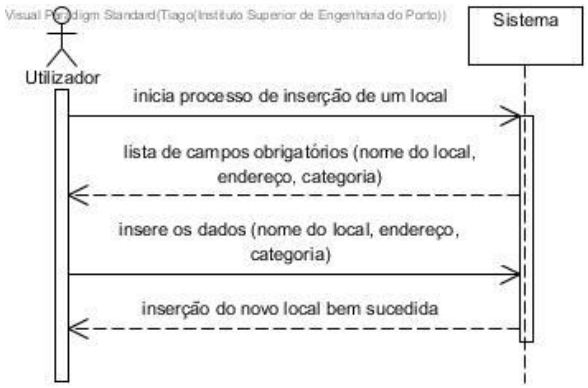

Figura 14 – UC3 SSD

4.1.4.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: Rapidez na adição do novo local e sem erros.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.

Pós-condições

O novo local é inserido no sistema.

Cenário de sucesso principal

1. O utilizador inicia o processo de inserção de um novo local.
2. O sistema solicita os dados necessários (nome do local, endereço, categoria).
3. O utilizador insere os dados solicitados.

4. O sistema valida os dados, cria o local e informa o utilizador do sucesso da operação.

Extensões

***a.** O utilizador solicita o cancelamento da operação.
O caso de uso termina.

4a. Dados mínimos obrigatórios em falta.
1. O sistema informa quais os dados em falta.
2. O sistema permite a introdução dos dados em falta (passo 2)

4b. O sistema deteta que os dados (ou algum subconjunto dos dados) já existem no sistema.
1. O sistema alerta o utilizador para o facto.
2. O sistema permite a reintrodução dos dados (passo 2)

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

- Quais os dados que em conjunto permitem detetar a duplicação de locais?

Como deverá ser validado o endereço introduzido?

4.1.5 UC4: Editar um local existente

4.1.5.1 Formato breve

O utilizador inicia o processo de edição de um local. O sistema apresenta uma lista dos locais existentes. O utilizador seleciona um dos locais presentes na lista. O sistema apresenta o local selecionado e dá a possibilidade de editar esse local. O utilizador seleciona a opção de editar o local. O sistema apresenta os campos atuais desse local e possibilita a edição. O utilizador edita os campos pretendidos e confirma. O sistema valida os dados, edita o local com os dados atualizados e informa o utilizador do sucesso da operação.

4.1.5.2 SSD

Figura 15 – UC4 SSD

4.1.5.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: Rapidez na edição do local e sem erros.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.

Pós-condições

O local é atualizado no sistema.

Cenário de sucesso principal

1. O utilizador inicia o processo de edição de um local.
2. O sistema apresenta uma lista dos locais existentes.
3. O utilizador seleciona um dos locais presentes na lista.
4. O sistema apresenta o local selecionado e dá a possibilidade de editar esse local.
5. O utilizador seleciona a opção de editar o local.
6. O sistema apresenta os campos atuais desse local (nome do local, endereço, categoria).
7. O utilizador edita os campos pretendidos e confirma.
8. O sistema valida os dados, edita o local com os dados atualizados e informa o utilizador do sucesso da operação.

Extensões

***a.** O utilizador solicita o cancelamento da operação.

O caso de uso termina.

2a. Não existem locais no sistema. A aplicação informa o utilizador do facto.

O caso de uso termina

8a. Dados mínimos obrigatórios em falta.

1. O sistema informa quais os dados em falta.
2. O sistema permite a introdução dos dados em falta (passo 4)

8b. O sistema deteta que os dados (ou algum subconjunto dos dados) já existem no sistema.

1. O sistema alerta o utilizador para o facto.
2. O sistema permite a reintrodução dos dados (passo 4)

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

-

4.1.6 UC5: Apagar um local existente

4.1.6.1 Formato breve

O utilizador inicia o processo de remoção de um local. O sistema apresenta uma lista dos locais existentes. O utilizador seleciona um dos locais presentes na lista. O sistema apresenta o local selecionado e dá a possibilidade de remover esse local. O utilizador seleciona a opção de remover o local. O sistema solicita que o utilizador confirme a remoção do local selecionado. O utilizador confirma. O sistema remove o local e informa o utilizador do sucesso da operação.

4.1.6.2 SSD

Figura 16 – UC5 SSD

4.1.6.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: Rapidez na remoção do local e sem erros.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.

Pós-condições

O local é removido do sistema.

Cenário de sucesso principal

1. O utilizador inicia o processo de remoção de um local.
2. O sistema apresenta uma lista dos locais existentes.
3. O utilizador seleciona um dos locais presentes na lista.
4. O sistema apresenta o local selecionado e dá a possibilidade de remover esse local.
5. O utilizador seleciona a opção de remover o local.
6. O sistema solicita que o utilizador confirme a remoção do local selecionado.
7. O utilizador confirma.
8. O sistema remove o local e informa o utilizador do sucesso da operação.

Extensões

***a.** O utilizador solicita o cancelamento da operação.

O caso de uso termina.

2a. Não existem locais no sistema. A aplicação informa o utilizador do facto.

O caso de uso termina**.**

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

- O que acontece às rotas previamente gravadas com o local removido associado?

4.1.7 UC6: Efetuar *tracking* da viagem

4.1.7.1 Formato breve

O utilizador inicia o processo de *tracking* da sua viagem. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução da origem da viagem. O utilizador seleciona um local. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução do destino da viagem. O utilizador seleciona um local. O sistema solicita a introdução do nome do novo trajeto. O utilizador insere o nome. O sistema inicia o *tracking* da viagem do utilizador. Ao chegar ao seu local de destino o utilizador finaliza o *tracking* da viagem. O sistema apresenta a rota efetuada e outras informações adicionais, solicitando a confirmação de gravação. O utilizador confirma. O sistema grava a rota efetuada adicionando informações adicionais (dia-da-semana e hora) e informa o utilizador do sucesso da operação.

4.1.7.2 SSD

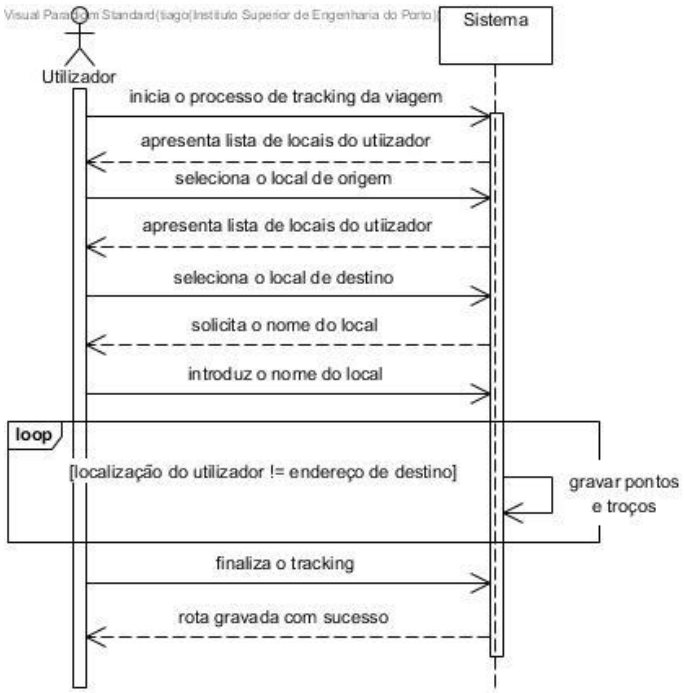

Figura 17 – UC6 SSD

4.1.7.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: *Tracking* da viagem preciso e a gravação da rota sem erros.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.
O dispositivo móvel deverá ter a informação de localização ativa.

Pós-condições

A rota da viagem do condutor é gravada no sistema.

Cenário de sucesso principal

1. O utilizador inicia o processo de *tracking* da sua viagem.
2. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução da origem da viagem.
3. O utilizador seleciona um local.

4. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução do destino da viagem.
5. O utilizador seleciona um local.
6. O sistema solicita a introdução do nome do novo trajeto.
7. O utilizador insere o nome.
8. O sistema inicia o *tracking* da viagem do utilizador.
9. Ao chegar ao seu local de destino o utilizador finaliza o *tracking* da viagem.
10. O sistema apresenta a rota efetuada e outras informações adicionais, solicitando a confirmação de gravação.
11. O utilizador confirma.
12. O sistema grava a rota efetuada adicionando informações adicionais (dia-da-semana e hora) e informa o utilizador do sucesso da operação.

Extensões

***a.** O utilizador solicita o cancelamento da operação.
O caso de uso termina.

2a. Não existem locais no sistema. A aplicação informa o utilizador do facto.
O caso de uso termina**.**

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

-

4.1.8 UC7: Solicitar a sugestão da melhor rota

4.1.8.1 Formato breve

O utilizador inicia o processo de solicitação da melhor rota. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução da origem da viagem. O utilizador seleciona um local. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução do destino da viagem. O utilizador seleciona um local. O sistema encontra o trajeto entre o par origem-destino selecionado e calcula a melhor rota tendo em conta as informações do dia-da-semana e hora atuais. O sistema apresenta o melhor trajeto ao utilizador.

4.1.8.2 SSD

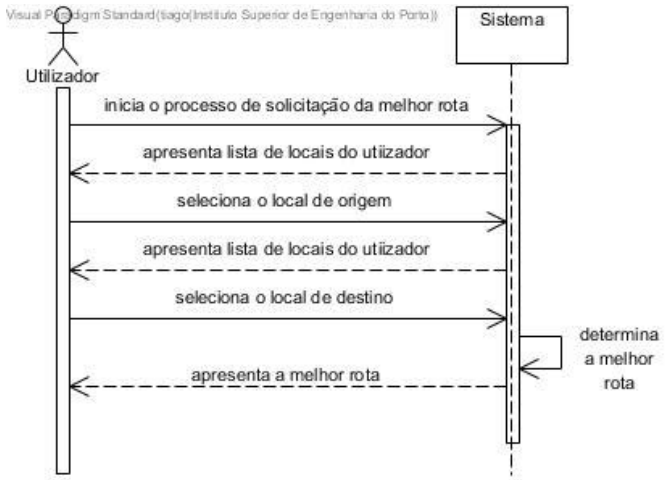

Figura 18 – UC7 SSD

4.1.8.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: Rota sugerida rapidamente e apresentada devidamente no mapa.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.
O dispositivo móvel deverá ter a informação de localização ativa.

Pós-condições

A melhor rota é apresentada ao condutor.

Cenário de sucesso principal

1. O utilizador inicia o processo de solicitação da melhor rota.
2. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução da origem da viagem.
3. O utilizador seleciona um local.
4. O sistema apresenta uma lista dos locais do utilizador e solicita a introdução do destino da viagem.
5. O utilizador seleciona um local.
6. O sistema encontra o trajeto entre o par origem-destino selecionado e calcula a melhor rota tendo em conta as informações do dia-da-semana e hora atuais.
7. O sistema apresenta o melhor trajeto ao utilizador.

Extensões

***a.** O utilizador solicita o cancelamento da operação.

O caso de uso termina.

2a. Não existem locais no sistema. A aplicação informa o utilizador do facto.

O caso de uso termina.

6a. Não existem rotas gravadas no sistema. A aplicação informa o utilizador do facto.

O caso de uso termina**.**

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

- Deve ser mostrada uma rota referente a outro dia-da-semana/hora caso não exista nenhuma com a mesma informação no momento em que o utilizador solicita a sugestão da melhor rota?

4.1.9 UC8: Visualizar o histórico de rotas

4.1.9.1 Formato breve

O utilizador inicia o processo de visualização do seu histórico de rotas. O sistema apresenta uma lista dos nomes dos trajetos do utilizador efetuados entre pares origem-destino. O utilizador seleciona um trajeto da lista. O sistema apresenta as rotas no mapa e informações adicionais do trajeto.

4.1.9.2 SSD

Figura 19 – UC8 SSD

4.1.9.3 Formato completo

Ator principal

Utilizador

Partes interessadas e seus interesses

Utilizador: Histórico bem organizado e visualização de rotas apresentadas devidamente no mapa.

Pré-condições

O dispositivo móvel deverá estar ligado à rede.

Pós-condições

O utilizador visualiza o seu histórico de viagens.

Cenário de sucesso principal

1. O utilizador inicia o processo de visualização do seu histórico de rotas.
2. O sistema apresenta uma lista dos nomes dos trajetos do utilizador efetuados entre pares origem-destino.
3. O utilizador seleciona um trajeto da lista.
4. O sistema apresenta as rotas no mapa e informações adicionais do trajeto.

Extensões

***a.** O utilizador solicita o cancelamento da operação.

O caso de uso termina.

2a. Não existem trajetos no sistema. A aplicação informa o utilizador do facto.
O caso de uso termina**.**

Requisitos especiais

-

Tecnologia e Lista de Variações dos Dados

-

Questões em aberto

-

4.2 Modelo de Domínio

O processo de construção do modelo de domínio é baseado nos casos de uso anteriormente descritos. Na Figura 20 é apresentado o diagrama do modelo de domínio proposto da solução.

O utilizador que contém um email e uma *password*, referentes ao seu registo, é responsável por criar os seus locais, cada um contendo a informação do nome, endereço e uma categoria que pode ser comum a vários locais.

O utilizador ao efetuar o *tracking* da sua viagem vai criar no final um trajeto que é constituído por dois locais, representando a origem e o destino do trajeto, e um conjunto de troços e pontos geográficos. Cada troço faz a conexão entre dois pontos geográficos.

Neste contexto entende-se por trajeto como o conjunto de pontos geográficos e troços gravados em todas as rotas efetuadas pelo utilizador entre o local inicial e final.

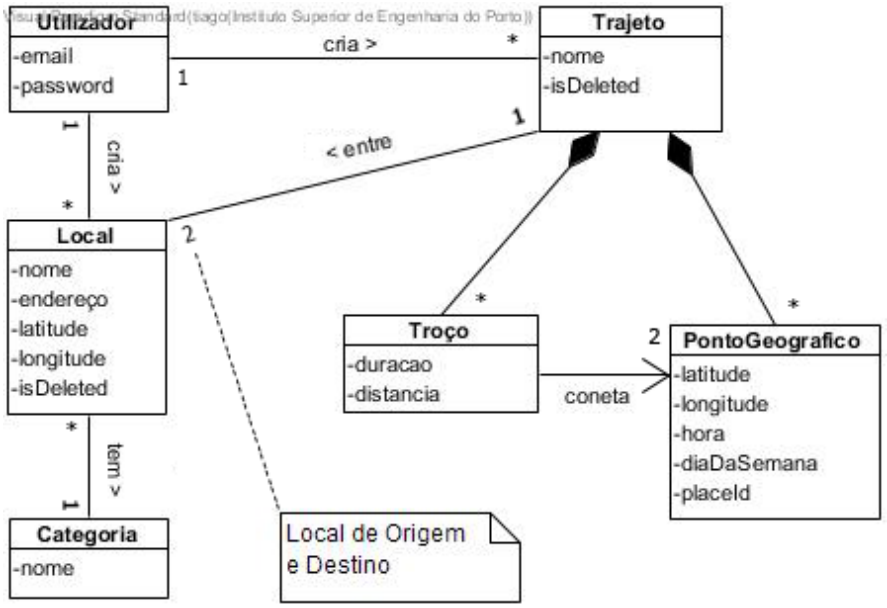

Figura 20 – Diagrama do Modelo de Domínio

4.3 *Design* Arquitetural

A fase do desenvolvimento do *design* arquitetural deve seguir as boas práticas da engenharia e requer de uma reflexão atenta uma vez que caso seja necessário efetuar alterações futuras, estas terão um elevado impacto no funcionamento de todo o projeto.

Durante a fase de definição da arquitetura do sistema foram avaliadas diferentes alternativas que vão ser de seguida detalhadas.

Decidiu-se usar o diagrama de componentes para apresentar as alternativas de arquitetura pois facilita a compreensão e a criação do *design* do sistema, explicita claramente a comunicação entre as partes interessadas e permite uma melhor manutenção do *design* aquando da alteração dos requisitos.

Nas figuras 21 e 22 são apresentados os diagramas de componentes dessas alternativas e será de seguida discutido as vantagens e desvantagens que foram o alvo de reflexão na escolha da arquitetura a aplicar no projeto.

