Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2020

Cristiano Marques Vagos

Sistema de Correção do Posicionamento de Condutores em Tempo-Real

Universidade de Aveiro
Ano 2020
Departamento de Eletrónica, Telecomunicações e Informática

Cristiano Marques Vagos

Sistema de Correção do Posicionamento de Condutores em Tempo-Real

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Manuel Matos Moreira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, em colaboração com a empresa Bus Terrace Technologies, S.A., com a supervisão do Mestre João Pedro Oliveira Pedrosa.

"*Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time.*"

- Thomas Edison

o júri

presidente

Prof. Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
professor auxiliar da Universidade de Aveiro

vogais

Prof. Doutor Alexandre Miguel Barbosa Valle de Carvalho
professor auxiliar da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor José Manuel Matos Moreira
professor auxiliar da Universidade de Aveiro

agradecimentos

Primeiro, quero agradecer à LUGGit pela oportunidade de poder realizar esta dissertação. Um grande agradecimento ao João Pedro Pedrosa, que me trouxe para a empresa e que esteve sempre disponível para me orientar, ajudar e motivar durante todo este processo, foste incansável. Agradeço ao Hugo Fonseca, por todos os conselhos e apoio dados durante o desenvolvimento. Aprendi muito convosco, e fizeram de mim um melhor profissional. Também agradecer ao Ricardo Figueiredo e ao Diogo Correia pelo voto de confiança, apoio e motivação, bem como a toda a restante equipa.

Um agradecimento especial ao Professor José Moreira por me ter orientado nesta dissertação da melhor forma possível, descomplicando o que para mim era inicialmente complicado. Obrigado Professor!

De seguida, quero agradecer aos meus pais, e aos meus avós Palmira e Paulo. Paula, eu não me esqueci de ti! Obrigado por estarem sempre ao meu lado, por me terem dado a oportunidade de seguir os meus sonhos e desejos, e por todos os sacrifícios que fizeram durante todos estes anos. Obrigado por nunca me terem deixado desistir, por terem sempre acreditado nas minhas capacidades, e por compreenderem o que esta etapa significa para mim. Devo-vos muito. Obrigado, muito obrigado família!

Agradecer à minha namorada, por todo o apoio, paciência, carinho, motivação e companhia durante estes anos. Obrigado por também compreenderes o quão importante esta etapa é para mim. Não tenho quaisquer dúvidas de que este caminho foi muito mais fácil contigo ao meu lado.

Quero também agradecer a todos os meus amigos (vocês sabem bem quem são) pela vossa amizade, paciência e apoio durante todos estes anos. Tenho a certeza de que sem vocês este percurso não seria de todo igual. Continuamos sempre juntos!

Finalmente, um obrigado muito especial a todos os professores que passaram pelo meu percurso académico desde a Nazaré até Aveiro, e por terem contribuído para a minha formação. Aprendi imenso com todos vós.

Esta dissertação teve a colaboração do projeto “MoST: Modelação, interrogação e visualização interativa de dados espácio-temporais”, com a referência POCI-01-0145-FEDER-032636.

palavras-chave

Trajetórias GPS, plataformas tecnológicas de mobilidade, posicionamento de condutores, análise de dados em tempo-real, indexação espacial, visualização de dados.

resumo

A utilização de plataformas tecnológicas de mobilidade é cada vez mais comum em contexto urbano, permitindo requisitar um meio de transporte em tempo-real através de uma aplicação móvel. Cada serviço é levado a cabo por um dos condutores nessas plataformas, sendo o tempo de realização um fator chave na satisfação do cliente. Neste contexto, o posicionamento dos condutores enquanto aguardam por um novo serviço é fundamental para a redução do tempo que demoram a chegar até ao cliente, o que irá aumentar a sua satisfação com o serviço.

Com esta dissertação propõe-se a criação de um sistema capaz de corrigir o posicionamento dos condutores fornecendo ferramentas que farão com que estes se consigam posicionar em locais estratégicos. Para conseguir esse objetivo serão analisadas e utilizadas ferramentas de otimização de rotas, indexação espacial e análise de dados geoespaciais, que serão posteriormente transformados em novas formas de visualização desses dados, através de *heatmaps*.

Este sistema foi desenvolvido e está em funcionamento na LUGGit, empresa que oferece um serviço de recolha e entrega de bagagens, com o objetivo de fazer com que os seus condutores (denominados de Keepers) se posicionem corretamente no seu contexto de utilização, o que fará com que sejam capaz de responder a novos pedidos de serviço num tempo inferior a 15 minutos.

keywords

GPS trajectories, technological mobility platforms, driver positioning, real-time data analysis, spatial indexing, data visualization.

abstract

The usage of technological mobility platforms is increasingly common in an urban context, allowing to request a transportation in real-time using a mobile app. Each service is provided by one of the platforms' drivers and the execution time is a crucial factor in customer satisfaction. In this context, the driver's positioning while waiting for a new service request is crucial to reduce the time needed to reach the customer, which will increase their satisfaction.

This dissertation proposes a system capable of correcting the drivers' position and providing tools that will allow them to wait in strategic locations. To reach that goal will analyze and use route optimization tools, spatial indexing, and geospatial data analysis, which will be later transformed into new visualization methods for showing them using heatmaps.

This system was developed and is in use at LUGGit, a company that delivers a real-time luggage pickup and delivery service, to allow their drivers (denominated Keepers) to be positioned correctly in their usage context, which will enable them to be able to answer new service requests in less than 15 minutes.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ACRÓNIMOS

API - Application Programming Interface

CART - Classification and Regression Trees

CH - Contraction Hierarchies

CI/CD - Continuous Integration and Continuous Deployment

CPU - Central Processing Unit

CSV - Comma Separated Values

DARP - Dial-a-Ride Problem

ELT - Extract Load Transform

EPSG - European Petroleum Survey Group

GCP - Google Cloud Platform

GPS - Global Positioning System

GTFS - General Transit Feed Specification

HTTP - Hypertext Transfer Protocol

JSON - JavaScript Object Notation

KPI - Key Performance Indicator

MAE - Mean Absolute Error

MLD - Multi-Level Dijkstra

OCDE - Organização para a Cooperação e Desenvolvimento Económico

OGC - Open Geospatial Consortium

OLAP - Online Analytical Processing

ORM - Object-Relational Mapping

OSM - OpenStreetMap

OSRM - Open Source Routing Machine

OTP - OpenTripPlanner

PDPTW - Pickup-and-Delivery Problem with Time Windows

PDVRPTW - Pickup-and-Delivery Vehicle Routing Problem with Time Windows

POI - Point of Interest

QA - Quality Assurance

REST - Representational State Transfer

RF - Random Forest

RPC - Remote Procedure Call

SDK - Software Development Kit

SGBD - Sistema de Gestão de Base de Dados

SGGD - Sistema Global de Grelhas Discretas

SITA - Société Internationale de Télécommunications Aéronautiques

SOA - Service Oriented Architecture

SQL - Structured Query Language

URL - Uniform Resource Locator

VRP - Vehicle Routing Problem

WGS - World Geodetic System

YAML - YAML Ain't Markup Language

1. Introdução

1.1. Contexto

As cidades nos dias de hoje estão em constante mutação e crescimento, sentindo sempre necessidade de se irem adaptando e inovando para conseguirem fazer frente ao constante aumento populacional. Segundo a OCDE, no ano de 2100 cerca de 85% da população mundial irá viver em cidades [1], aumento esse que irá trazer consequências ao nível da mobilidade urbana, dado que as infra estruturas terão de estar preparadas para um elevado aumento dos veículos em circulação [2]. Por exemplo, na Europa verificou-se nos últimos 5 anos um aumento de 5.7% nos veículos de passageiros em circulação [3], e espera-se que nos próximos anos esta estatística continue a sofrer alterações, já que a União Europeia anunciou a *carbon-neutrality*, medida que deverá ser atingida por todos os estados-membros até 2050 [4], o que já está a provocar uma quebra nas vendas de automóveis [5]. No que toca aos transportes públicos, estes estão cada vez mais a adaptar-se às exigências modernas, e em grandes centros urbanísticos já existe uma grande variedade de tipos de transporte, desde o uso do autocarro, o metropolitano e o comboio, estando inclusivamente a ocorrer cada vez mais investimentos para o seu desenvolvimento [2], notando-se neste sentido uma evolução progressiva dos sistemas de transporte em ambiente urbano. Contudo, também se tem verificado um uso acentuado de meios de transporte mais sustentáveis como a trotineta [6] e a bicicleta, bem como estratégias para o crescimento do uso da última [7]. Portanto, a grande variedade de opções de mobilidade existentes faz com que o utilizador decida o que para si é mais importante na hora de escolher o seu meio de transporte, de acordo com alguns requisitos (o primeiro ponto de destino, duração da viagem, preço, acessibilidade, entre outros).

Com o avanço da tecnologia, tornou-se também indispensável para qualquer pessoa a utilização do *smartphone*, cuja capacidade de processamento e de comunicação torna possível a criação e desenvolvimento de aplicações que facilitem em tarefas e situações do dia-a-dia, oferecendo ainda soluções para tornar mais cómodo o nosso quotidiano. Para além do mais, o *smartphone* tornou-se num verdadeiro assistente pessoal, possibilitando aceder a diversos tipos de informação através da Internet, criando um mundo mais interligado, motivando o aparecimento de novos serviços de transporte disponibilizados através de aplicações móveis, que permitem aos

utilizadores requisitar um meio de transporte em tempo-real. Ao contrário dos transportes públicos e outros meios de transporte como por exemplo os táxis, em que cada utilizador necessita de se deslocar aos locais de paragem para a sua utilização, as plataformas tecnológicas de mobilidade como a Uber[1], Lyft[2], Bolt[3], Cabify[4] e Kapten[5] promovem a comodidade do utilizador, oferecendo um serviço em tempo-real, a partir de e para qualquer lugar. O uso destas plataformas de mobilidade é feito através de aplicações móveis, fazendo com que o utilizador não precise de se deslocar, sendo o condutor da plataforma a deslocar-se até ele, transformando a sua localização atual numa paragem e num ponto de recolha para a utilização do transporte. Contudo, este novo paradigma traz novos desafios a nível logístico e tecnológico, tais como conseguir adaptar a procura com a oferta de serviços em tempo-real, sendo que os condutores deverão saber tirar o melhor proveito do uso da sua aplicação, já que estão dependentes de a utilizar para o seu meio de trabalho.

Nestas plataformas de mobilidade há um ator importante: o condutor. Sem ele não é efetuado qualquer serviço. Quando um utilizador requisita um novo serviço, a rede de condutores existente é notificada desse pedido, em conjunto com a informação do ponto de recolha e dos ganhos a obter ao concretizar o serviço. Após aceitação, o condutor deverá deslocar-se até à localização atual do utilizador para a realização do serviço. Aqui, a localização do condutor no momento em que um novo pedido de serviço ocorre é decisiva, sendo um dos fatores que interfere na escolha dos condutores candidatos a efetuar um novo serviço.

1.2. LUGGit

As plataformas de mobilidade têm como foco principal o transporte de pessoas, sendo também possível o transporte de bagagens, ainda que de alguma forma reduzido. Neste sentido surge a LUGGit[6], *startup* que propôs a realização desta dissertação, que tal como os exemplos referidos trata-se de uma plataforma tecnológica de mobilidade, onde a sua utilização é feita através de duas aplicações móveis: a aplicação do utilizador, que requisita o serviço; e a aplicação do condutor (doravante denominado de Keeper), que é responsável pela realização do serviço.

[1] https://www.uber.com/
[2] https://www.lyft.com/
[3] https://bolt.eu
[4] https://cabify.com
[5] https://www.kapten.com
[6] https://luggit.app/

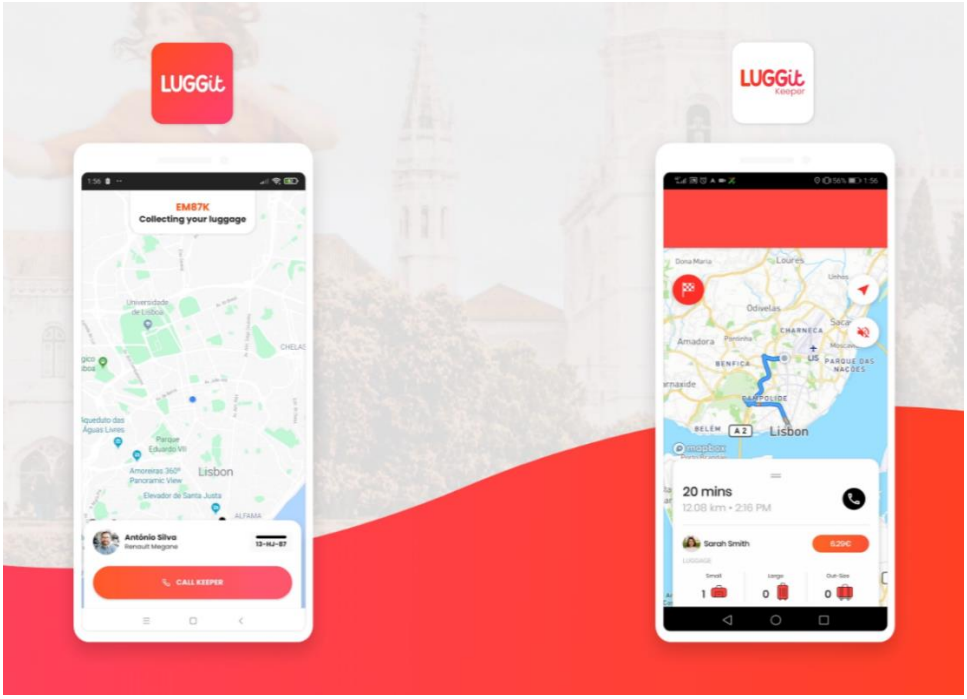

Figura 1 - Aplicação dos utilizadores da LUGGit (à esquerda) e aplicação dos Keepers da LUGGit (à direita)

Ao contrário das plataformas anteriormente referidas, a LUGGit foca-se apenas no transporte e armazenamento de bagagens, permitindo aos utilizadores através da sua aplicação móvel requisitar em tempo-real a recolha de bagagens, agendando a sua entrega quando e onde estes o desejarem. Desta forma o utilizador poderá viajar sem bagagens, retirando responsabilidade e desconforto ao transportar as bagagens consigo.

Os Keepers são responsáveis por recolher, guardar e entregar as bagagens ao utilizador a partir do momento em que estes recebem um novo pedido de serviço. Após a sua recolha, as bagagens são armazenadas num centro logístico, pois o período de tempo entre a recolha e a entrega pode justificar manter as bagagens armazenadas, libertando espaço no veículo do Keeper para este efetuar novos serviços.

Um dos focos das plataformas tecnológicas é a rapidez de serviço. Tratando-se de um serviço em tempo-real pretende-se garantir que o utilizador não atravesse longos períodos de espera, de modo a aumentar a sua satisfação. No caso da LUGGit, a localização atual de cada Keeper no momento de requisição de um novo serviço é um dos indicadores-chave que mais contribui para a redução do tempo de serviço. Assim, quanto mais perto o Keeper estiver do utilizador, menos tempo o utilizador terá de ficar à espera para que o serviço se concretize.

1.3. Motivação

Encarei pessoalmente o desenvolvimento de uma solução que permita corrigir o posicionamento dos Keepers da LUGGit como um excelente desafio. Todas as fases do mesmo, desde o levantamento de requisitos, passando pela idealização, desenho e a arquitetura do sistema a adotar, até à integração e manutenção em ambiente de produção permitiram-me colocar em prática muitos conceitos aprendidos ao longo do meu percurso académico, assim como abordar e experienciar novos conceitos e tecnologias.

A implementação e a integração de um sistema desta natureza em contexto empresarial também é um elemento enriquecedor, exigindo um maior cuidado e sentido crítico na análise e escolha de novas tecnologias. Esta dissertação fará também com que se possa avaliar o uso de diversas linguagens de programação, bem como ferramentas que auxiliem no desenvolvimento e na implantação de um sistema em ambientes *Cloud*, bem como preparar os componentes de *software* para um ambiente exigente.

Em última instância, este projeto fará com que a empresa consiga inovar e crescer, implementando funcionalidades e métodos que permitam a recolha e análise de dados dos seus Keepers para uso no presente e no futuro, otimizando uma parte fulcral do seu modelo de negócio. A solução na qual irei integrar o desenvolvimento e resultados desta dissertação já se encontra em produção, estando aplicada num contexto real de utilização. Este é um aspeto que também me motiva, pois é uma solução usada diariamente por vários utilizadores, cujo *feedback* contínuo contribuirá para a minha evolução tanto no desenvolvimento, na segurança, adaptabilidade e na própria manutenção do *software*.

1.4. Objetivos Gerais

Esta dissertação tem como principal objetivo construir um sistema que permita corrigir e reposicionar os Keepers da LUGGit em locais estratégicos, fazendo com que estes possam responder a um novo pedido de serviço em menos de 15 minutos. Ao mesmo tempo, isso fará com que o tempo de atendimento ao cliente seja o menor possível, aumentando a sua satisfação. Também fará com que a receita para o Keeper seja maximizada, uma vez que este poderá estar num local onde a probabilidade de haver serviços é maior. Este sistema será construído através da recolha de dados históricos e de dados em tempo-real obtidos a partir da aplicação utilizada pelos Keepers.

2. Trabalho Relacionado

Em primeiro lugar, e antes de se abordar uma solução no contexto desta dissertação, deve-se primeiro analisar o que existe atualmente no mercado, relativamente a plataformas tecnológicas de mobilidade. A LUGGit é uma delas, que à semelhança da Uber, Bolt e Cabify oferece um serviço em tempo-real através de uma aplicação para *smartphone*. Contudo, foca-se apenas na recolha e entrega de bagagens sendo que a sua solução é inovadora, não existindo atualmente no mercado competição a este modelo de negócio. No entanto, existem também outras plataformas que operam através de aplicações móveis que resolvem o problema do armazenamento de bagagens entre *check-ins*, não fornecendo um serviço em tempo-real como o da LUGGit.

De modo a obter uma otimização e correção do posicionamento dos Keepers no âmbito da operação da empresa devem ter-se em conta as suas localizações, bem como as localizações de novos pedidos, referenciando-as geograficamente através de indexação espacial, agregando várias localizações num único índice. A rota a efetuar pelo Keeper nos seus serviços deve também ser otimizada, o que permite saber quanto tempo a viagem entre duas localizações irá demorar. Todos estes fatores conjugados fazem com que seja possível por exemplo averiguar a possibilidade do Keeper, no futuro, ao estar mais corretamente posicionado poder atender a vários serviços em simultâneo através de pedidos encadeados, sugerir as suas próximas localizações para que este esteja o mais próximo possível do utilizador quando ocorrerem novos serviços, bem como será um fator importante para a seleção do Keeper ideal para os efetuar.

Neste capítulo discutem-se estes assuntos, introduzindo-os e dando um enquadramento generalizado sobre o que já existe, e como abordar a solução a este problema de otimização.

2.1. Plataformas Tecnológicas de Mobilidade e Produtos Similares à LUGGit

Quando nos deslocamos num contexto de viagem, é comum os viajantes trazerem consigo algum tipo de bagagem, de tamanho e quantidade variáveis. Conforme um estudo divulgado pela SITA em 2019, foram registadas no ano anterior 4.27 bilhões de malas durante o *check-in* em aeroportos [8] (Figura 2), sendo que o número de passageiros tem aumentado ano após ano. Estima-se de igual forma que o mercado de bagagens tenha um crescimento na ordem dos 7.3% até 2024 [9], acompanhando o crescimento a nível global da indústria do turismo [10], o que irá contribuir

para um maior uso de bagagens em viagem. De facto, a indústria do turismo e viagens tem sido recentemente afetada pelo crescente uso da tecnologia, sendo as aplicações para *smartphone* grandes impulsionadoras nesse crescimento [11], tornando a experiência de viagem diferente do habitual, uma vez que através de poucos cliques podemos reservar, planear e organizar uma viagem na totalidade.

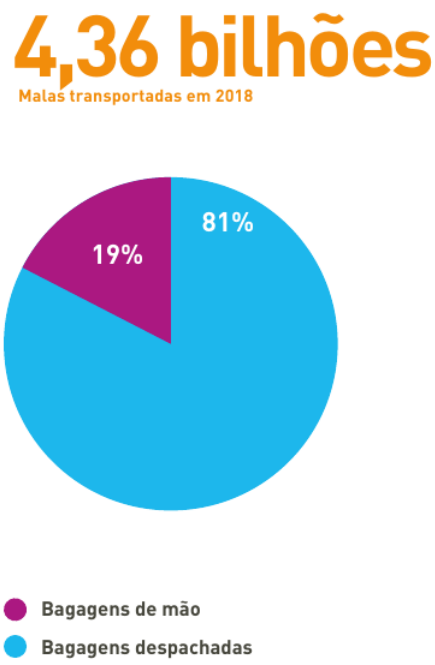

Figura 2 - Malas transportadas durante o ano de 2018 em aeroportos [8]

Um dos nichos de mercado responsáveis por este fenómeno é o chamado transporte *on-demand*, que permite a requisição de um serviço em tempo-real, sendo esse o principal foco das plataformas de mobilidade Uber, Lyft, Bolt, entre outras. O uso destas plataformas é feito em duas aplicações para *smartphone*, com funcionalidades distintas: uma aplicação para os utilizadores, onde irão requisitar e pagar pelo serviço, e uma outra aplicação para os condutores, onde irão aceitar e realizar os serviços requisitados, sendo pagos pela plataforma de acordo com os serviços realizados. O paradigma da oferta de serviços *on-demand* e *on-site* através de aplicações para *smartphone* veio mudar a forma como viajamos [12] e não só, fazendo com que sejam os prestadores de serviço a chegar diretamente até aos utilizadores em vez do contrário, o que normalmente acontece numa paragem de autocarro, por exemplo. Estas plataformas de mobilidade fazem com que um utilizador requisite uma viagem em tempo-real na sua localização atual, sendo ela atribuída e levada a cabo por um dos condutores que estejam disponíveis no momento a usar a plataforma através de veículos

ligeiros, que em Portugal podem ser identificados através do dístico TVDE[7]. Assim que o utilizador chega ao destino, é realizado o pagamento, e o condutor fica disponível para uma nova viagem [13]. No caso da Uber, que está disponível em 65 países e em mais de 600 cidades, traduz-se numa média diária de 14 milhões de viagens [14]. Já a Lyft, registou em Janeiro de 2018 a marca dos 23 milhões de utilizadores a nível mundial [15], sendo uma das mais importantes plataformas de transporte *on-demand* no mercado, também disponível através de uma aplicação para *smartphone*, sendo neste momento a maior concorrente da Uber no que ao transporte de pessoas *on-demand* diz respeito.

A LUGGit, empresa que propôs esta dissertação, é também uma dessas plataformas de mobilidade, sendo focada no transporte de bagagens, cujo modelo difere das plataformas anteriormente descritas. Enquanto estas se dedicam unicamente ao transporte de pessoas a partir de um local de origem para um local de destino, com o problema específico da LUGGit existe a necessidade de armazenar a bagagem de um utilizador entre a recolha e entrega através de um centro logístico, não existindo transporte de pessoas, só e apenas de bagagens. Num contexto em que o utilizador chegue a uma cidade às 9h da manhã e o seu check-in num alojamento seja às 15 horas, em vez deste esperar pela hora do check-in com as suas bagagens, pode delegar à LUGGit essa responsabilidade, permitindo que a bagagem seja recolhida em tempo-real e entregue no hotel na hora que desejar, ganhando tempo que poderá utilizar para conhecer uma cidade, por exemplo.

O conceito de armazenamento de bagagens utilizando aplicações para *smartphone* já é utilizado nos dias de hoje. Aplicações como a Vertoe[8], LuggageHero[9] e Stasher[10] resolvem o problema reunindo nas suas áreas de operação parceiros que facultam as suas próprias instalações como pontos de recolha e entrega de bagagens, no entanto fazem com que os utilizadores tenham que se deslocar até ao local, cobrando-lhes uma taxa de acordo com o tempo de armazenamento [16]. Outro exemplo é o da aplicação StoreMe[11], que apresenta o mesmo tipo de funcionamento usando cafés, lojas ou ginásios como os locais de armazenamento para as bagagens dos seus utilizadores [17]. Também existem plataformas que fazem a recolha e entrega de bagagens utilizando recursos próprios, através de uma reserva antecipada num website o que é o caso das

[7] https://imt-tvde.webnode.pt/
[8] https://vertoe.com/
[9] https://luggagehero.com/
[10] https://stasher.com/
[11] https://getstoreme.com/

empresas BoB[12], Luggage Driver[13], Luggagent[14], entre outras. O conceito apresentado pela LUGGit é de facto inovador neste sector, pois um dos condutores disponíveis na plataforma irá deslocar-se diretamente até ao utilizador, recolhendo as suas bagagens. Posteriormente, o condutor armazena as mesmas num centro logístico até minutos antes da entrega de volta ao utilizador, fazendo com que este não tenha que se deslocar até ao local onde as bagagens estão armazenadas, ao contrário das plataformas descritas acima. A LUGGit usa como nomenclatura para os seus condutores "Keeper" tratando-se de um guardião de bagagens, mantendo-as desde o momento que a recolha é feita até ao momento da entrega, que poderá ser feita por um outro Keeper disponível, não necessitando obrigatoriamente de ter feito a recolha inicial junto do utilizador, sendo também este um serviço em tempo-real. O uso de um centro logístico torna-se essencial para o contexto da LUGGit, uma vez que o período de tempo entre a recolha e a entrega poderá ser grande. Neste caso, é necessário manter as bagagens armazenadas temporariamente num armazém, de forma a maximizar a operação e capacidade logística do Keeper.

Nas plataformas Uber [13] e Lyft [18], o fator principal de seleção do condutor a realizar o serviço é a sua proximidade com o utilizador. Quanto mais próximo este estiver do utilizador, mais curta será a viagem, o que consequentemente significa um menor tempo de espera por parte do utilizador. Este é também o método utilizado pela LUGGit para atribuir um Keeper a um novo pedido. Este fator não deverá ser o único a considerar na escolha de um Keeper, até porque nem sempre o melhor caminho é o mais curto [19]. Fatores externos como o trânsito poderão também afetar o tempo de serviço. Por exemplo, comparando 2 condutores: um estando mais próximo do utilizador, e um outro estando mais longe do utilizador, caso o condutor mais distante tenha durante o seu trajeto até ao utilizador menor trânsito, este até poderá ser a escolha mais indicada para efetuar o serviço. Outros fatores como a capacidade de armazenamento do veículo também poderão ter influência na seleção de um Keeper, pois ao não ter espaço disponível para transportar mais bagagens não será capaz de efetuar um novo pedido de serviço.

Coloquemo-nos no lugar do condutor por um momento. Contrastando com os taxistas, que têm três tipos possíveis de atender a novos pedidos de serviço (estando estacionados na praça de táxis, em andamento - *hailing*, ou pré-contratados)[20], um condutor das plataformas de mobilidade

[12] https://bob.io/
[13] https://www.luggagedriver.com/
[14] https://www.luggagent.com/

apenas consegue receber um novo pedido através da própria plataforma, utilizando a aplicação própria criada para os condutores, necessitando de indicar que está de facto disponível para os receber. Desta forma, o método de seleção do condutor a efetuar um novo pedido de serviço deve ter principalmente em atenção a sua disponibilidade e só depois a proximidade com o utilizador que o requisitou. A localização e posicionamento dos condutores nas áreas de operação torna-se assim um dos fatores decisivos para o sucesso destas plataformas de mobilidade em tempo-real, sendo naturalmente um dos seus objetivos principais oferecer um tempo de resposta reduzido, atribuindo um dos condutores disponíveis a um novo pedido de serviço, o que irá consequentemente diminuir o tempo total da prestação do serviço, aumentando de igual forma a satisfação do utilizador e o potencial de lucro, tanto para a própria empresa como para os condutores. Este é um dos desafios propostos por esta dissertação: construir um sistema que permita corrigir e indicar o posicionamento dos Keepers da LUGGit de modo a otimizar a sua operação, ao mesmo tempo fazendo com que o seu tempo de resposta a um novo serviço seja o mais reduzido possível.

2.2. Indexação Espacial

Ao ser efetuado um novo pedido de serviço, são indicados previamente os locais de recolha e de entrega. Num contexto de análise de dados em bruto para sugerir locais aos condutores torna-se necessária a agregação de um conjunto de localizações através de áreas que facilmente sejam identificadas e obtidas por intermédio de indexação espacial.

De facto, a determinação de um conjunto de localizações ótimas para um condutor não é simples de obter, uma vez que não é possível saber com exatidão onde e quando é que os pedidos irão ocorrer, tratando-se este de um serviço em tempo-real. No entanto, torna-se possível fornecer uma estimativa, dado que se sabe à partida as localizações dos pedidos anteriormente realizados bem como as dos pedidos mais recentes através da recolha de todo o histórico de pedidos. Aqui entra também em discussão a capacidade de determinar em tempo-real a oferta e procura de condutores para uma dada localização, que vai posteriormente afetar a recomendação a fazer, o que já é utilizado nas plataformas aqui descritas.

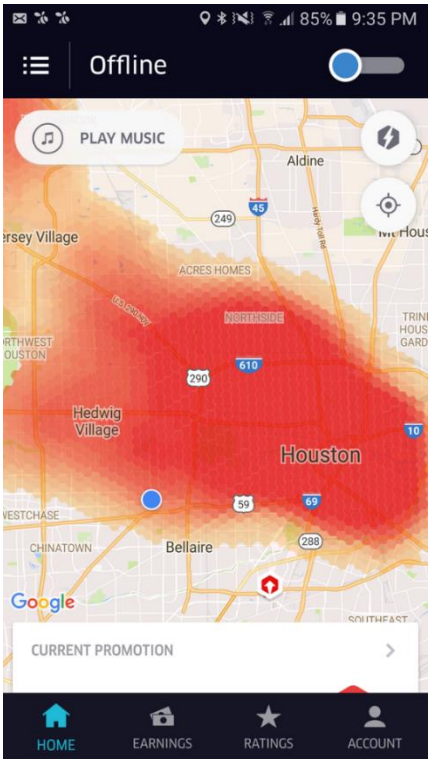

Figura 3 - Mapa de *surge pricing*, na aplicação Android dos condutores da Uber, em Houston (EUA) [21]

A Lyft, por exemplo, fornece *hot spots* que permitem aos condutores saber quais as áreas de maior procura de serviços [22], conceito esse que também é usado pela Uber [23] (Figura 3) na aplicação móvel dos condutores. Estas plataformas aplicam este conhecimento não só para direcionar os seus condutores para locais onde exista uma maior possibilidade de haver um novo serviço, mas também para aplicar aos seus utilizadores tarifas dinâmicas [24]. Numa área de operação poderá haver zonas onde haja um grande número de pedidos a ocorrer no momento, sendo que se tivermos em atenção o histórico de pedidos, essas mesmas zonas não reúnem tanta importância. Não havendo condutores suficientes junto desses locais para fazer face à procura dos mesmos e uma vez que o tempo de resposta e a proximidade é importante para reduzir o tempo de espera do utilizador, as plataformas de mobilidade alteram dinamicamente os preços do serviço. Tal ocorre que para que mais condutores estejam disponíveis nessas mesmas zonas face à necessidade atual dos utilizadores. Este é um caso comum, por exemplo, em locais junto de grandes eventos e espetáculos: poderão não ser locais onde frequentemente haja um grande número de pedidos no dia-a-dia, mas a necessidade atual sobrepõe-se aos dados históricos, tendo maior importância. O próprio modelo de negócio das plataformas e os potenciais ganhos dos condutores também beneficiam deste fenómeno, também conhecido como *surge pricing* [25], permitindo a prática de preços mais elevados ao utilizador, bem como fornecendo maiores ganhos aos condutores. Desta

forma, os condutores são incentivados a deslocarem-se para as localizações identificadas, adaptando a oferta e procura de condutores para novos serviços. Uma proposta de estratégia para um aproveitamento deste fenómeno para os condutores foi proposta recentemente por Guo et al. (2019)[26], através da agregação de vários dados urbanos, utilizando um modelo de regressão linear.