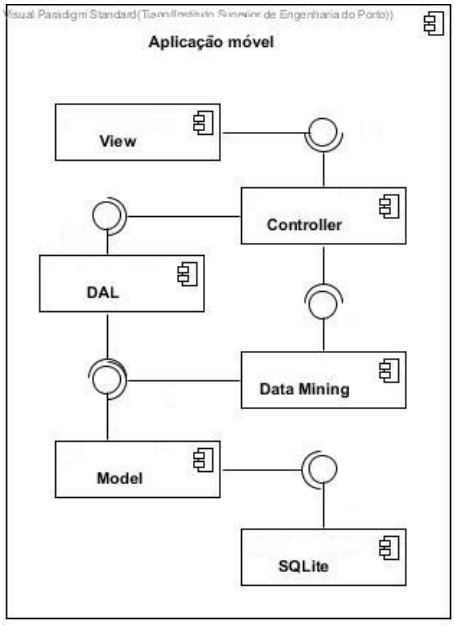

Figura 21 – Diagrama de Componentes 1

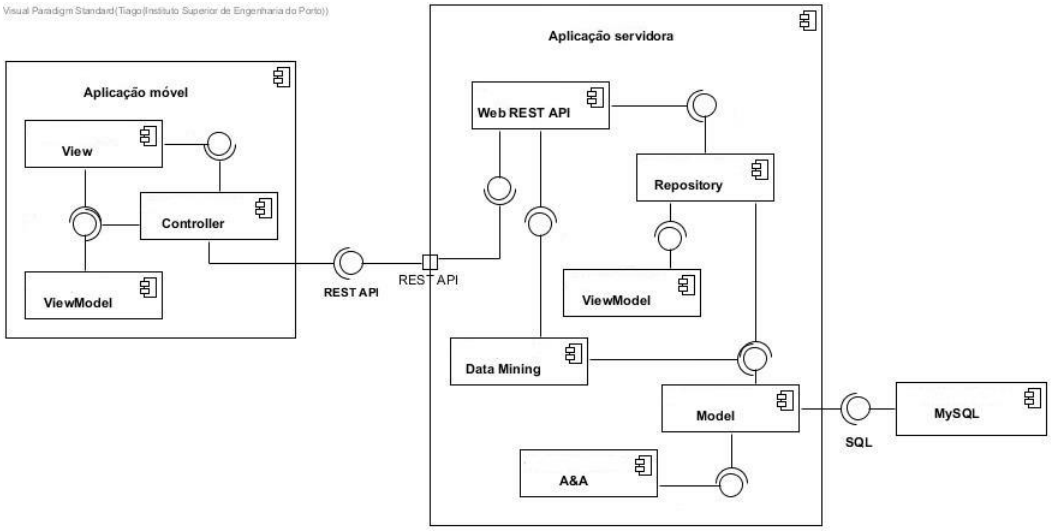

Figura 22 – Diagrama de Componentes 2

O primeiro diagrama apresenta uma arquitetura monolítica, onde apenas se iria desenvolver uma aplicação móvel como solução. Isto traria vantagens em termos do tempo de desenvolvimento e facilidade de implantação e uma interação entre componentes mais coesa, visto não existir comunicações com servidores externos.

Embora apresente vantagens, existem também muitas desvantagens neste tipo de arquitetura nomeadamente a sua difícil manutenção e a presença de pontos únicos de falha, onde um erro pode levar à falha geral do sistema. Existe também o problema de haver um uso extensivo de recursos devido a mecanismos como o *data mining* que poderá apresentar problemas para os dispositivos móveis com uma menor capacidade de processamento.

No diagrama são apresentados os vários componentes que seriam necessários para o sistema. A *View* representa a camada de apresentação para o cliente, que comunica com o *Controller* que gere as diversas funcionalidades da aplicação. Este componente comunica com uma camada de acesso aos dados, que por sua vez comunica com o *Model*. A base de dados neste caso seria *SQLite* que é a existente nos dispositivos móveis *Android*. Por fim o componente de *Data Mining* seria necessário na pesquisa dos caminhos para o utilizador.

A alternativa a esta arquitetura é exibida na Figura 22, onde é proposta uma arquitetura que engloba vários estilos arquiteturais chave:

- Cliente/servidor: sistema segregado em duas aplicações distintas, onde o cliente efetua pedidos ao servidor (Microsoft, s.d.);

- Baseada em componentes: *design* da aplicação decomposto em componentes lógicos ou funcionais reutilizáveis que expõem interfaces de comunicação bem definidas (Microsoft, s.d.);

- Arquitetura por camadas: componentes da aplicação organizados por camadas, onde cada camada desempenha um papel específico na aplicação (Microsoft, s.d.);

- *N-Tier*: cada segmento encontra-se localizado num sistema físico distinto (Microsoft, s.d.);

- Arquitetura orientada a serviços: as funcionalidades implementadas pela aplicação são expostas como serviços (Microsoft, s.d.).

O uso desta arquitetura torna o desenvolvimento mais fácil, sendo possível decompor e encapsular a complexidade mais facilmente. A separação por interesses facilita a reutilização. A manutenção também se torna mais simples devido ao baixo acoplamento entre as diversas camadas. Neste contexto a única desvantagem aparente é uma redução no desempenho devido à alta coesão e baixo acoplamento, sendo que esta arquitetura está mais otimizada para a modularização, facilidade de manutenção e reutilização.

No diagrama estão representados dois componentes relativos à aplicação móvel (cliente) e à aplicação servidora. A aplicação móvel é constituída por uma camada de apresentação que é controlada por um *Controller* e um componente *ViewModel* que funciona como uma camada intermédia entre a *View* e o *Model* e é usado como uma camada de abstração da lógica de negócio. O *Controller* comunica com o servidor através de serviços expostos pelo componente *Web REST API.* De modo a restringir a comunicação direta com a base de dados é usado um componente que aplica o padrão *Repository*. O *Model*, que contém a lógica de negócio da aplicação, comunica com o componente *A&A* que representa a camada que gere a autenticação dos clientes e as autorizações de acesso à aplicação. Por fim, é representada uma base de dados MySQL que será usada na persistência dos dados.

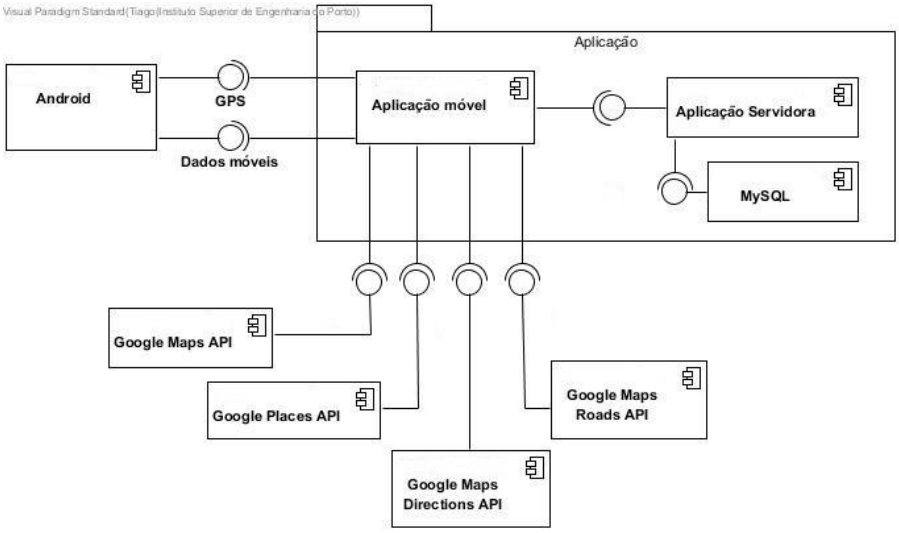

Figura 23 – Diagrama do modelo de componentes de alto nível

Na Figura 23 é apresentado um diagrama de componentes de alto nível do sistema onde é possível ver as interações da aplicação com alguns componentes do dispositivo móvel *Android*, necessários para o seu correto funcionamento, como a informação de localização (GPS) e os dados móveis. Na pesquisa de locais e na visualização, recolha e apresentação de rotas, a solução usa também serviços disponibilizados pela Google que estão presentes no diagrama.

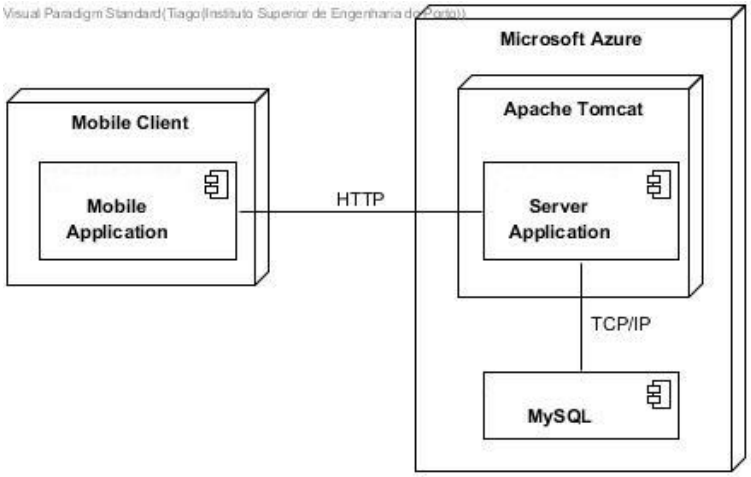

Figura 24 – Diagrama de implantação

Com o objetivo de dar uma visão geral da implantação física dos artefactos do sistema é apresentado um diagrama de implantação na Figura 24. A solução é composta por uma aplicação móvel que estará instalada no dispositivo móvel do cliente, que comunicará com a aplicação servidora implantada no servidor da Microsoft Azure em conjunto com uma base de dados *MySQL*.

4.3.1 UC1: Diagrama de Sequência

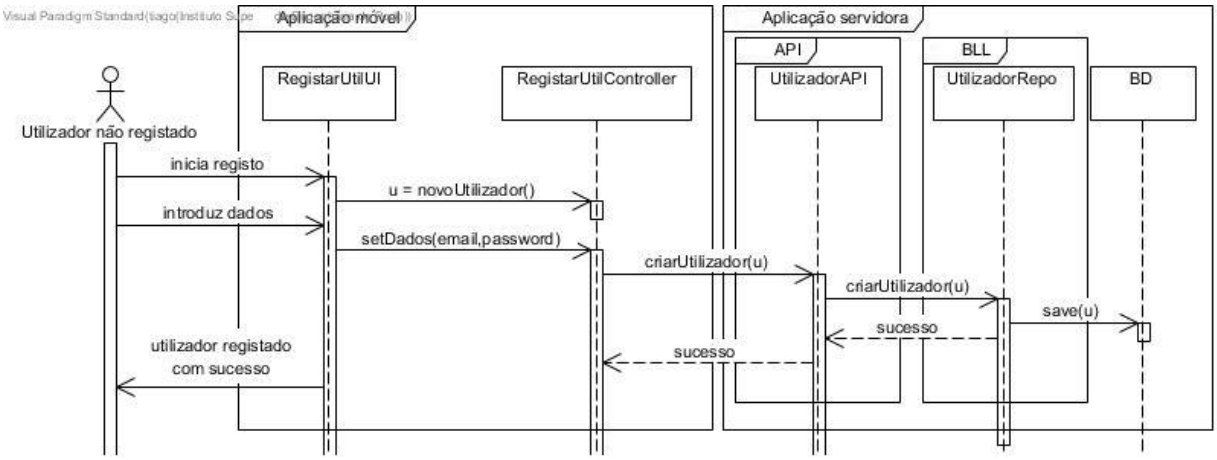

Figura 25 – Diagrama de sequência UC1

Na Figura 25 pode ser observado o diagrama de sequência do caso de uso relativo ao registo do utilizador no sistema. O registo é iniciado pelo utilizador na aplicação móvel e de seguida insere os dados solicitados (*email* e *password*). Com estes dados é feito um pedido à API da aplicação servidora que valida e cria o novo utilizador através do repositório. No final do processo é apresentada uma mensagem de sucesso ao utilizador.

4.3.2 UC2: Diagrama de Sequência

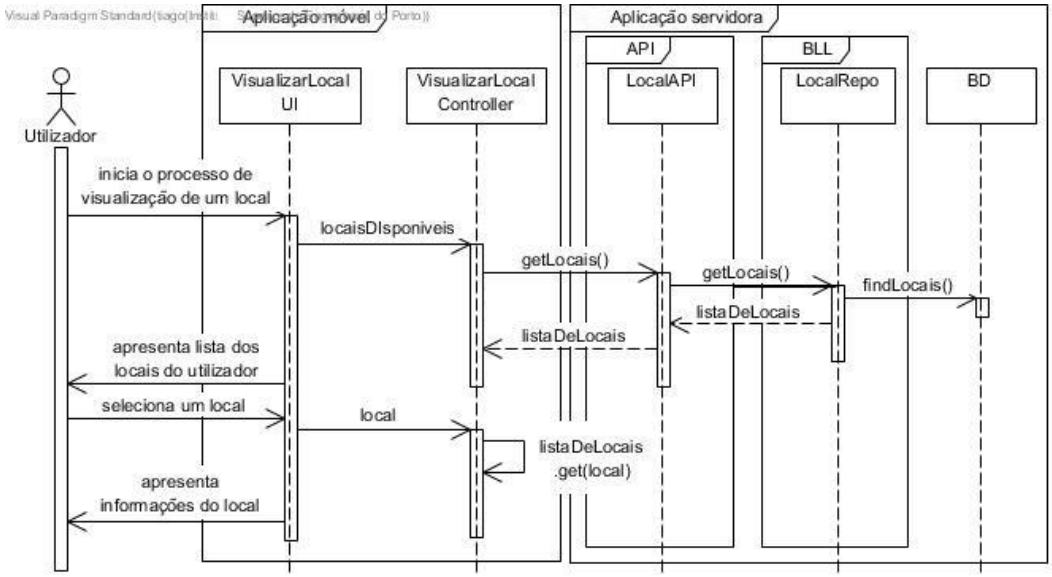

Figura 26 – Diagrama de sequência UC2

Na Figura 26 pode ser observado o diagrama de sequência do caso de uso relativo à visualização de um local. Ao iniciar este processo o sistema começa por comunicar com a aplicação servidora

para receber a lista de locais do utilizador. O utilizador de seguida seleciona o local que pretende visualizar. O sistema procura o local selecionado da lista de locais e apresenta-o ao utilizador.

4.3.3 UC3: Diagrama de Sequência

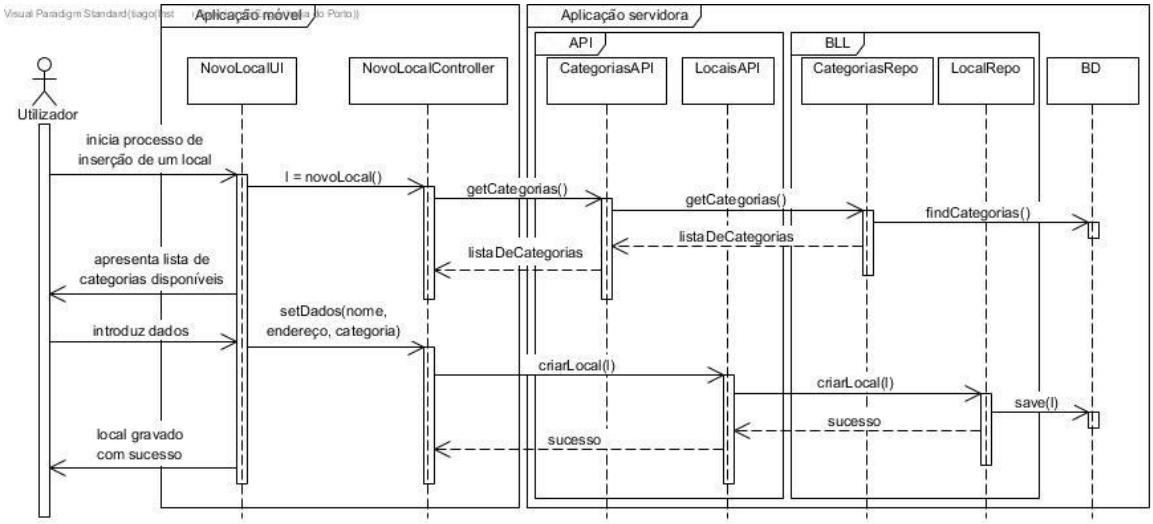

Figura 27 – Diagrama de sequência UC3

Na Figura 27 pode ser observado o diagrama de sequência do caso de uso relativo à criação de um novo local no sistema. Ao iniciar este processo o sistema começa por comunicar com a aplicação servidora para receber a lista de categorias existentes de modo a popular a UI com estes dados. O utilizador insere de seguida os dados necessários (nome e endereço) e seleciona uma das categorias apresentadas. Estes dados são depois enviados ao servidor que faz a sua persistência através do repositório. No final do processo é apresentada uma mensagem de sucesso ao utilizador.