Para obter e fornecer uma estimativa da localização ótima dos condutores, torna-se primordial obter as suas localizações em tempo-real. As localizações onde os pedidos ocorreram também são importantes, permitindo associar as localizações a uma dada região geográfica, agregando várias localizações. Assim, o uso de um Sistema Global de Grelhas Discretas - SGGD (em inglês *Discrete Global Grid System*) ganha especial importância (de Sousa et al., 2007)[27], permitindo identificar cada localização com um índice, referenciando uma região específica. Para este problema em específico, o SGGD Geodésico [28] é de facto o mais interessante, mapeando a superfície terrestre num poliedro. Como indicado por Sahr et al., com um SGGD Geodésico é possível obter em cada uma das faces uma partição espacial hierárquica, atribuindo pontos a grelhas de células. A agregação de localizações através de um índice também apresenta vantagens no armazenamento e manipulação de dados [29]. Com isto em mente, vamos agora analisar algumas soluções possíveis de indexação espacial que possam ser aplicadas no contexto desta dissertação, dando primazia a soluções *open-source* usadas pela comunidade, com desenvolvimento ativo, e que auxiliem a resolução de problemas relacionados com a indexação espacial.

2.2.1. Soluções Existentes

Já estão disponíveis bibliotecas que apresentam capacidades de indexação espacial, entre outras funcionalidades que permitem a manipulação de objetos geométricos como pontos (latitude e longitude), linhas, polígonos, entre outros. De seguida apresentam-se algumas bibliotecas existentes, que não só fornecem estas características, como também são interessantes do ponto de vista visual para a representação de regiões geográficas.

2.2.1.1. S2

Em 2017 surgiu por parte da Google a S2[15], uma biblioteca *open-source* que permite entre diversas funcionalidades [30] a manipulação de pontos, arestas e diversas formas como retângulos e discos. Permite também a indexação espacial através de um número inteiro de 64-bits, bem como

[15] https://s2geometry.io/

operações que permitem obter objetos próximos, calcular centroides e medir distâncias. A projeção é feita subdividindo o planeta Terra em várias células S2 [31], uma decomposição hierárquica de toda a sua superfície através de projeções esféricas tridimensionais.

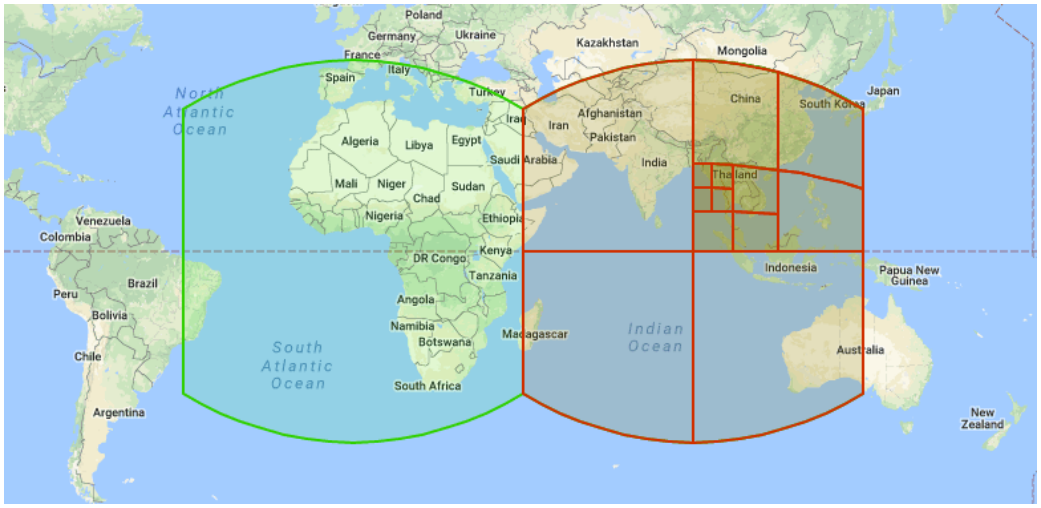

Figura 4 - Representação espacial do planeta Terra utilizando células S2 [32]

O tamanho destas células é variado consoante o seu nível, o que pode ir desde o nível 30 (em que cada célula ocupa 0.7cm²) até ao nível 0 (em que cada célula ocupa 85,011,012.19km²) [33], permitindo assim obter e analisar regiões conforme a granularidade desejada. Isto é particularmente interessante para aplicações que utilizam extensivamente estes dados, tais como a Uber [34], pois permite realizar consultas em dados espaciais de uma forma mais rápida utilizando sistemas de indexação como o S2, para por exemplo detetar quais os condutores mais próximos de um utilizador, o que também é útil no âmbito desta dissertação. Esta biblioteca foi construída na linguagem C++[16], e está disponível nas linguagens Go[17], Java[18] e Python[19].

2.2.1.2. H3

A própria Uber lançou-se também nesta matéria, criando em 2018 a sua própria biblioteca *open-source* de indexação espacial, o H3[20], que tal como a S2 subdivide hierarquicamente a superfície geográfica da Terra, mas desta vez utilizando hexágonos e pentágonos ao contrário da

[16] https://isocpp.org/
[17] https://golang.org/
[18] https://www.java.com/
[19] https://www.python.org/
[20] https://h3geo.org/

anterior. Esta ferramenta foi criada pela Uber para análise de dados às cidades onde esta plataforma opera através de regiões, bem como para facilitar a visualização de dados [35].

Figura 5 - Exemplo de mapeamento do estado da Califórnia (EUA) utilizando o H3 [35]

Para criar o sistema de grelha do H3 foi utilizada uma projeção do mapa *Dymaxion*, também conhecido por mapa de Fuller [36][37], que permite transformar a superfície esférica da Terra num icosaedro. Uma vez que não é possível mapear um icosaedro apenas com hexágonos, são usados pentágonos em cada um dos seus vértices, daí esta biblioteca utilizar pentágonos em combinação com hexágonos na sua representação [38]. O H3 é capaz de representar cerca de 16 resoluções distintas, mapeando cada hexágono desde os 0.9m² (resolução 15) até aos 4,250,546.85km² (resolução 0) de área [39]. Na Figura 5 pode-se ver um exemplo do mapeamento de um polígono através de hexágonos, cada um deles representando um índice espacial [40], o que também torna o H3 uma boa ferramenta de visualização. Esta biblioteca foi desenvolvida em linguagem C[21] e está disponível para várias outras linguagens, incluindo Go, Java, JavaScript[22], PHP[23] e Python.

[21] http://www.open-std.org/jtc1/sc22/wg14/
[22] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
[23] https://www.php.net/

2.2.1.3. OpenEAGGR

Outra biblioteca interessante implementando um SGGD é a OpenEAGGR[24] (*Open Equal Area Global Grid*), lançada em 2017 por parte da empresa Riskaware[25], que permite o uso de duas implementações diferentes de SGGD, sendo ambas semelhantes ao modo de mapeamento utilizado pelo H3, mapeando a superfície terrestre num icosaedro utilizando hexágonos. Uma das implementações diz respeito a um modelo da Terra usando um icosaedro, ao que a outra diz respeito à projeção de área igual de Snyder [41], também mapeando pontos num icosaedro, porém de forma distinta. Ao contrário das duas bibliotecas anteriormente descritas, o OpenEAGGR não tem um nível de granularidade predefinido para especificar a precisão das células, na conversão de uma localização (latitude / longitude) para uma célula DGGS [42]. Esta biblioteca está disponível nas linguagens C, C++, Java e Python, bem como tem extensões para *third-party softwares* como o PostgreSQL[26] e Elasticsearch[27].

2.2.1.4. Comparativo entre Soluções

As bibliotecas acima referidas tiveram em consideração o suporte a indexação espacial, permitindo a agregação de várias localizações a partir de um índice único. Como são bibliotecas *open-source* não existe nenhum custo associado ao seu uso, o que é um ponto positivo. Todas as soluções aqui referidas surgiram no mesmo período, sendo o H3 a biblioteca que mais tarde surgiu comparando com as restantes aqui descritas.

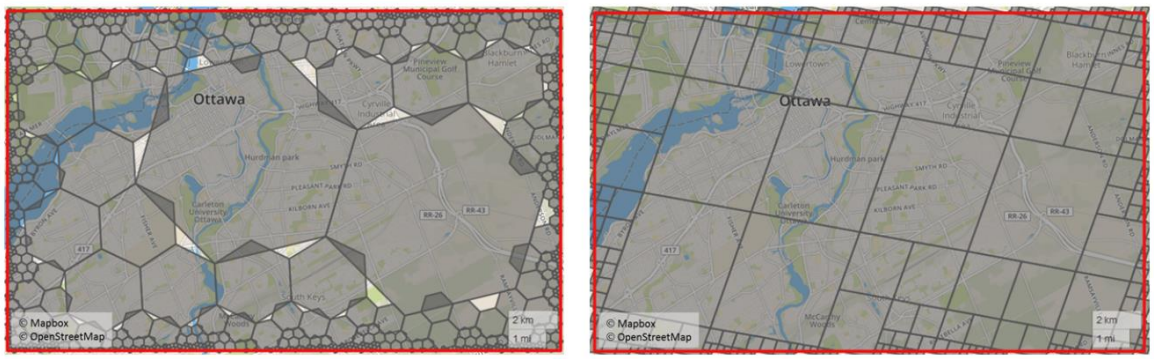

Figura 6 - Representação geográfica da região de Ottawa (Canadá) usando o H3 (à esquerda) e o S2 (à direita) [43]

[24] https://github.com/riskaware-ltd/open-eaggr
[25] https://www.riskaware.co.uk/
[26] https://www.postgresql.org/
[27] https://www.elastic.co/

Uma comparação direta entre bibliotecas, onde constam o S2, o OpenEAGGR e o H3 foi feita recentemente por Bondaruk et al. (2020)[43], discutindo o estado de arte destas implementações de SGGD. Entre as conclusões obtidas, de referir que das bibliotecas aqui mencionadas, o H3 é a biblioteca que está numa fase mais avançada de desenvolvimento, bem como apresenta performance superior nos testes efetuados. Todavia, e apesar de todas elas fornecerem funcionalidades básicas de manipulação de um SGGD, não cumprem à data do artigo total conformidade com os *standards* da OGC [44] no que aos requisitos técnicos diz respeito. Apesar disso, verifica-se que estas bibliotecas têm tido um desenvolvimento ativo e frequente desde o seu lançamento, incluindo cada vez mais funcionalidades ligadas ao uso de um SGGD bem como a possibilidade de resolver as lacunas identificadas, estando assim em conformidade com os *standards* já referidos.

Assim, ao utilizar uma destas formas de mapeamento geográfico pode-se associar um variado tipo de dados a uma dada região no mapa, reunindo características que nos permitam obter conhecimento tendo por base um conjunto de localizações. Estas bibliotecas também permitem usar as suas formas geométricas de células para fornecer um modelo de informação visual, o que no âmbito desta dissertação poderá fazer sentido, pois um dos objetivos é a criação de *heatmaps* (mapas de calor)[45] que ajudem visualmente os Keepers a posicionarem-se corretamente.

Solução *Open-Source*	Forma Geométrica de Projeção Geográfica	Forma Geométrica de Células	Linguagens Disponíveis	Funcionalidades
S2	Esférica	Retangular	C C++ Go Java Python	Indexação Espacial Operações em tipos geométricos *Point-in-Polygon*
H3	Icosaedro	Hexagonal	C / C++ Go Java JavaScript Python PHP R Ruby Rust Julia	Indexação Espacial Operações hierárquicas em índices Métodos de Pesquisa em grelha
OpenEAGGR	Icosaedro	Hexagonal	C / C++ Java Python	Indexação Espacial Operações hierárquicas em células Comparação de formas

Tabela 1 - Tabela comparativa entre soluções de indexação espacial

2.3. Otimização de Rotas

Na navegação de Keepers em ambiente urbano, a determinação da rota a efetuar, bem como a distância total e o tempo esperado de cada viagem são fatores que em conjunto poderão influenciar a escolha de um Keeper para realizar um novo pedido de serviço. A otimização de rotas poderá deste modo levar o Keeper a realizar cada viagem no menor tempo possível, minimizando assim o tempo de serviço.

Este problema foi inicialmente identificado em 1959 por Dantzig & Ramser [46], ao analisar uma frota de camiões na distribuição de combustível, e foi mais tarde apelidado e generalizado na

literatura como sendo um *Vehicle Routing Problem* (VRP) (Toth & Vigo, 2002) [47]. Desde aí, e após muitos trabalhos de investigação realizados neste campo, houve a necessidade de subdividir o VRP em várias variantes tendo em conta diversas características tais como o número e capacidade dos veículos, o tipo de pedidos para o transporte e os objetivos pretendidos para a otimização a fazer (S. Irnich et al., 2014)[48]. Dentro de todas as variantes existentes do VRP, a que mais se assemelha ao *modus operandi* da LUGGit é a que se apelida de *Pickup-and-Delivery Problem with Time Windows* (PDPTW) (Y. Dumas et al., 1991)[49], uma vez que se trata de um serviço de recolha e entrega de bagagens em tempo-real. O PDPTW acarreta a existência de janelas temporais (*Time Windows*) para a realização dos serviços de recolha e entrega, sob limitações de capacidade (uma vez que a capacidade de armazenamento de bagagens dentro do veículo é limitada): assim que é requerido um novo serviço este deve ser executado no menor espaço de tempo possível, sendo a recolha da bagagem feita imediatamente após o pedido de serviço, e a entrega feita na hora agendada pelo cliente, o que implica fazer a entrega sem atrasos.

Quando um utilizador requisita um novo serviço à LUGGit, é enviado um alerta de novo serviço aos Keepers mais próximos por intermédio da distância euclidiana, e assim que o primeiro deles o aceite, este deve dirigir-se de imediato junto do cliente para recolher a sua bagagem. Para a entrega, uma vez que a hora foi previamente agendada pelo cliente ao fazer o pedido de serviço, é necessário que esta seja feita o mais próximo possível da hora agendada. Este serviço implica assim ter uma janela temporal tanto para a recolha como para a entrega, com períodos de tempo bem definidos, contando com fatores externos tais como o trânsito, onde o objetivo é recolher e entregar as bagagens ao cliente o mais rápido possível para que este fique satisfeito, caso que torna este PDPTW como sendo do tipo um-para-um, dado que em cada pedido de um cliente é transportado um conjunto de bagagens de um ponto de partida para um ponto de destino. Este problema em específico, no caso de transporte de pessoas é conhecido como um *Dial-a-Ride Problem* (DARP), já discutido em detalhe por Cordeau & Laporte (2007)[50]. Uma vez que a LUGGit tem vários Keepers disponíveis para responder aos pedidos de serviço, este problema passa a ser um PDPTW para vários veículos, também conhecido na literatura como *Pickup-and-Delivery Vehicle Routing Problem with Time Windows* (PDVRPTW) (M. Battarra et al., 2014)[51].

Desde então, surgiram inúmeras propostas na literatura científica sobre como abordar e tentar resolver este problema, que por ser uma extensão do VRP trata-se de um problema do tipo *NP-hard*, o que permite a existência de aproximações e otimizações para resolver o mesmo. M.

Solomon (1984)[52] propõe o desenvolvimento de algoritmos heurísticos para otimização deste problema, recomendando o algoritmo de inserção de heurísticas. Bräysy & Gendreau [53][54] compilaram, discutiram e analisaram diversas formas de abordar o problema, com as mesmas sugestões apresentadas por Solomon, bem como alguns algoritmos metaheurísticos inspirados pela natureza, assunto que também foi discutido recentemente por Dixit et al. (2019)[55].

2.3.1. Soluções Existentes

Existem diversas soluções a este problema que permitem encontrar a melhor rota entre duas ou mais localizações, fornecendo também detalhes como a duração estimada, distância e instruções passo-a-passo da rota utilizando métodos e algoritmos sugeridos na literatura. De seguida analisam-se algumas dessas soluções existentes, sendo que se optou pela escolha de ferramentas *open-source* por não representarem custos na sua utilização, dado o número de pedidos que serão necessários para a determinação sucessiva da melhor rota a efetuar.

2.3.1.1. Open Source Routing Machine (OSRM)

O OSRM[28] [56] é um motor de encaminhamento *open-source* feito em C++, que utiliza dados oriundos do OpenStreetMap[29] para fornecer funcionalidades como:

- saber a rota mais rápida entre localizações;
- encontrar resultados mais próximos na rede de rua dado coordenadas (*map-matching*);
- calcular a duração/distância da rota mais rápida entre todos os pares de coordenadas (*route matrix*);
- resolver o problema do caixeiro-viajante usando heurísticas "*greedy*";
- gerar *tiles* vetorizados do Mapbox[30] com dados internos de encaminhamento.

Estas funcionalidades estão disponíveis para utilização através de uma REST API, em uma biblioteca C++ e também em NodeJS. O OSRM poderá ser instalado a partir do seu código fonte, ou via um *container* em Docker[31]. Atualmente, o OSRM é utilizado como *backend* [57] na Mapbox, uma plataforma de dados de localização, mais propriamente na sua Directions API [58], fornecendo as

[28] http://project-osrm.org/
[29] https://www.openstreetmap.org
[30] https://www.mapbox.com/
[31] https://www.docker.com/

mesmas funcionalidades descritas acima bem como outras funcionalidades extra, que requerem uma subscrição paga.

Os cálculos de rotas no OSRM são feitos a partir de uma de duas maneiras diferentes de pré-processamento: usando "Contraction Hierarchies" (CH) [59] e "Multi-Level Dijkstra" (MLD) [60], este último baseado no algoritmo de Dijkstra [61]. Estes algoritmos fazem com que seja possível obter resultados de uma forma mais rápida, uma vez que a maioria do cálculo entre distâncias já foi efetuada previamente, o que acontece quando fazemos um pedido a um dos serviços disponíveis. Porém, estes algoritmos apresentam diferenças que poderão afetar o funcionamento do OSRM, uma vez que só podemos escolher um deles. Segundo a documentação [62], o algoritmo CH é o mais indicado para cálculos de longas distâncias, sendo o algoritmo MLD recomendado como o predefinido no uso do OSRM. Recentemente, um utilizador do OSRM mostrou uma comparação entre estes dois algoritmos, onde foi possível verificar que o algoritmo CH apresenta melhores resultados com o custo de um maior tempo de pré-processamento prévio à execução do OSRM [63].

O OSRM torna também possível e por defeito, o uso de dados em tempo-real de trânsito [64], de forma aos cálculos de rotas poderem ser afetados pelo trânsito. No entanto, não nos é fornecido os dados de trânsito, sendo responsabilidade do utilizador fornecê-los e configurar o OSRM para os utilizar.

2.3.1.2. Valhalla

O Valhalla[32] é também um motor de encaminhamento *open-source* feito em C++, tendo pertencido à Mapzen[33], uma empresa que se dedicava à criação de serviços e ferramentas de mapeamento, que encerrou em Fevereiro de 2018, tendo-se tornado posteriormente um projeto da Linux Foundation [65], o que fez com que se tornasse *open-source*. Os seus programadores foram recentemente contratados pela Mapbox, um serviço competidor do Google Maps[34] [66]. Apresenta como funcionalidades:

- obter a rota mais rápida entre duas localizações, descritiva (*turn-by-turn*);
- obter a rota otimizada entre um conjunto de várias localizações;
- cálculo de matriz tempo-distância de um conjunto de várias localizações;

[32] https://github.com/valhalla/valhalla
[33] https://www.mapzen.com/
[34] https://www.google.com/maps/

- cálculo de regiões atingíveis dada a duração da viagem (*isochrones*);
- serviço de mapeamento de localizações a estradas / caminhos do OpenStreetMap (*map-matching*);
- serviço de localização de ruas / estradas dado uma localização;
- obter dados de elevação relativos a um conjunto de localizações, via *encoded polyline* [67] ou área.

Este motor de encaminhamento fornece estas funcionalidades através de uma REST API [68], utilizando JSON[35] como o modo de representação de informação. Os dados são obtidos principalmente a partir do OpenStreetMap e são transformados em *tiles*, que serão usadas para o cálculo das rotas, o que de acordo com os autores permite por exemplo requisitos de memória menores e o *download* de *tiles* para posterior reuso *offline* [69]. O mapeamento de todo o planeta para as *tiles* é feito a partir de três níveis hierárquicos, sendo que o nível 0 contém informações de estradas que são consideradas autoestradas, já o nível 1 contém estradas secundárias e o nível 2 estradas locais [70]. Na Figura 7 podemos ver um exemplo do cálculo de uma rota, onde se pode ver na prática o uso deste modo de representação.

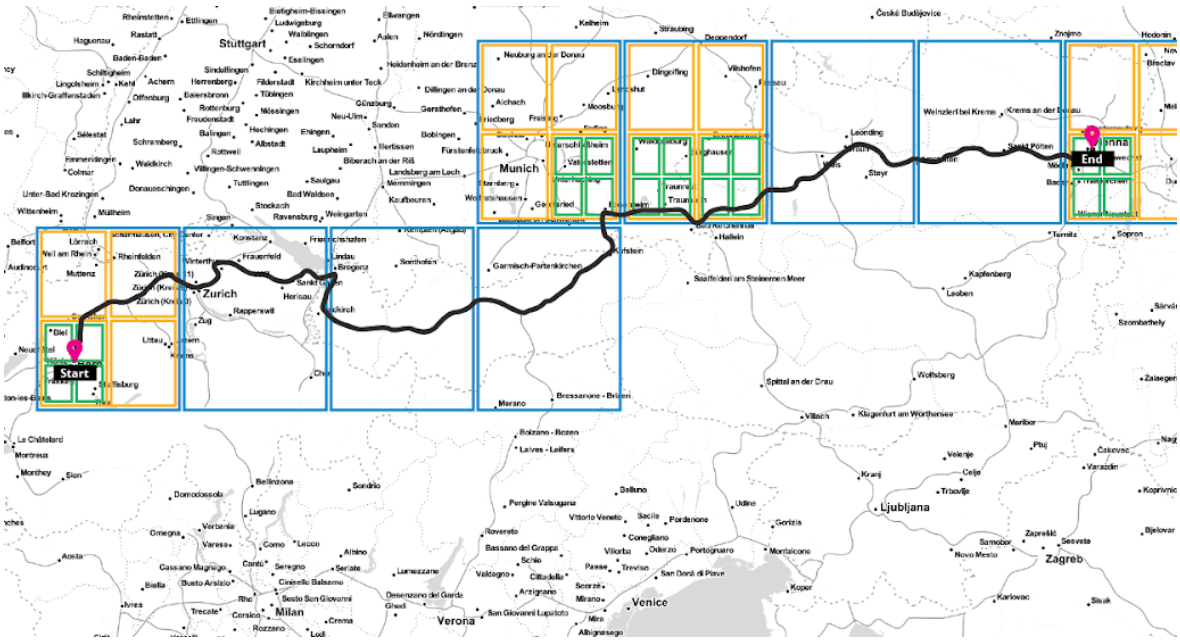

Figura 7 - Cálculo de rota do Valhalla com *tiles* [69]

[35] https://www.json.org

O dito cálculo é feito utilizando vários algoritmos baseados no algoritmo A* (A Star)[71], uma extensão do algoritmo de Dijkstra, permitindo encontrar caminhos de acordo com diversas características (localização em *tiles* semelhantes, características temporais, entre outras)[72], que poderão ser afetadas por custos dinâmicos [73], o que poderá permitir o cálculo de rotas tendo por base dados de trânsito. Neste campo, a Mapzen [74], em conjunto com diversos parceiros neste ramo fez parte do projeto OpenTraffic[36], levado a cabo pelo Banco Mundial em 2016 [75], uma iniciativa que tinha como objetivo medir o trânsito mundial a nível urbano. Esta plataforma[37] utilizava o Valhalla como um dos seus componentes principais, permitindo adicionar às suas *tiles* informação sobre os dados de trânsito recolhidos. Porém, este projeto deixou de ter atualizações um ano depois, sendo que para já, não há previsão de uma possível integração nativa de dados de trânsito no Valhalla que possam afetar os resultados de cálculo de rotas [76], uma vez que o suporte a dados do OpenTraffic foi também removido das tiles na sua última versão [77].

2.3.1.3. pgRouting

O pgRouting[38] é uma extensão para base de dados PostgreSQL[39] e PostGIS[40], que fornece funcionalidades de encaminhamento geoespacial. Incorpora a implementação de algoritmos de pesquisa como o Algoritmo de Johnson [78], Algoritmo de Floyd-Warshall [79][80], A* (A Star) e Algoritmo de Dijkstra, entre outros, o que permite uma total personalização de acordo com o caso de uso pretendido para o cálculo de rotas.

Como se trata de uma extensão para a base de dados PostgreSQL, o meio de utilização do pgRouting passa inevitavelmente por consultas (*queries*) diretas à base de dados utilizando a linguagem SQL, o que requer a ligação direta de um cliente ao SGBD. Desta forma torna-se também possível modificar esses dados, que são refletidos instantaneamente na sua utilização, sem haver necessidade de haver um novo processamento derivado dessas alterações, como por exemplo alterando o custo de uma dada rota.

A introdução de dados nesta ferramenta poderá ser totalmente personalizada, existindo no entanto algumas ferramentas auxiliares que permitem a integração de dados oriundos do

[36] http://opentraffic.io/
[37] https://github.com/opentraffic/otv2-platform
[38] https://pgrouting.org/
[39] https://www.postgresql.org/
[40] http://www.postgis.net/

OpenStreetMap como por exemplo o conversor *osm2pgrouting*[41] que como o nome indica insere esses dados diretamente no pgRouting. A partir daqui é possível executar *queries* que tenham como objetivo calcular o trajeto mais curto entre duas localizações, no OpenStreetMap identificadas por nós (*Nodes*)[81].

2.3.1.4. OpenTripPlanner

O OpenTripPlanner[42] (OTP) surgiu em 2009 como uma solução *open-source* multimodal de planeamento de viagens, sendo que a versão 1.0 foi lançada em 2016 [82]. Permite tanto o planeamento de viagens de transportes públicos, usando a especificação GTFS[43], bem como a nível rodoviário, usando para este meio os dados do OpenStreetMap, onde entre diversas funcionalidades também relacionadas com o uso combinado de transportes públicos se pode obter a rota mais rápida entre duas localizações. Esta ferramenta é inclusivamente utilizada em produção hoje em dia em locais como Los Angeles, Nova Iorque, Noruega e Finlândia, onde ajuda os seus utilizadores a planear as suas viagens utilizando também dados em tempo-real [83].

O OTP consiste em três componentes: um construtor de grafos, um motor de encaminhamento, ambos desenvolvidos em Java, e uma interface de utilizador desenvolvida em JavaScript, que expõe uma REST API [84]. O cálculo das rotas é feito utilizando os algoritmos A* (A Star) e Contraction Hierarchies (CH) baseadas em heurísticas [85].

2.3.1.5. Comparativo entre Soluções

As soluções de *Routing* acima referidas foram escolhidas tendo por base a sua adoção pela comunidade e pelo facto de estas serem *open-source*, não tendo nenhum custo associado à sua integração no âmbito desta dissertação e de trabalho futuro. De entre as cerca de quatro soluções mencionadas, três delas (OSRM, Valhalla, OpenTripPlanner) já expõem uma REST API, o que permite uma interação mais facilitada com as funcionalidades fornecidas, que são muito semelhantes entre si, enquanto que no pgRouting torna-se necessário um estudo mais aprofundado do seu modo de funcionamento para um uso adequado, dado que de todas é a solução que permite maior capacidade de personalização mas também a que apresenta uma curva de aprendizagem mais difícil, uma vez que as funcionalidades que por exemplo as soluções restantes apresentam teriam de ser

[41] https://github.com/pgRouting/osm2pgrouting
[42] http://www.opentripplanner.org/
[43] https://gtfs.org/

implementadas. Uma comparação entre o pgRouting, OSRM e OpenTripPlanner já foi feita em detalhe em 2014 [86], onde se considera como caso de uso utilizar as direções para uma aplicação de *smartphone* para pedestres com cegueira.

Tendo em conta o uso a que se destinam estas ferramentas para otimização de rotas, a decisão passa pela integração de uma das três soluções que já apresentam uma REST API implementada, dado que as suas funcionalidades vão de encontro ao pretendido. Neste sentido, das soluções mencionadas, o OSRM e o Valhalla são as duas soluções *open-source* mais utilizadas e reconhecidas na comunidade, também tendo em conta casos de uso como o Mapzen e o Mapbox, serviços que podem realmente ser vistos como competição à Directions API do Google Maps[44] pois propõem-se ao mesmo. Já o OpenTripPlanner, utilizado extensivamente em países com sistemas de transporte complexos, é uma solução viável para um motor de encaminhamento, no entanto o facto deste também ter em atenção a utilização de transportes públicos torna-o um pouco fora do contexto de utilização que é pretendido. A sua instalação e configuração também exigem um maior número de recursos comparativamente ao OSRM e o Valhalla. Uma comparação entre estas duas últimas soluções foi feita em 2019 pela Telenav [87], uma empresa que opera na indústria de navegação, onde podemos retirar o seguinte: o OSRM apesar de apresentar um *Routing* eficaz com suporte a dados de trânsito bem como ter alta performance, tem um uso de dados monolítico e maior consumo de memória, ao que o Valhalla apresenta um uso de dados flexível através das tiles, suporta a personalização de *Routing*, tem menor consumo de memória e é razoavelmente rápido no *Routing* ponto a ponto. Estes aspetos são importantes dado que o consumo de memória em exagero acarreta maiores custos operacionais.