4.3.4 UC4: Diagrama de Sequência

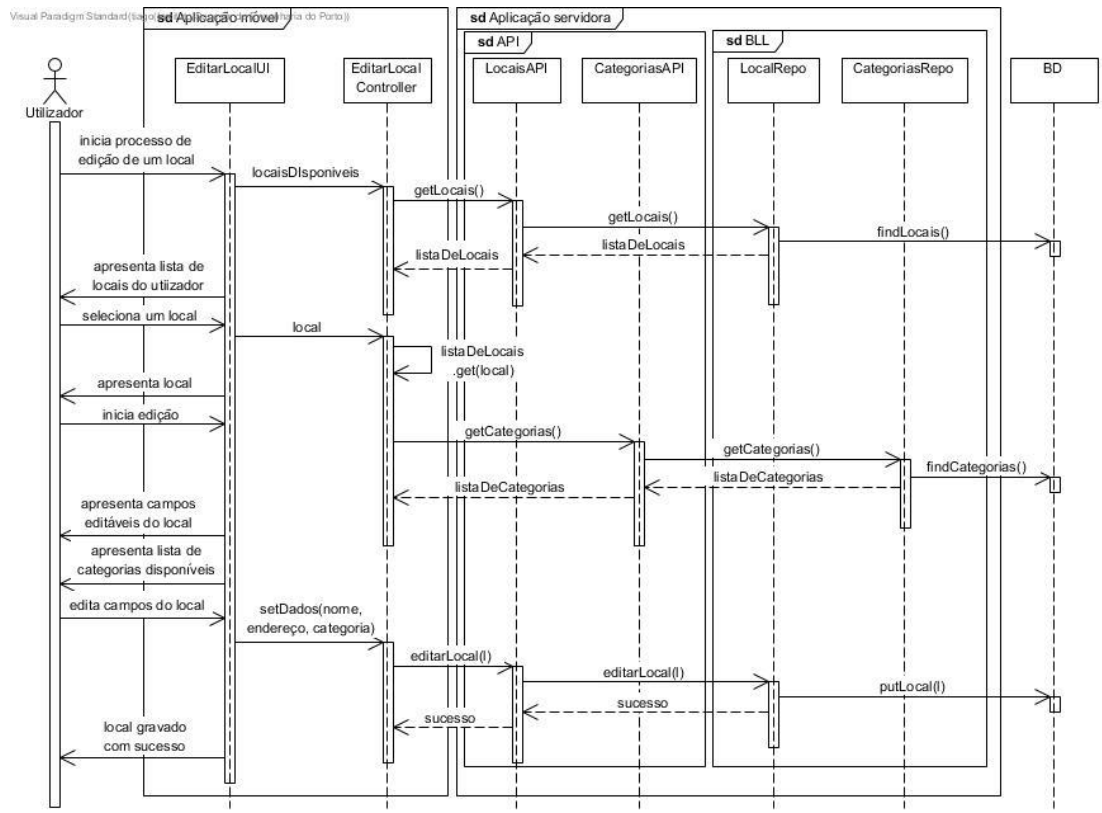

Figura 28 – Diagrama de sequência UC4

É mostrado na Figura 28 o diagrama de sequência do caso de uso relativo à edição de um local existente no sistema. No início do processo é efetuado um pedido a um serviço da aplicação servidora para serem retornados os locais existentes do utilizador. O utilizador seleciona o local que pretende editar. O servidor faz um novo pedido ao servidor para obter a lista de categorias disponíveis e apresenta ao utilizador os campos do local com a informação atual. O utilizador edita os campos pretendidos e confirma. Ao persistir o local com os novos dados, é apresentada uma mensagem de sucesso ao utilizador.

4.3.5 UC5: Diagrama de Sequência

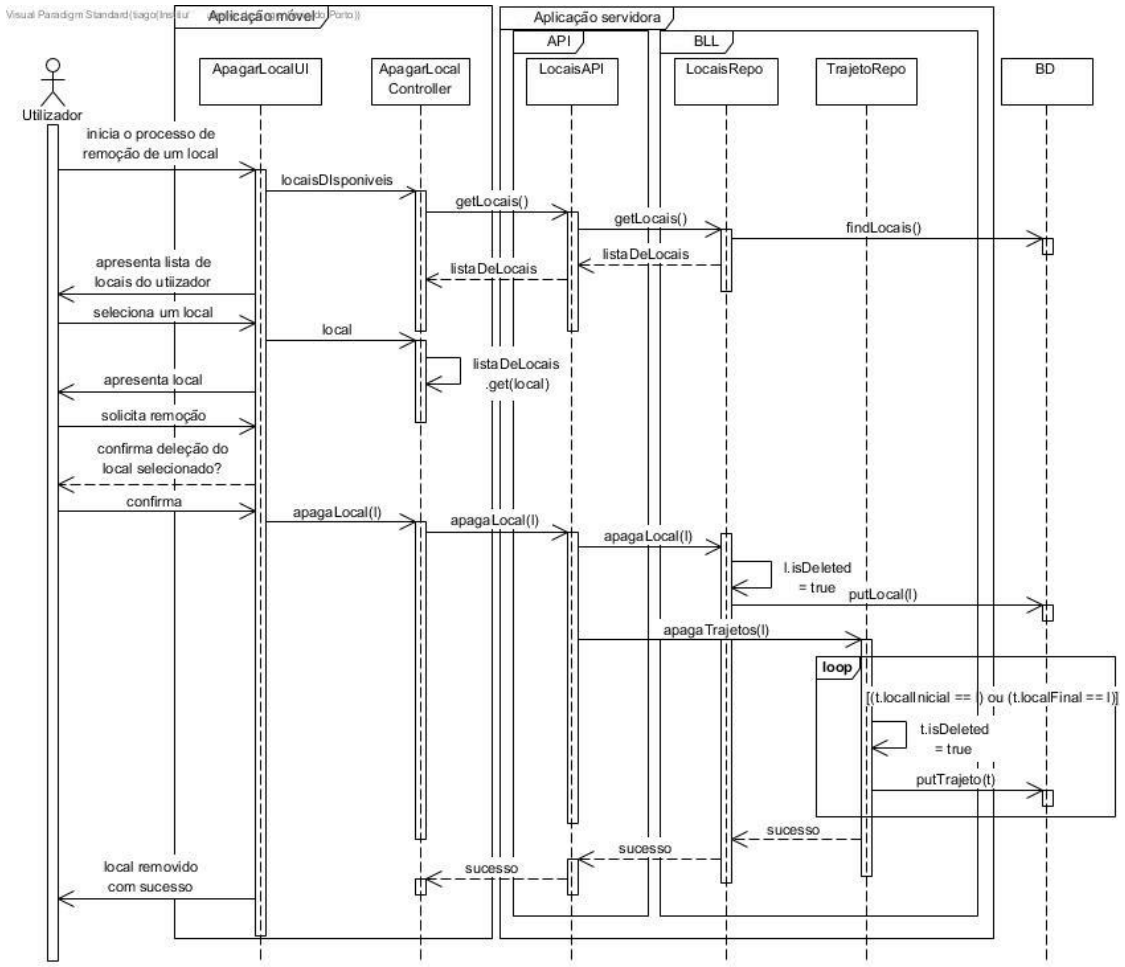

Figura 29 – Diagrama de sequência UC5

Na Figura 29 é apresentado o diagrama de sequência do caso de uso referente à remoção de um local existente no sistema. No início do processo é efetuado um pedido a um serviço da aplicação servidora para serem retornados os locais existentes do utilizador. O utilizador seleciona o local que pretende apagar. O servidor solicita a confirmação da remoção do local selecionado. Ao confirmar é feito um pedido a um serviço da aplicação servidora com a informação do local selecionado que vai alterar o a variável de *soft delete* de modo a remover o local e este é guardado novamente na base de dados, não voltando assim a aparecer na lista de locais do utilizador. Os trajetos associados a este local são assim também removidos do sistema, usando o mesmo processo dos locais. No final deste processo é apresentada uma mensagem de sucesso ao utilizador.

4.3.6 UC6: Diagrama de Sequência

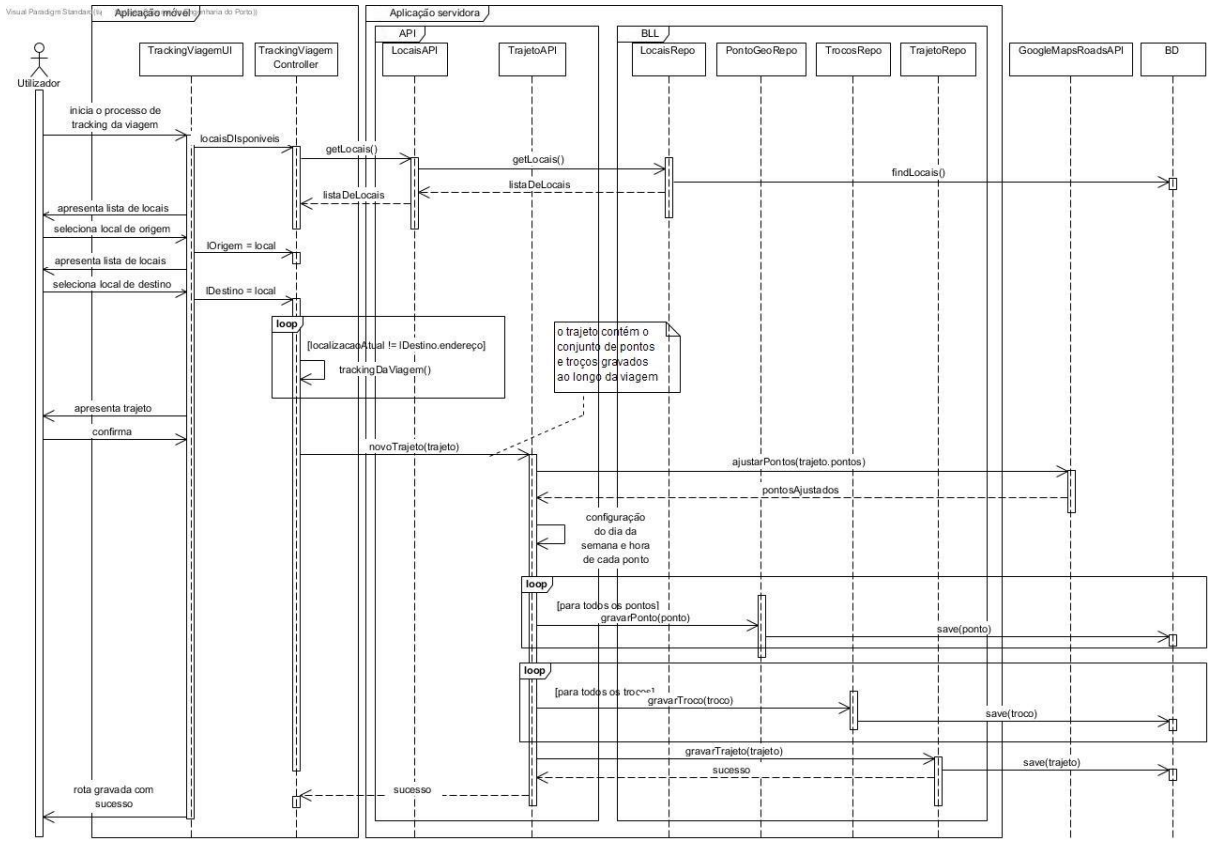

Figura 30 – Diagrama de sequência UC6

Na Figura 30 é apresentado o diagrama de sequência do caso de uso referente ao *tracking* de uma viagem. No início do processo é efetuado um pedido a um serviço da aplicação servidora para serem retornados os locais existentes do utilizador. O utilizador seleciona os locais de origem e destino do seu percurso. A aplicação móvel inicia o *tracking* da viagem do utilizador, gravando os pontos geográficos com um intervalo entre si, e a duração entre esses pontos (troços). Após o utilizador terminar o *tracking* na chegada ao endereço do local de destino é apresentado o trajeto criado. Após a confirmação do novo trajeto este é enviado para o servidor para ser persistido. Os pontos geográficos são depois ajustados com recurso à API da *Google Maps Roads* e é adicionada a informação relativa ao dia da semana e hora atuais. Estes pontos e os troços do trajeto são de seguida persistidos na base de dados e por fim é gravado o trajeto do condutor. No final do processo é apresentada uma mensagem de sucesso ao utilizador.

4.3.7 UC7: Diagrama de Sequência

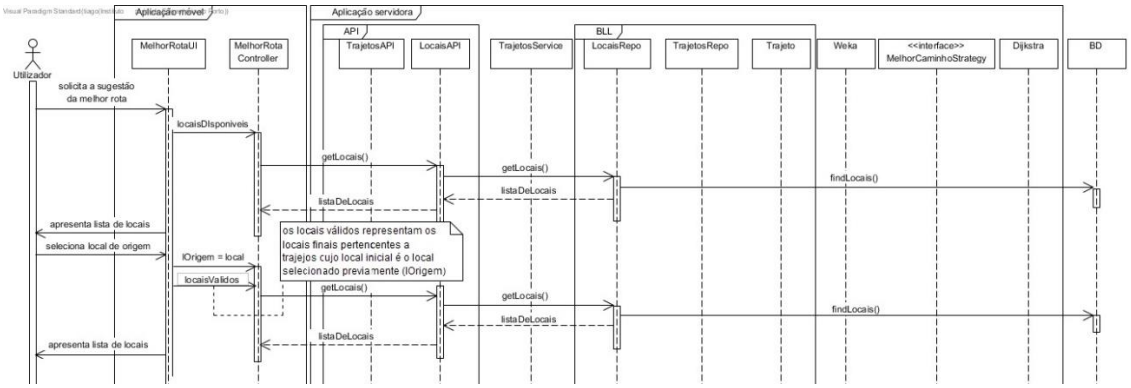

Figura 31 – Diagrama de sequência UC7 (Parte 1)

Na Figura 31 é apresentada a primeira parte o diagrama de sequência do caso de uso relativo à solicitação da melhor rota. No início do processo é efetuado um pedido a um serviço da aplicação servidora para serem retornados os locais existentes do utilizador. O utilizador seleciona o local de origem do seu percurso. De seguida é feito um novo pedido ao servidor de modo a retornar os locais finais pertencentes a trajetos já efetuados pelo utilizador com o local de origem igual ao do local anteriormente selecionado.

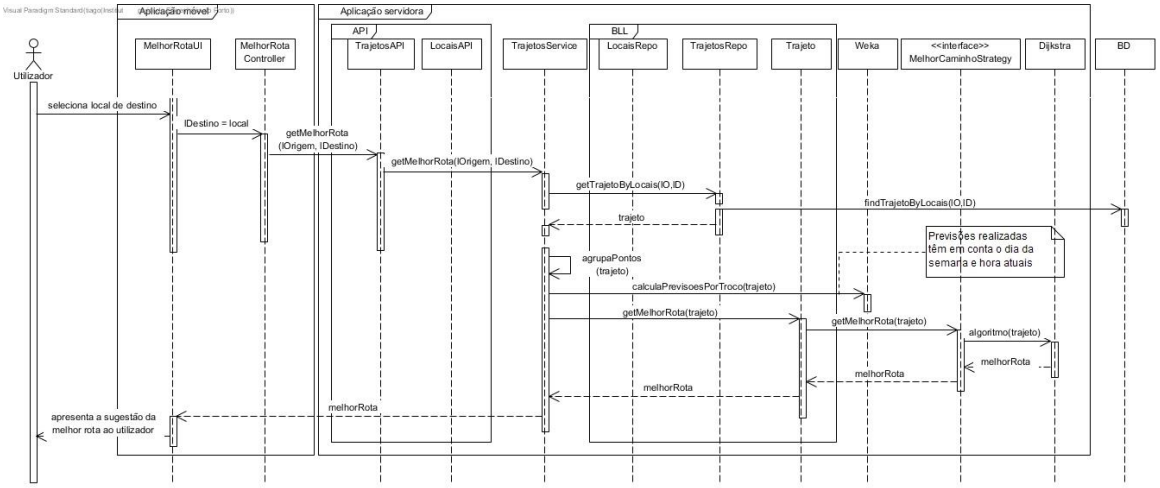

Figura 32 – Diagrama de sequência UC7 (Parte 2)

Na Figura 32 é apresentada a segunda parte do diagrama de sequência deste caso de uso. Após selecionados os locais de origem e destino é usado o repositório de trajetos para encontrar o trajeto do utilizador com os locais enviados. Os pontos geográficos do trajeto encontrado são depois agrupados com base nas suas coordenadas e são calculadas as previsões da duração entre troços com base no dia da semana e hora atuais através do uso do Weka.

Posteriormente a serem calculadas as previsões de todos os troços é de seguida usado o algoritmo de Dijkstra para encontrar a melhor rota. Esta rota é depois apresentada ao utilizador.

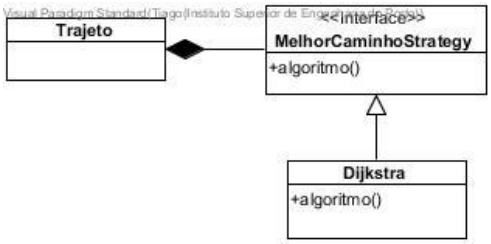

Figura 33 – Diagrama de classes do padrão *Strategy*

De modo a tornar a aplicação extensível no futuro é usado o padrão *Strategy* na aplicação do algoritmo de otimização. Na Figura 33 pode ser observado o diagrama de classes relativos à aplicação deste padrão. O objeto *Trajeto* possui a interface *MelhorCaminhoStrategy* que contém uma função usada para aplicar o algoritmo pretendido ao conjunto de pontos geográficos e troços do trajeto. Esta interface é depois implementada pelos algoritmos de otimização pretendidos, como o atualmente implementado de Dijkstra.

4.3.8 UC8: Diagrama de Sequência

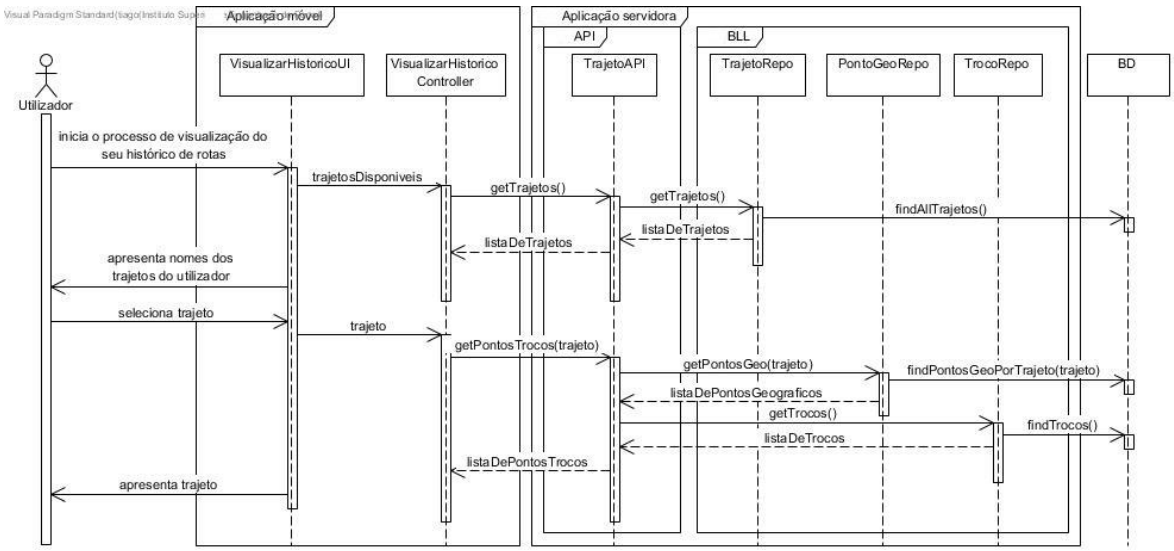

Figura 34 – Diagrama de sequência UC8

Na Figura 34 é apresentado o diagrama de sequência do caso de uso referente ao *tracking* de uma viagem. No início do processo é efetuado um pedido ao servidor que retorna a lista de trajetos do utilizador e são apresentados os nomes dos diversos trajetos. Ao selecionar um trajeto é feito um novo pedido ao servidor que irá encontrar todos os pontos geográficos e troços referentes ao trajeto selecionado. Estes pontos são depois organizados e apresentados no mapa ao utilizador.

4.4 Mockups

Durante o processo de *design* foi também feito um esboço dos componentes principais da camada de apresentação da aplicação móvel. Na primeira imagem da Figura 35 é exibido o menu lateral da aplicação que concede o acesso às diferentes funcionalidades da aplicação. No segundo ecrã é apresentada a vista que o utilizador terá durante o *tracking* da viagem, onde poderá ver a sua localização no mapa. Por fim são expostos os ecrãs que apresentam a lista de locais e a criação de um novo local respetivamente.

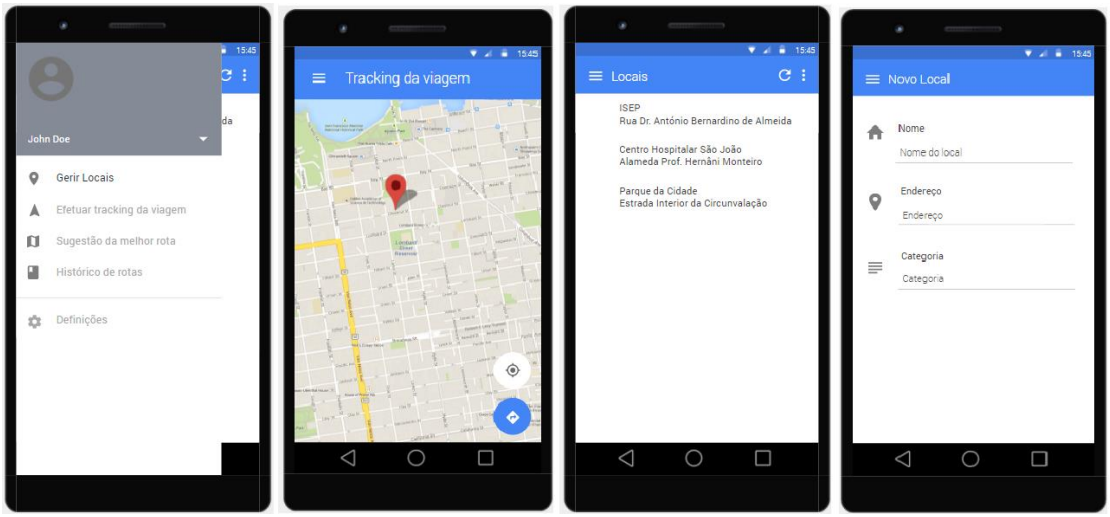

Figura 35 – *Mockups* da aplicação

Após o utilizador se ter registado na aplicação, as funcionalidades disponíveis vão-lhe ser apresentadas num menu lateral apresentado na primeira figura. Caso solicite efetuar o *tracking* da sua viagem irá ser apresentado um mapa, como pode ser visto na segunda figura, com a informação da localização do utilizador. A partir deste mapa o seu trajeto entre a origem e o destino previamente selecionados irá ser desenhado durante a viagem.

O menu lateral apresenta também a opção de gerir locais. Na terceira figura é apresentado o modelo de ecrã para a visualização da lista de locais para o condutor. Ao selecionar um destes locais vão ser apresentadas as opções de edição e remoção do local. Caso o utilizador solicite a criação de um novo local ser-lhe-á apresentado o ecrã presente na última figura com os campos necessários à criação do novo local.

5 Desenvolvimento e Implementação

Neste capítulo será documentado o trabalho elaborado durante o desenvolvimento e a implementação das aplicações abordadas neste projeto. Também serão aqui abordados os detalhes relativos às ferramentas e tecnologias utilizadas, configurações da base de dados e do servidor da implantação. Nas secções 5.1 e 5.2 são descritos detalhes gerais das aplicações de modo a evitar repetição de informação na descrição do desenvolvimento.

5.1 Aplicação Servidora

A parte servidora da aplicação é usada nos processos de persistência e armazenamento dos dados enviados pela aplicação móvel. Oferece também suporte em operações mais complexas que não possam ser executadas nos dispositivos por possuírem uma menor capacidade de processamento.

O servidor foi desenvolvido em Java através do uso da *framework* Spring Boot, desenhada para simplificar o desenvolvimento e construção da aplicação (Woods, 2014). Uma explicação mais detalhada desta *framework* será apresentada de seguida nesta secção.

De modo a gerir as dependências da aplicação e para a automatização dos *builds* foi utilizado o Maven (Maven - Welcome to Apache Maven, s.d.).

Ao executar a aplicação o Spring Boot inicia automaticamente uma instância embutida do *Apache Tomcat* (Apache Tomcat - Welcome!, s.d.) que pode ser facilmente configurada.

5.1.1 Spring Boot

O Spring Boot torna mais fácil a criação de aplicações *stand-alone*, baseadas em Spring (Spring, s.d.) e prontas para produção que podem simplesmente ser executadas.

Esta *framework* traz vantagens ao remover todas as configurações baseadas em XML e fornece anotações como meio de configurar os comportamentos do Spring Framework.

Ao incorporar o Tomcat, Jetty (Jetty - Servlet Engine and Http Server, s.d.) ou Undertow (Undertow - JBoss Community, s.d.) diretamente torna desnecessária a implantação de ficheiros WAR.

Uma das características do Spring Boot consiste no uso de técnicas para definir *beans*, que constituem os "objetos que formam a base da aplicação e que são geridas pelo Spring IoC" (Spring Bean Definition), e as respetivas dependências injetadas. Através do uso da anotação *@ComponentScan* para procurar os *beans* é possível encontrar e registar automaticamente todos os componentes da aplicação existentes. Para efeitos de injeção de construtores é usada a anotação *@Autowired* (Webb, et al.).

5.1.1.1 Camada de Serviço

A camada de serviço "encapsula a lógica de negócio da aplicação, controlando transações e coordenando respostas na implementação das suas operações" (Service Layer).

```
@RestController
@RequestMapping(path = "/api/trajetos")
public class TrajetoController  {
```

Figura 36 - API de trajetos

Para a criação dos serviços REST o *Spring Framework* permite, através da anotação *@RestController*, a criação de controladores para gerir a entrada de pedidos HTTP. Os métodos do serviço são mapeados usando anotações *@RequestMapping* que permitem especificar o tipo de método e o URL de acesso (Webb, et al.). Um serviço de exemplo criado é apresentado na Figura 36.

Quando se torna necessário efetuar pedidos a uma interface remota, estes podem-se tornar dispendiosos o que implica reduzir o número de chamadas e, assim, aumentar a quantidade de dados passados em cada chamada.

A solução é a criação de um objeto de transferência de dados que agregue todos os dados necessários para a chamada e que possa ser serializável.

Embora a principal função do DTO seja agrupar os dados referentes a várias chamadas em uma única chamada, outra vantagem é o encapsulamento do mecanismo de serialização para transferir os dados pela rede. Assim, estes objetos escondem essa lógica do restante código permitindo também a sua fácil manutenção (Data Transfer Object).

```
public class TrajetoDTO {

    private Long ID;
    private String nome;
    private List<PontoGeograficoVM> pontosGeograficos;
    private List<TrocoVM> trocos;
    private Long localInicialId;
    private Long localFinalId;
    private Long utilizadorId;
```

Figura 37 - DTO de trajetos

Na Figura 37, é possível observar o objeto de transferência de dados criado para passar a informação dos trajetos. Como cada Trajeto é constituído por vários objetos foi criado um DTO que armazenasse todos eles e tornasse necessário a execução de apenas uma chamada.

Os mecanismos de estruturação de dados diferem entre os objetos e as bases de dados relacionais, onde coleções e heranças não estão presentes. Assim, quando é necessário construir um modelo de um objeto com muita lógica de negócio envolvida torna-se necessário usar mecanismo que organizem os dados e os comportamentos adjacentes.

"O padrão *Data Mapper* é uma camada de *software* que separa os objetos em memória dos da base de dados" (Data Mapper). A sua responsabilidade reside na transferência de dados entre ambos, isolando-os mutualmente.

```
public LocalVM addLocal(LocalVM localVM) {

...

l = localRepo.save(l);
return modelMapper.map(l, LocalVM.class);
}
```

Figura 38 - Mapeamento de um local

Na Figura 38, é apresentado um excerto de código onde é mapeado um local para o respetivo DTO. Este mapeamento foi executado com o auxílio da biblioteca *ModelMapper* (ModelMapper - Simple, Intelligent, Object Mapping, s.d.) que torna este processo mais simples.

5.1.1.2 Camada de Serviço de Dados

A camada de serviço de dados disponibiliza os serviços de acesso aos dados, encapsulando de um ponto de vista de operações de negócio, o acesso aos dados isolando a camada de negócio (Sousa).

O padrão repositório (DDD) faz a mediação entre o domínio e as camadas de mapeamento de dados, simulando uma coleção de objetos de domínio em memória.

Conceptualmente, o repositório encapsula o conjunto de objetos persistidos numa base de dados e as operações executadas sobre eles, fornecendo uma visão mais orientada a objetos da camada de persistência.

Este padrão suporta também o objetivo de alcançar uma separação mais bem definida e uma dependência unidirecional entre as camadas de domínio e mapeamento de dados (Repository).

```
public interface LocalRepo extends JpaRepository<Local, Long> {

@Query("select local from Local local where local.utilizador.id = ?1")
List<Local> findAllLocaisByUtilizador(Long utilizadorId);
```

Figura 39 - Repositório de locais

O *Spring Boot* também permite uma criação simplificada de repositórios através da anotação *@Repository* e onde apresenta diversas opções de interfaces. Neste projeto foi escolhido usar a interface *JpaRepository* (Gierke & Darimont) que já implementa os métodos básicos de acesso à base de dados. Em métodos mais específicos a *framework* oferece a possibilidade da implementação de métodos que permitem a criação de *queries* usando a linguagem *query* JPA através da anotação *Query* (Gierke & Darimont), como pode ser visto detalhadamente na Figura 39.

```
@Entity
@Table(name = "categoria")
public class Categoria implements Serializable {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long ID;
private String nome;
```

Figura 40 - Classe de modelo de categoria

Nas classes pertencentes ao modelo da aplicação foi usada a anotação *@Entity* para estas serem reconhecidas pela aplicação, como pode ser visto na Figura 40. Através do uso de anotações é ainda possível definir o tipo de estratégia usada para a geração do ID e a definição de relacionamentos com outras entidades.

Quando se torna necessário efetuar pedidos a uma interface remota, estes podem-se tornar dispendiosos o que implica reduzir o número de chamadas e, assim, aumentar a quantidade de dados passados em cada chamada.

A solução é a criação de um objeto de transferência de dados que agregue todos os dados necessários para a chamada e que possa ser serializável.

Embora a principal função do DTO seja agrupar os dados referentes a várias chamadas em uma única chamada, outra vantagem é o encapsulamento do mecanismo de serialização para transferir os dados pela rede. Assim, estes objetos escondem essa lógica do restante código permitindo também a sua fácil manutenção (Data Transfer Object).

5.2 Aplicação Móvel

A aplicação móvel desenvolvida consome os serviços disponibilizados pela aplicação servidora de modo a expor a informação recebida ao utilizador. É também responsável por efetuar o *tracking* das viagens do utilizador através dos diversos sensores incluídos no dispositivo.

Esta aplicação foi desenvolvida em *Java* onde se usou como ambiente de desenvolvimento *Android Studio* (Android Studio and SDK tools, s.d.) que permite desenvolver para a plataforma Android.

Com o intuito de utilizar as APIs desenvolvidas pela Google foram adicionadas configurações na aplicação móvel para integrar estes serviços, assim como as chaves geradas de acesso.

5.3 Casos de Uso

Nesta secção são apresentados os detalhes da implementação de cada caso de uso, onde se descrevem as opções tomadas, as ferramentas usadas e o modo de implementação, complementado por alguns excertos de código e imagens da aplicação desenvolvida.

5.3.1 UC1: Registar Utilizador

Com o objetivo de implementar autenticação na aplicação foi usado o *Spring Security* (Spring Security, s.d.), uma *framework* presente no *Spring Boot* que fornece autenticação e autorização para aplicações Java. Autenticação referindo-se ao processo de identificação dos utilizadores presentes no sistema e autorização o processo que verifica os acessos disponíveis para cada utilizador (Alex, et al., 2017).

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
        .and()
            .authorizeRequests()
            .antMatchers("/api/login").permitAll()
            .antMatchers("/api/registo").permitAll()
            .antMatchers("/api/validar").permitAll()
            .antMatchers("/api/**").hasAuthority("UTILIZADOR")
```

Figura 41 - Configurações de segurança

De modo a garantir a autenticação no acesso aos serviços REST da aplicação servidora foram criadas configurações de segurança personalizadas como se pode ver na Figura 41. Assim, todos os serviços, à exceção do registo, *login* e validação, exigem a autenticação dos utilizadores no acesso aos recursos internos do servidor.

Ao nível da autenticação, o *Spring Security* oferece uma grande variedade de modelos de autenticação. Para este projeto optou-se por usar a autenticação *JSON Web Token* (JWT).

- JWT

"O *JSON Web Token* (JWT) é um padrão aberto (RFC 7519) que define um modo compacto e autónomo de transmitir informação entre as duas partes interessadas como um objeto JSON" (JSON Web Token Introduction).

O processo de utilização começa pelo utilizador se autenticar com sucesso no servidor usando as suas credenciais de acesso. Após a sua autenticação o servidor retorna um *JSON Web Token* que deverá ser guardado localmente. Em todas as restantes comunicações com o servidor este *token* deverá ser enviado no cabeçalho (*header*) da mensagem o que servirá para o servidor validar a autenticação do utilizador e permitir o seu acesso a recursos protegidos.

Além deste tipo de autenticação ter um fluxo simples, tem também a vantagem de reduzir a carga no servidor e permitir uma melhor integração com aplicações móveis.

A implementação deste caso de uso começou pela criação das classes de modelo do Utilizador e respetivas autorizações. Embora na aplicação não seja necessário o uso de autorizações a sua implementação é necessária nesta *framework*.