[44] https://developers.google.com/maps/documentation/directions/start

Solução *Open-Source*	Linguagem Utilizada	REST API	Algoritmos de Pesquisa	Tem suporte nativo a dados de trânsito?
OSRM	C++	Sim	CH MLD	Sim
Valhalla	C++	Sim	A* Dijkstra	Não
pgRouting	SQL	Não	Johnson Dijkstra Floyd-Warshall A*	Sim, mas requer desenvolvimento por parte do utilizador
OTP	Java	Sim	A* CH	Não

Tabela 2 - Tabela comparativa entre as soluções de *Routing open-source* mencionadas

2.4. Previsão da Duração de Viagem entre dois Locais

Conforme discutido no capítulo anterior, a otimização da rota a ser efetuada pelo condutor é uma característica importante: no âmbito da correção do posicionamento e em outros aspetos operacionais é necessário saber com uma margem temporal aceitável o tempo que um condutor demora a dirigir-se entre dois locais. Trata-se de um fator decisivo para por exemplo poder-se selecionar numa área de operação o melhor condutor a efetuar um novo serviço.

Todas as soluções mostradas no capítulo 2.3.1 permitem obter a duração prevista para o trajeto a efetuar entre dois locais através das suas APIs. No entanto, os conjuntos de dados que estas soluções utilizam para a estrutura de dados que calcula os trajetos não têm (*out of the box*) um conhecimento real de fatores externos como o trânsito, estradas cortadas por obras ou outro incidente rodoviário. Para além disso, o número de pedidos a serem feitos às APIs dessas ferramentas num contexto de otimização torna-se excessivo, uma vez que para cada viagem possível terá de se efetuar um ou mais pedidos de modo a obter uma otimização generalizada, o que não é exequível, da mesma forma que não se torna uma solução escalável com o aumento de pedidos

concorrentes. O conhecimento adquirido pelos condutores durante os trajetos já feitos e a sua experiência de navegação e orientação é também um fator a reter. Um condutor experiente, por exemplo um taxista, sabe de atalhos numa cidade, os dias em que há mais serviços, entre outras características só obtidas pela sua experiência em várias horas de condução [88]. Em Londres, os pretendentes a taxista (os famosos *Black Taxis*) têm de superar um dos desafios mais difíceis da indústria, conhecido por "*The Knowledge*". Uma série de testes em que os condutores deverão saber 320 rotas, 25 mil ruas e 20 mil pontos de interesse da capital inglesa sem qualquer tipo de sistema de navegação para o auxiliar, pois não é permitida a sua utilização [89].

A análise deste tipo de dados, recolhidos neste contexto, através da circulação rodoviária no dia-a-dia dos taxistas torna-se extremamente útil para compreender, aprender e prever futuros acontecimentos. Um desses exemplos é a previsão de uma viagem entre dois locais, que poderá ser prevista tendo em conta diversos fatores que de outra forma não são trivialmente conseguidos utilizando uma das soluções discutidas anteriormente. O que aqui se propõe é a criação de um modelo de previsão criado por aprendizagem automática através deste tipo de dados, uma vez que se pretende prever a duração do trajeto entre dois locais numa área de operação da LUGGit, recolhendo dados através dos Keepers da plataforma. Yang et al. (2010)[90] por exemplo já mostraram a validade de um método de regressão ao construir um modelo para previsão de tempos de viagem utilizando dados veiculares.

Um dos métodos utilizados na aprendizagem automática para regressão e classificação generalizada é a árvore de decisão [91], onde cada nó interno corresponde a um teste a um atributo (*feature*) do conjunto de dados, cada um dos ramos da árvore o resultado desse teste, e cada nó-folha a decisão tomada após análise de todos os atributos por parte do algoritmo, dada a classe (*label*) pretendida [92]. As árvores de decisão são construídas através da totalidade do conjunto de dados, e caso este seja de tal forma complexo, poderá não tomar a decisão ótima global à medida que esta é construída, o que por exemplo acontece com o algoritmo CART [93], um dos algoritmos usados para escolha dos atributos de teste. Então, para minimizar potenciais erros de decisão, Tin Kam Ho (1995)[94] propôs a criação de um conjunto de árvores de decisão, onde o conjunto de dados a analisar por cada uma dessas árvores é aleatório, método que denominou de *Random Forest*. O resultado deste método é obtido através de um sistema de votos, onde no caso de se tratar de classificação a classe vencedora será escolhida, ao que para a regressão será considerada a média de todos os resultados.

O uso de *Random Forests* para previsão de tempo de viagem entre locais já foi discutido na literatura. Elhenawy et al. (2014)[95] usaram este método com uma matriz de velocidades através de dados recolhidos via GPS. Zhang & Haghani (2015)[96] compararam as RF com outro método derivado das árvores de decisão de regressão, *Gradient Boosting*. E recentemente, Song & Zhou (2020)[97] sugeriram um modelo de previsão usando RF conjugando um algoritmo de *clustering*, de modo a agregar regiões de alta densidade. São exemplos que comprovam o uso deste método, o que é interessante avaliar no âmbito desta dissertação.

3. Especificação de Requisitos e Arquitetura

Pretende-se com esta dissertação propor uma solução que permita aos Keepers corrigir o seu posicionamento para localizações com maior possibilidade de ocorrência de novos pedidos. Ao mesmo tempo, pretende-se também melhorar a seleção de Keepers de modo a que estes consigam chegar até junto do cliente em menos de 15 minutos, o que irá reduzir o tempo de serviço no geral.

Neste capítulo abordam-se os requisitos necessários para a construção do sistema, bem como a arquitetura dos vários componentes a serem desenvolvidos para atingir os objetivos propostos.

3.1. Objetivos Específicos

O problema a resolver tem importância para o contexto operacional da LUGGit, sendo este um serviço em tempo-real. A solução aqui proposta irá permitir à empresa reduzir o tempo de resposta a novos serviços, aumentando a satisfação do cliente bem como a sua rentabilidade devido à introdução de pedidos encadeados para cada Keeper, o que tem um impacto direto na utilização da mesma.

Apesar já existirem soluções integradas nas plataformas de mobilidade relativamente à correção de posicionamento dos seus condutores, a inexistência de informação de domínio público sobre o modo como estas resolvem o problema permite-nos sugerir uma solução. Pretende-se também resolver este problema utilizando tecnologias e ferramentas de uso livre, com um maior foco em tecnologias *open-source*.

A nível pessoal, considero este problema um excelente desafio, tanto a nível logístico e tecnológico, sendo que me dará a possibilidade de poder trabalhar com diversas tecnologias, e que em última instância, permitindo ajudar a empresa a crescer, sendo o progresso e inovação um ponto fulcral numa *startup*.

Assim, de acordo com o problema identificado e descrito acima, definiu-se os seguintes objetivos a cumprir no âmbito desta dissertação:

- Objetivo 1: Analisar os trajetos ótimos a percorrer pelos Keepers na realização de serviços da LUGGit;
- Objetivo 2: Detetar o posicionamento atual dos Keepers e guardar novas localizações e pedidos de serviço;
- Objetivo 3: Corrigir o posicionamento dos Keepers através de *heatmaps* e análise de rotas;
- Objetivo 4: Selecionar os Keepers em melhores condições de posicionamento para responder a um pedido em menos de 15 minutos.

3.2. User Stories

Uma vez que a solução proposta se trata de lógica aplicacional situada no *backend*, sem contacto direto com o utilizador final da aplicação, as User Stories criadas estão intrinsecamente ligadas ao ponto de vista do programador, que irá utilizar APIs que serão criadas de forma a este poder interagir com o sistema. Todas as chamadas às APIs criadas serão apenas acessíveis por utilizadores autorizados, o que implica a utilização de um *token* de autenticação.

US1: Como utilizador autorizado, posso obter o caminho mais próximo entre duas coordenadas GPS.

US2: Como utilizador autorizado, posso obter sugestões de locais próximos de uma coordenada GPS.

US3: Como utilizador autorizado, posso obter uma coordenada GPS tendo por base uma morada ou local.

US4: Como utilizador autorizado, posso saber se me encontro dentro de uma área de operação da LUGGit.

US5: Como utilizador autorizado, posso obter a lista de áreas de operação disponíveis na LUGGit.

US6: Como utilizador autorizado, posso obter a lista de parceiros da LUGGit próximos de uma coordenada GPS.

US7: Como utilizador autorizado, posso obter a lista de sugestões de pontos de interesse próximos de uma coordenada GPS.

US8: Como utilizador autorizado, posso obter a lista de pontos de encontro próximos de uma coordenada GPS.

US9: Como utilizador autorizado, posso enviar a minha localização atual.

US10: Como utilizador autorizado, posso obter um *heatmap* de uma área de operação tendo por base o estado da operação da LUGGit em tempo-real.

US11: Como utilizador autorizado, posso obter um *heatmap* de uma área de operação tendo por base o estado da operação da LUGGit no passado.

US12: Como utilizador autorizado, posso obter uma lista dos Keepers mais próximos de uma coordenada GPS.

3.3. Desenho da Solução

Toda a solução da LUGGit encontra-se baseada numa arquitetura orientada a microserviços, de modo a tornar cada componente aplicacional distinto dos demais e acima de tudo para promover a escalabilidade, fornecendo transparência e autonomia nas operações a realizar por cada serviço desenvolvido. Assim, cada microserviço a ser desenvolvido deverá ter uma lógica e funcionamento independentes, e de modo a atingir os objetivos definidos no capítulo 3.1 vão ser desenvolvidos três microserviços no âmbito desta dissertação.

Para podermos encontrar o trajeto ótimo a percorrer pelos Keepers, e desta forma atingir-se o Objetivo 1, necessitamos de um serviço capaz de receber localizações, analisar e indicar posteriormente a melhor rota a efetuar, e assim criou-se o *Serviço de Routing*, que também será útil para determinar o tempo que um condutor demora a efetuar um trajeto entre duas localizações, algo que será útil para atingir o Objetivo 4.

O tratamento de dados de contexto geoespacial torna-se também fulcral para a operação do Keeper, sendo necessário providenciar informação auxiliar que lhe permita realizar serviços da maneira mais rápida e eficaz possível, fornecendo pontos de encontro com o cliente na recolha e

entrega, bem como a deteção e contextualização de dados geoespaciais, por via das localizações dos utilizadores e pedidos, sendo que para esse efeito foi criado o *Serviço de Geofencing*.

Para armazenarmos e posteriormente procedermos à análise de dados que nos permita corrigir o posicionamento dos Keepers, criou-se um serviço responsável não só pela recolha de dados bem como pelo seu processamento, análise e criação de *heatmaps* que forneçam informação visual auxiliar ao Keeper, surgindo o *Serviço de Correção de Posicionamento de Keepers*, que em conjunto com os serviços restantes fará com que seja possível resolver os objetivos restantes.

Foi também desenvolvido no âmbito desta dissertação um modelo de aprendizagem automática que permitiu prever com base em dados reais de condução a duração de viagens entre duas localizações. Apesar de não ser um dos objetivos desta dissertação, procedeu-se ao seu desenvolvimento como caso de estudo, o que permitiu analisar e comparar resultados com o *Serviço de Routing*, bem como validar um método de seleção de Keepers tendo em conta o Objetivo 4.

3.3.1. Requisitos Não-Funcionais

A infraestrutura da LUGGit foi pensada desde o início da sua criação com objetivos bem definidos ao nível da escalabilidade. Todo o produto, lógica e consequentemente o *software* está pensado para uma perspetiva de longo prazo, o que se transpôs para o trabalho a desenvolver no âmbito desta dissertação. Deste modo, todos os requisitos não-funcionais descritos em seguida são os mesmos para todos os serviços desenvolvidos.

RNF-1: Documentação e Versionamento - O serviço deverá expor uma API bem-definida e documentada, de modo a ser fácil de utilizar e de ser compreendida. Ao mesmo tempo deverá ter em atenção o seu versionamento, de modo a suportar retrocompatibilidade para posteriores alterações.

RNF-2: Estabilidade - O serviço deverá ser fiável, de maneira a que as suas operações não comprometam o seu uso esperado, tratando-se de um serviço que será usado em tempo-real em aplicações móveis. O tratamento de erros deverá ser abordado cuidadosamente, e deverá ser bem-definido e documentado, através de códigos e mensagens de erro que permitam identificar de uma forma direta e eficaz o erro, quando ocorrido. Da mesma forma, o serviço também deverá ser capaz

de validar a informação enviada a partir da API, de modo a garantir a integridade da informação, evitando erros.

RNF-3: Redundância - Perante uma ocorrência de erros internos, o serviço deverá ser capaz de recuperar dos mesmos e continuar a sua execução.

RNF-4: Segurança - O serviço deverá ser seguro, na medida em que deverá permitir apenas o seu uso a utilizadores autorizados para o efeito. Todas as comunicações deverão ser feitas usando protocolos que permitam uma comunicação segura.

RNF-5: Performance - O serviço deverá ser rápido a responder a pedidos da API, uma vez que será utilizado em tempo-real em condições de execução exigentes, decorrendo da utilização em aplicações móveis, cuja conectividade poderá ser limitada.

RNF-6: Escalabilidade - O serviço deverá ser escalável, de modo a corresponder à rapidez de resposta esperada, referida no RNF-5, bem como ser capaz de lidar com uma grande quantidade de pedidos e de utilizadores, tanto em simultâneo (pedidos concorrentes) como acumulados (pedidos sequenciais), uma vez que se espera que a LUGGit aumente cada vez mais o número de cidades em operação, o que irá aumentar o número de utilizadores, sejam clientes ou condutores.

3.3.2. Serviço de Routing

Motivação e Requisitos

Para as pessoas que circulam todos os dias em veículos numa determinada cidade, torna-se cada vez mais útil o uso de equipamentos eletrónicos que permitam saber a localização atual a cada instante, obtendo indicações sobre o caminho que estes deverão seguir para chegar ao seu destino de forma mais rápida.

A maioria dos fabricantes de automóveis estão progressivamente a incorporar nos veículos sistemas de navegação *on-board* cada vez mais avançados, com integração facilitada ao *smartphone* e com o uso de aplicações [98] também com recurso aos sistemas operativos Apple Carplay[45] e

[45] https://www.apple.com/ios/carplay/

Android Auto[46], específicos para automóveis. No caso específico da LUGGit, o *smartphone* é o dispositivo principal de uso, tanto do ponto de vista do utilizador que pretende utilizar o serviço tanto do ponto de vista dos Keepers, onde irão receber novos pedidos de serviço, aceitar esses mesmos pedidos e dirigirem-se até ao utilizador, sendo também necessário disponibilizar ao Keeper o trajeto a ser percorrido até chegar ao utilizador (seja na recolha ou na entrega) e ao centro logístico para armazenamento das bagagens.

Torna-se assim essencial o desenvolvimento de um serviço de Routing (encaminhamento) que permita obter o trajeto a ser percorrido pelo Keeper, bem como detalhes associados ao mesmo, como o tempo esperado para efetuar aquele trajeto e a distância a percorrer. Outro dos requisitos necessários a incorporar neste serviço é a funcionalidade de *geocoding*, isto é, a obtenção de uma coordenada a partir de uma morada ou local (ao que é chamado de *Forward Geocoding*), e vice-versa, ou seja, a obtenção de uma morada ou local (ao que é chamado de *Reverse Geocoding*).

Neste sentido, a primeira solução foi usar os serviços da Google, nomeadamente as APIs Directions, Places Autocomplete, Places Detail e Geocoding do Google Maps[47], que no caso específico da Directions API permite calcular o trajeto entre um ponto de origem e de destino fornecendo no resultado dados como a distância total do caminho, o tempo estimado de viagem, bem como a disponibilização de uma *encoded polyline* que permite representar o caminho previsto num mapa, codificando as várias localizações do trajeto num conjunto de carateres [67]. As APIs de *geocoding* do Google Maps, nomeadamente a Geocoding API, permitem-nos efetuar pedidos de *Forward* e *Reverse Geocoding* e tal como descrito acima, a API Places Autocomplete permite-nos obter sugestões de moradas/locais de acordo com especificações como por exemplo o raio de localização, língua, entre outros, sendo que a API Places Detail nos permite obter detalhes sobre um local em específico, através do identificador de um local atribuído por esse serviço.

O Google Maps é atualmente um dos serviços de mapas mais utilizado no mundo. No entanto, o uso de um destes serviços *online* acarreta maiores custos, tais como o custo por número de chamadas à API, entre outros. Fez-se então uma investigação sobre as soluções *open-source* de Routing e de *geocoding* já existentes, onde o maior foco deve ser aplicado em soluções que pudessem ser implementadas internamente, de forma a reduzir os custos operacionais bem como

[46] https://www.android.com/auto/
[47] https://developers.google.com/maps/documentation/

ter em atenção aspetos importantes como o desempenho, estabilidade, e a possibilidade de importar dados externos, como por exemplo dados referentes ao trânsito, informação que poderá ser no futuro de extrema importância, para tornar este serviço no mais completo e assertivo possível, uma vez que poderá ter influência no trajeto a indicar ao Keeper.

3.3.2.1. Requisitos Funcionais

RF-1: O serviço deve fornecer uma API acessível a partir de um URL.

RF-2: O serviço deve fornecer um *endpoint*[48] na sua API para obter o trajeto mais curto entre duas localizações (*Directions*), onde deve ter em atenção as seguintes características: ser capaz de evitar portagens, ferries e autoestradas no cálculo do seu trajeto; permitir calcular o trajeto com localizações intermédias entre o início e o destino; a unidade de medida do utilizador (unidade métrica ou imperial); na resposta, deve indicar a duração e distância do trajeto calculado, bem como fornecer uma *encoded polyline* que represente o trajeto resultante.

RF-3: O serviço deve fornecer um *endpoint* na sua API para obter uma localização (latitude, longitude) tendo por base uma morada/local (*Forward Geocoding*).

RF-4: O serviço deve fornecer um *endpoint* na sua API para obter uma morada/local tendo por base uma localização (latitude, longitude) (*Reverse Geocoding*).

RF-5: Para que este serviço seja de facto um serviço de *Routing* completo, mesmo não necessário no contexto desta dissertação, foi adicionado um *endpoint* na sua API para obter um conjunto de moradas/locais e sua localização, tendo por base um nome (*Autocomplete*), onde os seus resultados deverão ser influenciados pela localização e um raio máximo de procura, bem como deverão ser limitados por um número máximo de resultados, indicado no pedido.

RF-6: O serviço deve fornecer um *endpoint* na sua API para obter um conjunto de polígonos que indiquem o trajeto máximo possível a efetuar a partir de uma localização dentro de um período de tempo finito (*Isochrone*).

[48] URL que representa um canal de ligação entre o cliente e o servidor através de uma API.

RF-7: Cada *endpoint* deve ter em atenção a linguagem do utilizador, e caso sejam utilizados vários fornecedores intermédios de serviço, deverá indicar qual foi o fornecedor da resposta.

RF-8: O serviço deverá ser capaz de verificar o estado operacional de cada fornecedor de serviço, quando estão vários em existência, e neste caso deverá escolher um deles, que esteja disponível.

3.3.2.2. Soluções de *Geocoding* Existentes

De forma a tornar este serviço o mais completo possível, resolveu-se estudar soluções de g*eocoding* que possam oferecer um serviço de procura e resolução de locais, que apesar de não se enquadrar no âmbito desta dissertação permitirá por exemplo obter locais por intermédio de localizações num contexto de análise de dados. De seguida apresentam-se e analisam-se algumas soluções encontradas.

3.3.2.2.1. Pelias

O Pelias[49] é um *geocoder open-source*, inicialmente criado como um projeto da Mapzen, e que se tornou *open-source* após o encerramento da Mapzen em 2017 (à semelhança do Valhalla, discutido anteriormente). Utiliza diversas fontes de dados distintas, que são importados para um motor de pesquisa em Elasticsearch e posteriormente expostos através de uma REST API, interagindo com serviços que em conjunto fornecem resultados de *Forward Geocoding* e *Reverse Geocoding*.

A importação de dados é proveniente do OpenStreetMap, que importa os seus nós (*Nodes*) e ruas (*Ways*), bem como dados do OpenAddresses[50], que reúne moradas a nível mundial, do Who's On First[51] e do Geonames[52], que reúnem informações de áreas administrativas (continentes, países, distritos / estados, cidades, entre outros) e de locais em geral.

[49] https://pelias.io/
[50] https://openaddresses.io/
[51] https://www.whosonfirst.org/
[52] http://www.geonames.org/

3.3.2.2.2. Nominatim

O Nominatim[53] é o *geocoder* oficial do OpenStreetMap. Consiste numa base de dados PostgreSQL bem como numa REST API feita em PHP [99]. Utiliza como fonte de dados principal o OpenStreetMap, mas também suporta outras fontes de dados, como o TIGER (Censos dos EUA)[100], GB Postcodes (códigos postais do Reino Unido) [101], bem como do Wikipédia/Wikidata [102]. Após a importação dos dados, estes são processados e indexados na base de dados utilizando *triggers* SQL. Assim passam a estar prontos a utilizar através da REST API, que fornece as funcionalidades de *Forward Geocoding* e *Reverse Geocoding*.

3.3.2.2.3. Photon

O Photon[54] é um *geocoder open-source*, projeto criado pelo website Komoot[55], que é um planeador de atividades *outdoor*. Desenvolvido utilizando o motor de pesquisa Elasticsearch, e uma REST API desenvolvida em Java [103], teve o seu primeiro lançamento em 2014. Utiliza como fonte de dados o OpenStreetMap, todavia também permite importar dados que estejam no Nominatim, aumentando os dados à sua disposição para resultados multilingue.

3.3.2.2.4. Mimirsbrunn

O Mimirsbrunn[56] é um *geocoder open-source* desenvolvido em Rust[57] que também utiliza o motor de pesquisa Elasticsearch, e que tal como os restantes serviços já mencionados utiliza uma REST API que permite acesso aos seus dados, tais como pontos de interesse, ruas, moradas e paragens de transportes públicos. Este projeto surgiu pelas mãos da Navitia[58], uma empresa que oferece serviços de mapeamento, à semelhança da Mapzen e da Mapbox, empresas anteriormente mencionadas.

[53] http://nominatim.org/
[54] http://photon.komoot.de/
[55] https://www.komoot.de/
[56] https://github.com/CanalTP/mimirsbrunn
[57] https://www.rust-lang.org/
[58] https://www.navitia.io/

3.3.2.2.5. Comparativo entre soluções

Tal como referido no comparativo entre soluções de Routing, os requisitos mantém-se: encontrar uma solução que possa ser implementada internamente e que não envolva custos na sua utilização e integração. Neste sentido, as opções *open-source* aqui mencionadas são as mais utilizadas e em constante manutenção e atualizações por parte da comunidade. Todas as soluções oferecem uma REST API, bem como as mesmas funcionalidades, pelo que a comparação mais justa deve ser feita através de alguns detalhes que acabam por as distinguir entre si.

Em termos da tecnologia de armazenamento e tratamento dados utilizada, três destas soluções optaram pela utilização do Elasticsearch, uma ferramenta de pesquisa de texto, que permite fornecer "*out of the box*" uma pesquisa rápida aos dados nele inseridos, em comparação com o PostgreSQL, que não está otimizado para pesquisas de texto [103].

No que diz respeito às fontes de dados, o Pelias tem vantagem, permitindo incorporar quatro fontes de dados diferentes, o que num contexto de fornecer resultados para a operação de *Autocomplete* é de facto muito vantajoso. De facto, todas as soluções usam dados provenientes do OpenStreetMap, sendo que aqui quanto maior for a quantidade de dados a incluir na pesquisa maior será o número de resultados possíveis a dar ao utilizador. No entanto, comparando com as restantes soluções, o Pelias é a única que não suporta dados multilingue, o que prejudica os resultados a obter nas suas funcionalidades. Já o Nominatim, sendo o *geocoder* oficial do OpenStreetMap, em conjunção com dados obtidos via Wikipédia/Wikidata permite obter maior volume de resultados. Se a este facto podermos juntar a capacidade por parte do Photon poder importar dados diretamente do Nominatim temos duas hipóteses válidas para uma possível implementação a escolher, ao contrário do Mimirsbrunn, que apenas permite importar dados do OpenStreetMap.

Solução *Open-Source*	Fontes de Dados	Armazenamento / Tratamento de Dados	Linguagem	Suporte Multilingue
Pelias	OSM OpenAddresses Who's On First Geonames	Elasticsearch	Node.js Go	Não
Nominatim	OSM Wikipedia/Wikidata	PostgreSQL	C PHP SQL	Sim
Photon	OSM Nominatim	Elasticsearch	Java	Sim
Mimirsbrunn	OSM	Elasticsearch	Rust	Sim

Tabela 3 - Tabela comparativa entre as soluções de *geocoding open-source* mencionadas

3.3.2.3. Discussão

Apesar de um dos requisitos ser a utilização de ferramentas de uso e integração livre, ao instalar soluções *open-source* tanto de Routing como de *geocoding*, não quer dizer que esta seja de facto a melhor solução para o problema. Como já foi mencionado, a primeira solução implementada passou por adotar os serviços da Google, nomeadamente a Google Maps Platform, para fornecer os dados que necessitamos, tanto para Routing como para *geocoding*. Contudo, trata-se de uma solução que é paga, o que não vai de acordo com os requisitos definidos.

Existe uma grande variedade de serviços *Web* que fornecem o mesmo tipo de funcionalidades, nos quais o que os distingue é a quantidade de dados que reúnem comparativamente com os dados do OpenStreetMap, ainda que sendo soluções pagas. Por exemplo, o Google My Business[59] faz com que a localização de qualquer empresa consiga estar presente nas suas plataformas, como o Google

[59] https://www.google.com/business/

Maps, e consequentemente, disponível nas suas APIs. A aquisição do Waze[60] por parte da Google [103], aplicação que outrora foi vista como grande concorrente ao Google Maps, demonstra que a Google está empenhada em melhorar o Google Maps. O Waze é uma aplicação conhecida pela cooperação entre os seus utilizadores, permitindo reportar ocorrência como acidentes de viação, radares, operações STOP, e fornecer dados precisos de trânsito.

Já a TomTom[61], que foi durante muitos anos uma das principais marcas no mercado dos sistemas de navegação GPS, ainda hoje reúne um grande número de utilizadores, através de parcerias com grandes marcas do ramo automóvel[62] que usam os seus serviços de localização nos seus dispositivos de *infotainment*, bem como através da oferta de serviços *Web* para *developers*, fazendo com que também seja uma solução paga a considerar. Outros serviços pagos como o Mapbox, HERE[63], MapQuest[64] e GraphHopper[65] têm ofertas de Routing e *geocoding*, e poderiam também ser perfeitamente soluções viáveis e adequadas ao problema, uma vez que não requerem manutenção e atualizações como as soluções *open-source* anteriormente referidas.

Portanto, para desenvolver este serviço optou-se por usar o Valhalla como solução de Routing, fornecendo os trajetos ótimos entre várias localizações, bem como o uso de *isochrones*, que serão úteis para percebermos a distância a que os Keepers se encontram do utilizador no âmbito de um serviço. Para fornecer as funcionalidades de *geocoding*, escolheu-se combinar duas soluções distintas: o Nominatim e o Photon. O uso destas duas soluções em simultâneo permite fornecer funcionalidades extra ao mesmo tempo que é aproveitada a tecnologia de pesquisa de texto usando uma ferramenta especificamente criada para o efeito, o que é o caso do Elasticsearch.

3.3.2.4. Arquitetura Proposta

Tendo em conta os requisitos discutidos anteriormente, propõe-se a seguinte arquitetura para o serviço de Routing. Este serviço será utilizado pelas duas aplicações da LUGGit, tanto pelo utilizador (o cliente do serviço) como pelo Keeper (para obtenção de rotas e métricas). As tecnologias utilizadas neste serviço serão discutidas mais em detalhe no capítulo de desenvolvimento da solução.

[60] https://www.waze.com/
[61] https://www.tomtom.com/
[62] https://www.tomtom.com/industries/automotive/
[63] https://www.here.com/
[64] https://www.mapquest.com/
[65] https://www.graphhopper.com/

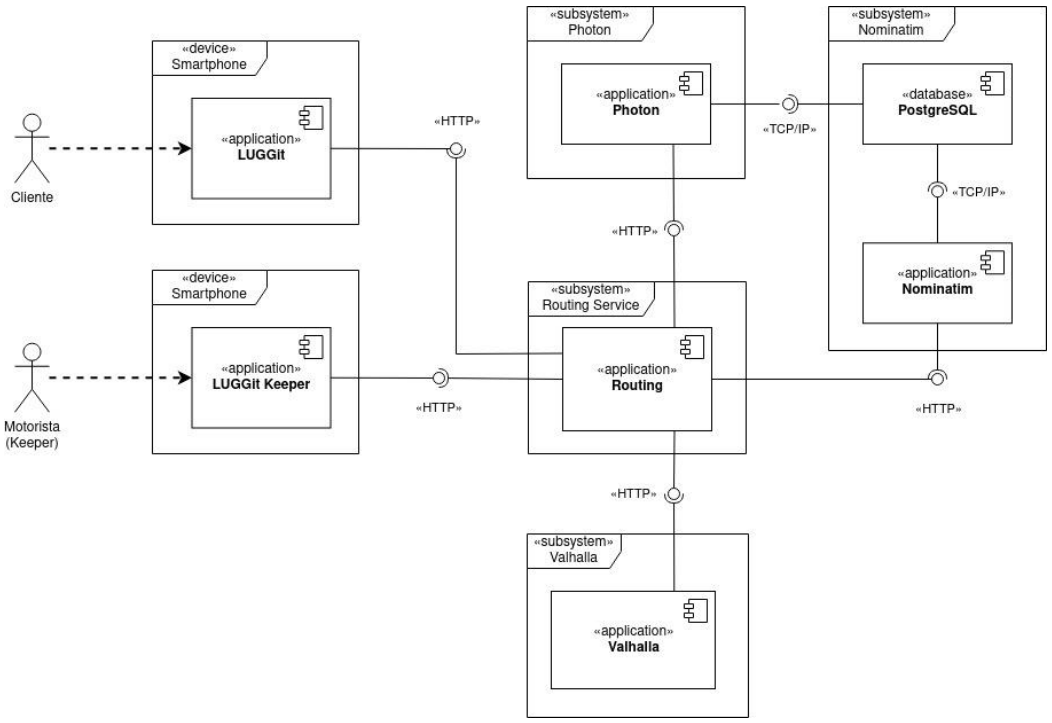

Figura 8 - Arquitetura Geral do serviço de Routing

3.3.3. Serviço de Geofencing

Motivação e Requisitos

A localização é um fator chave para as aplicações da LUGGit, cuja operação dos serviços em cada uma das cidades disponíveis é oferecida através de áreas denominadas de áreas de operação. Cada uma dessas áreas de operação (por exemplo, Lisboa ou Porto) representa a zona onde é possível requisitar um serviço da LUGGit, tendo locais e pontos de interesse que são relevantes em termos operacionais.

Por exemplo, os aeroportos fazem parte desse tipo de locais, sendo um dos principais pontos de entrada e saída no país por parte de turistas e viajantes, dado que estes na sua maioria trazem consigo bagagens, e portanto tornam-se no público-alvo ideal. Assim, estes pontos de interesse apresentam um conjunto de características e dados importantes à operação, tais como locais próprios de paragem para veículos externos (exceto táxis e autocarros), custos e limites de paragem e estacionamento de veículos, entre outras características. Durante um serviço da LUGGit pretende-se fornecer ao utilizador, quer seja um cliente ou um Keeper, toda a informação que este necessitar para utilizar a sua aplicação respetiva da forma mais simples e eficaz possível, para que desse modo

este saiba exatamente o que fazer quando por exemplo chega ao aeroporto. Neste caso, o cliente necessita de saber qual a localização do Keeper, se tem de se deslocar até junto dele, e da mesma forma o Keeper deve saber qual o sítio onde deve parar ou estacionar enquanto espera pelo cliente, se o estacionamento é gratuito ou não, qual o melhor local para se encontrar com o cliente, entre outras informações que no âmbito de um serviço possam ser úteis. Tal como os aeroportos, existem locais como estações de comboios, assim como locais onde haja no momento ocorrência de eventos, sendo igualmente importantes para a operação, uma vez que estes poderão ter características específicas (por exemplo estradas ou circulação cortada) e é necessário agilizar e facilitar da forma mais eficiente possível o encontro entre o cliente e o Keeper, tendo em vista uma recolha e entrega das malas mais facilitada.

Portanto, existe uma necessidade real de tratar os dados geoespaciais para que o Keeper possa realizar o processo de recolha e de entrega da melhor forma. Para este não perder tempo ao procurar um local de paragem ou estacionamento para se encontrar com o cliente, deverá ser indicado um ou mais pontos de encontro. Também, no caso de um dos locais de recolha ou entrega coincidir com os casos apresentados anteriormente, deverão ser facultadas características extra que permitam ao Keeper ter um conhecimento real do terreno, o que fará com que o serviço ocorra da forma desejada. Obter o conhecimento dos locais e pontos de interesse numa área de operação torna-se assim fundamental para a operação, o que irá contribuir para atingir os objetivos propostos.

Outra funcionalidade que poderá ser usada neste âmbito é a deteção e contextualização de localizações e pesquisa geoespacial. Saber se um Keeper ou um utilizador está de facto numa das áreas de operação, se este se encontra numa área onde seja necessário fornecer outro tipo de dados conforme discutimos anteriormente, procurar ou contextualizar locais são funcionalidades que poderão ser feitas através da definição de uma ou várias *geofences*, isto é, a definição de uma cerca (*fence*, em Inglês) geográfica representando cada área pretendida.

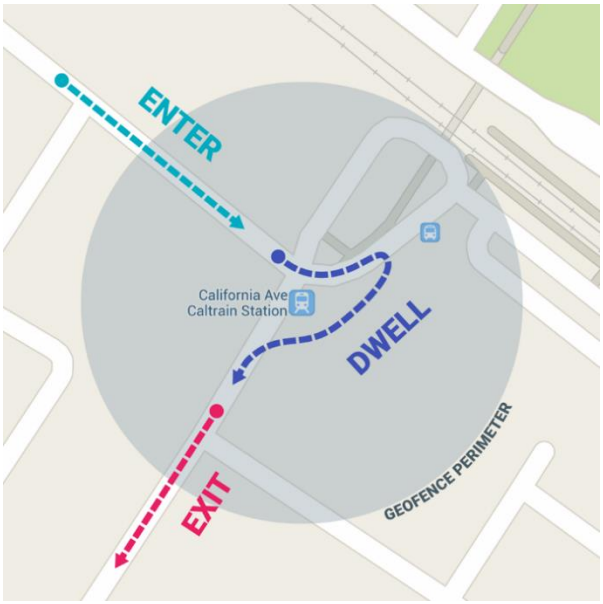

Figura 9 - Exemplo de uma *geofence* [103]

Assim, o desenvolvimento de um serviço que permita fornecer informação com contexto geoespacial é importante no âmbito desta dissertação, e para o futuro. Não só a deteção de localizações e regiões geográficas próximas que permitam indicar o posicionamento ao Keeper na realização de recolhas e entregas, como também fornecer dados contextualizados de pontos de interesse que possam ser úteis para a interação de ambos os utilizadores (seja o cliente ou o Keeper) nas aplicações da LUGGit, como por exemplo parceiros (unidades hoteleiras e de alojamentos que recomendam o serviço aos seus hóspedes), dados esses que poderão ser fornecidos visualmente nas aplicações, sugerindo por exemplo pontos de encontro entre o cliente e o Keeper, ou para uma posterior análise através de outros serviços, como por exemplo o Serviço de Correção de Posicionamento de Condutores, como se explicará adiante.

3.3.3.1. Requisitos Funcionais

RF-1: O serviço deve fornecer uma API acessível a partir de um URL.

RF-2: O serviço deve fornecer um *endpoint* que com base numa localização indique se esta se encontra dentro de uma área de operação da LUGGit.

RF-3: O serviço deve fornecer um *endpoint* que com base numa localização e num raio de procura devolva os pontos de interesse mais próximos.

RF-4: O serviço deve fornecer um *endpoint* que com base numa localização devolva os parceiros mais próximos.

RF-5: O serviço deve fornecer um *endpoint* que com base numa localização devolva os pontos de encontro mais próximos.

RF-6: O serviço deve fornecer um *endpoint* que devolva as áreas de operação (cidades) disponíveis.

RF-7: Cada *endpoint* deve ter em atenção a linguagem do utilizador.

3.3.3.2. Arquitetura Proposta

Vista Geral

Tendo em conta os requisitos discutidos para este serviço, propõe-se a seguinte arquitetura, que irá consistir no uso de duas bases de dados geoespaciais: o Tile38[66] e o PostgreSQL com a extensão PostGIS. O Tile38 será utilizado para fornecer e armazenar dados utilizados no momento, sendo que o uso do PostgreSQL em conjunto com o PostGIS será utilizado para armazenamento em bruto, com maior quantidade de dados comparativamente ao anterior. À semelhança do serviço de Routing, este serviço será utilizado pelas duas aplicações da LUGGit, e os detalhes relativamente à sua implementação serão abordados adiante no capítulo de desenvolvimento.

[66] https://tile38.com/

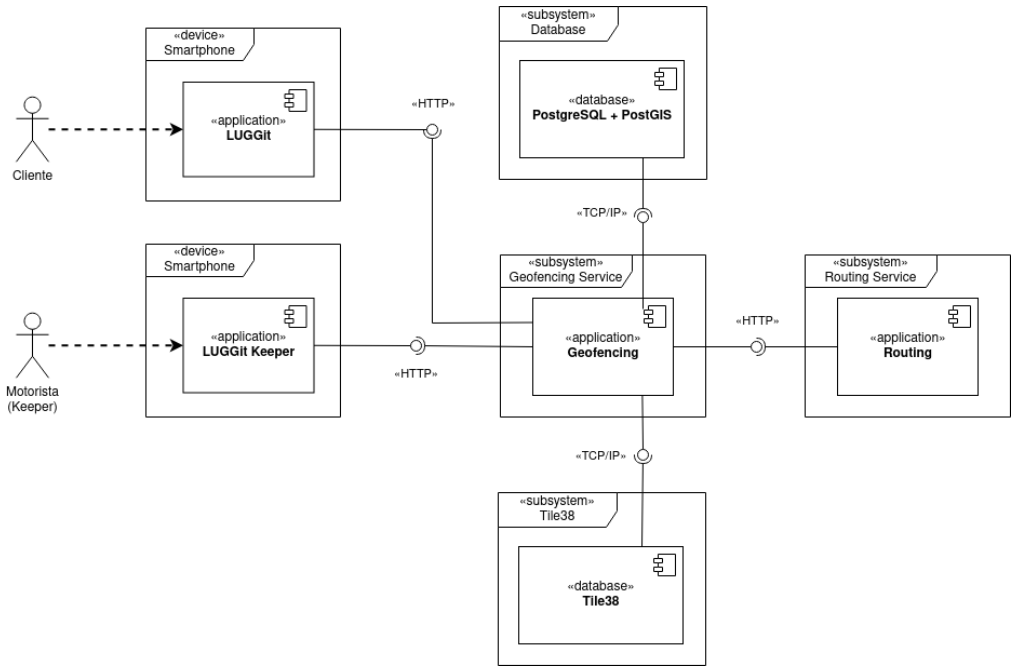

Figura 10 - Arquitetura Geral do serviço de Geofencing

Esquema de Dados

De modo a poder-se guardar informação relativa a áreas de operação, aeroportos e seus terminais, eventos, estações bem como os seus respetivos pontos de encontro e traduções foi necessário proceder-se à sua modelação de dados, cujo diagrama está disponível no Anexo A.

3.3.4. Serviço de Correção de Posicionamento de Condutores

Motivação e Requisitos

O grande objetivo desta dissertação é poder corrigir o posicionamento dos condutores com base em dados recolhidos em tempo-real e no histórico da operação, ou seja, indicar aos Keepers quais os locais indicados para estes se dirigirem, de modo a aceitar o próximo pedido de serviço o mais rápido possível. Não só são importantes os locais onde os Keepers circulam durante o seu trabalho, como também obter locais marcados pela ocorrência de eventos relacionados com o próprio serviço, de forma a identificá-los de uma melhor forma: quando e onde foi a recolha, a entrega, entre outros. Também assim se torna mais fácil o cálculo de métricas que constituem meios de avaliação de desempenho, não só dos Keepers como ao que o negócio e operação diz respeito.

Não existe nenhuma informação pública sobre o modo como as plataformas tecnológicas de mobilidade sugerem locais aos seus condutores. A Uber por exemplo sugere localizações que na verdade não são as ótimas nos casos em que ocorre o fenómeno *surge pricing* [104], para que estes possam estar distribuídos ao longo de uma área de operação. A LUGGit, no momento em que este documento foi escrito, opera nas duas maiores cidades de Portugal: Lisboa e Porto, e conta com um conjunto reduzido de condutores em comparação com algumas das plataformas já aqui mencionadas, que estão há mais tempo disponíveis no mercado. Todavia, tem tido um crescimento substancial desde a sua criação e entrada no mercado, seja através de clientes ou dos próprios Keepers registados. Isto faz com que a construção de uma plataforma interna de gestão, monitorização e otimização de Keepers seja de facto uma necessidade para a escalabilidade da própria empresa e para a maximização da sua operação. Neste sentido, o desenvolvimento de *software* deverá ir de encontro aos objetivos da empresa, construindo soluções que acompanhem o seu crescimento.

Torna-se assim necessário implementar um sistema que seja capaz de recolher dados enviados por todos os Keepers durante a execução da sua própria aplicação. De seguida, esses dados deverão ser processados para que se possam sugerir localizações tendo por base serviços recentes e históricos através de *heatmaps*, bem como armazenar dados estatísticos sobre os Keepers, usando esse mesmo conhecimento para prever o tempo médio de viagem entre duas localizações. Neste último caso, os dados são também úteis para a gestão, otimização e controlo da própria operação da empresa, sendo interessante fazer com que este sistema não só armazene toda a informação sobre os Keepers, como também forneça métricas interessantes de análise do próprio serviço, ou seja *data warehousing*, neste caso sugerindo-se o método ELT (Extract Load Transform)[104]. Para a transformação dos dados, torna-se também interessante a integração de APIs externas que reúnam importância para o contexto de um serviço da LUGGit, nomeadamente o estado do tempo (uma vez que poderá afetar as métricas, e consequentemente a operação), bem como voos marcados para os aeroportos das respetivas áreas de operação, o que permite efetuar uma análise mais direta entre os serviços relativamente aos horários de partidas/chegadas, sendo os aeroportos locais onde por norma existe maior possibilidade de haver um serviço. Ao mesmo tempo, este sistema terá que ir de encontro aos objetivos da empresa no imediato, o que irá acontecer através do aumento do número de condutores à medida que mais cidades e áreas de operação são adicionadas à sua oferta.

3.3.4.1. Requisitos Funcionais

RF-1: O serviço deve fornecer uma API acessível a partir de um URL.

RF-2: O serviço deve fornecer um *endpoint* na sua API que receba a localização atual dos Keepers, bem como a sua marca temporal.

RF-3: O serviço deve fornecer um *endpoint* na sua API que receba a localização dos Keepers na ocorrência de eventos, bem como a sua marca temporal.

RF-4: O serviço deverá detetar se um Keeper se encontra ou não online, caso este não envie a sua localização após um determinado número de minutos de inatividade.

RF-5: O serviço deverá fornecer um *endpoint* na sua API que receba o estado atual de um serviço de recolha/entrega da LUGGit.

RF-6: O serviço deverá processar o estado de serviços de recolha/entrega, calculando métricas importantes inerentes à operação, como:

- tempo de confirmação do pedido, por parte do cliente;
- tempo de cancelamento do pedido, por parte do cliente;
- tempo de aceitação do pedido, por parte do Keeper;
- tempo de recolha do pedido, por parte do Keeper;
- tempo entre a recolha do pedido até ao centro logístico;
- tempo de entrega do pedido, por parte do Keeper;
- tempo de espera para entrega ao cliente, por parte do Keeper;
- tempo de atraso do pedido na entrega;
- tempo de atraso do Keeper;
- tempo de atraso do cliente.

RF-7: O serviço deverá recolher dados meteorológicos para que estes sejam associados diretamente a um serviço de recolha/entrega, através de uma API externa.

RF-8: O serviço deverá recolher dados relativos aos voos nos aeroportos em áreas de operação, para avaliar a possibilidade de associação de data e hora de partidas e chegadas com novos serviços de recolha/entrega.

RF-9: O serviço deverá gerar heatmaps para as áreas de operação, disponibilizando-os através de um endpoint tendo em conta dados em tempo-real, de acordo com um ou mais eventos de serviço.

RF-10: O serviço deverá gerar heatmaps para as áreas de operação, disponibilizando-os através de um *endpoint* tendo em conta dados históricos, de acordo com um ou mais eventos de serviço.

RF-11: O serviço deverá gerar heatmaps para as áreas de operação, disponibilizando-os através de um *endpoint* combinando dados recentes (tempo-real) bem como dados históricos, de acordo com um ou mais eventos de serviço.

3.3.4.2. Arquitetura Proposta

Vista Geral

Tendo em conta os requisitos apontados para este serviço, apresenta-se na figura seguinte a sua arquitetura. Escolheu-se separar o serviço em dois microserviços, denominados *Keeper Optimizer* e *Keeper Processor*, de modo a usar-se uma arquitetura baseada numa fila de mensagens. Desta forma o primeiro serviço emite e o segundo processa as mensagens, com o Apache Kafka[67] a ser responsável pela gestão, armazenamento e processamento das mesmas, permitindo a recolha e processamento de dados de uma forma assíncrona. Por sua vez, os dados serão armazenados numa base de dados PostgreSQL com a extensão PostGIS, cujo esquema de dados será explicado adiante. Para relacionar e integrar os dados recebidos utilizando o método ELT, é usada a Google Cloud Firestore[68] como fonte de dados complementar, sendo esta a base de dados principal em utilização. Este serviço também será utilizado pelas duas aplicações da LUGGit, apresentando modos de interação distintos. Sendo este serviço responsável por recolher e processar informação útil para os Keepers, estes farão um contacto direto com ambos os serviços, ao contrário do cliente, cuja interação será feita ao emitir eventos através da API da LUGGit. São usadas APIs externas para obter dados como os feriados, meteorologia e aeroportos, que serão relacionados com os dados existentes do serviço, o que permitirá efetuar análises futuras. Também serão usados os serviços de Routing e

[67] https://kafka.apache.org
[68] https://firebase.google.com/docs/firestore

Geofencing, de modo a complementar os dados obtidos e processados. No capítulo 4.5 será discutido o desenvolvimento mais em detalhe.

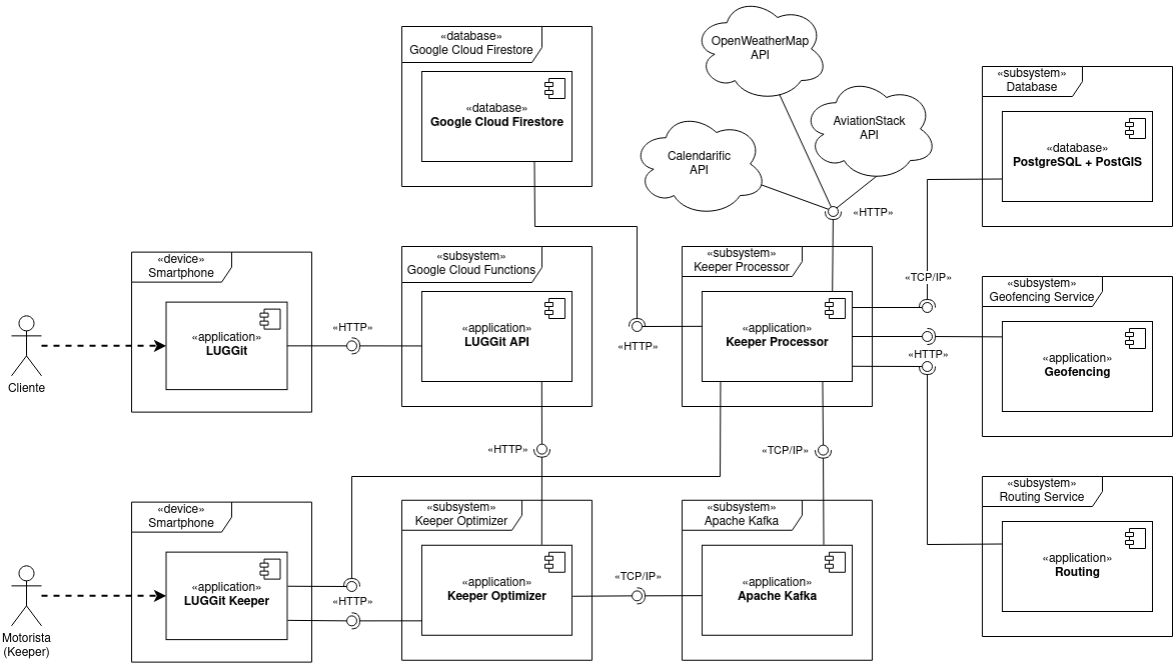

Figura 11 - Arquitetura Geral do Sistema de Correção de Posicionamento de Condutores

Esquema de Dados

Para este serviço, e dado que existe a necessidade de armazenar métricas para fornecer estatísticas relativamente aos serviços efetuados pela LUGGit, efetuou-se a modelação da base de dados usando um esquema de *data warehousing*, denominado de ELT, através da criação de *data marts*. Pretende-se que os dados sejam extraídos da base de dados já em operação por parte da LUGGit, a Google Cloud Firestore, carregando-os e transformando-os para inserção no modelo de dados relacional que aqui se propõe criar. O diagrama da base de dados está disponível no Anexo A.

3.3.5. Previsão da Duração de Viagem entre dois locais

Para conseguirmos obter um Keeper em melhor posição para efetuar um novo serviço (assim que ele é requisitado pelo utilizador), e garantir que este chega junto do cliente em menos de 15 minutos, temos que ter conhecimento constante da sua localização numa área de operação. Contudo, não se sabe o estado atual de uma cidade. O trânsito pode variar de acordo com a altura do dia, do ano, e até mesmo com o estado do tempo, e estes são fatores que influenciam o tempo que um Keeper demora a efetuar um trajeto entre dois locais. Deste modo, discutir e mostrar uma

solução a este problema analisando dados históricos de condutores torna-se de facto útil tendo em vista o cumprimento do objetivo.

No momento em que esta dissertação foi escrita, Portugal e o Mundo atravessavam um estado de confinamento levado a cabo pelo coronavírus, que parou toda a população, com limitações ao nível do transporte e circulação de pessoas sob risco de contágio. Como consequência e sob regras estatais as pessoas ficaram em casa, pelo que a operação da LUGGit parou durante meses, não havendo dados reais por parte dos Keepers para este estudo. Em 2015, a Geolink lançou um concurso[69] cujo objetivo era prever o destino de uma viagem de táxi a partir do local de origem na cidade do Porto, Portugal, fornecendo para o efeito dados obtidos a partir de 442 taxistas entre 2013 e 2014. Estes dados fornecidos pelo concurso foram usados para analisar e posteriormente prever através de um modelo de *machine learning* a duração de uma viagem entre dois locais na cidade do Porto, uma das cidades onde a LUGGit opera.

Os dados obtidos a partir do modelo serão mais tarde comparados com os dados fornecidos pelo serviço de Routing, de modo a podermos aferir potenciais diferenças entre os mesmos, e discutir qual o seu impacto no âmbito desta dissertação, com vista a cumprir o Objetivo 4, conforme referido no capítulo 3.1. Este modelo será apenas alvo de estudo, não sendo integrado em nenhum dos serviços já referidos.

3.4. Arquitetura Geral da Solução Proposta

A infraestrutura da LUGGit está montada em torno de produtos da Google, nomeadamente a Google Cloud Platform[70] (GCP), bem como o Google Firebase[71], que é um *Mobile Backend as a Service* (MBaaS) que fornece ferramentas úteis ao desenvolvimento de aplicações como autenticação, armazenamento, entre outras.

O *deployment* de todos os componentes de *software* da LUGGit é feito na GCP em *clusters* criados para as diferentes fases de desenvolvimento (*Development*, *Staging/QA* e *Production*). Cada um deles executa o Google Kubernetes Engine[72], permitindo instanciar várias máquinas virtuais que

[69] http://www.geolink.pt/ecmlpkdd2015-challenge/index.html
[70] https://console.cloud.google.com/
[71] https://firebase.google.com/
[72] https://cloud.google.com/kubernetes-engine

em conjunto executam o Kubernetes[73], um orquestrador de microserviços, em que cada um dos microserviços da empresa é posto em execução. São também usados produtos do Google Firebase, nomeadamente o Cloud Firestore como a base de dados principal, bem como uma API *serverless* montada a partir das Google Cloud Functions.

Posto isto, bem como o discutido ao longo deste capítulo, apresenta-se na Figura 12 a arquitetura geral da solução proposta por esta dissertação, numa vista geral, integrada na infraestrutura *Cloud* da LUGGit.

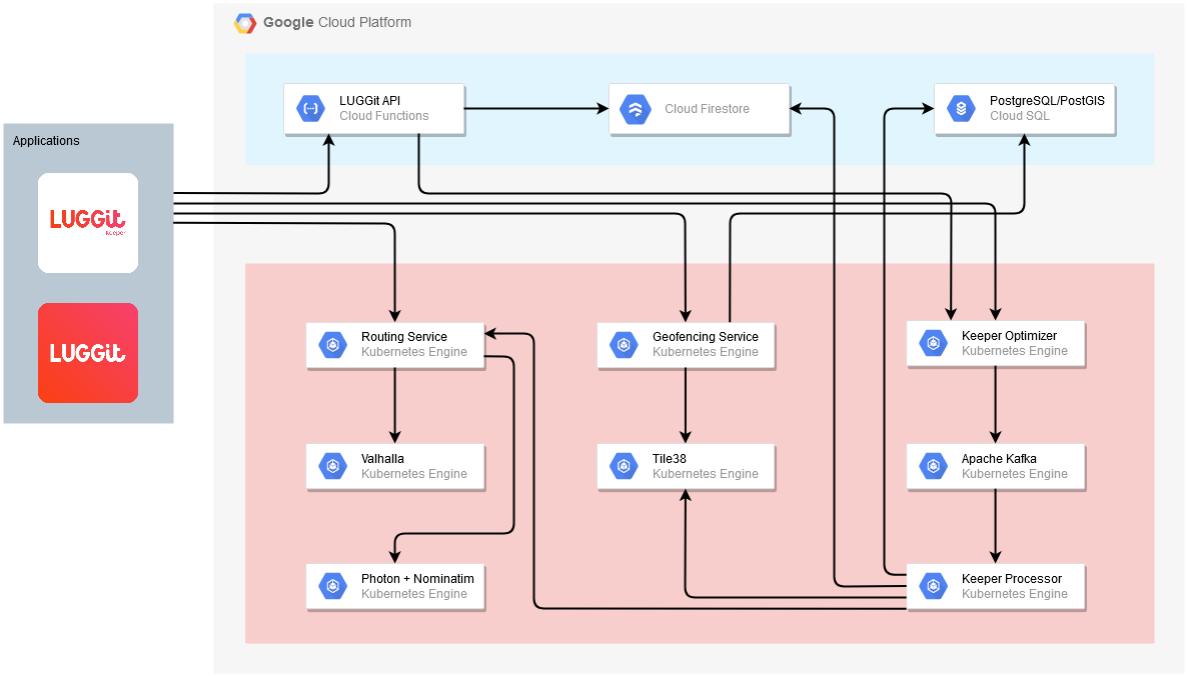

Figura 12 - Arquitetura Geral da Solução Proposta por esta dissertação (a vermelho), numa vista geral, integrada na infraestrutura *Cloud* da LUGGit

[73] https://kubernetes.io/

4. Desenvolvimento da Solução

Neste capítulo vai-se abordar o desenvolvimento da solução proposta para esta dissertação, através da integração das soluções de otimização de rotas, indexação espacial, bem como outras soluções que em conjunto permitem cumprir os requisitos anteriormente identificados e de modo a atingir os objetivos propostos. Aqui o foco será mais na parte técnica, discutindo-se tecnologias, ferramentas que auxiliaram o desenvolvimento, bem como os microserviços e outras funcionalidades que foram criadas.

4.1. Requisitos de Desenvolvimento

4.1.1. Ambiente de Desenvolvimento

Para o desenvolvimento foi escolhido o Visual Studio Code[74] da Microsoft, um editor de código open-source e de uso livre. Permite o uso de diversas linguagens bem como a instalação de extensões criadas tanto pela própria Microsoft como pela comunidade que permitem auxiliar o desenvolvimento.

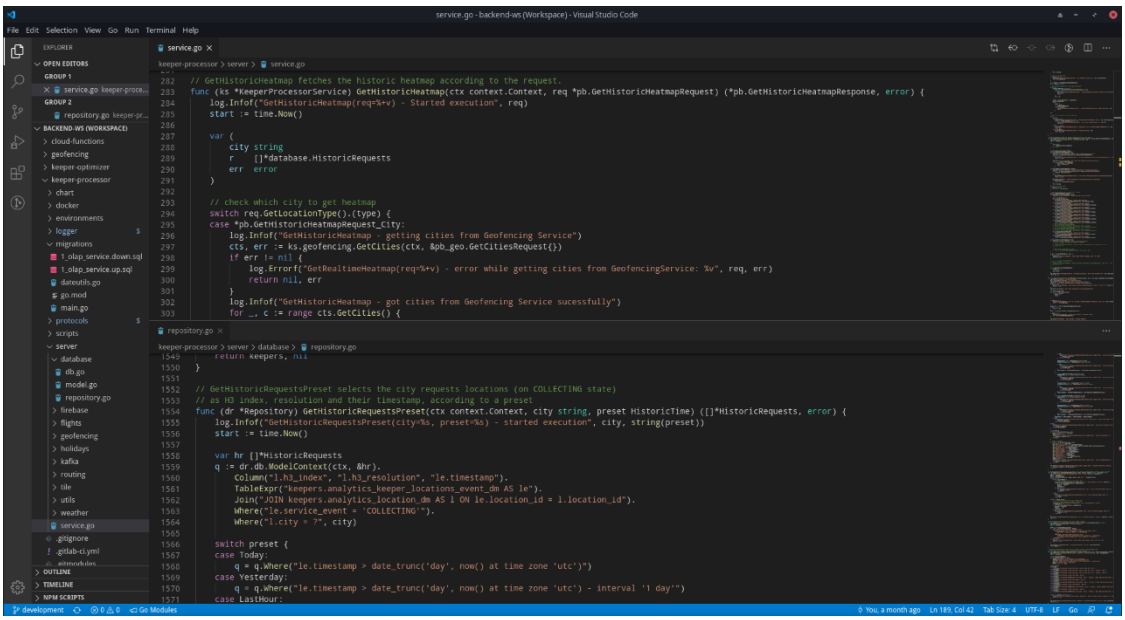

Figura 13 - Ambiente de desenvolvimento com o Visual Studio Code

[74] https://code.visualstudio.com/

4.1.2. Planeamento, Gestão de Projeto e Controlo de Versões

Para a gestão, organização e planeamento das tarefas a realizar durante o desenvolvimento foi utilizado o ClickUp[75], uma plataforma de produtividade, que permite seguir uma abordagem de gestão de projetos usando por exemplo os métodos Agile, Scrum ou Kanban [105]. Na LUGGit segue-se uma abordagem personalizada na gestão de projetos, sendo que um exemplo é mostrado na Figura 14. Para cada projeto de desenvolvimento é criada uma secção indicando qual é a *release* (lançamento) onde as tarefas são colocadas, sendo que visível na barra lateral esquerda a "*Next Release*" como sendo o lançamento atual, a "*Pending Release*" o próximo lançamento após o atual, e "*Backlog*" indicando tarefas que não são prioritárias no imediato. As tarefas são movidas para os separadores à medida que o seu estado de desenvolvimento é atualizado.

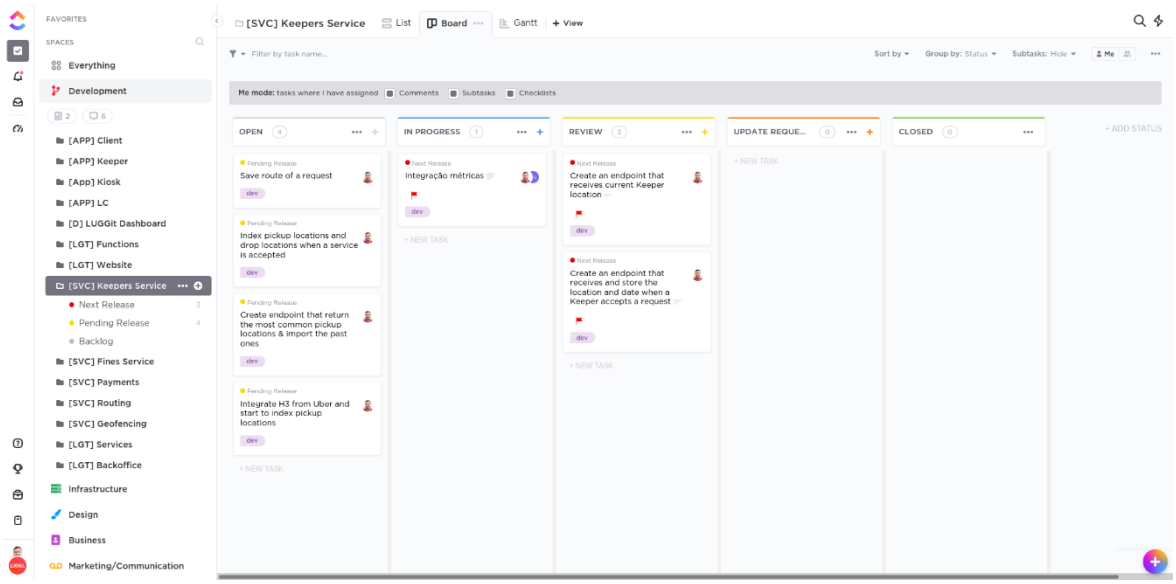

Figura 14 - Gestão de projeto utilizando o ClickUp

Para o armazenamento do código foram utilizados repositórios no GitLab[76] para cada projeto, sendo que o *software* usado para o seu controlo de versões foi o Git[77], sendo que cada *commit* efetuado nos repositórios seguiu a convenção Conventional Commits[78].