```java
@Entity
@Table(name = "utilizador")
public class Utilizador implements Serializable {

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

    @Email
    @Size(min = 5, max = 100)
    @Column(length = 100, unique = true)
    private String email;

    @JsonIgnore
    @NotNull
    @Size(min = 60, max = 60)
    @Column(name = "password",length = 60)
    private String password;
```

Figura 42 - Classe de modelo de Utilizador

Na Figura 42, é apresentado um excerto de código da classe modelo do Utilizador. Como credenciais de acesso foi definido um email e uma password. Através do uso das anotações do *Spring* foram também feitas restrições a estes atributos tal como a anotação 'unique = true' na coluna do email o que impõe a sua unicidade.

```java
long now = (new Date()).getTime();

Date validity = new Date(System.currentTimeMillis() + 864_000_000);

return Jwts.builder()
 .setSubject(authentication.getName())
 .claim(AUTHORITIES_KEY, authorities)
 .signWith(SignatureAlgorithm.HS512, secretKey)
 .setExpiration(validity)
 .compact();
```

Figura 43 - Criação do *token* JWT

Algumas classes de configuração para o JWT foram criadas incluindo o componente *TokenProvider* usado na criação e validação dos *tokens*. É possível observar na Figura 43, o método usado na criação de um *token* com informação relativa à sua data de expiração, autorizações e a chave secreta usada para gerar o *hash* da palavra-passe.

Por fim, foi criado um serviço REST para permitir o registo de utilizadores na aplicação móvel. Ao receber um pedido para registar um novo utilizador a sua palavra-passe é cifrada utilizando o método acima descrito e é feita a sua persistência na base de dados.

```
@PostMapping("/login")
public ResponseEntity<JWTToken> authorize(@Valid @RequestBody UtilizadorVM
utilizadorVM) {

UsernamePasswordAuthenticationToken authenticationToken =
   new UsernamePasswordAuthenticationToken(utilizadorVM.getEmail(),
utilizadorVM.getPassword());

        ...

        authentication = this.authenticationManager
                              .authenticate(authenticationToken);

        ...


SecurityContextHolder.getContext().setAuthentication(authentication);
String jwt = tokenProvider.createToken(authentication);
HttpHeaders httpHeaders = new HttpHeaders();
     httpHeaders.add(JWTConfigurer.AUTHORIZATION_HEADER, "Bearer " + jwt);
return new ResponseEntity<>(new JWTToken(jwt), httpHeaders, HttpStatus.OK);
```

Figura 44 - Serviço de *login*

Para se efetuar a autenticação dos utilizadores já registados foi criado um serviço REST de *login*, apresentado na Figura 44. Este serviço recebe as credenciais de acesso do utilizador e valida-as, o que irá gerar e retornar um *JSON Web Token* em caso de sucesso.

Na aplicação móvel existe um mecanismo de validação do *token* do utilizador, caso este exista, que é executado assim que a aplicação é aberta. Caso a validade do *token* se confirme o utilizador é redirecionado para a atividade principal da aplicação, caso contrário é apresentada a atividade de *login*.

Para efetuar o registo na aplicação móvel foi criada uma atividade que permite a introdução das credenciais do utilizador que são enviadas através de um POST ao servidor usando a biblioteca *Volley* presente no *Android*. Caso seja retornada uma mensagem de sucesso o utilizador é notificado e redirecionado para a atividade de *login*.

Na atividade de *login* são solicitadas novamente as credenciais de acesso do utilizador e é feito um novo pedido ao servidor. Em caso de sucesso é retornado um *JSON Web Token* que é guardado na memória do dispositivo através do uso da interface *SharedPreferences* (SharedPreferences) presente no *Android* que permite guardar e editar pares valor-chave com segurança.

5.3.2 UC2: Visualizar um local

A implementação deste caso de uso começou pela criação da classe Local e da classe Categoria, ao qual existe uma associação. Para ambas as classes foram também criados os respetivos DTO's de modo a serem usados pelos serviços REST.

```
public class LocalDTO {

    private Long ID;
    private String nome;
    private String endereco;
    private Long categoriaId;
```

Figura 45 - DTO de locais

Na Figura 45, é apresentado um excerto do DTO de local onde é possível observar os seus atributos. Com o fim de simplificar e reduzir a informação enviada entre o servidor e a aplicação móvel, este DTO apenas necessita do identificador de Categoria, em oposição ao objeto inteiro, que será usado para retornar a categoria selecionada durante a criação de um novo local.

Este caso de uso deve apresentar uma lista de locais por utilizador para permitir a seleção de um local para visualizar. Assim, foi necessário criar um método no repositório do Local que retornasse todos os locais pelo ID do utilizador atual.

```
public interface LocalRepo extends JpaRepository<Local, Long> {

    @Query("select local from Local local where local.utilizador.id = ?1")
    List<Local> findAllLocaisByUtilizador(Long utilizadorId);
```

Figura 46 - Repositório de locais

Na Figura 46 é apresentado o método criado para esse efeito onde foi utilizada a anotação *Query* que permite a introdução de uma *query* personalizada. A este método é passado por parâmetro o ID do utilizador que solicitou a visualização dos seus locais.

Para ser encontrado o identificador do utilizador que efetuou o pedido foi criado um método auxiliar que encontra o seu ID através do *token* enviado no cabeçalho do pedido.

Na aplicação móvel o utilizador ao solicitar a execução deste caso de uso o sistema apresenta a lista de locais com a possibilidade de seleção de um local. Após a seleção de um Local são apresentados os campos do local e as opções para efetuar a sua edição ou remoção.

5.3.3 UC3: Adicionar um novo local

De modo a ser possível ao utilizador ver a lista de categorias existentes no sistema e possibilitar a seleção de um deles durante a criação de um local, foi criado um método GET que retorna a lista de categorias disponíveis para seleção.

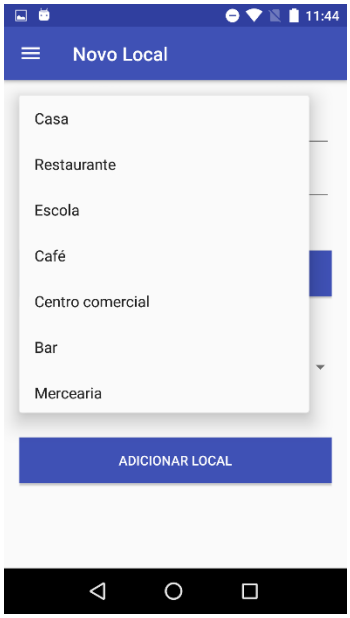

Figura 47 – *Spinner* de categorias

Na aplicação móvel, este serviço é chamado imediatamente após o utilizador solicitar a execução deste caso de uso e é apresentada a lista retornada num *Spinner*, possibilitando a seleção de uma das categorias apresentadas, tal como é apresentado na Figura 47. Foi também criado um método auxiliar para obter as coordenadas do local através do endereço introduzido pelo utilizador.

A criação do novo local é feita através da chamada a um método POST que recebe o DTO do local e o persiste na base de dados do servidor, retornando sucesso no final da execução.

5.3.4 UC4: Editar um local existente

No sentido de implementar este caso de uso foi criado na aplicação servidora um novo método PUT usado para editar locais. Analogamente ao UC3, este método recebe um DTO de Local e efetua a sua persistência na base de dados.

Durante a execução do UC2, o sistema apresenta ao utilizador a possibilidade de editar o Local que está a visualizar. Ao iniciar a sua execução a aplicação apresenta os campos necessários à introdução de um novo Local já editados com os dados do Local selecionado. Após o utilizador editar os campos pretendidos o sistema faz um novo pedido ao servidor para gravar o Local com os novos dados.

5.3.5 UC5: Apagar um local existente

Ao executar o caso de uso 2 (5.3.2), a aplicação móvel apresenta a possibilidade de remover o local em questão.

Para a remoção dos locais foi implementado um *soft delete* em oposição à remoção normal dos objetos. De seguida é apresentada uma explicação de como foi implementado este processo.

Na aplicação servidora foi criado um método DELETE que recebe como parâmetro o identificador do local a remover e efetua o seu *soft delete*, incluindo todos os trajetos associados com esse local.

5.3.5.1 Soft Delete

O *soft delete* no âmbito da base de dados é uma técnica que permite a remoção de um objeto sem remover o seu registo da base de dados. A sua implementação implica a criação de um atributo (*flag*) na tabela existente que indica se um registo foi ou não apagado (Soft Delete). Todas as *queries* que envolvam o fluxo ativo da aplicação são também personalizadas para ignorar entradas com a *flag* ativa (Hedge, 2015).

```
@Entity
@Table(name = "local")
@Where(clause="is_deleted=0")
public class Local implements Serializable {
```

Figura 48 - Cabeçalho da classe de modelo de locais

No Spring Boot foi utilizada a anotação *@Where*, que pode ser observada na Figura 48, que verifica o estado de cada registo, omitindo-o caso se encontre com a *flag* ativa.

A implementação do *soft delete* traz vantagens em termos do desempenho da aplicação, fácil implementação e principalmente a fácil restauração dos dados apagados (Hedge, 2015).

5.3.6 UC6: Efetuar o *tracking* da viagem

A implementação deste caso de uso começou pela criação da classe de modelo do trajeto e do correspondente DTO na aplicação servidora. Para simplificar os pedidos efetuados entre a aplicação móvel e o servidor alguns objetos foram substituídos pelo correspondente ID no objeto de transferência de dados.

Na aplicação móvel foram criados dois fragmentos de modo a permitir ao utilizador selecionar os locais inicial e final respetivamente correspondentes ao trajeto em que pretende efetuar o *tracking*. Após a seleção de ambos os locais é efetuado um pedido ao servidor que verifica a existência de um trajeto já criado com os locais selecionados. Caso o servidor não encontre nenhum trajeto, o sistema solicita a introdução do nome do novo trajeto.

Figura 49 – Interface gráfica do *tracking* da viagem

A Figura 49 apresenta a interface gráfica que a aplicação móvel mostra ao iniciar o *tracking* da viagem onde é exibido um mapa contendo os locais inicial e final marcados e a posição atual do utilizador. Ao longo da viagem são traçados os pontos e troços do trajeto no mapa e são simultaneamente criados os objetos correspondentes aos pontos geográficos e aos troços contendo a duração entre esses pontos.

Na aplicação móvel foi utilizada a *Google Maps API* para permitir obter a localização do utilizador. Para serem obtidas as várias coordenadas relativas à localização do utilizador, para posteriormente serem guardadas como pontos geográficos, foi usada a *FusedLocationProviderClient* (FusedLocationProviderClient). Este serviço estima a localização combinando todos os fornecedores de localização disponíveis, incluindo o GPS e a localização da rede (Location services).

```
mLocationRequest.setInterval(5000);
mLocationRequest.setFastestInterval(5000);
mLocationRequest.setSmallestDisplacement(metersAccuracy);

mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

...

mFusedLocationProviderClient.requestLocationUpdates(mLocationRequest,
mLocationCallback, Looper.myLooper());

mMap.setMyLocationEnabled(true);
```

Figura 50 - Configurações da classe que retorna a localização do utilizador

Assim, foi implementada esta classe utilizando as definições presentes na Figura 50, para receber durante toda a viagem a localização do condutor.

Ao finalizar o *tracking* da viagem é apresentado o trajeto efetuado pelo condutor e outras informações adicionais e é solicitada a confirmação para a gravação do trajeto. Os dados do novo trajeto são depois enviados para o servidor para este ser persistido.

Com o objetivo de melhorar a qualidade dos pontos geográficos do novo trajeto é usada a Google Maps Roads API. Este serviço aborda o problema em que dados imprecisos ou insuficientes causem a criação de trajetos estranhos que transitem por cima de obstruções enquanto o utilizador se encontre em movimento.

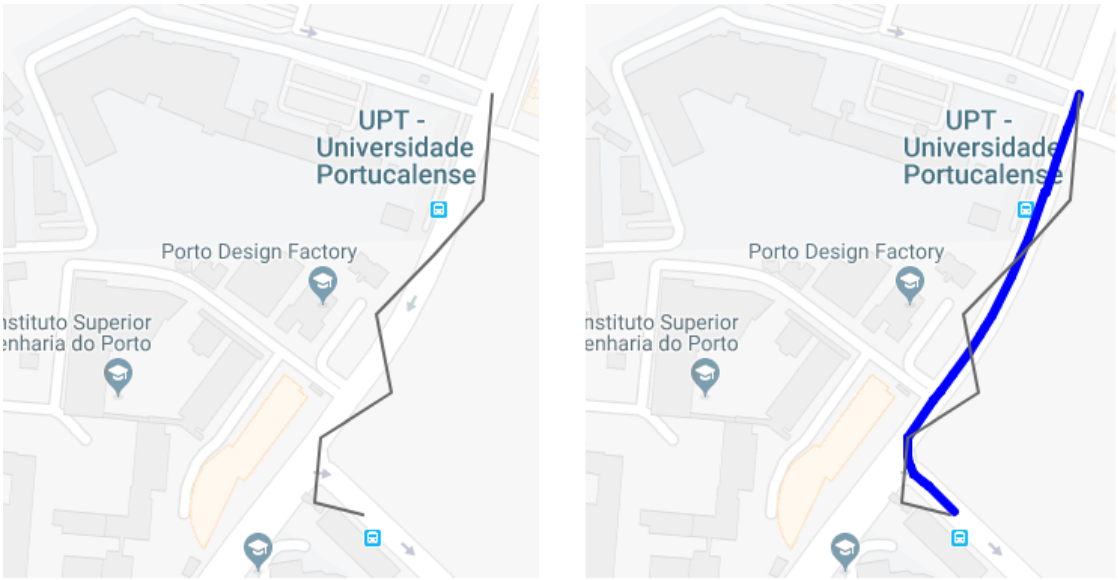

Figura 51 – Exemplo do uso da Google Maps Roads API

Na Figura 51, é apresentado um exemplo da utilização deste serviço onde é observável, no quadro da esquerda, um exemplo de um trajeto guardado que apresenta algumas imprecisões nas coordenadas dos seus pontos geográficos. No quadro da direita é apresentado o resultado do uso da Google Maps Roads API que ajusta os pontos do trajeto à estrada correspondente.

Assim, foi criado um método auxiliar na aplicação servidora para efetuar uma chamada à Google Maps Roads API. Este serviço recebe uma lista de coordenadas referentes aos pontos geográficos agregados do trajeto e retorna uma lista contendo as coordenadas ajustadas e um identificador da rua de cada ponto.