[75] https://clickup.com/
[76] https://about.gitlab.com/
[77] https://git-scm.com/
[78] https://www.conventionalcommits.org

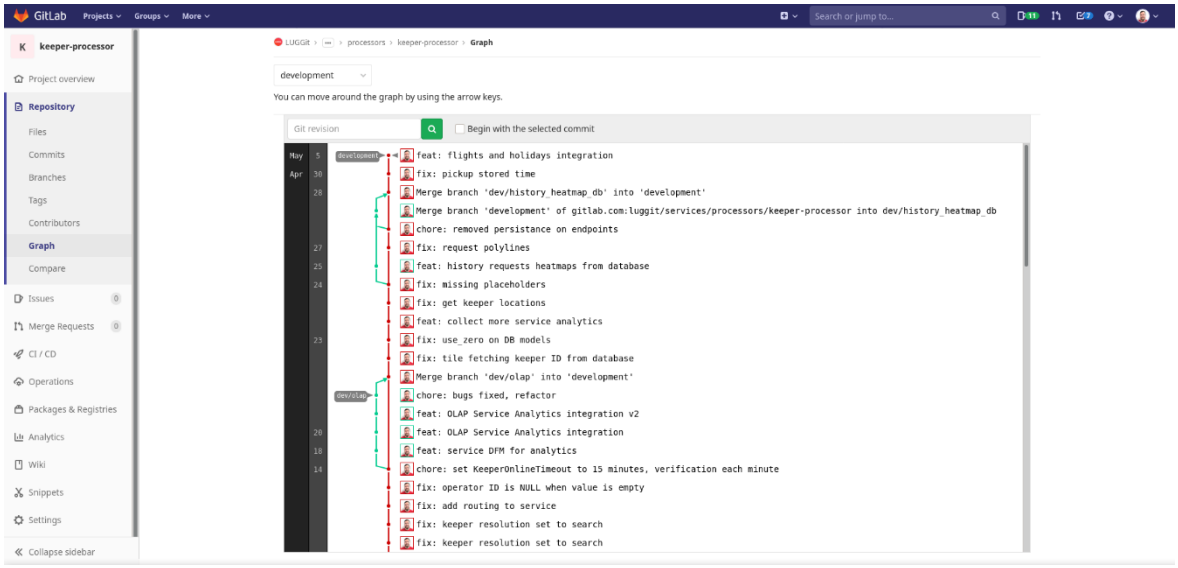

Figura 15 - Armazenamento de código e controlo de versões usando GitLab e Git

4.1.3. Ambiente de Execução, Orquestração e Comunicação entre Serviços

Todo o software desenvolvido na empresa está alojado em ambiente *Cloud*, mais propriamente na Google Cloud Platform. Como tal, houve também a necessidade de adaptar o *software* desenvolvido no âmbito desta dissertação para que este pudesse ser integrado nesse ambiente. Cada um dos serviços e componentes de *software* é compilado, e os seus executáveis são colocados em *containers* Linux [106], utilizando o Docker como ferramenta de virtualização.

Na Figura 16 pode-se ver um exemplo de uma *Dockerfile*, um ficheiro usado pelo Docker para a construção de uma imagem Docker [108]: um conjunto de instruções necessárias para a construção de um *container* Docker, onde a imagem Docker é executada. Cada linha nesse ficheiro corresponde a uma *layer*, sendo que um conjunto de *layers* constroem uma imagem Docker, que mais tarde poderá ser executada num *container*.

No caso desta *Dockerfile* em específico é usada uma funcionalidade do Docker chamada *multi-stage build*, isto é, a construção de uma imagem separada por etapas. A primeira etapa consiste na compilação da aplicação, onde todas as bibliotecas e dependências necessárias para a compilação e construção da aplicação são descarregadas e instaladas. Uma vez terminada esta etapa, não é necessário ter estes componentes instalados pois a aplicação fica disponível através de um executável binário gerado pelo Go.

Para executar a aplicação basta agora executar esse ficheiro executável numa imagem limpa, sendo que se usou a imagem do sistema operativo Alpine Linux[79], que contém o mínimo necessário para a execução da aplicação. É aí que entra a segunda etapa, que consiste na cópia do ficheiro executável a partir da primeira etapa, indicando a execução desse ficheiro como ponto de entrada (*entrypoint*) da imagem Docker criada, que se trata do Serviço de Correção de Posicionamento de Condutores.

```
FROM golang:1.12-alpine as builder
RUN apk add --no-cache ca-certificates git
RUN apk add --update go git build-base
ENV PROJECT luggit.app/src/keeper-processor
ENV GO111MODULE=on
WORKDIR /svc/$PROJECT

# restore dependencies
COPY . .
RUN go mod vendor

COPY . .
RUN go build -gcflags='-N -l' -o /keeper-processor .

FROM alpine as release
RUN apk add --no-cache ca-certificates
COPY --from=builder /keeper-processor /keeper-processor
ENTRYPOINT ["/keeper-processor"]
```

Figura 16 - *Dockerfile* para a imagem Docker do Serviço de Correção de Posicionamento de Condutores

Uma vez que é usada a Google Cloud Platform como ambiente *Cloud* na infraestrutura, é necessário enviar a imagem Docker gerada para um *image registry*, um diretório onde constam imagens Docker necessárias para a execução dos *containers*. No caso da Google Cloud Platform, as imagens são enviadas para o Container Registry[80], onde são assinaladas com uma *tag* (marca) identificativa da versão atual do *software*, neste caso do serviço.

A orquestração de todos os *containers* existentes é feita usando o Kubernetes, mais propriamente o Google Kubernetes Engine. Por forma a garantir um maior controlo na comunicação entre cada um dos serviços, é usado o Istio[81], uma *service mesh framework* que faz com que seja possível controlar o tráfego, aplicar políticas de segurança, bem como monitorizar cada um dos serviços nela existentes através do Kubernetes [108].

[79] https://alpinelinux.org/
[80] https://cloud.google.com/container-registry
[81] https://istio.io/

Como durante o desenvolvimento existiu a necessidade de integrar diversos componentes como bases de dados e outros sistemas utilizou-se o Docker Compose[82] para auxiliar o desenvolvimento em ambiente local. O Docker Compose é uma ferramenta que permite a instanciação de uma *stack* (plataforma) completa em num ambiente local utilizando o Docker como ferramenta de virtualização. Desta forma a instalação, configuração e integração de componentes de *software* externos foi facilitada.

4.1.4. Integração e Implantação Contínua de Software (CI/CD)

De modo a facilitar a integração e implantação de software de uma forma contínua durante as diferentes fases de desenvolvimento, foi usado o GitLab CI/CD[83], uma ferramenta incluída no GitLab que permite automatizar tarefas que vão desde a compilação e *packaging* da aplicação, passando pelos testes de *software* e finalizando na implantação/instalação no ambiente de execução.

Assim que ocorre uma atualização de código num dos ramos do repositório no GitLab, é executada uma *pipeline* que por sua vez executa ordenadamente cada uma das tarefas (*stages*) nela especificadas, verificando e garantindo qualidade ao software de uma forma ágil. Caso uma das tarefas não seja concluída, a *pipeline* falha, o que significa que o código mais recente ainda não está pronto para ser implantado e ser colocado em funcionamento em ambiente de (pré)produção, pelo que deverá ser corrigido, o que depois levará a uma nova *pipeline*.

[82] https://docs.docker.com/compose/
[83] https://docs.gitlab.com/ee/ci/README.html

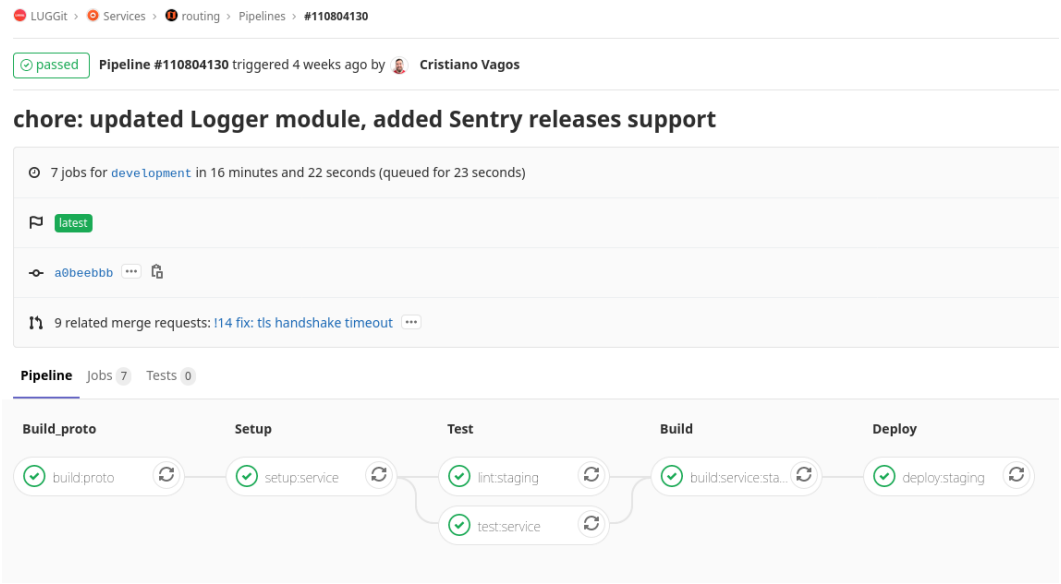

Figura 17 - Exemplo de uma *pipeline* usando o GitLab CI/CD

Logo após executar as diversas fases da *pipeline*, a aplicação é colocada numa imagem Docker, cuja execução será feita num *container*, conforme discutimos anteriormente. Neste caso, uma vez que as *pipelines* do GitLab CI/CD são executadas a partir um *container*, a construção da imagem terá de ser feita utilizando o Kaniko[84], uma ferramenta que permite a construção de imagens Docker a partir de *containers*, de uma *Dockerfile*, ou a partir de um *cluster* Kubernetes.

A implantação e posterior instalação e configuração do serviço (*deployment*) no Google Kubernetes Engine fica a cargo do Helm[85], que com base num *chart*, um conjunto de ficheiros YAML que descrevem recursos a serem usados pelo Kubernetes, permite a criação de uma *stack* completa, desde as aplicações em si até às bases de dados, gestão de rede e políticas de segurança a implementar para um funcionamento correto e seguro do serviço. Neste caso, para cada serviço há que especificar variáveis de ambiente, as portas a expor por cada um dos *containers*, volumes de armazenamento, entre outros aspetos que sejam necessários para a execução do serviço.

[84] https://github.com/GoogleContainerTools/kaniko
[85] https://helm.sh/

4.1.5. Monitorização e Reporte de Erros

Algo inevitável no desenvolvimento de *software* é o facto de que eventualmente, haverá falhas e erros durante a sua execução. Assim, foi usado o Sentry[86] como plataforma de *error reporting*, reportando em tempo-real erros através de notificações por email e pelo Slack[87], uma plataforma usada para comunicação da equipa. Todos os serviços desenvolvidos têm o SDK do Sentry integrado para que todos os erros de *software* sejam monitorizados.

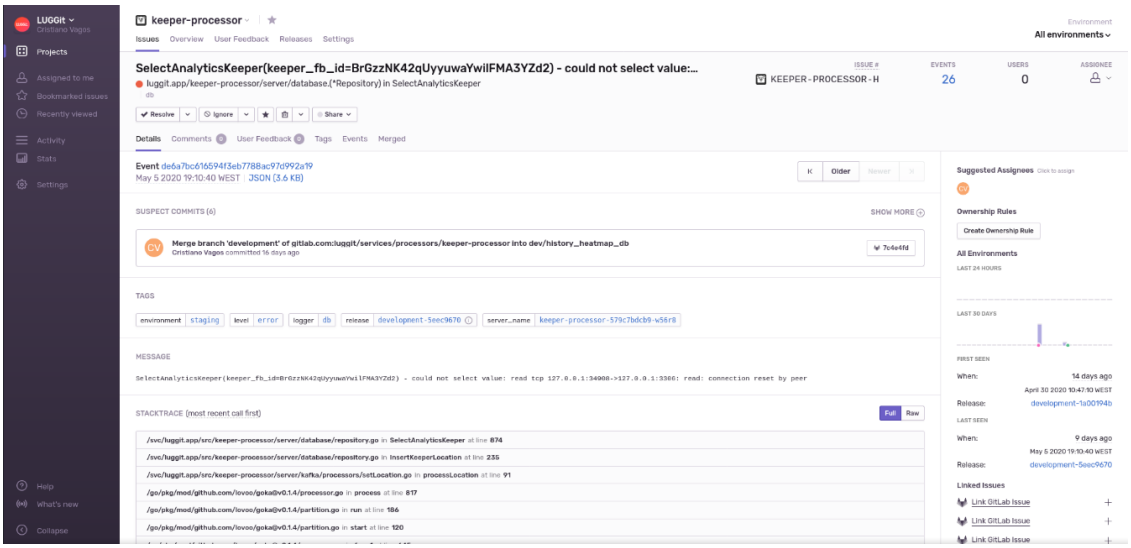

Figura 18 - Relatório de erro reportado pelo Sentry

4.2. Tecnologias Escolhidas

Nesta secção irão ser apresentadas e discutidas as tecnologias escolhidas para o desenvolvimento dos serviços apresentados, sendo que serão abordadas tecnologias comuns a alguns serviços.

A nível pessoal, senti-me motivado para escolher tecnologias que pudessem ser úteis para a resolução do problema desta dissertação. Apesar de algumas das tecnologias escolhidas já terem sido abordadas durante o meu percurso académico, escolhi algumas tecnologias modernas e com uma adoção crescente na comunidade, o que constituiu um desafio extra. A própria empresa motivou-me à pesquisa e integração dessas tecnologias.

[86] https://sentry.io/
[87] https://slack.com/

Optou-se por desenvolver os serviços utilizando a linguagem de programação Go, pelo facto desta ser uma linguagem adotada em ambientes que exijam performance, como é o caso dos serviços que se desenvolveram, e também por já existir uma *codebase* construída com esta linguagem, o que facilitou a aprendizagem. Utilizou-se o gRPC[88] como *framework* de comunicação pelo facto de utilizar sobretudo métodos RPC, o que tem vantagens ao nível da performance permitindo fornecer uma API de baixa latência e alta taxa de transferência, com vista ao cumprimento dos requisitos não-funcionais, discutidos no capítulo 3.3.1.

Outras tecnologias como o PostGIS e Tile38 foram escolhidas devido ao seu suporte geoespacial, sendo o uso combinado de bases de dados relacionais e não-relacionais beneficiado também ao nível da performance e escalabilidade. Também foi utilizado o Apache Kafka para permitir a emissão e respetivo processamento a eventos que ocorram durante a realização de um serviço da LUGGit, através de fluxos de mensagens.

O método de indexação espacial utilizado foi o H3, por fornecer características que permitem fornecer as funcionalidades pretendidas no desenvolvimento, e estar disponível para a linguagem de programação Go.

Para a análise dos dados que nos permitam obter um modelo de *machine learning* onde seja possível prever a duração da viagem entre dois locais, foi utilizada a linguagem de programação Python, em conjunto com ferramentas de aprendizagem automática como o scikit-learn[89], manipulação de *data frames* com o pandas[90]. O modelo foi criado e treinado através da ferramenta gratuita Google Colab[91], que permite a execução e visualização de um *notebook* a partir de um servidor externo.

[88] https://grpc.io/
[89] https://scikit-learn.org/
[90] https://pandas.pydata.org/
[91] https://colab.research.google.com/

4.2.1. Go

Uma das características de uma arquitetura baseada em microserviços é o facto de se poder desenvolver em qualquer linguagem, sendo que o protocolo de comunicação é o fator mais importante, sendo um elo de ligação.

Todos os serviços já existentes na infraestrutura que já se encontrava implementada previamente a esta dissertação foram desenvolvidos utilizando como linguagem de programação principal o Go (também conhecida por Golang), uma linguagem de programação criada na Google em 2012 por Robert Griesemer, Rob Pike, e Ken Thompson [109], que apresenta como pontos a favor o facto desta ser uma linguagem minimalista, com ferramentas internas para testes e *benchmark*, deteção de condições de corrida, deteção de pacotes não-usados, e é também considerada uma linguagem com grande performance devido ao facto desta ser uma linguagem compilada - um programa em Go é um executável binário, sem ser necessário nenhum *middleware* específico para o executar [110][111]. Outra funcionalidade interessante introduzida pelo Go são as *goroutines* [112], fios de execução que possibilitam a execução de funções de uma forma assíncrona, sendo que o meio de comunicação entre elas poderá ser feito através de canais de comunicação, conhecidos por *channels* [113].

De forma a manter-se na empresa uma *codebase* uniforme ao nível do *backend*, e tendo em conta as vantagens mencionadas anteriormente, resolveu-se assim adotar esta linguagem de programação para o desenvolvimento de todos os serviços.

4.2.2. gRPC e Protocol Buffers

O gRPC é uma *framework* de RPC *open-source* criada pela Google em 2015, criada inicialmente para interligar microserviços através de uma infraestrutura baseada em chamadas de procedimentos remotos. Hoje, para além dos objetivos iniciais para a qual esta foi criada, trata-se de uma *framework* que suporta múltiplas plataformas, e fornece funcionalidades que garantem eficiência, segurança e fiabilidade [114]. Através desta plataforma, um cliente poderá invocar um método diretamente num servidor localizado numa máquina diferente tal e qual como se tratasse de um objeto local, daí o seu uso ser adequado em microserviços. Assim, no lado do servidor é implementada uma interface, ao mesmo tempo que recebe e processa novos pedidos através de um servidor gRPC, ao que no lado do cliente é instanciado um *stub* que fornece os mesmos métodos disponíveis no servidor,

permitindo desta forma o uso de chamadas de procedimentos remotos. O gRPC utiliza a versão 2 do protocolo HTTP (HTTP/2), bem como suporta todas as combinações de pedidos existentes: unitários (*Unary*, pedido-resposta), *Streaming* unidirecional e *Streaming* bidirecional, permitindo assim fornecer comunicação em tempo-real.

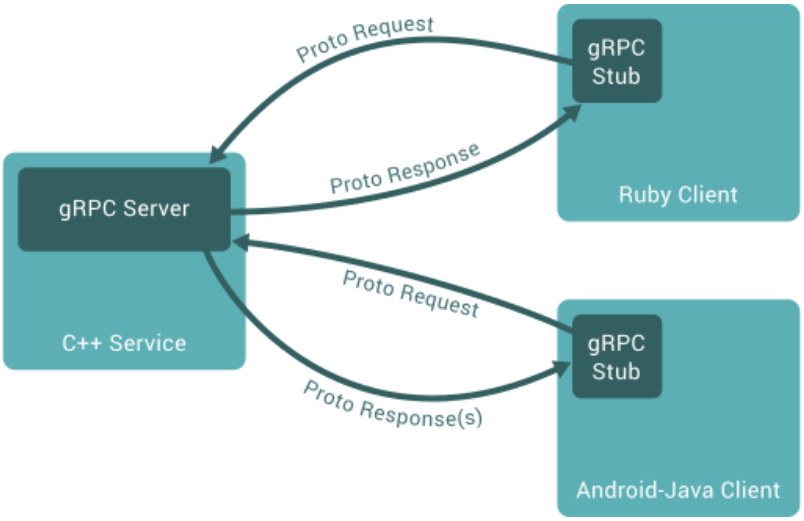

Figura 19 - Comunicação cliente-servidor utilizando gRPC [115]

Por omissão, o gRPC usa Protocol Buffers[92] (também conhecido por Proto/Protobuf) como mecanismo de serialização[93] de dados estruturados, uma plataforma universal disponível para várias linguagens e de alta performance. Os Protocol Buffers foram criados pela Google em 2001 [116] como uma forma eficiente de serialização sendo que a versão atual da linguagem (proto3)[117], usada para o desenvolvimento dos serviços apresentados nesta dissertação, permite a geração de código para linguagens como o Java, C++, Python, Ruby, JavaScript, Objective-C, C# e Go a partir de um único ficheiro, o que é feito através da ferramenta *protoc*, criada especificamente para compilar os ficheiros "*.proto*", que contêm a especificação das *Messages* (mensagens, conjunto de dados estruturado) e dos *Services* (serviço a ser criado, com o conjunto de métodos a ser chamado). Após a compilação destes ficheiros, pode-se escrever e ler dados em qualquer uma das linguagens de programação descritas com este formato, que em conjunto com o gRPC apresenta vantagens comparativamente ao uso de APIs HTTP usando JSON, ao custo dos pedidos gRPC não serem legíveis pelos humanos, pela forma como as mensagens são codificadas [118]. Um exemplo de um ficheiro

[92] https://developers.google.com/protocol-buffers
[93] Conversão de uma estrutura de dados em *bytes*.

".proto" pode ser visto na Figura 20, onde um serviço com um método RPC é definido, especificando a mensagem a receber e a enviar na resposta.

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Figura 20 - Exemplo de um ficheiro ".proto" definindo um serviço gRPC [115]

Contudo, apesar de apresentar vantagens a nível de performance e na comunicação entre microserviços, o gRPC não é suportado por exemplo, por navegadores Web, que tradicionalmente utilizam o protocolo HTTP (versão 1) assim como REST APIs para a comunicação. Então, para resolver esse problema foi criado o grpc-gateway [119], um *reverse proxy* que "traduz uma REST API em HTTP para gRPC", gerando um servidor que interpreta pedidos em HTTP e os converte em pedidos para o serviço gRPC. Esta ferramenta pode ser incorporada facilmente num serviço gRPC através da colocação de anotações na declaração do método no ficheiro "*.proto*", procedendo-se à geração do código novamente através da ferramenta *protoc*, especificando os parâmetros adequados para compilação do *gateway*. Desta forma torna-se possível a um microserviço fornecer uma REST API compatível com gRPC. Esta ferramenta também inclui um gerador de documentação para o Swagger UI[94], onde podemos testar e visualizar os métodos disponíveis na API gerada.

[94] https://swagger.io/tools/swagger-ui/

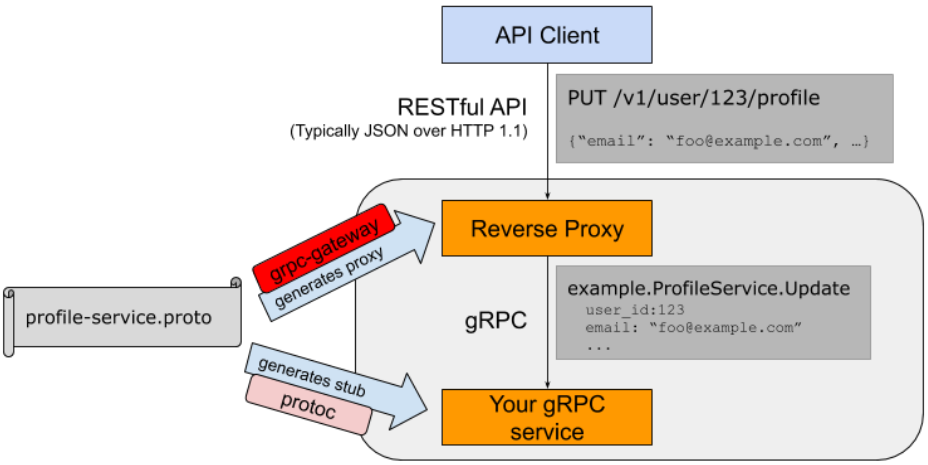

Figura 21 - Funcionamento da ferramenta grpc-gateway [119]

Para facilitar a geração dos ficheiros Protocol Buffers que definem o protocolo de comunicação nos serviços apresentados nesta dissertação, foi usada a imagem Docker *namely/protoc-all*[95], que contém todas as ferramentas necessárias para compilar e gerar o código que permite a implementação de um serviço gRPC, incluindo o grpc-gateway e extras como validação automática de mensagens e campos utilizando a ferramenta protoc-gen-validate[96], tendo-se contribuindo para a mesma ao integrar esta funcionalidade na imagem Docker [119]. A utilização de uma ferramenta deste género através de uma imagem Docker faz com que seja facilitada a compilação e geração de Protocol Buffers para uso nos serviços feitos no âmbito desta dissertação, ao ser integrada na *pipeline* CI/CD [120].

```
message DirectionsRequest {
    luggit.common.geo.Point origin              = 1 [(validate.rules).message.required = true];
    luggit.common.geo.Point destination         = 2 [(validate.rules).message.required = true];
    repeated luggit.common.geo.Point waypoints  = 3;
    repeated AvoidType avoid                    = 4 [(validate.rules).repeated.items.enum.defined_only = true];
    MeasureUnit unit                            = 5 [(validate.rules).enum.defined_only = true];
}
```

Figura 22 - Exemplo de validação automática num ficheiro ".proto" usando a ferramenta protoc-gen-validate

Assim, combinando o gRPC com a possibilidade de fornecer uma REST API através da mesma especificação dos ficheiros Protocol Buffers de cada serviço consegue-se oferecer uma API uniforme, fazendo com que tanto sistemas modernos como sistemas *legacy*, que não suportem ainda esta tecnologia consigam comunicar com os serviços criados.

[95] https://github.com/namely/docker-protoc
[96] https://github.com/envoyproxy/protoc-gen-validate

4.2.3. PostGIS

O PostGIS é uma extensão *open-source* criada para a base de dados relacional PostgreSQL, que adiciona suporte, bem como funções espaciais tais como a distância, área, união, intersecção, entre outros a objetos geográficos. Houve necessidade de utilizar esta extensão por esta fornecer suporte à manipulação e armazenamento de pontos e polígonos numa base de dados relacional.

A integração desta extensão faz com que seja possível lidar com dados espaciais, que no âmbito desta dissertação são importantes, nomeadamente o armazenamento de localizações, polígonos e o uso de funções espaciais tais como por exemplo a codificação de linhas utilizando o algoritmo *encoded polyline* para as representar mais facilmente. O sistema de coordenadas usado foi o EPSG:4326[97], ao que corresponde o sistema de referência espacial WGS 84, o mesmo utilizado no GPS (latitude e longitude).

A base de dados PostgreSQL foi instalada na infraestrutura Cloud a partir do serviço de base de dados Cloud SQL[98], disponível na Google Cloud Platform. Existe uma opção no momento da instalação correspondente ao PostGIS que está disponível para a instalação.

4.2.4. Tile38

O Tile38 é uma base de dados *open-source* baseada no Redis[99], sendo portanto uma base de dados não-relacional *in-memory*[100], contudo apresentando características geoespaciais como *geofencing* em tempo-real, pesquisa espacial (*Point-in-Polygon*, procura de objetos próximos, entre outros), suporte a formatos como o GeoJSON[101] e pontos (latitude/longitude), bem como fornece notificações de eventos suportando diversos protocolos de comunicação como o gRPC, Apache Kafka, HTTP, entre outros. Nesta base de dados foram inseridas as áreas de operação disponíveis na LUGGit, bem como os dados geoespaciais dos aeroportos e seus terminais através de polígonos em objetos GeoJSON, bem como alguns pontos de interesse para a determinação de locais próximos de uma dada localização.

[97] https://spatialreference.org/ref/epsg/wgs-84/
[98] https://cloud.google.com/sql
[99] https://redis.io/
[100] Base de dados que utiliza principalmente a memória principal como método de armazenamento.
[101] https://geojson.org/

Apesar de ser uma base de dados *in-memory*, onde os dados são guardados e mantidos em memória, obtendo maior performance comparativamente a escritas em disco, o Tile38 também tem persistência em disco à semelhança do Redis, e permite não só o uso de uma simples instância como também replicação de dados ao usar um sistema *leader-follower* [121] através de várias instâncias. Contudo, a grande vantagem do Tile38 passa pelo suporte a *geofences* estáticas [122] bem como dinâmicas (*roaming geofences*)[123], bem como algoritmos de pesquisa espacial [124]. Esta base de dados também possibilita a notificação de eventos através de *webhooks*, onde a ocorrência de um evento é enviada através de um pedido síncrono ou através de *channels* pelo uso do padrão *Publish-Subscribe* [125], um método de envio de mensagens assíncrono.

4.2.5. Apache Kafka

O Apache Kafka é uma plataforma *open-source* de *streaming* distribuído, oferecendo como funcionalidades o armazenamento e processamento de *streams* (fluxos) de mensagens em tempo-real, utilizando o padrão *Publish-Subscribe*, sendo um *message broker*. Apresenta como grandes vantagens a sua performance, escalabilidade, durabilidade e tolerância a falhas [126].

A Figura 23 mostra as APIs disponíveis para utilização e o seu contacto com um *cluster* Kafka: os *Producers* são produtores/remetentes de fluxos de mensagens; os *Consumers* os seus consumidores, ficando à escuta de novas mensagens provenientes dos *Producers*; os *Stream Processors* são responsáveis por processar os fluxos; e os *Connectors* são pontos de ligação para bases de dados externas, permitindo a extração ou armazenamento de dados.

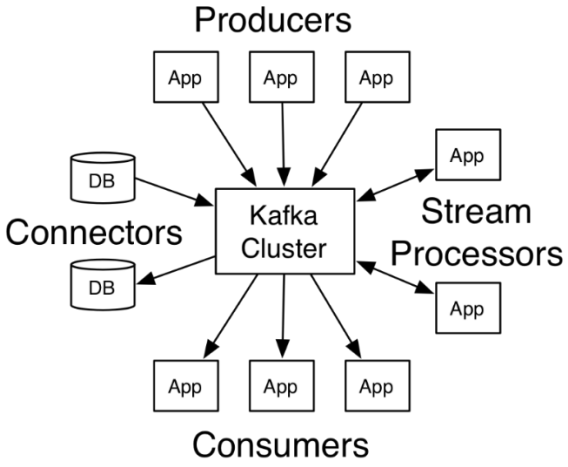

Figura 23 - Interação entre um *cluster* Kafka através de APIs [127]

No Kafka, um ou mais fluxos de mensagens são enviados pelos *Producers* para um *cluster*, que os mantém e os envia para os *Consumers*. Tal é feito através de um *topic* (tópico), para o qual os *Producers* enviam a mensagem, e onde um ou mais *Consumers* a recebem, dado que um *topic* permite a subscrição de um ou mais *Consumers*. Cada mensagem enviada consiste num par chave-valor, onde a chave é usada para definir a partição do *topic*, que pode estar distribuída por várias máquinas. O Kafka garante persistência e replicação de dados ao escrever cada uma das mensagens no disco assim que estas são recebidas.

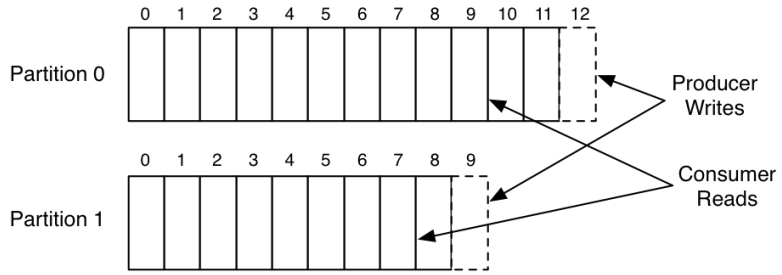

Figura 24 - Exemplo da interação de Kafka *Producers* e *Consumers* num *topic* [126]

O uso de um *message broker* como o Kafka torna-se importante no contexto desta dissertação, permitindo utilizar o conceito de troca de mensagens, o que oferece a possibilidade de se poder reagir à ocorrência de eventos durante um serviço de uma forma assíncrona, sendo também vantajoso ao nível da concorrência, permitindo lidar com grandes quantidades de mensagens [126]. A emissão e posterior processamento de mensagens neste sentido será importante, como poderemos ver adiante no Serviço de Correção de Posicionamento de Condutores, que faz uso deste conceito.

4.2.6. Python

O Python é uma linguagem de programação de alto-nível criada por Guido Van Rossum em 1991, considerada uma linguagem de uso generalizado. O Python é uma das linguagens de *scripting* mais utilizadas tanto para análise de dados como para aprendizagem automática, dada a existência de imensas ferramentas, e bibliotecas para essas áreas. Consequentemente, existe também muita documentação e conhecimento partilhado pela comunidade, tornando-se assim mais fácil a criação

e implementação de um modelo de *machine learning* utilizando esta linguagem, como o que veremos adiante nesta dissertação.

4.2.7. pandas

O pandas é uma biblioteca *open-source* criada para a linguagem Python em 2008 pela AQR[102], sendo utilizada para análise e manipulação de dados. Permite ler e escrever de e para diversos formatos como CSV e SQL, suporta a manipulação rápida e eficiente de *data frames* e permite agrupar, juntar e consultar enormes conjuntos de dados, entre outras funcionalidades [128]. Esta biblioteca foi utilizada para auxiliar no desenvolvimento do modelo de *machine learning*, ao carregar os dados obtidos, pré-processar e manipular esses mesmos dados de uma forma mais facilitada através de *data frames*.

4.2.8. scikit-learn

O scikit-learn é uma biblioteca *open-source* que fornece métodos e ferramentas de *machine learning* para Python. Foi criada em 2007 por David Cournapeau e desde então conta com diversas contribuições por parte da comunidade, reunindo cada vez mais funcionalidades, que permitem analisar e criar modelos de previsão tendo a partir de um conjunto de dados, fornecendo métodos de classificação, regressão, *clustering*, redução dimensional, seleção de modelos bem como ferramentas de pré-processamento. Escolheu-se esta ferramenta pelo facto de esta possuir todas as ferramentas necessárias para o desenvolvimento do modelo de *machine learning*, desde os métodos de treino até à validação do mesmo.

[102] https://www.aqr.com/

4.3. Serviço de Routing

O serviço de Routing tem como objetivo fornecer funcionalidades que permitam determinar trajetos ótimos entre duas localizações, bem como a resolução de locais a partir de texto, que apesar de não ser uma funcionalidade requerida no âmbito desta dissertação foi incluída, o que será útil num contexto futuro de análise de dados. Além deste serviço ser utilizado em ambas as aplicações da empresa, também fornece informações que por exemplo o cálculo de preço de um serviço, entre outras métricas usada em vários serviços da infraestrutura da LUGGit. Após a análise de soluções de Routing que permitissem obter os trajetos, métricas e distâncias entre duas localizações, a escolha recaiu sobre o Valhalla.

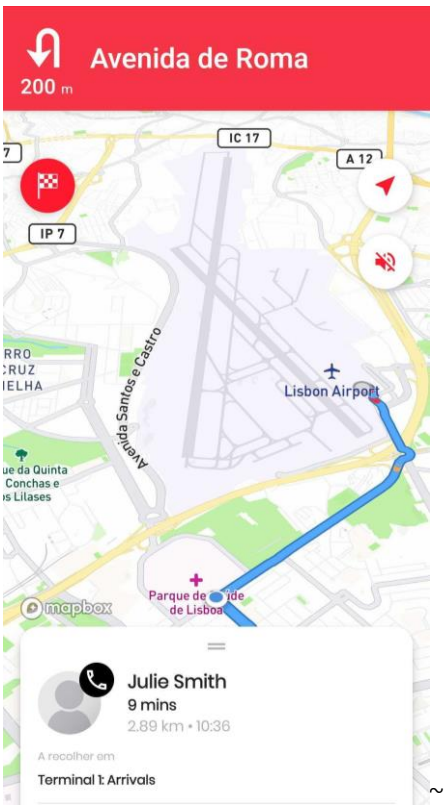

Figura 25 - Uso do *endpoint* Directions na aplicação dos Keepers da LUGGit

Relativamente ao *geocoding*, sendo a pesquisa de locais para recolha e entrega uma funcionalidade usada na aplicação do utilizador, torna-se necessário fornecer neste serviço *Forward Geocoding* e *Reverse Geocoding*, bem como *Autocomplete*. A solução escolhida recaiu sobre a integração de duas soluções: Nominatim e Photon.

Figura 26 - Uso do *endpoint* Directions na aplicação dos utilizadores da LUGGit

4.3.1. Tecnologias Usadas

O serviço de Routing foi desenvolvido utilizando a linguagem de programação Go, usando o gRPC como *framework* de comunicação para a sua API, obtendo dados a partir das REST APIs das ferramentas de Routing e de *geocoding* escolhidas.

A solução de Routing escolhida para obter trajetos entre duas localizações foi o Valhalla, uma solução que oferece exatamente o que é pretendido, com uma vasta REST API disponível e que comparando com as restantes soluções consome menos recursos, o que é benéfico para os custos operacionais da empresa. Para além de fornecer direções entre dois locais, também fornece *isochrones*, o que será útil para poder calcular a distância máxima possível dentro de um determinado período de tempo. Apesar de ser *open-source*, o Valhalla está disponível na Mapbox como uma ferramenta paga [129], isto porque a Mapbox fornece funcionalidades extra que não estão disponíveis na versão *open-source*, bem como retira a responsabilidade operacional de instalação e manutenção desta ferramenta. O Valhalla foi assim instalado na infraestrutura da LUGGit utilizando *containers* Docker e um *chart* Helm para instalação e *upgrade* no Kubernetes. O

Valhalla usa os dados do OpenStreetMap, que estão disponíveis no website da Geofabrik[103]. Estes dados devem ser transformados em *tiles* usando uma ferramenta disponibilizada no próprio Valhalla. A configuração do Valhalla é feita a partir de um ficheiro JSON, onde se definem as pastas com as *tiles*, entre outros. A atualização dos dados é feita através de um *cron job* executado no Kubernetes [130], que executa este processo diariamente, pois os dados disponibilizados no website da Geofabrik também são atualizados diariamente.

De forma a completar as funcionalidades deste serviço, no sentido de substituir completamente o uso da Google Maps Platform, foram acrescentadas funcionalidades de *geocoding* através de duas ferramentas, o Photon e o Nominatim, que tal como o Valhalla foram instaladas e configuradas internamente na empresa utilizando um Helm *chart*. O Nominatim é o *geocoder* utilizado pelo OpenStreetMap, e utiliza uma base de dados em PostgreSQL para armazenamento de dados bem como fornece uma REST API onde é possível obter *Forward* e *Reverse Geocoding*. Tal como o Valhalla, o Nominatim utiliza dados do OpenStreetMap. Estes dados são depois inseridos na base de dados, para posteriormente serem obtidos e disponibilizados através da REST API do Nominatim. O Photon é também um *geocoder*, fornecendo as mesmas funcionalidades de *geocoding* que o Nominatim, porém apresenta vantagens ao utilizar o Elasticsearch como armazenamento de dados, que como já discutimos tem maior performance comparando com o PostgreSQL. O Photon funciona através de uma aplicação Java, que importa os dados a partir da base de dados do Nominatim para o Elasticsearch, fornecendo também uma REST API com as mesmas funcionalidades de *geocoding*. Optou-se por manter as duas ferramentas em funcionamento, uma vez que o Nominatim apresenta características extra nas funcionalidades da sua REST API que o Photon não oferece, como por exemplo obter *Reverse Geocoding* a uma localização com diferentes níveis de *zoom*, funcionalidade importante para encontrar a estrada/rua mais próxima de uma dada coordenada.

Durante a execução do serviço são feitos pedidos de teste constantes aos serviços externos para assegurar o seu funcionamento, pois o serviço de Routing depende diretamente dos mesmos, fornecendo APIs que vão ser usadas nas aplicações da LUGGit.

[103] https://download.geofabrik.de/europe/portugal.html

4.3.2. Funcionalidades

O serviço de Routing limita-se a disponibilizar funcionalidades semelhantes às fornecidas pela Google Maps Platform, nomeadamente direções entre dois locais (ao que se denominou *Directions*), auxiliar e completar pesquisas de locais através de texto (ao que se denominou *Autocomplete*), obter um POI tendo por base texto (ao que se denominou *Geocoding*), obter um local/ponto de interesse através de coordenadas (ao que se denominou *ReverseGeocoding*) bem como obter regiões que definem a área máxima a atingir num determinado período de tempo (*Isochrones*). Tanto os pedidos como as respostas estão no formato JSON.

Figura 27 - *Endpoints* disponíveis no serviço de Routing

Todos os endpoints são versionados de modo a suportar retrocompatibilidade no desenvolvimento de futuras versões, e são feitos com o método POST [131].

4.3.2.1. Directions

O *endpoint* Directions tem como objetivo obter direções entre dois locais, isto é, entre duas coordenadas (latitude e longitude). Apresenta como campos obrigatórios as localizações de origem e destino, e contém campos opcionais que permitem efetuar filtragem e limitação no trajeto obtido.

Figura 28 - Exemplo de pedido ao *endpoint* Directions

Os campos *origin* e *destination* referem-se a localizações, representando os locais de origem e destino respetivamente, o campo *waypoints* é um *array* (conjunto) de localizações representando locais de paragem entre os locais de origem e destino, o campo *avoid* é um *array* representando tipos de trajetos a evitar (ferries - FERRY, auto-estradas - MOTORWAY e portagens - TOLL), e o campo *unit* a indicação da unidade de medida a ser utilizada (métrica - METRIC ou imperial - IMPERIAL).

Caso o corpo do pedido seja válido, é feito um pedido ao Valhalla através da sua REST API, e caso a resposta não tenha erros, transforma os dados e envia a resposta, onde constam os campos *duration* (duração da viagem em segundos), o objeto *distance* que contém os campos *unit* com a unidade de medida, e o campo *value* com a distância (milhas, se unidade imperial; quilómetros, se unidade métrica), o campo *route* com a *encoded polyline* do trajeto a efetuar, e finalmente o campo *provider* que indica que o pedido veio do Valhalla.

```
{
  "duration": 764,
  "distance": {
    "unit": "METRIC",
    "value": 10.623
  },
  "route":
"gcdnlAju|oOdEvRtOf]~CjFde@tf@lExEnSnTpQxRfSlTlUbWpLzJpCbClErB|ElBxFxAzFhAtE\\xGJ`HYpHeAjF}@xG{DtYyO
t|BgtApBUjLkGzFYtDBpCd@hBl@bA^jAh@|@d@tEpEnD`HfCdJdAzEGxQl@`RrA`_@vRj|BpHlx@vSbt@bPrTvNxMhRnIxQ~EbuA
tDpLhBzFhBpBdA|@fBd@dKaC|c@GfSd@pMF`Cl@jIrA|Q`RfhBbFzs@t@xMUbEOlCcA`Bm@zBO`CFdCd@~B|@vM{AzR{AhLuExWe
@rC{Pzz@yLtl@sGvZ}Itb@iH|V_DbHaCnDs@lAkBBaBj@sAtA}@tB]fCqCpEaBbBcBlA{ZjK{Fr@_NfAmPLcQ_@oNu@oNy@wl@aH
mrAsOka@wEaHHcPhBgh@nJoIj@yLMeKQum@cA}d@u@c[}@eEeAaCu@gDWaCiAyBEyBPqBl@yChByAlCwDxCkF`EmUbFaB\\kB`@i
B`@yBf@iCj@gDp@gDt@}Dz@uE`A{EbAuFlAg]pH_~C|p@sFlAeFdAmEbA}Dx@wDv@wCr@iCj@yBd@iB\\cB^iB`@yGvAsGBsFy@y
Ba@{AoAeEqEyByCyGuLeEgIkAiBsAaB{AgB{AcByBgBoCuB_DoBgDsBeFcB{EsA{Fs@}Ea@cFAcGd@wDd@mD`@uE`Ag`Cji@wSvG
}XhMqWzNs`@zZk[hX_c@t]w]z[qCbCgDlFqBrCkF|GkA~A}@Z{@l@m@x@eElBaCNqCEyBc@gNeG}E_C}DmCoDuDmDcGaCyGwDsO{
AgE}DkZw]
{kCmUqfBwR}iAmKso@c|BagQaImi@gMgs@wIca@ePoi@gNe_@iRuc@cVid@yVc`@{Vm^axCkaEuTsa@kFuPkBeSE}Hr@qh@wCkWs
FeTaH}NsUma@aDaMFqNNwG`BoG^mHm@iHkBeGgDmEyAaGeAiDs@sBWiBUsA?gB?
cAFo@Lw@dFcOnMuK`SaQxBcCtDgGlAwDhBqBjAwGN_C\\gFOgFuD}KcGoOkFuP]cD}@_HsAwX]uFmE_g@TuPl@}LFsD\\yBG_C]w
B{@cBkA_AsBi@qBViBrAeAlB]hC?xB\\hBl@|AjAlArAf@rALrAWdArCLxA",
  "provider": "VALHALLA"
}
```


Figura 29 - Exemplo de resposta ao *endpoint* Directions

Em suma, a interação deste *endpoint* apresenta-se a seguir:

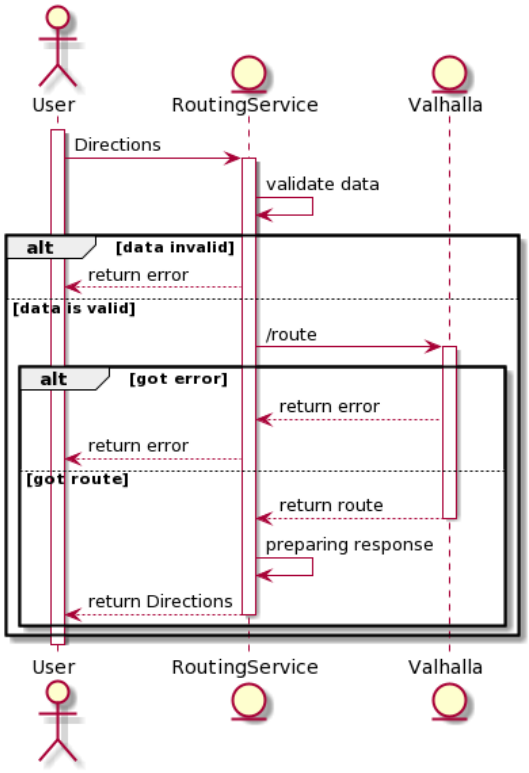

Figura 30 - Diagrama de interação do *endpoint* Directions

4.3.2.2. Autocomplete

O *endpoint* Autocomplete tem como objetivo obter uma lista de localizações a partir de texto, com foco na localização atual de forma a evitar resultados díspares da mesma. Apresenta como

campos obrigatórios o texto a procurar bem como coordenadas de uma localização que permite filtrar os resultados encontrados tendo em conta a proximidade. Permite também ajustar o raio de pesquisa, o número de resultados bem como obter resultados noutra linguagem.

Figura 31 - Exemplo de pedido ao *endpoint* Autocomplete

O campo *query* corresponde ao texto a pesquisar, o campo *location* indica qual a localização por onde devemos basear o pedido (de forma aos resultados serem focados/próximos da localização indicada), o campo *radius* é o raio ao qual devemos limitar a procura, o campo *limit* indica o número de resultados a devolver, e *language* a linguagem dos resultados.

Da mesma forma, caso o pedido seja válido, é feito um pedido ao Photon através da sua REST API, e caso a resposta não tenha erros, transforma a resposta obtida e envia. Os resultados são enviados num *array* que contém o nome do local, bem como a sua estrutura (nome, código postal, cidade, estado e país), bem como a sua localização em coordenadas, o seu tipo (de acordo com as *tags*[104] do OpenStreetMap), um ícone representativo do tipo de localização encontrado, bem como o fornecedor de serviço (neste caso o Photon).

[104] https://wiki.openstreetmap.org/wiki/Tags

```
{
  "results": [
    {
      "name": "Rua da Mota",
      "address_components": [
        {
          "name": "Rua da Mota",
          "types": [
            "name"
          ]
        },
        {
          "name": "3830-142",
          "types": [
            "postalcode"
          ]
        },
        {
          "name": "Ílhavo",
          "types": [
            "city"
          ]
        },
        {
          "name": "Centro",
          "types": [
            "state"
          ]
        },
        {
          "name": "Portugal",
          "types": [
            "country"
          ]
        }
      ],
      "point": {
        "lat": 40.6091393,
        "lon": -8.691823
      },
      "types": [
        "highway",
        "tertiary"
      ],
      "icon": "https://storage.googleapis.com/static.luggit.app/services/routing/geocode.png"
    },
    {…5}
  ],
  "provider": "PHOTON"
}
```


Figura 32 - Exemplo de resposta ao *endpoint* Autocomplete

Assim, obtém-se a seguinte interação:

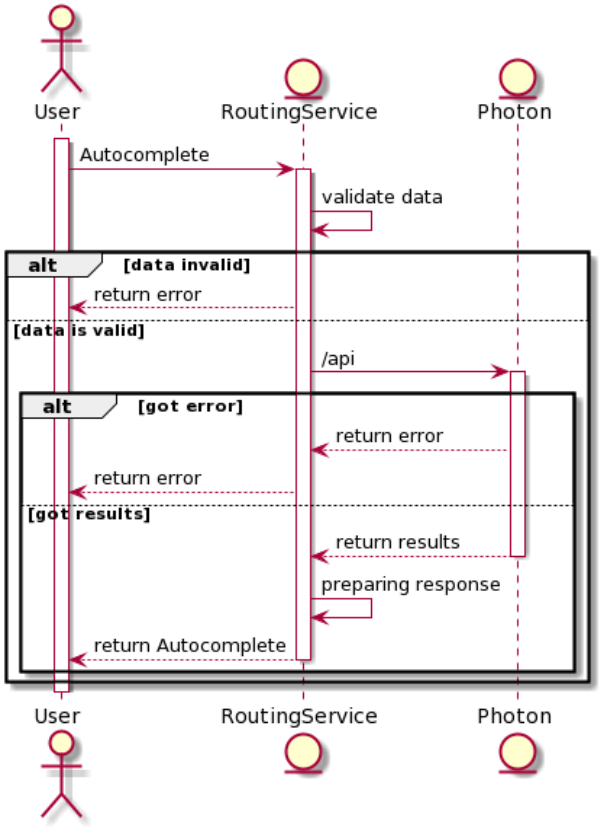

Figura 33 - Diagrama de interação do *endpoint* Autocomplete

4.3.2.3. Geocoding

O *endpoint* Geocoding tem como objetivo obter uma localização a partir da introdução de texto. O campo obrigatório é o texto a pesquisar, sendo que a linguagem é opcional.

```
{
  "query": "Avenida Vieira Guimarães",
  "language": "pt"
}
```

Figura 34 - Exemplo de pedido ao *endpoint* Geocoding

No caso deste *endpoint*, apenas é necessária a *query* indicando o texto que se pretende encontrar a localização, bem como a linguagem na qual o resultado deverá ser mostrado.

O funcionamento e a resposta deste *endpoint* é muito semelhante ao *endpoint* Autocomplete, na qual a diferença está neste *endpoint* só devolver um resultado, ao contrário do anterior que poderá devolver vários resultados.


```
{
  "results": {
    "name": "Avenida Vieira Guimarães",
    "point": {
      "lat": 39.5995705,
      "lon": -9.0676272
    },
    "address_components": [
      {
        "name": "Avenida Vieira Guimarães",
        "types": [
          "name"
        ]
      },
      {
        "name": "2450-109",
        "types": [
          "postalcode"
        ]
      },
      {
        "name": "Nazaré",
        "types": [
          "city"
        ]
      },
      {
        "name": "Centro",
        "types": [
          "state"
        ]
      },
      {
        "name": "Portugal",
        "types": [
          "country"
        ]
      }
    ],
    "types": [],
    "icon": "https://storage.googleapis.com/static.luggit.app/services/routing/geocode.png"
  },
  "provider": "PHOTON"
}
```


Figura 35 - Exemplo de resposta ao *endpoint* Geocoding

Assim, a interação deste *endpoint* é a seguinte:

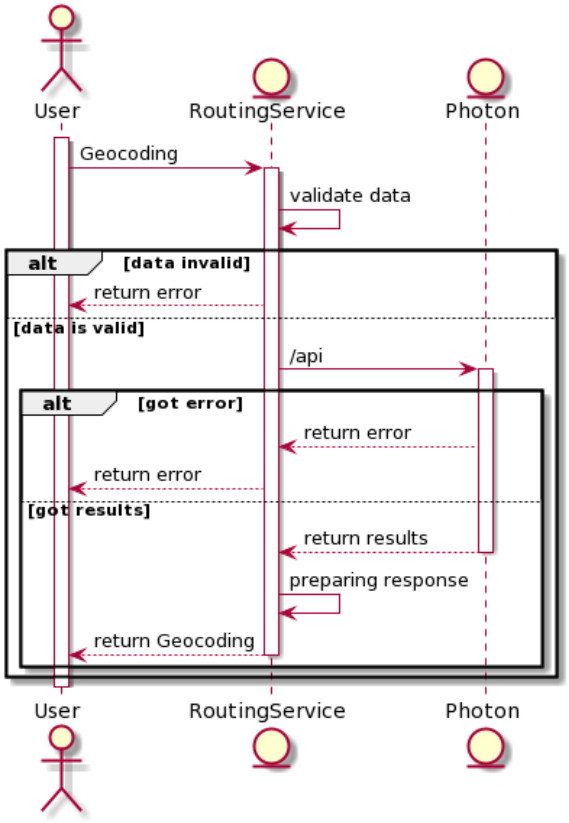

Figura 36 - Diagrama de interação do *endpoint* Geocoding

4.3.2.4. ReverseGeocoding

O *endpoint* ReverseGeocoding tem como objetivo obter um local/ponto de interesse a partir de uma localização (latitude e longitude). O ponto para o qual se pretende obter a localização é campo obrigatório, e os restantes campos opcionais.

```
{
  "point": {
    "lat": 40.618648,
    "lon": -8.666003
  },
  "zoom": "BUILDING",
  "language": "pt"
}
```

Figura 37 - Exemplo de pedido ao *endpoint* ReverseGeocoding

Ao contrário do que acontece com o *endpoint* Geocoding, aqui pretende-se o oposto, obter com base numa localização o local / ponto de interesse. A precisão do resultado a obter poderá ser ajustada através do campo *zoom*.

Tal como os *endpoints* anteriores, a resposta é muito semelhante, mostrando todos os componentes da morada que perfaz o local encontrado.


```
{
  "results": {
    "name": "PCI - Estacionamento Principal",
    "point": {
      "lat": 40.6180957,
      "lon": -8.666445586282196
    },
    "address_components": [
      {
        "name": "PCI - Estacionamento Principal",
        "types": [
          "name"
        ]
      },
      {
        "name": "Via do Conhecimento",
        "types": [
          "street"
        ]
      },
      {
        "name": "3830-352",
        "types": [
          "postalcode"
        ]
      },
      {
        "name": "Ílhavo",
        "types": [
          "city"
        ]
      },
      {
        "name": "Centro",
        "types": [
          "state"
        ]
      },
      {
        "name": "Portugal",
        "types": [
          "country"
        ]
      }
    ],
    "types": [],
    "icon": "https://storage.googleapis.com/static.luggit.app/services/routing/generic_business.png"
  },
  "provider": "PHOTON"
}
```


Figura 38 - Exemplo de resposta ao *endpoint* ReverseGeocoding

O diagrama é ligeiramente diferente, uma vez que caso o campo zoom seja especificado no pedido, o pedido será feito ao Nominatim, uma vez que na REST API do Photon não é possível especificar este parâmetro.

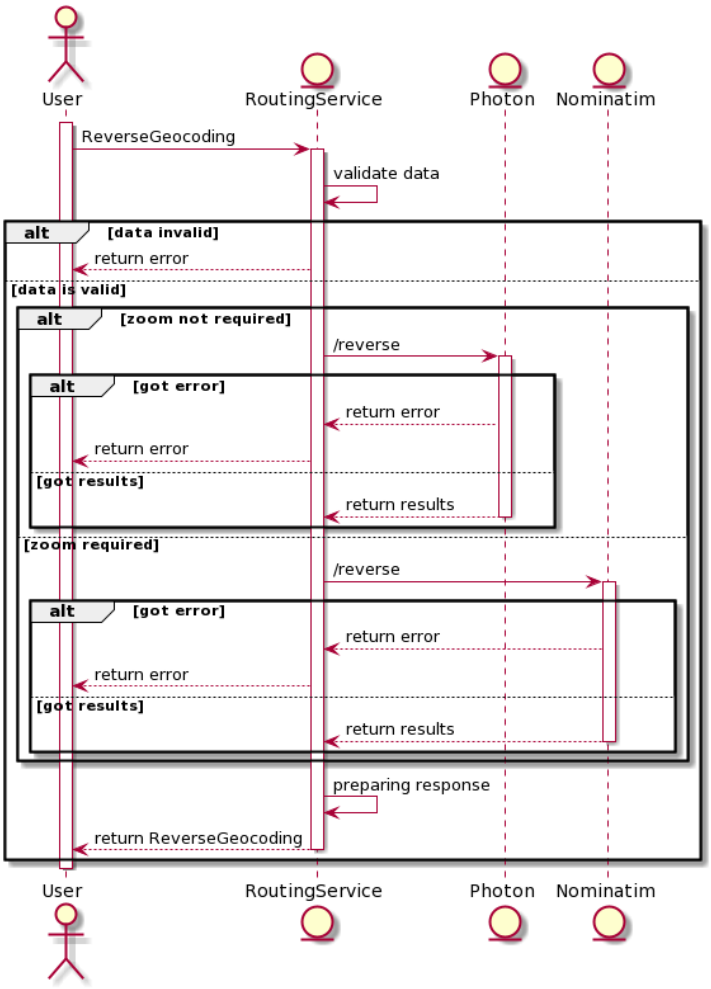

Figura 39 - Diagrama de interação do *endpoint* ReverseGeocoding

4.3.2.5. Isochrone

O *endpoint* Isochrone permite-nos obter regiões máximas atingíveis até um certo período de tempo a partir de uma dada localização. Na Figura 40 podemos ver um exemplo de 4 regiões sendo que a região verde corresponde a 15 minutos, a região amarela a 30 minutos, a região laranja a 45 minutos e a região a vermelho a 60 minutos.

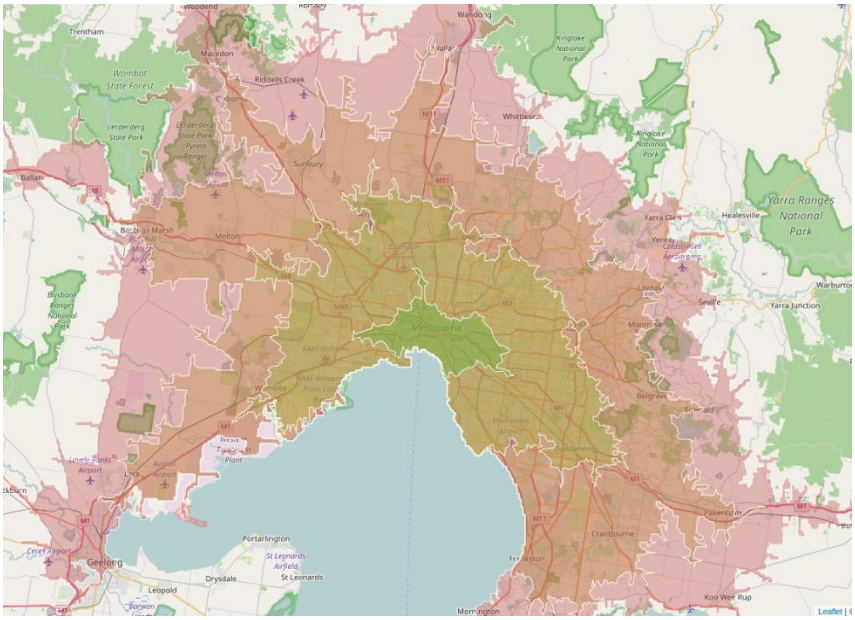

Figura 40 - Exemplo de uma *isochrone* na cidade de Melbourne [132]

Para o pedido é necessária a localização de onde o cálculo será feito, bem como um conjunto de até 4 períodos de tempo (número limitado pelo Valhalla) referentes ao número de polígonos a obter. Todos os campos são obrigatórios.

```
{
    "location": {
        "lat": 40.618648,
        "lon": -8.666003
    },
    "times": [10, 15, 20, 30]
}
```

Figura 41 - Exemplo de um pedido ao *endpoint* Isochrone

Na resposta, são obtidos polígonos no formato GeoJSON que correspondem às regiões encontradas, bem como a indicação do tempo definido para cada uma delas.

```
{
  "polygons": [
    {
      "time": 30,
      "polygon": "{\"coordinates\":[[[-8.565706,40.886944],[-8.542004,40.817501],[-8.498003,40.802166],
[-8.468003,40.826965],[-8.528136,40.748508],[-8.482003,40.782669],[-8.454336,40.704311],[-8.417262,40.702648],
[-8.434199,40.686451],[-8.366003,40.704342],[-8.42829,40.684647],[-8.374003,40.652344],[-8.336367,40.694649],
[-8.322003,40.665428],[-8.273876,40.682774],[-8.332004,40.648315],[-8.434234,40.652878],[-8.45705,40.600643],
[-8.397722,40.596645],[-8.444488,40.548649],[-8.444003,40.494415],[-8.472383,40.498646],[-8.438003,40.457268],
[-8.524282,40.490925],[-8.462583,40.368649],[-8.492003,40.32629],[-8.525472,40.471176],[-8.586003,40.42802],
[-8.610003,40.474964],[-8.604003,40.445122],[-8.666505,40.406651],[-8.629343,40.378647],[-8.702538,40.391186],
[-8.748003,40.284874],[-8.705686,40.364647],[-8.720003,40.398132],[-8.747647,40.360645],[-8.751705,40.440945],
[-8.794593,40.456646],[-8.749783,40.644428],[-8.637939,40.656586],[-8.622004,40.71954],[-8.584003,40.682537],
[-8.553918,40.69656],[-8.579867,40.752781],[-8.631764,40.762646],[-8.597223,40.768646],[-8.633445,40.794647],
[-8.589207,40.814644],[-8.610319,40.864964],[-8.559142,40.857784],[-8.565706,40.886944]]],\"type\":\"Polygon\"}"
    },
    {
      "time": 20,
      "polygon": "{\"coordinates\":[[[-8.542004,40.756123],[-8.536003,40.699665],[-8.560023,40.730667],
[-8.566003,40.685608],[-8.606003,40.704102],[-8.656003,40.646168],[-8.749512,40.644154],[-8.768003,40.526821],
[-8.748003,40.551083],[-8.705743,40.536907],[-8.736226,40.520649],[-8.693573,40.512646],[-8.702164,40.440647],
[-8.66222,40.510864],[-8.586901,40.499542],[-8.584003,40.563114],[-8.552632,40.522018],[-8.556635,40.551277],
[-8.510003,40.534519],[-8.526003,40.568821],[-8.560354,40.560646],[-8.524003,40.638664],[-8.578003,40.63438],
[-8.584925,40.666645],[-8.546448,40.6502],[-8.554331,40.676975],[-8.525611,40.681042],[-8.518003,40.651604],
[-8.511091,40.685734],[-8.461584,40.678646],[-8.519748,40.694904],[-8.522003,40.72966],[-8.538241,40.686405],
[-8.542004,40.756123]]],\"type\":\"Polygon\"}"
    },
    {
      "time": 15,
      "polygon": "{\"coordinates\":[[[-8.555656,40.692993],[-8.586723,40.666645],[-8.563623,40.628647],
[-8.594661,40.625305],[-8.564003,40.562717],[-8.587577,40.577076],[-8.602563,40.534649],[-8.664224,40.510426],
[-8.63912,40.540646],[-8.6743,40.548943],[-8.680003,40.50428],[-8.722253,40.552647],[-8.691855,40.584793],
[-8.752354,40.560295],[-8.748913,40.643555],[-8.714004,40.658955],[-8.684003,40.635696],[-8.622746,40.692646],
[-8.555656,40.692993]]],\"type\":\"Polygon\"}"
    },
    {
      "time": 10,
      "polygon": "{\"coordinates\":[[[-8.609632,40.667019],[-8.610216,40.570648],[-8.682003,40.556385],
[-8.719954,40.586647],[-8.693798,40.60685],[-8.728004,40.588245],[-8.746455,40.635098],
[-8.609632,40.667019]]],\"type\":\"Polygon\"}"
    }
  ]
}
```


Figura 42 - Exemplo de resposta ao *endpoint* Isochrone

Assim, obtém-se o seguinte diagrama de interação:

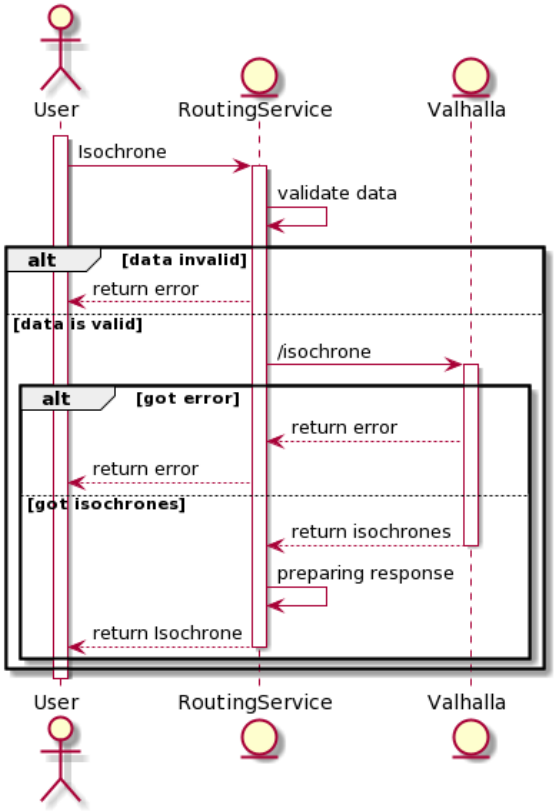

Figura 43 - Diagrama de interação do *endpoint* Isochrone

4.4. Serviço de Geofencing

O serviço de Geofencing é um serviço criado com o objetivo de fornecer informação com contexto geoespacial, utilizando funcionalidades de *geofencing* e de procura espacial, o que se torna útil no âmbito desta dissertação. Este serviço permite detetar e contextualizar localizações de utilizadores e/ou serviços, bem como indicar os locais ideais para um Keeper se encontrar com um cliente durante uma recolha ou entrega.