```
RestTemplate restTemplate = new RestTemplate();
SnappedPoints response = restTemplate.getForObject(url,
SnappedPoints.class);

 for(SnappedPoint sp : response.getSnappedPoint()) {

 PontoGeograficoVM ponto = pontosGeograficos.get(sp.getOriginalIndex());

   ponto.setLatitude(sp.getLocation().getLatitude());

   ponto.setLongitude(sp.getLocation().getLongitude());
                                        ponto.setPlaceId(sp.getPlaceId());

 }

}
```

Figura 52 - Chamada ao serviço da Google Maps Roads

A Figura 52 apresenta um excerto do método criado onde é possível observar a chamada à API através de uma classe *template* chamada *RestTemplate* (RestTemplate). Esta classe torna mais fácil a interação com a maior parte dos serviços *RESTful*, permitindo ainda efetuar o *bind* dos dados a classes de domínio personalizadas (Consuming a RESTful Web Service).

Finalmente as informações correspondentes ao dia da semana e hora do trajeto são adicionadas aos pontos geográficos antes de estes serem persistidos.

5.3.7 UC7: Solicitar a sugestão da melhor rota

A fim de encontrar a melhor rota do utilizador para um dado dia da semana e hora é necessário primeiro tratar os dados dos trajetos efetuados previamente pelo utilizador.

Ao iniciar este caso de uso o utilizador começa por selecionar na aplicação móvel o local inicial do trajeto da lista dos seus locais existentes. De seguida é efetuado um pedido ao servidor de modo a obter a lista de locais existentes que pertençam a um trajeto previamente efetuado e com o mesmo local inicial ao selecionado previamente.

Após selecionados os dois locais é feita uma chamada à API da aplicação servidora para se obter o melhor trajeto que irá de seguida ser apresentado no mapa ao utilizador.

No servidor o primeiro passo consiste em encontrar o trajeto com os pontos inicial e final enviados pelo utilizador.

De seguida são apresentadas as várias opções que foram testadas de modo a encontrar a melhor solução no agrupamento dos pontos, que encontra interseções entre as diversas rotas efetuadas do utilizador.

A primeira opção a ser testada incluía no método de agrupamento os identificadores da rua de cada ponto geográfico que são guardados durante o (5.3.6), através da Google Maps Roads API.

Em conjunto com este identificador e de modo a ser possível avaliar a distância entre os dois pontos geográficos, foi criado um método auxiliar com o intuito de calcular a distância entre os dois pontos através dos seus valores da latitude e longitude.

Assim, todos os pontos geográficos presentes no trajeto são comparados entre si e são agrupados caso se verifique a igualdade entre os identificadores da rua e uma distância mínima de 15 metros.

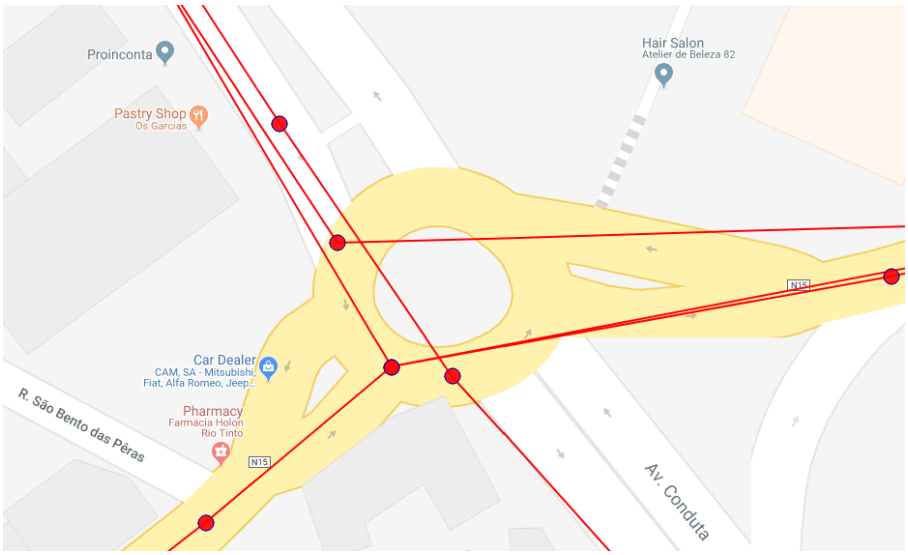

Figura 53 – Pontos geográficos agrupados através do método 1

Através deste método surgiu um problema em zonas de interseção onde os pontos geográficos situavam-se a uma distância menor do que a usada durante a comparação, mas apresentavam diferentes identificadores de rua o que não permitia o seu agrupamento, tal como pode ser visto na Figura 53.

Na segunda opção utilizado foi descartado o uso do identificador da rua e apenas se compara as distâncias entre os pontos geográficos do trajeto. Neste método foram também testadas diferentes distâncias mínimas para efetuar o agrupamento dos pontos de modo a atingir a melhor solução. Visto que os resultados do agrupamento dos pontos geográficos podem variar de trajeto para trajeto, foram efetuados vários testes com trajetos diferentes de modo a obter a melhor solução geral para todos os trajetos.

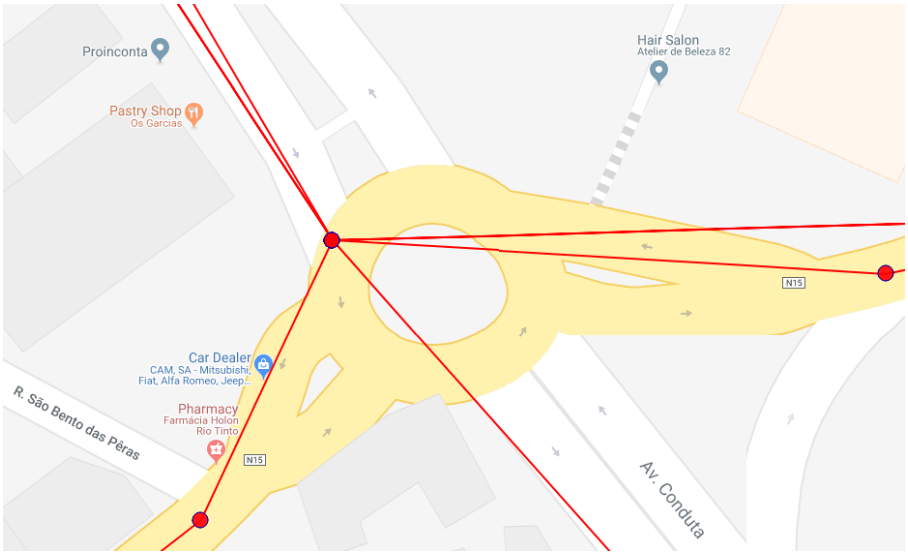

Figura 54 – Pontos geográficos agrupados através do método 2

Na Figura 54, é apresentado um exemplo do agrupamento de pontos usando este segundo método em comparação com a primeira opção tomada.

O passo seguinte após efetuado o agrupamento dos pontos geográficos do trajeto consistiu no cálculo das previsões da duração para cada troço através do uso do Weka.

```
ArrayList<Attribute> attrs = new ArrayList<>();
            attrs.add(new Attribute("duracao"));
            attrs.add(new Attribute("minutos"));
            attrs.add(new Attribute("diaDaSemana"));
            attrs.add(new Attribute("trab/folga"));
```

Figura 55 - Criação dos atributos usados na previsão

Na Figura 55 são apresentados os atributos usados para criar o modelo de regressão. A duração é referente ao tempo demorado pelo utilizador a percorrer a distância entre dois pontos geográficos, o atributo minutos apresenta a hora do dia convertida em minutos, o diaDaSemana exibe um inteiro alusivo ao dia da semana convertido num número de 1 a 7 e o trab/folga evidencia se o trajeto foi efetuado a um dia da semana ou a um sábado ou domingo onde os níveis de trânsito podem variar significativamente.

Após configurados os atributos do modelo percorre-se todos os pontos geográficos do trajeto e são encontrados todos os troços entre cada dois pontos. Com os troços encontrados são criadas instâncias para cada troço para ser possível posteriormente criar o modelo de regressão.

```
RandomForest rf = new RandomForest();
rf.setOptions(Utils.splitOptions("-P 100 -I 500 -num-slots 1 -K 0 -M 1.0 -V
0.001 -S 1"));
rf.buildClassifier(train);

trocosPrevisao.add(new Troco(ponto1, ponto2,
rf.classifyInstance(test.instance(0))));
```

Figura 56 - Criação do modelo de regressão

Na Figura 56 é apresentada a criação do modelo de previsão (usando o algoritmo de regressão *Random Forest*) da duração do troço entre os dois pontos atuais.

No final do ciclo, são encontradas as previsões de todos os troços do trajeto que irão servir para encontrar o melhor caminho através do algoritmo de *Dijkstra*.

5.3.8 UC8: Visualizar o histórico de rotas

Ao solicitar a visualização do histórico de rotas, a aplicação móvel começa por fazer uma chamada ao serviço de trajetos da aplicação servidora que retorna a lista de trajetos efetuados pelo utilizador. Para minimizar o tamanho deste pedido o servidor apenas envia os nomes e identificadores dos trajetos.

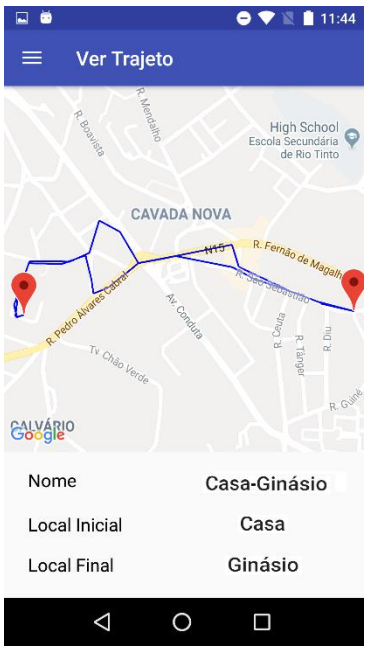

Figura 57 – Interface gráfica da visualização de um trajeto

É dada ao utilizador a possibilidade de escolher um dos trajetos retornados para visualizar as suas informações detalhadas. Ao selecionar um deles é feito um novo pedido ao servidor com

o ID do trajeto selecionado. Posteriormente, a aplicação móvel apresenta a informação do nome, locais iniciais e finais e os pontos e troços marcados no mapa como pode ser visualizado na Figura 57.

5.4 Implantação

Assim que a aplicação servidora atingiu um estado estável foi efetuada a sua implantação no servidor Microsoft Azure.

O primeiro passo consistiu em criar um novo servidor no portal Microsoft Azure para ser posteriormente implantada a aplicação. De seguida, foi criado um recurso *Web App* que foi configurada a versão *Java* da aplicação, o servidor *Tomcat*, autorizações, entre outras configurações necessárias para a implantação.

Finalmente, foi criado um ficheiro JAR da aplicação após serem executados e validados todos os testes desenvolvidos e foi adicionado o ficheiro a este recurso.

Após implantada a aplicação servidora o URL gerado pelo *Azure* foi configurado na aplicação móvel para aceder aos seus serviços.

6 Testes e resultados

Neste capítulo são documentados todo o tipo de testes efetuados durante o desenvolvimento das aplicações. De seguida é apresentada a avaliação da solução onde são identificadas as métricas usadas, a metodologia usada na avaliação e os resultados obtidos.

6.1 Testes

Com o objetivo de garantir a qualidade da solução foram criados vários testes ao longo do desenvolvimento deste projeto. Entre estes testes destacam-se os testes de unidade, testes de integração e testes funcionais.

Adotou-se uma estratégia incremental nesta fase, começando com os testes unitários, avançando para os testes de integração e culminando nos testes funcionais que exercitam a totalidade do sistema desenvolvido. Nas próximas secções são detalhadas cada uma das classes de teste.

6.1.1 Testes Unitários

Na literatura de *software* os testes unitários são normalmente definidos como uma unidade com a menor quantidade de código possível que possa ser adequadamente testada (Parkin).

Estes testes são normalmente vistos como um teste de “caixa branca”, isto é, um método que testa as estruturas internas, em vez de avaliar a conformidade com algum conjunto de requisites (Parkin).

Os testes unitários são fundamentais em sistemas com algum nível de complexidade, onde se torna ineficiente e inefetivo testar o sistema total como uma “grande caixa preta” (Parkin).

Devido à grande importância existente no correto funcionamento das API's criadas na aplicação servidora foram efetuados testes a todos os serviços desenvolvidos.

```
@RestController
@RequestMapping(path = "/api/locais")
public class LocalController {

   @Autowired
   private LocalService localService;

   @GetMapping()
   public List<LocalVM> findAll() {
           return localService.getLocais();
   }
```

Figura 58 - API de locais

De seguida é apresentado um exemplo de um teste unitário criado neste projeto. Na Figura 58 está presente o serviço usado para retornar os locais existentes do utilizador que se pretende testar.

```
@WebMvcTest(controllers = {LocalController.class}, secure=false)
public class LocalControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private LocalService localService;

    ...

    @Test
    public void givenLocais_whenGetLocais_thenReturnJsonArray()
      throws Exception {


          Mockito.when(categoriaRepo.findOne((long)1))
                           .thenReturn(new Categoria((long)1, "Casa"));

        LocalVM localVM = new LocalVM((long)1, "LocalTest",
                                      "EnderecoTest", (long)1, "Casa");

        List<LocalVM> allLocais = Arrays.asList(localVM);

        Mockito.when(localService.getLocais()).thenReturn(allLocais);

        mvc.perform(get("/api/locais")
          .contentType(MediaType.APPLICATION_JSON))
          .andExpect(status().isOk())
          .andExpect(jsonPath("$[0].nome", is(localVM.getNome())));
    }
```

Figura 59 - Teste à API de locais

De modo a criar um teste focado apenas no código da API foi criado um *mock*[4] para cada classe de serviço dependente. Na Figura 59 é apresentado um excerto do teste criado onde pode ser observada a anotação *@MockBean* que fornece uma implementação *mock* do serviço de locais e dos repositórios necessários para o funcionamento da API.

A anotação *@WebMvcTest* presente cabeçalho no teste criado auto configura a infraestrutura MVC para os testes unitários tratando também da configuração do *MockMvc*, o que oferece uma maneira simples de testar as API's sem criar um servidor completo HTTP (Testing in Spring Boot, 2018).

No método apresentado é testado o método GET, no entanto este pode ser substituído por outros existentes conforme seja necessário.

6.1.2 Testes de Integração

"Os testes de integração determinam se as unidades de *software* desenvolvidas de forma independente funcionam corretamente quando conectadas entre si" (Fowler, Integration Test, 2018).

Após se efetuarem os testes unitários, que testam os módulos isoladamente, o próximo passo são os testes de integração onde vários módulos são combinados, seja em todo o sistema ou em subsistemas significativo, com o objetivo de testar se estes funcionam juntos como esperado.

Como os testes de integração dependem da camada de acesso aos dados, como o JPA, é necessário criar uma base de dados de menor dimensão e mais rápida, diferente da usada em produção.

Assim, foi configurada uma base de dados H2 (H2 Database Engine, s.d.) que persiste os dados em memória de modo a executar isoladamente estes testes.

[4] "The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing (Fowler, Mocks Aren't Stubs, 2007).

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class LocalRepoTest extends ApplicationTests {

@Autowired
private TestEntityManager entityManager;

    ...

@Test
public void returnLocaisByUtilizador() {

//persisted on startup

// given
Utilizador utilizadorTest = new Utilizador((long)1, "util@teste.com",
"passwordteste");

Local local1 = new Local ((long)1, "LocalTeste 1", "EnderecoTest 1",
categoriaRepo.findOne((long)1), utilizadorTest);
Local local2 = new Local ((long)2, "LocalTeste 2", "EnderecoTest 2",
categoriaRepo.findOne((long)1), utilizadorTest);

// when
List<Local> locaisRetornados =
localRepo.findAllLocaisByUtilizador((long)1);

// then
List<Local> listaExpectavel = new ArrayList<>();
listaExpectavel.add(local1);
listaExpectavel.add(local2);

assertThat(locaisRetornados, is(listaExpectavel));
```

Figura 60 - Teste de integração

Na Figura 60 é apresentado um excerto de um teste de integração desenvolvido. De modo a criar uma ponte entre as características de teste do Spring Boot e do JUnit foi adicionada a anotação *@RunWith(SpringRunner.class)* para esse efeito e para fornecer uma configuração padrão necessária para os testes da camada de acesso aos dados é usada a anotação *@DataJpaTest* (Testing in Spring Boot, 2018).

Para efeitos de teste são criados automaticamente alguns registos na base de dados ao iniciar cada teste de integração que são posteriormente usados para verificar o correto funcionamento dos repositórios.

6.1.3 Testes Funcionais

Os testes funcionais asseguram que todos os aspetos do *software* funcionam corretamente e de acordo com os requisitos do projeto (Angerer, 2016). Durante estes testes são inseridos dados válidos e inválidos de modo a validar o correto funcionamento da aplicação

Para garantir que os requisitos inicialmente propostos eram atingidos foram planeados os testes que seriam necessários efetuar na aplicação móvel para validar o seu correto funcionamento. Como as ferramentas existentes para desenvolver testes em Android apresentam uma elevada complexidade de implementação e a solução desenvolvida contém interfaces gráficas bastante complexas, os testes foram executados manualmente num ambiente semelhante ao do consumidor final.

As validações contínuas e os testes realizados em simultâneo com o desenvolvimento permitiram validar e identificar erros durante a extensão deste projeto.

6.2 Avaliação da solução

Com o objetivo de avaliar a solução desenvolvida, nomeadamente a qualidade das previsões por troço e a aplicação do algoritmo de Dijkstra para obter a melhor rota, foi escolhida uma rota aleatória para armazenar os vários trajetos do utilizador em dias e horas diferentes para posteriormente testar o funcionamento da aplicação e avaliar os resultados obtidos.