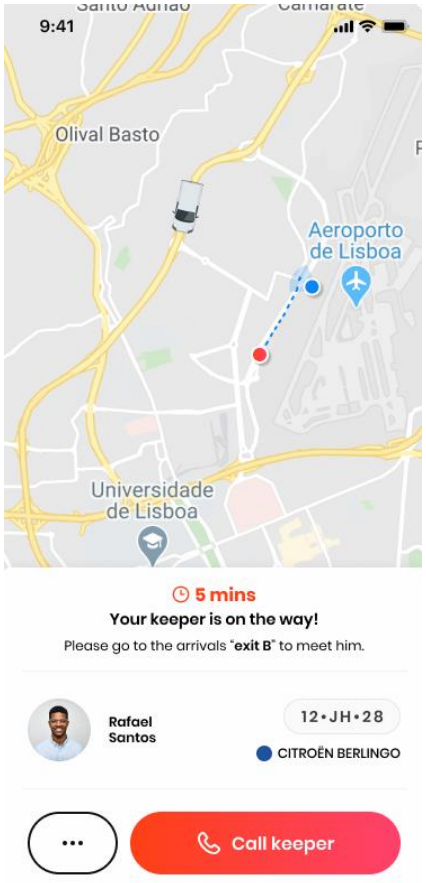

Figura 44 - Uso do *endpoint* GetMeetingPoints na aplicação dos utilizadores da LUGGit

Da mesma forma, é também importante obter pontos de interesse (como por exemplo parceiros hoteleiros da LUGGit) próximos da localização atual do utilizador para fornecer sugestões de pontos de recolha e/ou entrega, o que facilita a experiência de utilização da aplicação, bem como usar *geofences* para detetar e reagir a eventos. Neste sentido optou-se pela utilização de duas bases de dados com funcionalidades geoespaciais, nomeadamente o Tile38 e o PostGIS.

4.4.1. Tecnologias Usadas

Para o desenvolvimento deste serviço, optou-se também pela linguagem Go bem como pela *framework* gRPC para o desenvolvimento da API, e como já referido anteriormente, a área de operação de uma cidade onde é possível utilizar e requisitar um serviço da LUGGit é dada por um polígono, uma região fechada. Uma vez que a execução da aplicação dos utilizadores da LUGGit poderá ser feita em qualquer lugar, é necessária a deteção da localização atual numa das áreas de operação disponíveis, bem como em locais como os aeroportos, onde o estacionamento e respetiva

paragem dos veículos dos Keepers num serviço é limitada (e em alguns locais proibida) e apresenta custos mediante o tempo de paragem. As áreas de operação definidas são também as áreas onde os Keepers operam e atuam, sendo importantes de referir no âmbito desta dissertação. Este problema é chamado de *Point-in-Polygon* e consiste na deteção de um ponto, dado pelas coordenadas atuais do utilizador nos polígonos que definem cada uma destas regiões fechadas.

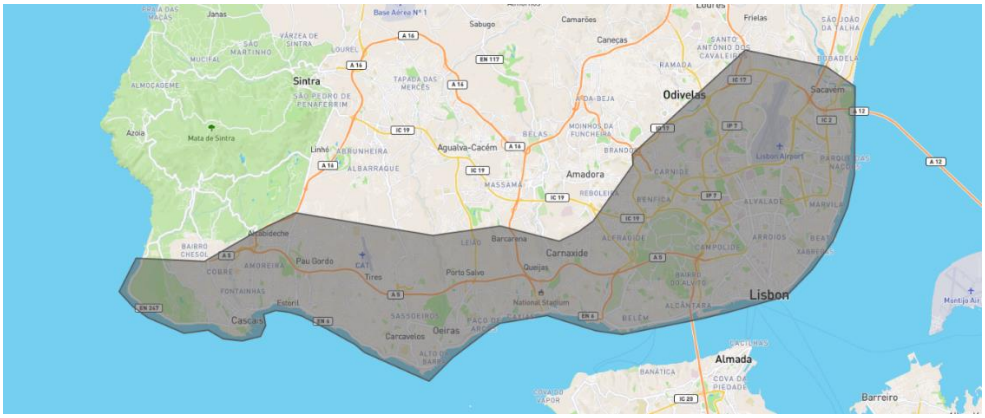

Figura 45 - Área de operação da LUGGit na cidade de Lisboa

Dado que se pretende guardar informação relativamente às áreas de operação disponíveis, bem como detalhes importantes das mesmas como o seu nome, moeda, e até mesmo o polígono que a define, utilizou-se uma base de dados relacional para o armazenamento dos dados necessários. A escolha recaiu sobre a base de dados PostgreSQL em conjunto com a extensão PostGIS, suportando tipos de dados geoespaciais como polígonos, pontos, entre outros, bem como oferece uma variedade de funções capazes de manipular estes tipos de dados. A criação e manipulação de dados no serviço é feita utilizando um ORM, nomeadamente o go-pg[105]. Uma vez que poderá haver alterações à base de dados durante os sucessivos lançamentos do serviço para ambiente de produção, usou-se uma ferramenta de gestão de migrações fornecida pelo go-pg chamada migrations[106], que assegura que o estado da base de dados está atualizado para a versão da migração mais recente.

Para podermos detetar pontos de interesse mais próximos, como por exemplo parceiros que recomendem aos seus clientes o serviço da LUGGit, e detetar se estamos dentro de uma área de operação e por sua vez dentro de um terminal de aeroporto, é usado o Tile38, uma base de dados

[105] https://github.com/go-pg/pg
[106] https://github.com/go-pg/migrations

não-relacional com funcionalidades geoespaciais, bem como de *geofencing*. Como se trata de uma base de dados baseada no Redis, utiliza sobretudo a memória principal para armazenamento, o que faz com que as consultas sejam mais rápidas, servindo de cache. Assim, no arranque do serviço de Geofencing, são feitas consultas à base de dados PostgreSQL para obter os polígonos representativos das áreas de operação, dos aeroportos e dos seus terminais, para que estas consultas geoespaciais possam ser feitas, preenchendo o Tile38 com os dados atualizados da base de dados.

4.4.2. Funcionalidades

O serviço de Geofencing fornece sobretudo funcionalidades geoespaciais, fazendo uso do PostGIS e Tile38, nomeadamente deteção de áreas de operação e de locais (ao que se denominou *CheckAvailability*), obter a lista de cidades disponíveis (ao que se denominou *GetCities*), obter um ponto de encontro dada uma localização (ao que se denominou *GetMeetingPoints*), obter a lista de parceiros mais próximos de uma localização (ao que se denominou *GetPartners*) e finalmente, obter a lista de pontos de interesse como eventos, estações de comboio / metro, entre outras (ao que se denominou *GetSuggestions*).

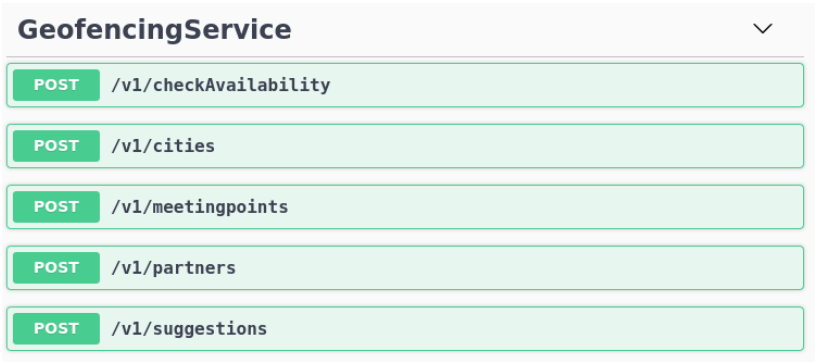

Figura 46 - *Endpoints* disponíveis no serviço de Geofencing

Tal como mostrado no capítulo 4.3.2, os *endpoints* são versionados de modo a suportar retrocompatibilidade entre versões bem como utilizam o método HTTP POST.

4.4.2.1. CheckAvailability

O *endpoint* CheckAvailability é responsável por indicar se uma localização está dentro de área de operação, o que torna possível fazer desde logo um pedido de serviço. É o primeiro pedido a ser efetuado na aplicação dos utilizadores da LUGGit. Tem como campo obrigatório a localização.

Figura 47 - Exemplo de pedido ao *endpoint* CheckAvailability

Para a execução do pedido é necessário o envio da localização (campo *location*), bem como da linguagem (campo *lang*) que define a obtenção de campos na resposta como o nome da cidade, ou a descrição de um ponto de encontro num aeroporto, como se pode ver na Figura 48. As linguagens disponíveis neste momento são Inglês e Português, no formato ISO 639-1:2002 [133], mas como vimos no capítulo 3.3.3.2 a arquitetura foi pensada para suportar diversas linguagens.

Na resposta, obtém-se a indicação de se de facto estamos dentro de uma área de operação (campo *available*), a cidade onde estamos, e caso estejamos num aeroporto indica dados referentes ao mesmo, como o ponto de encontro com o Keeper na eventualidade de um serviço na zona, dados referentes ao estacionamento / paragem no aeroporto, bem como a sua localização.

Figura 48 - Exemplo de resposta ao *endpoint* CheckAvailability

Assim, obtemos a seguinte interação:

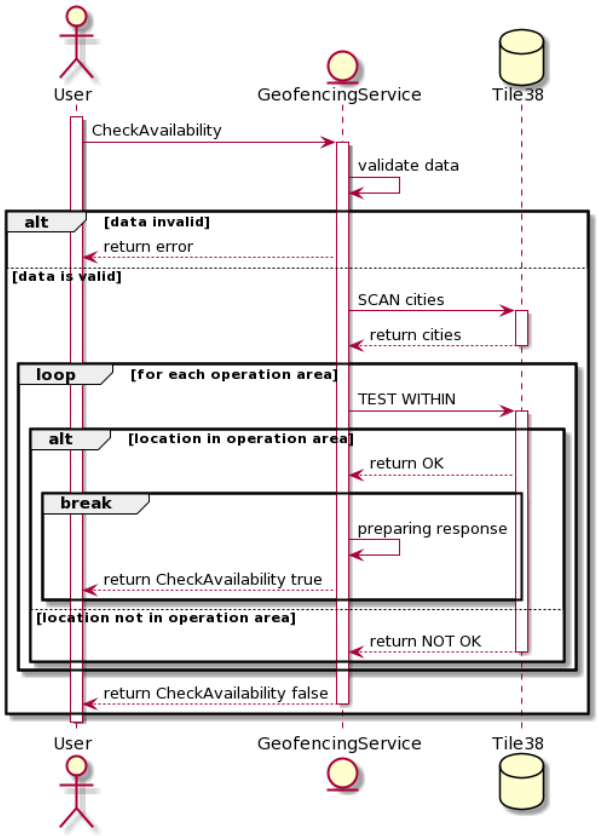

Figura 49 - Diagrama de interação do *endpoint* CheckAvailability

4.4.2.2. GetCities

O *endpoint* GetCities é responsável pela obtenção das cidades onde os serviços da LUGGit estão disponíveis. Para o pedido apenas é necessário fornecer a linguagem (através do campo *lang*).

```
{
    "lang": "pt-PT"
}
```

Figura 50 - Exemplo de pedido ao *endpoint* GetCities

Na resposta, obtêm-se as cidades disponíveis, bem como a localização dos seus aeroportos e a área de operação que as define, em formato GeoJSON.

```
{
  "cities": [
    {
      "id": "1",
      "name": "Lisboa",
      "airports": [
        {
          "name": "Aeroporto da Portela",
          "location": {
            "lat": 38.779444,
            "lon": -9.136111
          }
        }
      ],
      "slug": "lisbon",
      "area": "{\"type\":\"Polygon\",\"coordinates\":[[[-9.0898132,38.7961056],[-9.1055202,38.8046676],[-9.1495514,38.8116903],
[-9.2111778,38.7674019],[-9.2107487,38.762918],[-9.2324638,38.7392223],[-9.2401886,38.7345358],[-9.2508316,38.7305186],
[-9.2827606,38.7369461],[-9.3193245,38.7329289],[-9.3943405,38.7424358],[-9.4171715,38.7337324],[-9.4437361,38.7219142],
[-9.4809437,38.7234208],[-9.4902992,38.7091898],[-9.4830894,38.7029272],[-9.4745064,38.6980374],[-9.4546366,38.6914389],
[-9.4416761,38.6926447],[-9.4336081,38.6890271],[-9.4230938,38.6880557],[-9.4131804,38.690166],[-9.4105196,38.694621],
[-9.4115067,38.6995111],[-9.4049406,38.7010182],[-9.39713,38.7001474],[-9.3906927,38.6985733],[-9.3779039,38.6934486],
[-9.3618536,38.6866487],[-9.3570042,38.6834328],[-9.3321562,38.6747892],[-9.3215132,38.6710701],[-9.3087244,38.6810877],
[-9.2886829,38.6930132],[-9.2828894,38.6957933],[-9.2737055,38.6968986],[-9.2571831,38.6989082],[-9.2308331,38.6925443],
[-9.2164135,38.6908024],[-9.1890335,38.6948889],[-9.1731548,38.6975015],[-9.1577053,38.7005158],[-9.1272354,38.7073479],
[-9.1131592,38.7185993],[-9.1018295,38.7305186],[-9.0938473,38.7449127],[-9.0901566,38.7585677],[-9.0898132,38.7961056]]]}"
    },
    {
      "id": "2",
      "name": "Porto",
      "airports": [
        {
          "name": "Aeroporto Francisco Sá Carneiro",
          "location": {
            "lat": 41.236939,
            "lon": -8.670278
          }
        }
      ],
      "slug": "porto",
      "area": "{\"type\":\"Polygon\",\"coordinates\":[[[-8.728123,41.269936],[-8.729324,41.263744],[-8.725376,41.255486],
[-8.729153,41.2449],[-8.729839,41.238575],[-8.725719,41.233154],[-8.722286,41.225925],[-8.720055,41.21004],
[-8.719196,41.203197],[-8.714218,41.19661],[-8.710785,41.191315],[-8.71336,41.18524],[-8.714561,41.179817],
[-8.71233,41.175167],[-8.710785,41.171677],[-8.70718,41.170383],[-8.69791,41.17581],[-8.693962,41.171032],[-8.6816025,41.1488],
[-8.679543,41.144276],[-8.6706161,41.1283748],[-8.668642,41.1114341],[-8.6684704,41.1082007],[-8.6283875,41.1075539],
[-8.6066723,41.107748],[-8.5879612,41.1040616],[-8.567791,41.0995989],[-8.5527706,41.0953946],[-8.5502386,41.0943273],
[-8.5502386,41.0942788],[-8.5502601,41.0943273],[-8.5494661,41.0965265],[-8.5479212,41.1007631],[-8.5466337,41.104288],
[-8.5462904,41.1049509],[-8.5448742,41.1069072],[-8.5441232,41.1078935],[-8.5436082,41.1085078],[-8.5428786,41.1092192],
[-8.5425568,41.1097689],[-8.5421491,41.1104318],[-8.5416126,41.1113209],[-8.5415053,41.1118221],[-8.5415483,41.1120646],
[-8.5415697,41.1121455],[-8.5415268,41.1122101],[-8.5410333,41.1126143],[-8.5404539,41.1127436],[-8.5395956,41.1132286],
[-8.5385013,41.1139884],[-8.532257,41.11609],[-8.521099,41.11816],[-8.511658,41.119713],[-8.504105,41.12178],
[-8.500328,41.1762],[-8.527451,41.21185],[-8.543072,41.273033],[-8.727264,41.270325],[-8.728123,41.269936]]]}"
    }
  ],
  "lang": "pt-PT"
}
```


Figura 51 - Exemplo de resposta ao *endpoint* GetCities

O diagrama de interação deste *endpoint* é o seguinte:

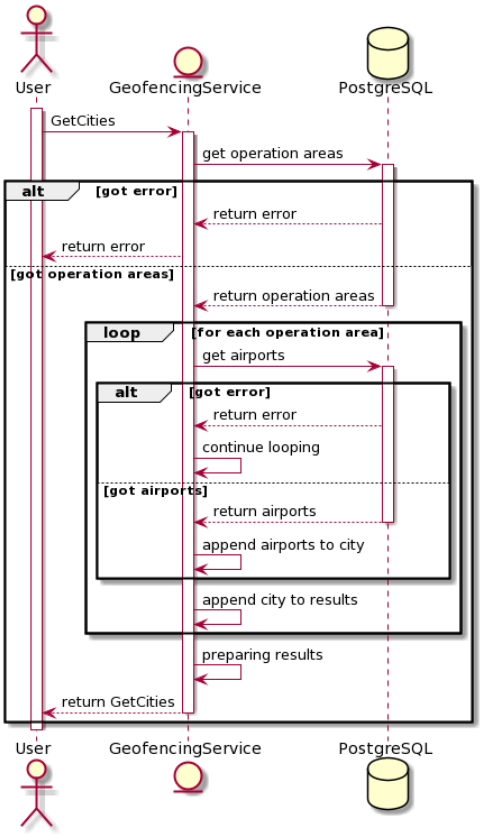

Figura 52 - Diagrama de interação do *endpoint* GetCities

4.4.2.3. GetMeetingPoints

O *endpoint* GetMeetingPoints tem como objetivo fornecer pontos de encontro próximos de uma localização, e para tal são utilizados pontos de interesse como os parceiros da LUGGit, pontos de paragem em terminais de aeroportos, locais junto a eventos, bem como a rua/estrada mais próxima da localização, na qual é utilizado o Serviço de Routing. Apenas é obrigatório definir a localização como parâmetro.

```
{
    "location": {
        "lat": 38.711661,
        "lon": -9.130539
    },
    "radius": 100,
    "lang": "pt-PT"
}
```

Figura 53 - Exemplo de pedido ao *endpoint* GetMeetingPoints

O pedido aceita como parâmetros a localização (com o campo *location*), o raio de procura (através do campo *radius*), bem como a linguagem (campo *lang*).

A resposta a este pedido é quando bem-sucedida, uma lista de pontos de interesse e os seus pontos de encontro. Neste caso, a resposta contém diversos pontos de interesse, nomeadamente parceiros da LUGGit (alguns resultados foram ocultados), bem como a rua mais próxima, onde podemos ver pelo campo *type* de cada um dos resultados obtidos.

```
{
  "results": [
    {
      "name": "Guest Inn Alfama IV",
      "meeting_points": [
        {
          "name": "Guest Inn Alfama IV",
          "description": "",
          "address": "R. de São Miguel 85 3º, 1100-543 Lisboa",
          "location": {
            "lat": 38.7116942,
            "lon": -9.1308437
          }
        }
      ],
      "type": "partner"
    },
    {…3…},
    {…3…},
    {…3…},
    {…3…},
    {…3…},
    {
      "name": "Alfama Lovely Duplex",
      "meeting_points": [
        {
          "name": "Alfama Lovely Duplex",
          "description": "",
          "address": "R. do Vigário 90 2º Andar, 1100-617 Lisboa",
          "location": {
            "lat": 38.7124642,
            "lon": -9.1300657
          }
        }
      ],
      "type": "partner"
    },
    {
      "name": "Largo de Santa Luzia",
      "meeting_points": [
        {
          "name": "Largo de Santa Luzia",
          "description": "",
          "address": "Largo de Santa Luzia, 1100-411, Lisboa",
          "location": {
            "lat": 38.711677050375,
            "lon": -9.1305810350125
          }
        }
      ],
      "type": "road"
    }
  ],
  "lang": "pt-PT"
}
```


Figura 54 - Exemplo de resposta ao *endpoint* GetMeetingPoints

Deste modo, apresenta-se a seguir o diagrama de interação deste *endpoint*:

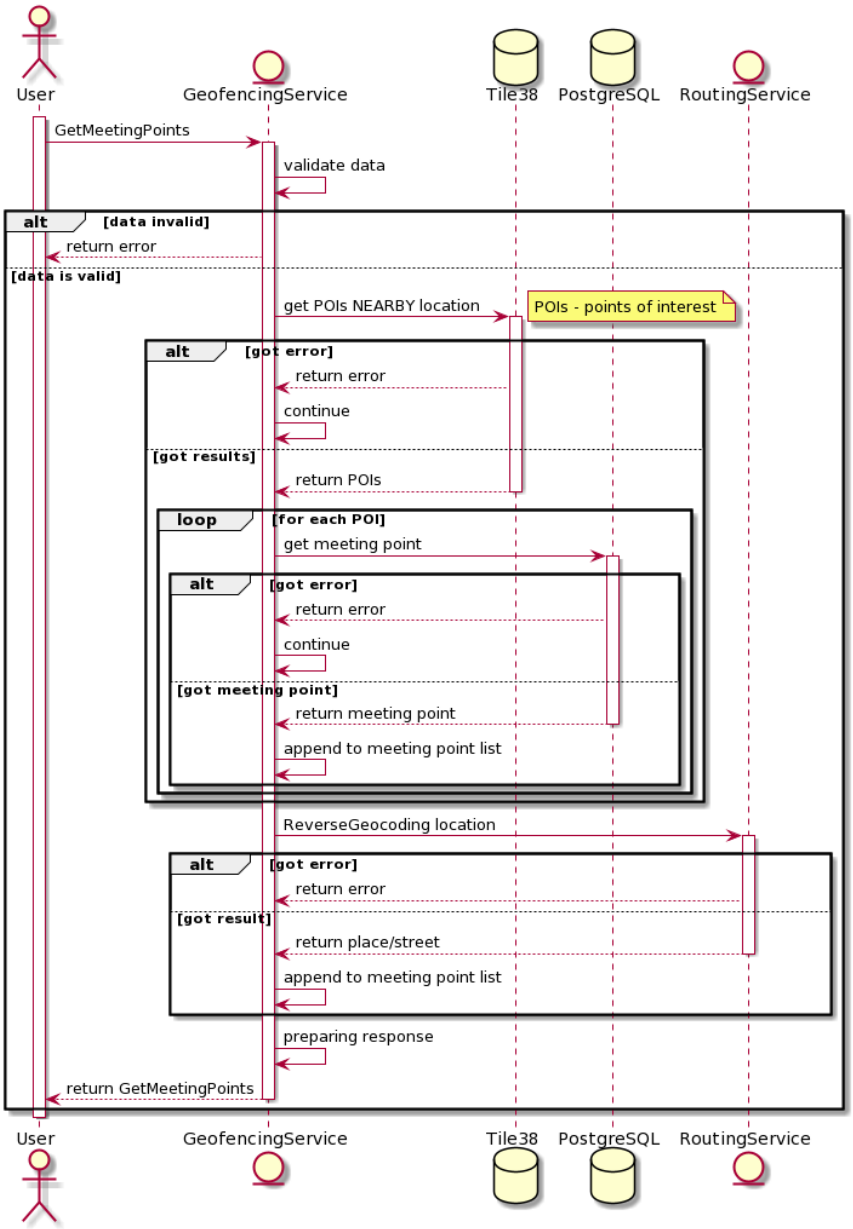

Figura 55 - Diagrama de interação ao *endpoint* GetMeetingPoints

4.4.2.4. GetPartners

O *endpoint* GetPartners retorna uma lista de parceiros próxima de uma localização. Os parceiros da LUGGit são sobretudo entidades ligadas ao alojamento, que recomendam os serviços da LUGGit aos seus clientes. É obrigatório definir a localização para qual se pretende encontrar a lista de parceiros.

```
{
    "location": {
        "lat": 41.239191,
        "lon": -8.671088
    },
    "limit": 3
}
```


Figura 56 - Exemplo de pedido ao *endpoint* GetPartners

Para obter a lista de parceiros mais próxima de um local, é necessário indicar no pedido a localização (campo *location*), bem como o número de resultados máximo que se pretende obter.