6.2.1 Métricas

Nesta secção apresentam-se algumas das métricas mais utilizadas para avaliar os resultados da aplicação de algoritmos de regressão aos dados de teste.

Supondo que existem *N* pontos de dados de teste. O erro entre a duração prevista e a real para a i-ésima amostra é representado por $e_i = \hat{y}_i - y_i$. As métricas usadas para medir o desempenho do modelo são:

*Mean absolute error (*erro absoluto médio) – quantidade usada para medir a aproximação das previsões aos valores verdadeiros. Neste contexto irá fornecer a diferença entre cada previsão da duração e do valor real atingido. Caso os valores se apresentem espalhados em redor da média, a variação será menor do que o erro absoluto médio, caso contrário o quadrado da distância levará a variações maiores (Weka Error Measurements).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} e_i \qquad (1)$$

Root mean squared error (raiz quadrada do erro quadrático médio) – mede as diferenças entre os valores previstos por um modelo ou um estimador e os valores observados. Representa o desvio padrão da amostra das diferenças entre valores previstos e valores observados. Agrega as magnitudes dos erros das previsões em vários momentos numa única medida de poder preditivo (Weka Error Measurements).

É considerada uma boa medida de precisão, mas apenas na comparação de erros da previsão de modelos diferentes para uma variável específica e não entre variáveis, visto que é dependente da escala (Weka Error Measurements).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2} \quad (2)$$

Relative absolute error (erro relativo absoluto) – este erro é relativo ao preditor simples, que é apenas uma média dos valores atuais. Este erro apresenta o erro total absoluto em vez do erro quadrático total, por isso, o erro relativo absoluto normaliza o erro total absoluto ao dividi-lo pelo erro total absoluto do preditor simples (Relative Absolute Error).

$$RAE = \frac{\sum_{i=1}^{N}|\widehat{e_i} - e_i|}{\sum_{i=1}^{N}|\overline{e_i} - e_i|} \times 100 \quad (3)$$

Root relative squared error - este erro normaliza o erro quadrático total ao dividi-lo pelo erro quadrático total do preditor simples. Ao aplicar a raiz quadrada do erro quadrático relativo reduz-se o erro às mesmas dimensões das da quantidade a ser prevista (Root Relative Squared Error).

$$RRSE = \frac{\sum_{i=1}^{N}(\widehat{e_i} - e_i)^2}{\sum_{i=1}^{N}(\overline{e_i} - e_i)^2} \times 100 \quad (4)$$

6.2.2 Comparação de algoritmos

Durante a fase de desenvolvimento apresentada na secção 5.3.7, foi necessário escolher o algoritmo mais apropriado para efetuar as previsões de cada troço. Assim, foram comparados algoritmos com base no estudo feito durante a fase de análise apresentada na secção 2.1.1.

Para a comparação dos algoritmos foi criado um *dataset* de teste composto por 18 instâncias, com dados aproximados a uma situação real de previsão para um troço de um trajeto, com o objetivo de avaliar o comportamento e precisão de todos os algoritmos de forma idêntica.

Como metodologia de avaliação foram utilizados, em todos os exemplos, 70% dos dados como dados de treino e os restantes 30% para teste.

Tabela 3 – Erros relativos obtidos para os algoritmos testados

	MAE	**RMSE**	**RAE**	**RRSE**
Linear Regression	2.22	3.70	49.62	60.64
SVM	2.63	4.19	58.82	68.65
Random Forest	0.38	0.54	9.00	9.44
M5P	1.75	2.87	42.13	50.36

Na Tabela 3, são apresentados os resultados obtidos no modelo de regressão para cada algoritmo e os respetivos erros. Para este teste foram usados os parâmetros padrão de cada algoritmo que são apresentados de seguida.

No algoritmo *Linear Regression* (LinearRegression) foi usado o método de seleção de atributos M5 (-S 0), um valor de *ridge* de 1.0E-8 (-R 1.0E-8) e 4 casas decimais (-num-decimal-places 4). No algoritmo M5P foram utilizadas 4 instâncias mínimas por folha e manteve-se desativado o uso de árvores não podadas e previsões não suavizadas (M5P).

No algoritmo SVM (SMOreg) foi utilizada a constante de complexidade 1 (-C 1.0), um filtro de normalização dos dados de treino (-N 0), a classe RegSMOImproved para resolver o problema de otimização quadrática (-I RegSMOImproved), o valor de 0.001 como parâmetro de tolerância para verificar o critério de paragem (-T 0.001), um épsilon para erro de arredondamento de 1.0E-12 (-P 1.0E-12), um parâmetro épsilon na função de perda insensível a épsilon de 0.001 (-L 0.001), um número aleatório de *seed* 1 (-W 1) e o *kernel* PolyKernel (-K PolyKernel).

No algoritmo *Random Forest* (RandomForest) foi utilizado um modelo (*bag*) de tamanho 100 (-P 100), 100 árvores (-I 100), 1 *slot* de execução (-num-slots 1), 1 instância por folha (-M 1.0), uma proporção mínima para a divisão de 0.001 (-V 0.001) e o valor 1 como *seed* de geração de números aleatórios (-S 1).

Com base nestes resultados, verificou-se que para o conjunto de dados usados na previsão da duração de cada troço o algoritmo *Random Forest* teve o melhor desempenho, por apresentar um menor valor de erro de meio segundo (0.37) em comparação com as restantes opções e um erro relativo absoluto também bastante baixo (9%).

Após escolher o algoritmo mais adequado efetuaram-se algumas afinações aos hiperparâmetros com o objetivo de reduzir o erro nas previsões. Após vários testes serem efetuados verificou-se que apenas o número de árvores da floresta (parâmetro numIteractions) baixava os erros dos resultados obtidos. Para este parâmetro, vários valores foram testados pois embora um elevado número de árvores melhore as previsões obtidas, a adição de árvores causa um considerável aumento no tempo de processamento (Fraj, 2017).

Assim, ficou definido como o algoritmo de previsão o *Random Forest* com 500 árvores (-I 500), onde se obteve um erro médio absoluto de 0.34 e um erro relativo absoluto de 8.32%, que evidencia valores mais baixos do que os previamente obtidos.

6.2.3 Dataset

O armazenamento dos diferentes trajetos foi efetuado através da aplicação móvel desenvolvida. A recolha foi feita em dias diferentes e no mesmo intervalo de horas de modo a permitir uma melhor leitura e avaliação dos resultados. Alguns destes trajetos foram repetidos várias vezes para aumentar o número e variância dos dados recolhidos.

Na escolha dos locais inicial e final do trajeto de teste foram tidos em conta dois fatores:

- A diversidade das rotas distintas possíveis;

- O número de interseções possíveis entre os vários trajetos.

Isto torna viável a aplicação do algoritmo de *Dijkstra* após efetuadas as previsões para cada troço.

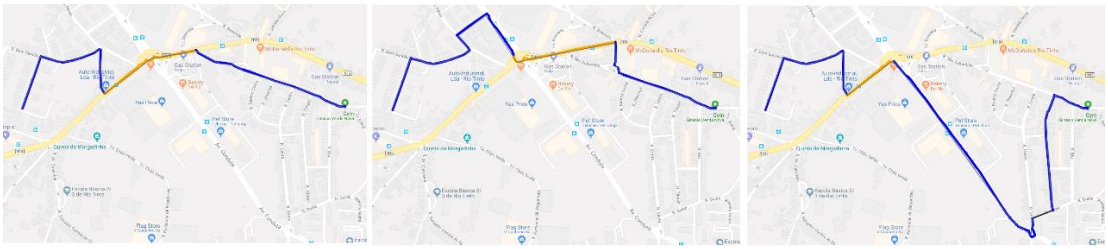

Figura 61 – Exemplos de trajetos possíveis entre os locais inicial e final

Na Figura 61, são apresentados exemplos de trajetos entre os dois locais escolhidos para efetuar a avaliação da solução. Nos vários quadros estão presentes alguns dos trajetos possíveis na deslocação entre o local de origem e destino. No entanto, outras rotas distintas das que são exibidas foram guardadas.

Como já foi explicado na secção 765.3.7 o conjunto de dados armazenados para efetuar posteriormente a previsão tem os atributos: duração do troço, hora do dia (convertido em minutos), dia da semana (convertido no inteiro correspondente entre 1 e 7) e a informação se o dia do armazenamento corresponde a um dia de trabalho ou a um fim de semana (convertido num inteiro de 0 ou 1).

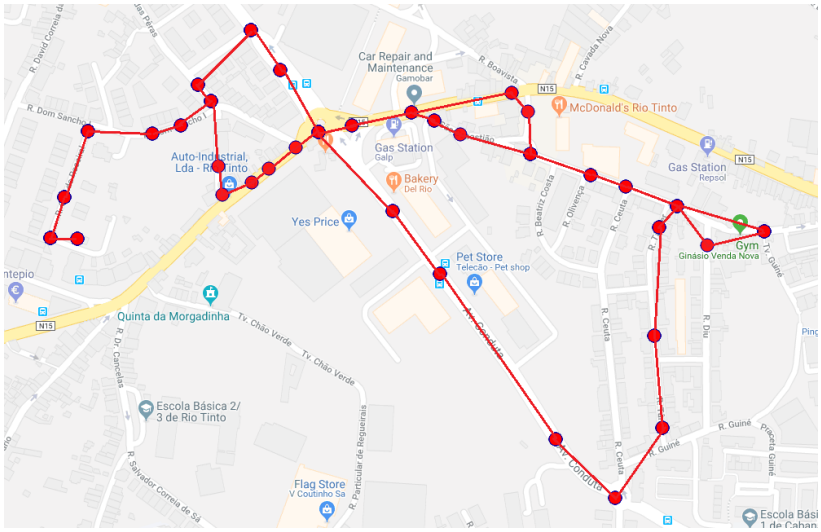

Figura 62 – Conjunto dos trajetos recolhidos com os seus pontos geográficos agrupados

No total foram efetuados 12 trajetos o que agrega um total de 207 pontos e 226 troços. Como já foi referido na secção 5.3.7, o tratamento inicial dos dados consistiu em efetuar o agrupamento dos pontos geográficos, produzindo o resultado apresentado na Figura 62.

No resultado obtido observou-se 31 pares de pontos geográficos com vários troços em comum, que são usados para calcular a previsão da duração de cada um deles.

6.2.4 Resultados

No conjunto de dados foram encontrados troços singulares onde o cálculo da previsão não era possível e, por isso, não foram considerados para avaliar a regressão.

Em todas as previsões foi utilizado o algoritmo *Random Forest*, como foi explicado na secção 6.2.2, e como metodologia de avaliação foram usados 70% dos dados para treino do modelo e os restantes 30% para teste porque, em testes iniciais, mostrou ter um bom desempenho perante este conjunto de dados.

Tabela 4 – Média de erros obtida nas previsões

Modelo	**MAE**	**RMSE**	**RAE**	**RRSE**
Troço	1.03	1.33	37.20	46.15

Assim, foi calculada a média de erros obtidos na previsão de cada troço obtendo-se os resultados apresentados na Tabela 4. A média apresentada refere-se à previsão de 31 troços no total. No geral, a solução desenvolvida revela uma alta precisão nas previsões, com um erro máximo absoluto baixo (1 segundo).

Tabela 5 – Erros das previsões de cada troço

Troço	Nº de Instâncias	MAE	RMSE	RAE	RRSE
1	4	0.70	0.82	16.52	18.65
2	4	2.02	2.31	26.93	27.87
3	4	0.67	0.70	13.40	13.93
4	4	0.73	0.80	13.90	15.24
5	4	0.86	0.97	21.40	22.93
6	4	0.67	0.70	13.40	13.93
7	4	0.79	0.95	14.36	17.18
8	5	0.95	1.29	37.95	49.95
9	4	1.87	2.12	24.08	25.34
10	7	0.87	1.22	49.39	52.69
11	3	0.76	0.93	65.51	72.10
12	2	0.99	1.40	49.60	70.15
13	5	1.49	2.03	21.22	26.21
14	2	0.51	0.73	49.60	70.15
15	3	0.35	0.43	39.80	46.78
16	5	0.92	1.07	24.03	25.88
17	4	0.54	1.07	28.64	49.43
18	6	2.93	3.61	54.88	52.89
19	4	1.53	1.84	87.17	89.99
20	6	0.29	0.40	6.10	7.94
21	4	2.99	3.88	46.94	51.18
22	4	2.02	2.31	26.93	27.87
23	3	1.33	1.87	49.97	63.58
24	2	0.00	0.00	49.60	70.15
25	2	1.24	1.75	49.60	70.15
26	2	0.74	1.06	49.60	70.15
27	3	0.30	0.51	33.45	53.89
28	2	1.24	1.75	49.60	70.15
29	3	1.30	1.59	65.04	73.96
30	2	0.00	0.00	49.60	70.15
31	10	0.43	1.21	24.00	40.24

Embora este seja o resultado geral, verificou-se alguma variância nas previsões individuais devido ao número de instâncias usadas em cada uma delas, como pode ser visto na Tabela 5, o que afeta também o coeficiente de correlação. Um valor também relativamente baixo é observado no erro relativo absoluto (37,20%) o que vem confirmar a boa precisão nas previsões efetuadas.

Posteriormente a analisar os resultados obtidos nas previsões e para testar o bom funcionamento da solução foram solicitadas previsões em dias diferentes.

Ao analisar os dados recolhidos observou-se uma menor duração num dos trajetos efetuados ao domingo em comparação com os restantes dias da semana. Assim, a primeira previsão foi efetuada a uma segunda-feira e a segunda previsão a um domingo.

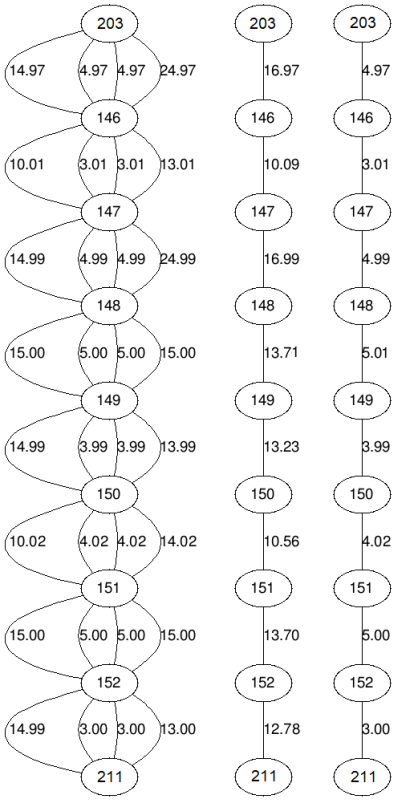

Figura 63 – Durações dos troços do trajeto em formato de um grafo

Na Figura 63, são apresentados três excertos dos dados no formato de um grafo. No quadro da esquerda são apresentados todos os troços entre os pontos que foram agrupados onde é possível observar as várias durações obtidas em dias diferentes. No quadro seguinte, tem-se a previsão efetuada para o primeiro dia (segunda-feira) e por fim a previsão da duração dos troços a um domingo.

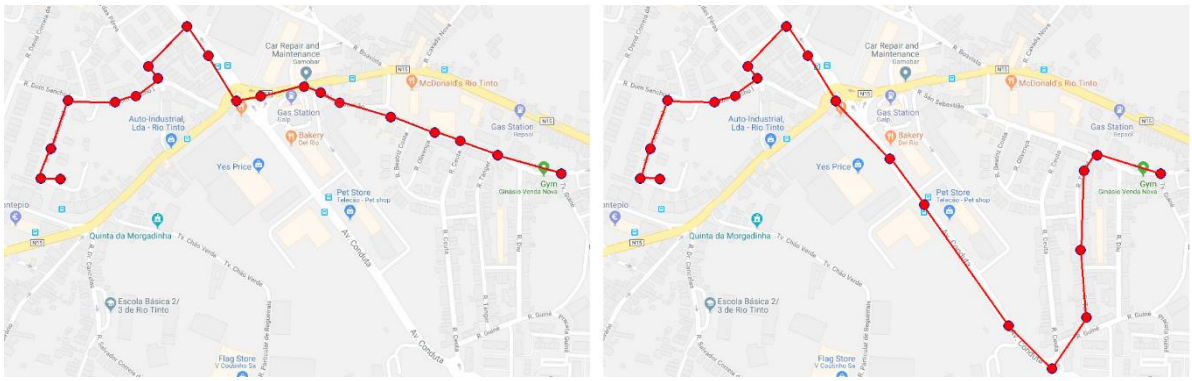

Figura 64 – Sugestão do melhor trajeto a uma segunda e a um domingo

Pode observar-se com clareza a variação da duração dos troços em dias distintos. Neste caso, pelo facto de ao domingo não existir tanto trânsito, o trajeto efetuou-se com maior rapidez. Por fim, os melhores trajetos solicitados são apresentados na Figura 64, onde para o mesmo conjunto de dados são sugeridos dois trajetos distintos em dias da semana diferentes.

7 Conclusão

Neste capítulo são apresentadas as conclusões desta dissertação. Na secção 7.1 é efetuada uma descrição sucinta do trabalho desenvolvido, realçando os objetivos atingidos. As limitações da solução desenvolvida e as ideias de trabalho futuro são descritas na secção 7.2.

7.1 Síntese e Objetivos Alcançados

O presente projeto surgiu da intenção de melhorar o processo de planeamento de rotas após verificar a existência de um aumento do tempo despendido nas deslocações de automóvel. O congestionamento resulta do crescente número de carros vendidos e ao facto da maioria da população viver em áreas urbanas. Embora exista um vasto leque de aplicações que através de diferentes métodos e técnicas tentem minimizar este problema, verificou-se uma limitação comum às soluções que consistia nas sugestões não se basearem nos trajetos e características de cada condutor.

Numa primeira fase foram analisadas as várias APIs disponibilizadas pela Google, que seriam necessárias durante o desenvolvimento, assim como outras ferramentas relevantes de modo a escolher a mais apropriada para o projeto. Na fase final da investigação foram também

estudadas técnicas de *data mining* para regressão e foi ainda estudado o problema do caminho mais curto.

Durante a fase de *design* foram documentados todos os casos de uso funcionais e não funcionais da solução, comparou-se diferentes abordagens arquiteturais e foram construídos diferentes modelos de apoio à fase de implementação.

A solução foi construída na fase de implementação e desenvolvimento em simultâneo com vários testes que garantiram a qualidade do sistema desenvolvido.

Finalmente, as aplicações desenvolvidas foram devidamente avaliadas através de um conjunto de dados de teste e respetivas métricas e os resultados obtidos foram analisados.

Os objetivos propostos inicialmente para este projeto foram atingidos, tendo sido ainda acrescentados novos objetivo no decorrer do mesmo.

Como inicialmente proposto, foram desenvolvidas uma aplicação móvel e uma aplicação servidora. A aplicação móvel desenvolvida permite a criação, edição e remoção dos locais de cada utilizador. Estes locais são criados para marcar o início e fim de cada trajeto.

O *tracking* de viagens permite ao condutor armazenar um conjunto de coordenas geográficas do seu percurso que são depois enviadas para o servidor. Estas viagens podem depois ser consultadas no histórico de viagens.

Para devolver ao utilizador o melhor trajeto entre um par origem-destino são feitas as previsões das durações entre cada troço, usando um modelo de regressão aprendido, e é aplicado de seguida o algoritmo de *Dijkstra* que encontra o trajeto mais rápido.

A aplicação servidora desenvolvida trata de fatores como a segurança, assim como da persistência e armazenamento de locais e trajetos de cada utilizador. É também responsável por efetuar o cálculo do melhor trajeto através de técnicas de *data mining*, enviando o resultado para a aplicação móvel.

7.2 Limitações e Trabalho Futuro

Concluindo o trabalho e observando numa perspetiva autocrítica construtiva, constatam-se as seguintes limitações e potenciais necessidades de desenvolvimento futuro:

- No *tracking* da viagem do utilizador apesar do atual meio de guardar os vários pontos e troços durante a viagem ser eficaz, o desenvolvimento de um método que permitisse uma distância mais consistente entre pontos iria permitir um melhor resultado no agrupamento de pontos do trajeto.

- O método criado para agrupar os pontos geográficos antes do cálculo da previsão embora seja eficaz não é o mais eficiente e, por isso, um trabalho futuro seria implementar um método que apresentasse menor tempo de execução o que diminuiria o tempo de espera do utilizador.

- Em relação ao cálculo do melhor trajeto entre dois locais com base no dia da semana e hora do dia, seria interessante usar os trajetos persistidos de todos os utilizadores do sistema. Isto iria fornecer uma maior quantidade e diversidade dos dados usados no cálculo das previsões.

- No cálculo da melhor rota, acrescentar atributos como o estado do tempo, ocorrência de acidentes, ou obras na estrada, poderia melhorar o modelo de previsão de cada troço através do Weka.

8 Referências

Abiy, T., Pang, H., Khim, J., Ross, E., & Williams, C. (s.d.). *Dijkstra's Shortest Path Algorithm*. Obtido de Brilliant: https://brilliant.org/wiki/dijkstras-short-path-finder/

Alex, B., Taylor, L., Winch, R., Hillert, G., Grandja, J., & Bryant, J. (2017). *Spring Security Reference.* Obtido de Spring: https://docs.spring.io/spring-security/site/docs/5.0.5.RELEASE/reference/htmlsingle/#what-is-acegi-security

Android Studio and SDK tools. (s.d.). Obtido de Android Developers: https://developer.android.com/studio/

Angerer, M. (26 de Outubro de 2016). *Functional vs. Regression Testing – Know The Difference.* Obtido de resultspositive: https://resultspositive.com/functional-vs-regression-testing/

Apache Tomcat - Welcome! (s.d.). Obtido de Apache Tomcat: http://tomcat.apache.org/

Bambrick, N. (24 de Junho de 2016). *upport Vector Machines for dummies; A Simple Explanation.* Obtido de http://blog.aylien.com/support-vector-machines-for-dummies-a-simple/

Consuming a RESTful Web Service. (s.d.). Obtido de Spring: https://spring.io/guides/gs/consuming-rest/

Cooper, R. G. (1993). *Winning at New Products.* Addison-Wesley.

Data Mapper. (s.d.). Obtido de martinfowler: https://martinfowler.com/eaaCatalog/dataMapper.html

Data Mining Algorithms. (s.d.). Obtido de Microsoft: https://technet.microsoft.com/en-us/library/ms175595(v=sql.110).aspx

Data Transfer Object. (s.d.). Obtido de martinfowler: https://martinfowler.com/eaaCatalog/dataTransferObject.html

Dijkstra's Algorithm for Single Source Shortest Paths in. (2016). Obtido de Institute of Advanced Study: http://www.math.ias.edu/~pooyahat/Dijkstra.pdf

Dijkstra's algorithm. (s.d.). Obtido de Oxford Reference: http://www.oxfordreference.com/view/10.1093/oi/authority.20110803095718501

Escudeiro, P., & Bidarra, J. (2008). *Quantitative Evaluation Framework (QEF).* Revista Ibérica de STI.

Escudeiro, P., & Escudeiro, N. (2012). Evaluation of Serious Games in Mobile Platforms with QEF. *IEEE Internation Conference on Wireless, Mobile and Ubiquitous Technology in Education.*

Firebase Realtime Database. (s.d.). Obtido de Firebase: https://firebase.google.com/docs/database/

Fowler, M. (2 de Janeiro de 2007). *Mocks Aren't Stubs.* Obtido de MartinFowler: https://martinfowler.com/articles/mocksArentStubs.html

Fowler, M. (16 de Janeiro de 2018). *Integration Test.* Obtido de MartinFowler: https://martinfowler.com/bliki/IntegrationTest.html

Fraj, B. M. (17 de Dezembro de 2017). *In Depth: Parameter tuning for Random Forest.* Obtido de Medium: https://medium.com/@mohtedibf/in-depth-parameter-tuning-for-random-forest-d67bb7e920d

FusedLocationProviderClient. (s.d.). Obtido de Google APIs for Android: https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

Gama, J., Carvalho, A. P., Faceli, K., Lorena, A. C., & Oliveira, M. (2015). *Extração de Conhecimento de Dados - Data Mining.* Lisboa: Sílabo.

Geppert, A., & Wimmers, E. L. (s.d.). *Consistency Constraints in Database Middleware.*

Gierke, O., & Darimont, T. (s.d.). *Spring Data JPA - Reference Documentation.* Obtido de Spring: https://docs.spring.io/spring-data/jpa/docs/1.4.3.RELEASE/reference/html/

Global car sales 1990-2017. (Junho de 2017). Obtido de Statista: https://www.statista.com/statistics/200002/international-car-sales-since-1990/

Google Maps Android API. (s.d.). Obtido de Google Developers: https://developers.google.com/maps/documentation/android-api/

Google Maps Directions API. (s.d.). Obtido de Google Developers: https://developers.google.com/maps/documentation/directions/

Google Places API. (s.d.). Obtido de Google Developers: https://developers.google.com/places/

Gupta, S. (2015). A Regression Modeling Technique on Data Mining. *Internation Journal of Computer Applications*, 27-29.

H2 Database Engine. (s.d.). Obtido de H2: http://www.h2database.com/html/main.html

Hammes, D., Medero, H., & Mitchell, H. (2014). *Comparison of NoSQL and SQL Databases in the Cloud.* AIS Electronic Library.

Han, J., Kamber, M., & Pei, J. (2012). *Data Mining - Concepts and Techniques.* Morgan Kaupmann.

Hedge, R. (28 de Junho de 2015). *Soft Delete vs Hard Delete.* Obtido de Abstraction.blog: http://abstraction.blog/2015/06/28/soft-vs-hard-delete

Hinai, A. H. (2016). *A Performance Comparison of SQL and NoSQL Databases for Large Scale Analysis of Persistent Logs.*

INRIX 2016 Global Traffic Scorecard. (2016). Obtido de INRIX: http://inrix.com/resources/inrix-2016-global-traffic-scorecard/

Introduction to the Google Maps Roads API. (s.d.). Obtido de Google Developers: https://developers.google.com/maps/documentation/roads/intro

Jacobs, F. (12 de Maio de 2016). *Are Waze and Google Maps Displacing Radio Traffic Reports?* Obtido de Jacobs media strategies: http://jacobsmedia.com/stay-tuned-traffic-weather-smartphone/

Jetty - Servlet Engine and Http Server. (s.d.). Obtido de Eclipse: https://www.eclipse.org/jetty/

JSON Web Token Introduction. (s.d.). Obtido de jwt.io: https://jwt.io/introduction/

Koen, P. A., Ajamian, G. M., Boyce, S., Clamen, A., Fisher, E., Fountoulakis, S., . . . Seibert, R. (s.d.). *Fuzzy Front End: Effective Methods, Tools, and Techniques.*

Land, S., & Fischer, S. (27 de Agosto de 2012). *RapidMiner 5: RapidMiner in academic use.* Obtido de RapidMiner: https://docs.rapidminer.com/downloads/RapidMiner_RapidMinerInAcademicUse_en.pdf

LinearRegression. (s.d.). Obtido de Weka: http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/LinearRegression.html

Location services. (s.d.). Obtido de GitBook: https://google-developer-training.gitbooks.io/android-developer-advanced-course-concepts/unit-4-add-geo-

features-to-your-apps/lesson-7-location/7-1-c-location-services/7-1-c-location-services.html

M5P. (s.d.). Obtido de Weka: http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/M5P.html

Maps - Navigation & Transit. (s.d.). Obtido de Google Play: https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=en

Massachusetts Institute of Technology. (s.d.). *Big O notation.* Obtido de Massachusetts Institute of Technology: http://web.mit.edu/16.070/www/lecture/big_o.pdf

Maven - Welcome to Apache Maven. (s.d.). Obtido de Maven: https://maven.apache.org/

Microsoft Azure Cloud Computing Platform & Services. (s.d.). Obtido de Micrososft Azure: https://azure.microsoft.com/en-us/?v=18.02

Microsoft. (s.d.). *Chapter 3: Architectural Patterns and Styles*. Obtido de Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/ee658117.aspx

Misa, T. (2010). An Interview with Edsger W. Dijkstra. *Communications of the ACM*.

ModelMapper - Simple, Intelligent, Object Mapping. (s.d.). Obtido de ModelMapper: http://modelmapper.org/

MySQL. (s.d.). Obtido de MySQL: https://www.mysql.com/

Nations, U. (2016). *The World's Cities in 2016.*

Nicola, S., & Ferreira, E. P. (2012). A Novel Framework for Modeling Value for the Customer, an Essay on Negotiation. *Internation Journal of Information Technology & Decision Making*, 661-703.

Norris, D. (15 de Novembro de 2013). *Rapid Miner - a potential game changer*. Obtido de Bloor: https://www.bloorresearch.com/2013/11/rapidminer-a-potential-game-changer/

Number of smartphone users worldwide. (s.d.).

OpenShift: Container Application Platform by RedHat. (s.d.). Obtido de OpenShift: https://www.openshift.com

Parkin, R. (s.d.). *Software Unit Testing.* Obtido de IV & V Australia: http://condor.depaul.edu/sjost/hci430/documents/testing/UnitTesting.pdf

Popa, B., & Popescu, D. (2016). Analysis of algorithms for shortest path problem in parallel. *Carpathian Control Conference.* IEEE.

Pore, S. S., & Pawar, S. B. (2015). Comparative Study of SQL & NoSQL Databases. *International Journal of Advanced Research in Computer Engineering & Technology*, 1747-1753.

Quinlan, J. R. (2006). *Learning with continuous classes.* Singapura: World Scientific.

R Foundation. (s.d.). *The R Project for Statistical Computing*. Obtido de R: https://www.r-project.org/

RandomForest. (s.d.). Obtido de Weka: http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html

Relative Absolute Error. (s.d.). Obtido de gepsoft: https://www.gepsoft.com/gxpt4kb/Chapter10/Section2/SS15.htm

Repository. (s.d.). Obtido de martinfowler: https://martinfowler.com/eaaCatalog/repository.html

RestTemplate. (s.d.). Obtido de Spring Framework: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

RForge. (s.d.). *JRI - Java/R Interface*. Obtido de RForge: https://www.rforge.net/JRI/

RForge. (s.d.). *Rserve - Binary R server*. Obtido de RForge: https://www.rforge.net/Rserve/

Root Relative Squared Error. (s.d.). Obtido de gepsoft: https://www.gepsoft.com/gxpt4kb/Chapter10/Section1/SS07.htm

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operation Research*, 9-26.

Service Layer. (s.d.). Obtido de martinfowler: https://www.martinfowler.com/eaaCatalog/serviceLayer.html

Shalizi, C. (2006). *Statistics 36-350: Data Mining.*

SharedPreferences. (s.d.). Obtido de Android Developers: https://developer.android.com/reference/android/content/SharedPreferences

SMOreg. (s.d.). Obtido de Weka: http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMOreg.html

Soft Delete. (s.d.). Obtido de CFWheels: https://guides.cfwheels.org/docs/soft-delete

Sousa, P. (s.d.). *Arquitectura de Aplicação e modelo 3 camadas.* Obtido de ISEP: http://www.dei.isep.ipp.pt/~psousa/aulas/EINF/EINF-2006-01.pdf

Spring. (s.d.). Obtido de Spring: https://spring.io/

Spring Bean Definition. (s.d.). Obtido de tutorialspoint: https://www.tutorialspoint.com/spring/spring_bean_definition.htm

Spring Security. (s.d.). Obtido de Spring: https://spring.io/projects/spring-security

Testing in Spring Boot. (24 de Junho de 2018). Obtido de Baeldung: http://www.baeldung.com/spring-boot-testing

The University of Waikato. (s.d.). *Weka 3: Data Mining Software in Java*. Obtido de Weka The University of Waikato: https://www.cs.waikato.ac.nz/ml/weka/

Undertow - JBoss Community. (s.d.). Obtido de Undertow: http://undertow.io/

Warwick Manufacturing Group. (2007). *Quality Function Deployment.*

Waze - GPS, Maps, Traffic Alerts & Live Navigation. (s.d.). Obtido de Google Play: https://play.google.com/store/apps/details?id=com.waze&hl=en

Webb, P., Syer, D., Long, J., Nicoll, S., Winch, R., Wilkinson, A., . . . Bhave, M. (s.d.). *Spring Boot Reference Guide.* Obtido de Spring: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-spring-beans-and-dependency-injection

Weka Error Measurements. (s.d.). Obtido de Computer Science Student Server: https://katie.mtech.edu/classes/csci347/Resources/Weka_error_measurements.pdf

Witten, I. H., & Frank, E. (2016). *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann.

Woods, D. (18 de Março de 2014). *Exploring Micro-frameworks: Spring Boot.* Obtido de InfoQ: https://www.infoq.com/articles/microframeworks1-spring-boot

Yang, X., Liu, D., & Lin, C. (2014). Shortest path algorithm based on distance comparison. *Geoscience and Remote Sensing Symposium.* IEEE.

Zeithaml, V. A. (Julho de 1988). Consumer Perceptions of Price, Quality, and Value: A Means-End Model and Synthesis of Evidence. *Journal of Marketing*, 2-22.

Zhan, F. B., & Noon, C. E. (1998). *Shortest Path Algorithms: An Evaluation Using Real Road Networks.*