A resposta é dada por uma lista de parceiros, onde constam informações como o email, telefone, foto, morada e a localização do parceiro.


```
{
  "partners": [
    {
      "name": "BnBird - Porto Prestige Flat",
      "email": "info@bnbird.com",
      "phone": "+351 215800096",
      "nif": "",
      "photo": "https://bnbird.com/wp-content/uploads/2016/02/bnbird-text-logo.png",
      "address": "R. Eugénio de Castro 238 Apt 513, 4100-225 Porto",
      "location": {
        "lat": 41.163354,
        "lon": -8.6501837
      }
    },
    {
      "name": "Lovely Stay - Serralves Apartment",
      "email": "booklisbon@lovelystay.com",
      "phone": "",
      "nif": "",
      "photo": "",
      "address": "Rua de Tangêr, Porto",
      "location": {
        "lat": 41.1618159,
        "lon": -8.6599992
      }
    },
    {
      "name": "Lovely Stay - Bessa Leite Residence",
      "email": "booklisbon@lovelystay.com",
      "phone": "",
      "nif": "",
      "photo": "",
      "address": "Rua António Bessa Leite, Porto",
      "location": {
        "lat": 41.1583851,
        "lon": -8.6451472
      }
    }
  ]
}
```


Figura 57 - Exemplo de resposta ao *endpoint* GetPartners

O funcionamento deste *endpoint* está descrito na seguinte figura:

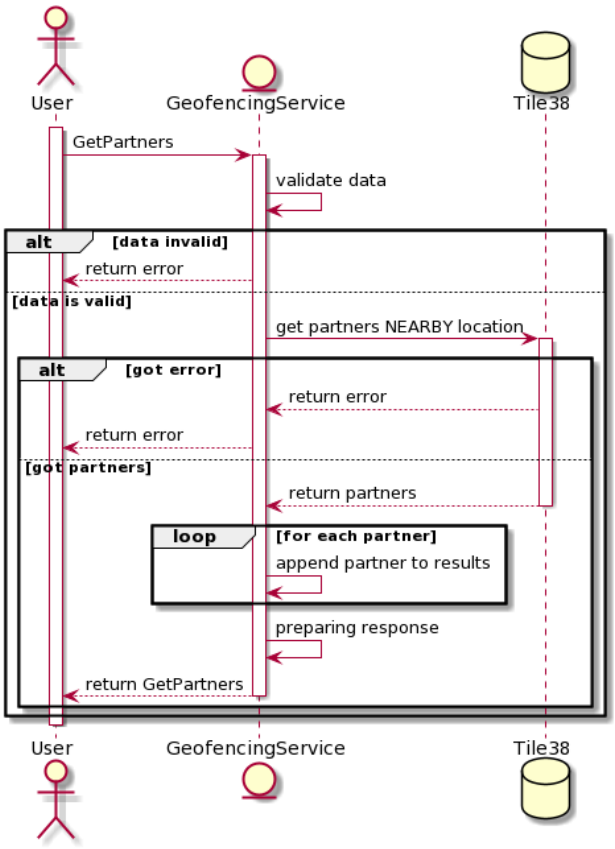

Figura 58 - Diagrama de interação do *endpoint* GetPartners

4.4.2.5. GetSuggestions

O *endpoint* GetSuggestions fornece uma lista de pontos de interesse mistos (parceiros, aeroportos, estações de comboio / metro, eventos, entre outros), próxima de uma localização (campo *location*), onde é possível alargar o raio de pesquisa (campo *radius*), limitar o número máximo de resultados (campo *limit*), bem como ordenar os resultados pelo tipo de pontos de interesse (campo *sort*). Apenas é obrigatório indicar a localização.

```
{
    "location": {
        "lat": 41.239191,
        "lon": -8.671088
    },
    "radius": 10000,
    "sort": "partner",
    "limit": 3
}
```

Figura 59 - Exemplo de pedido ao *endpoint* GetSuggestions

A resposta a este *endpoint* revela os pontos de interesse sugeridos nas proximidades da localização, ordenadas pelo tipo, que conforme vimos acima é referente aos parceiros. Para cada um dos pontos de interesse conseguimos saber também a sua localização, bem como a morada e um ícone identificativo.


```
{
  "suggestions": [
    {
      "name": "Lovely Stay - Serralves Apartment",
      "location": {
        "lat": 41.1618159,
        "lon": -8.6599992
      },
      "address": "Rua de Tangér, Porto",
      "icon": "https://storage.googleapis.com/static.luggit.app/services/geofencing/partner.png",
      "type": "partner",
      "id": "292"
    },
    {
      "name": "Terminal 1",
      "location": {
        "lat": 41.237357,
        "lon": -8.670376
      },
      "address": "",
      "icon": "https://storage.googleapis.com/static.luggit.app/services/geofencing/airport.png",
      "type": "airport",
      "id": "3"
    },
    {
      "name": "Porto Airport",
      "location": {
        "lat": 41.236939,
        "lon": -8.670278
      },
      "address": "",
      "icon": "https://storage.googleapis.com/static.luggit.app/services/geofencing/airport.png",
      "type": "airport",
      "id": "2"
    }
  ]
}
```


Figura 60 - Exemplo de resposta ao *endpoint* GetSuggestions

O seguinte diagrama de interação mostra o seu funcionamento:

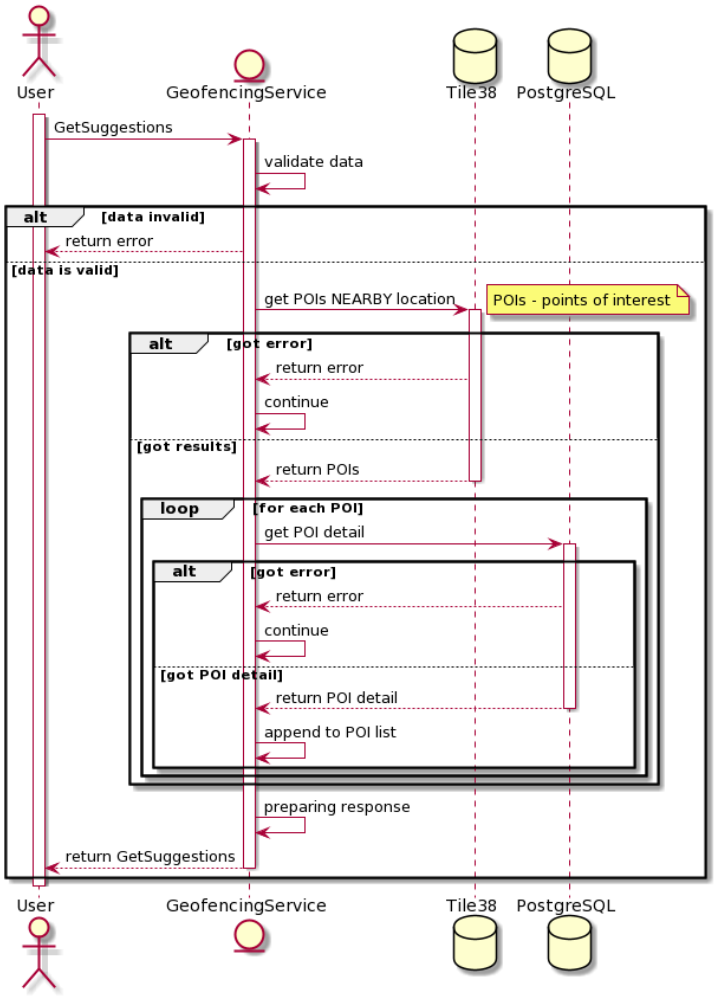

Figura 61 - Diagrama de interação do *endpoint* GetSuggestions

4.5. Serviço de Correção de Posicionamento de Condutores

O serviço de Correção de Posicionamento de Condutores é um dos grandes objetivos desta dissertação, levando os Keepers a corrigirem o seu posicionamento numa área de operação através de uma análise de dados feita a partir de pedidos efetuados historicamente e em tempo-real. Em última instância são os *heatmaps* gerados por este serviço com recurso ao H3 que irão ter influência no posicionamento dos Keepers, sendo indicadores visuais que lhes permitem escolher zonas estratégicas numa área de operação.

Figura 62 - Uso dos *heatmaps* na aplicação dos Keepers da LUGGit

Ao mesmo tempo, torna-se necessário recolher a localização atual dos Keepers bem como detetar a sua presença em tempo-real, de modo a reunir informação que nos permita selecionar e notificar um Keeper que possa responder a um novo pedido de serviço em menos de 15 minutos. Assim, optou-se por desenvolver um serviço que não só recolhe e processa eventos de uma forma rápida e escalável usando o Apache Kafka, como também armazena dados relativos a serviços de recolha/entrega da LUGGit através de uma *data warehouse* que recolhe e fornece métricas úteis tanto de um ponto de vista operacional como de um ponto de vista estratégico, e para tal usou-se o PostgreSQL em conjunto com o PostGIS.

4.5.1. Desenvolvimento do Serviço e Tecnologias Usadas

Tal como os serviços abordados anteriormente, este serviço foi desenvolvido usando a linguagem de programação Go, e a *framework* de comunicação gRPC. Como já foi discutido, a recolha de dados dos Keepers é muito importante, não só para a empresa ter controlo da sua operação, mas também para tarefas de análise e processamento de dados, como abordado nesta dissertação.

Até ao momento, não existia na empresa nenhum método que permitisse recolher um histórico de localizações dos Keepers, sendo que ao utilizar a Cloud Firestore apenas era recolhida a localização mais recente. Para o desenvolvimento deste serviço era fulcral a implementação de uma infraestrutura capaz de receber as localizações dos Keepers de maneira recorrente e constante durante a execução da aplicação, sem prejuízo de alta latência ou de sobrecarga quando escalada a vários utilizadores. Esta foi a razão que levou ao uso do Apache Kafka, que através do padrão *Publish-Subscribe* permite emitir e receber mensagens de uma forma rápida e escalável. Para que esta solução se torne escalável, cada mensagem terá de ser emitida por uma entidade, e por sua vez recebida e processada por uma outra entidade.

Assim, optou-se por separar cada uma destas entidades em dois microserviços, onde um é responsável pela emissão de mensagens (ao que se chamou de *Keeper Optimizer*), e outro é responsável pela recepção e respetivo processamento (ao que se chamou de *Keeper Processor*), o que pode ser visto na Figura 63. O serviço *Keeper Optimizer* emite a mensagem, e o processamento é feito pelo *Keeper Processor*, podendo esse processamento ser demorado. Desta forma o envio e processamento da localização não é sentido pelo Keeper ao utilizar a sua aplicação.

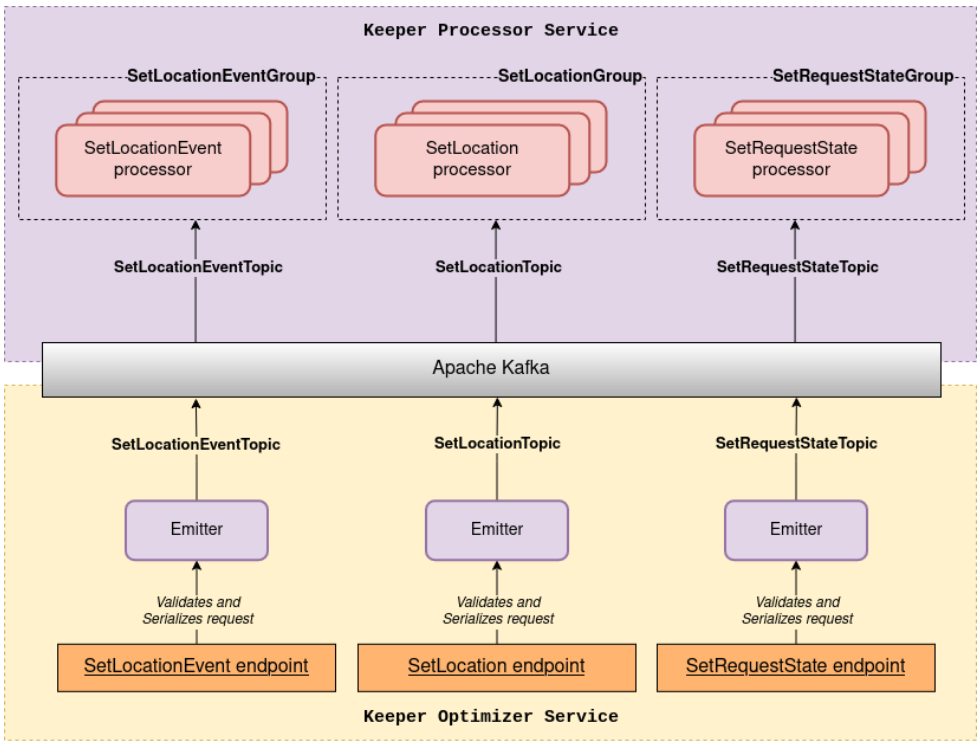

Figura 63 - Separação do serviço de Correção de Posicionamento de Condutores nos microserviços Keeper Optimizer e Keeper Processor

Como referido no capítulo 4.2.5, as mensagens são enviadas para um tópico. Neste caso, o *Keeper Optimizer* emite uma mensagem para um tópico de acordo com o *endpoint*, e esse mesmo tópico é subscrito por um grupo de processadores no *Keeper Processor*. Assim que uma nova mensagem chega, um dos processadores desse tópico é notificado pelo Kafka e consome a mensagem, que consiste num par chave-valor. Uma vez que neste serviço se usam Protocol Buffers como mecanismo de serialização, sempre que uma nova mensagem chega ao *Keeper Optimizer* esta é validada, serializada e enviada para o tópico, usando como chave a marca temporal do momento de emissão e como valor o conjunto de *bytes* provenientes da serialização da mensagem. No lado do *Keeper Processor* é feito o oposto, convertem-se os *bytes* do valor da mensagem, voltando à mensagem originalmente enviada pelo *Keeper Optimizer*, que por sua vez é processada.

Para a interação e desenvolvimento dos componentes que interagem com o Kafka nos dois microserviços foi usada a biblioteca Goka[107], que utiliza conceitos que permitem abstrair o funcionamento do Kafka. No Goka, os *emitters* são, como o nome indica, emissores que enviam mensagens chave-valor para o Kafka. Os *processors* são um conjunto de funções que consomem as mensagens recebidas, podendo também emitir mensagens para o Kafka. Quando se usam grupos de *processors*, os *processor groups*, permite-se a escalabilidade para várias instâncias bem como tolerância a falhas, onde quando um *processor* falha este é substituído por um outro criado pelo próprio Goka. O estado de um *processor group* é mantido numa *group table*, que também utiliza mensagens chave-valor como método de armazenamento e de leitura, que é feita pelas *views*, que se definem como sendo *caches* locais de uma *group table*.

Foram criados três *endpoints* diferentes para envio de mensagens: *SetLocation*, que recebe frequentemente as localizações dos Keepers durante a execução da aplicação; *SetLocationEvent*, que recebe as localizações dos Keepers quando ocorre um evento do serviço (Keeper aceitou o serviço, Keeper chegou ao local de recolha, entre outros); e o *endpoint SetRequestState*, que é chamado quando um serviço da LUGGit muda o seu estado, o que nem sempre envolve uma ligação com o Keeper.

No lado do *Keeper Processor* ocorre o processamento das mensagens provenientes dos tópicos respetivos de cada *endpoint*. O que se pretende é não só fornecer *heatmaps* como também

[107] https://github.com/lovoo/goka

criar uma *data warehouse* interna que reúna dados de modo a serem fornecidas métricas relativamente a cada serviço efetuado levando a que se possam contextualizar as localizações, bem como permitir saber a cada instante onde se situam os Keepers, qual o seu estado, entre outros. Algo igualmente necessário numa perspetiva de análise de dados é a capacidade de se poder relacionar esses mesmos dados com características externas ao serviço, de modo a que se possam criar eventuais campanhas de *marketing*, bem como preparar a operação logística antecipadamente, prevendo um maior acréscimo de serviços a ocorrer em feriados, por exemplo. Deste modo escolheu-se integrar dados provenientes de fontes externas. É usada a API Calendarific[108] para fornecer os feriados, associando-os às datas; a API do OpenWeatherMap[109] é usada para providenciar o estado do tempo atual, de modo a este ser associado a um serviço; e a API aviationstack[110] para consultar dados referentes aos voos nos aeroportos das áreas de operação, tornando possível relacionar a data e hora tanto das partidas como das chegadas de voos com os momentos de ocorrência de serviços.

Com esta recolha e processamento de dados, conseguem-se construir os *heatmaps* através de dados históricos bem como dados em tempo-real, onde se optou por considerar como sendo tempo-real os dados da última hora. Estes *endpoints* fornecidos pelo *Keeper Processor* são o *GetRealtimeHeatmap*, que retorna um *heatmap* com dados em tempo-real provenientes do Kafka, o *GetHistoricHeatmap*, que retorna um *heatmap* com dados históricos, sendo o período histórico definido no pedido em si, e finalmente o *endpoint GetHeatmap*, que retorna um *heatmap* que combina dados em tempo-real com dados históricos, que é feita através da atribuição de pesos estáticos para cada uma das componentes. Para o fornecimento dos dados históricos foram selecionados os dados da última semana, sendo também possível a sua alteração. Para a agregação de localizações através da indexação espacial utilizou-se o H3, e para o armazenamento dos dados utilizou-se a base de dados PostgreSQL com a extensão PostGIS, bem como a base de dados geoespacial Tile38. Como também se sabe a localização dos Keepers a cada momento, torna-se também possível a implementação de um *endpoint* que indique quais os Keepers em melhor posição para um novo pedido de serviço, ao que se chamou *GetNearKeepers*.

[108] https://calendarific.com/
[109] https://openweathermap.org/
[110] https://aviationstack.com/

4.5.2. Funcionalidades

Keeper Optimizer

O Keeper Optimizer torna-se assim um serviço que fornece *endpoints* para a emissão de eventos que sejam posteriormente processados pelo Keeper Processor. Os *endpoints* disponíveis neste serviço são, pela ordem mostrada na Figura 64 o *SetLocationEvent*, o *SetLocation* e o *SetRequestState*.

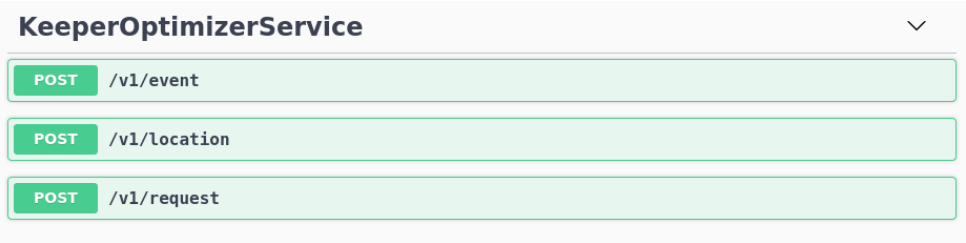

Figura 64 - *Endpoints* disponíveis no Keeper Optimizer

Keeper Processor

Por sua vez, o Keeper Processor é um serviço responsável por processar as mensagens obtidas a partir do Keeper Optimizer, que como já vimos são enviadas através do *message broker* Apache Kafka. Uma vez que este serviço irá receber e lidar com esses dados nomeadamente a construção dos *heatmaps* de acordo com as localizações bem como a ligação à base de dados, houve a necessidade de disponibilizar *endpoints* neste serviço. Os *endpoints* disponíveis pela ordem que são mostrados na Figura 65 são o *GetHeatmap*, *GetHistoricHeatmap*, *GetNearKeepers* (através de duas versões, uma feita tendo por base as localizações próximas dos Keepers utilizando o H3, e uma outra fazendo uso das *isochrones* devolvidas pelo serviço de Routing) e o *endpoint GetRealtimeHeatmap*.

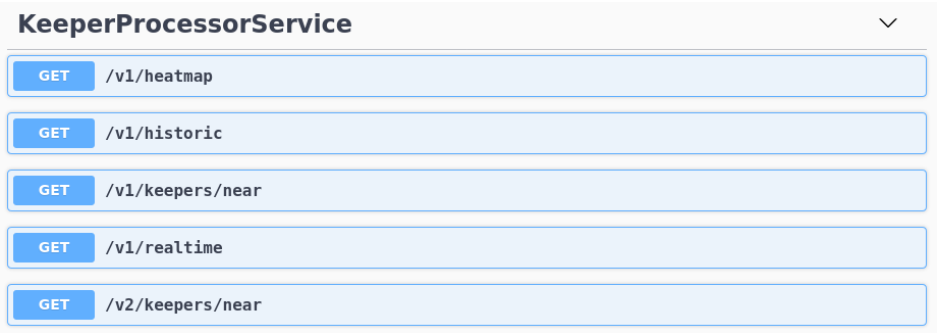

Figura 65 - *Endpoints* disponíveis no Keeper Processor

4.5.2.1. SetLocation

O *endpoint* SetLocation é responsável por enviar, em média a cada 15 segundos, a localização atual do Keeper quando este assinala na aplicação que está *online*, ou seja, que está disponível para efetuar novos serviços.

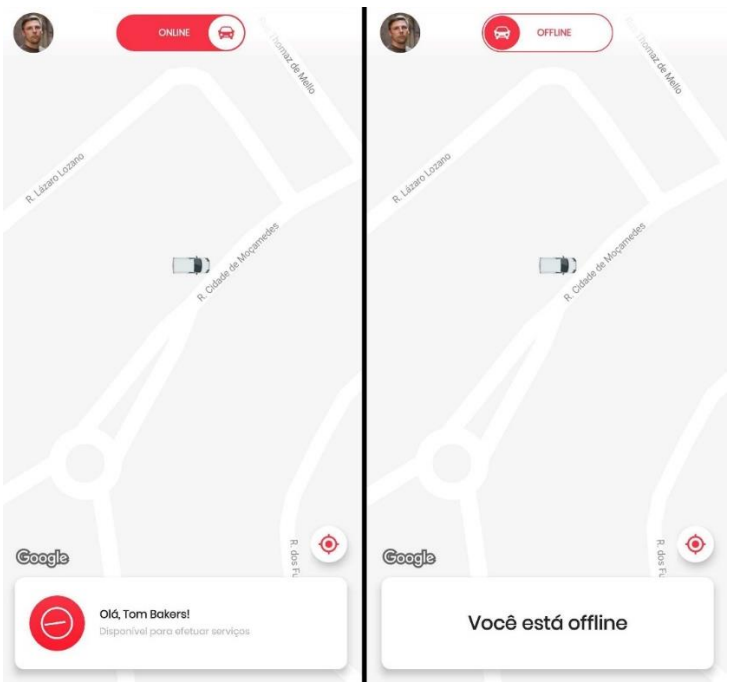

Figura 66 - Indicação do estado *online* na aplicação dos Keepers da LUGGit

Para o pedido, é enviado o identificador do Keeper (campo *keeper_id*) na Cloud Firestore, a principal base de dados utilizada pela LUGGit. Também é enviada a localização atual (campo *location*) bem como a marca temporal de quando a localização foi enviada (campo *timestamp*), que poderá

ser enviada seja através do formato Unix ou pelo formato ISO 8601 [134] de data e hora completa. Quando o Keeper está a efetuar um serviço, é indicado também o identificador desse serviço. Não é enviada nenhuma resposta específica a este pedido.

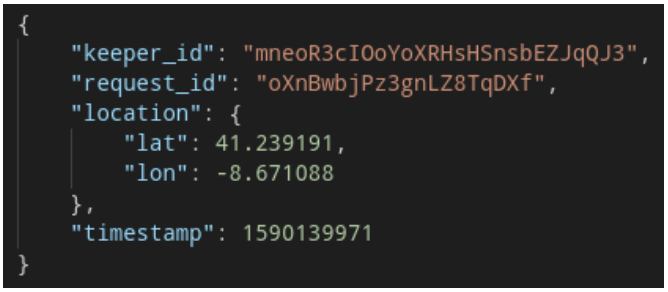

Figura 67 - Exemplo de pedido ao *endpoint* SetLocation

De seguida podemos ver os diagramas de interação que representam como é que a localização é enviada para o Kafka pelo Keeper Optimizer e posteriormente processada pelo Keeper Processor. No lado do Keeper Optimizer, quando o Keeper indica que está *online* é enviada em média a cada 15 segundos a sua localização para o Keeper Optimizer, que por sua vez recebe o pedido, serializa-o utilizando Protocol Buffers, ou seja o mesmo formato no qual o pedido foi enviado, e envia-o para o Kafka através do tópico criado especificamente para esse efeito, sendo a chave a marca temporal do pedido e o valor a mensagem serializada. Quando o Keeper marca como estando *offline* na aplicação ou sai da mesma não é enviada qualquer localização, e o processo é assim interrompido. O estado *online*/*offline* do Keeper é verificado recorrentemente, e caso um Keeper tenha 15 minutos de inatividade, o que acontece quando deixa de enviar a sua localização, a base de dados é atualizada com o momento em que este deixou de estar online.

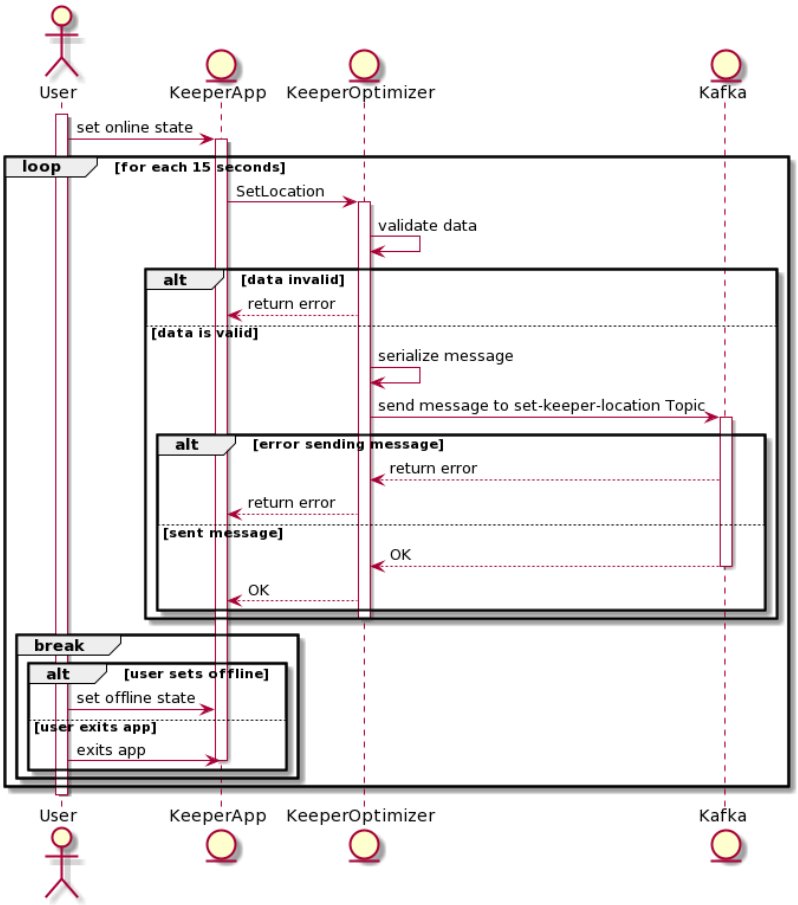

Figura 68 - Diagrama de interação do *endpoint* SetLocation (lado do Keeper Optimizer)

Do lado do Keeper Processor, é recebida a mensagem, é feita a deserialização de volta ao formato Protocol Buffers, e por sua vez inserida a localização do Keeper nas bases de dados PostgreSQL e Tile38.

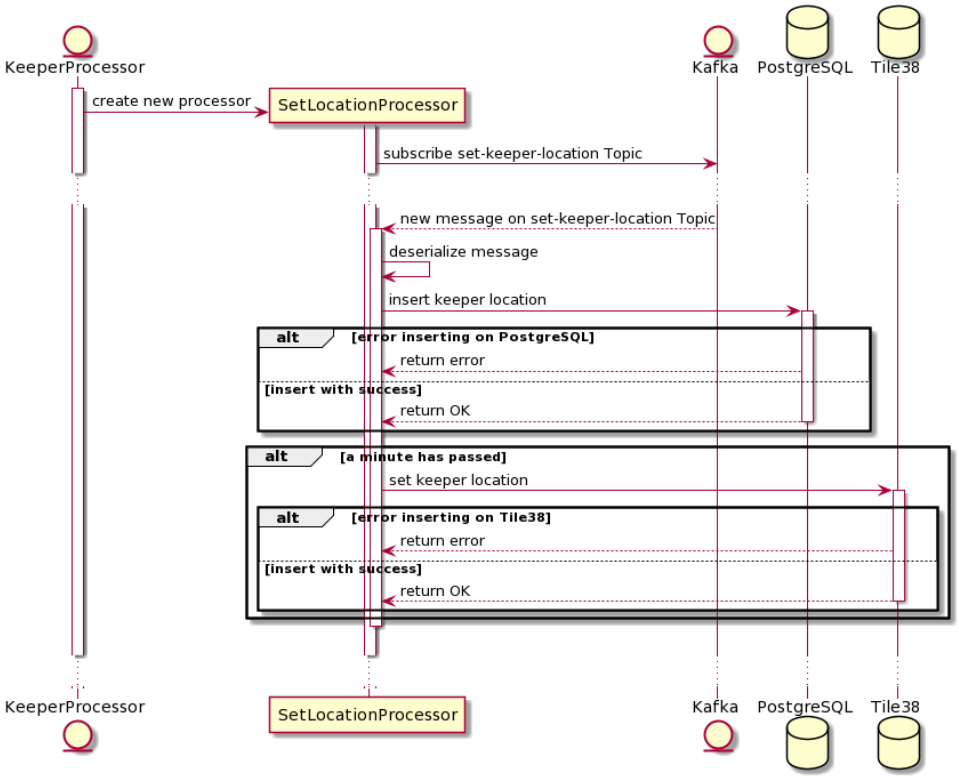

Figura 69 - Diagrama de interação do *endpoint* SetLocation (lado do Keeper Processor)

4.5.2.2. SetLocationEvent

O *endpoint* SetLocationEvent é muito semelhante ao *endpoint* SetLocation, visto anteriormente. A diferença é que este *endpoint* é chamado quando o Keeper muda o estado de um serviço, por exemplo quando aceita um novo serviço, quando chega ao local de recolha, entre outros. Optou-se por separar esta lógica do *endpoint* SetLocation pelo facto de esta comunicar com um tópico diferente, que não é notificado com a mesma frequência, bem como para facilitar o *debug*.

```
{
    "keeper_id": "mneoR3cIOoYoXRHsHSnsbEZJqQJ3",
    "request_id": "oXnBwbjPz3gnLZ8TqDXf",
    "location": {
        "lat": 40.618648,
        "lon": -8.666003
    },
    "timestamp": 1590139971,
    "event": "COLLECTING"
}
```

Figura 70 - Exemplo de pedido ao *endpoint* SetLocationEvent

No caso deste pedido, existem duas diferenças relativamente com o pedido feito no SetLocation. Neste caso, uma vez que se trata de eventos relacionados com o serviço em si o campo *request_id* é neste endpoint obrigatório, o que também acontece com o campo *event*, que indica o evento enviado. Este campo corresponde a um enumerado com os diversos estados possíveis num serviço da LUGGit, sendo validado previamente de forma automática, conforme explicado no capítulo 4.2.2.

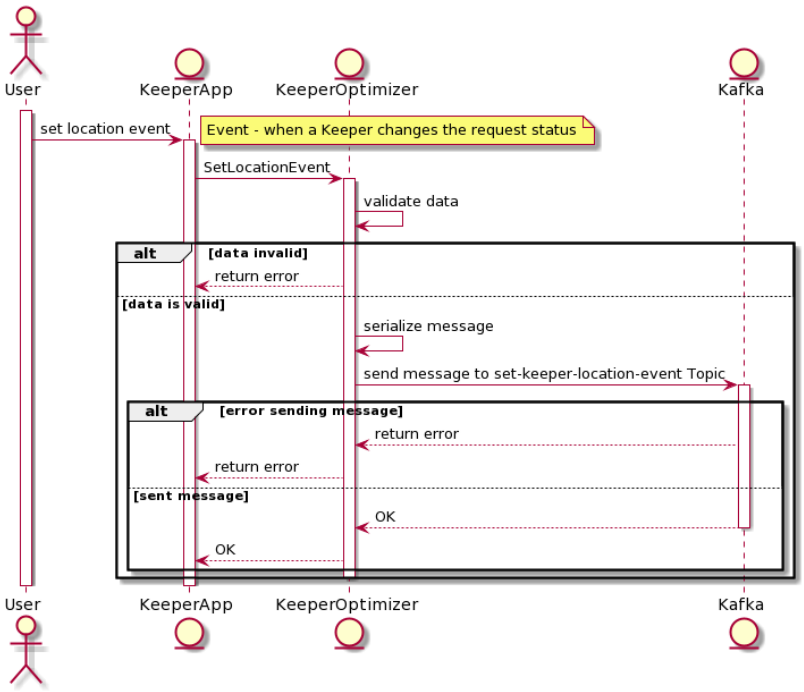

Figura 71 - Diagrama de interação do *endpoint* SetLocationEvent (lado do Keeper Optimizer)

Do lado do Keeper Processor, o procedimento é semelhante ao que acontece com o SetLocation, só que como se recebem mensagens para os diversos eventos que ocorrem durante um serviço da LUGGit, torna-se possível agir de acordo com os mesmos.

No caso do evento *COLLECTING*, que é enviado quando um Keeper aceitou um serviço, sabemos assim a localização da recolha e podemos assim começar a construir um *heatmap* em tempo-real utilizando o H3 para agregar várias localizações numa região. Como já vimos no capítulo 2.2.1.2, o H3 tem várias resoluções com áreas variadas. Neste caso, uma vez que se trata de uma localização proveniente de um serviço, ela é enviada para a construção do *heatmap* na maior resolução possível (resolução 15) de modo a obtermos o máximo de precisão, o que corresponde a 0.9 metros quadrados. Depois, ao processar o *heatmap* podemos ajustar quais as resoluções

pretendidas para visualização, o que veremos adiante. Assim é enviada uma mensagem para o tópico responsável por construir e atualizar o heatmap em tempo-real.

Na Figura 72 podemos ver um exemplo de um pedido no ponto de vista do Keeper, bem como os eventos do serviço na qual estes são responsáveis. Uma vez que os Keepers circulam livremente numa área de operação, poderão receber pedidos a qualquer instante, pressupondo que estão *online*, isto é, disponíveis para o fazer. Num serviço, após o evento *COLLECTING*, que é enviado quando um Keeper aceita um novo serviço, o Keeper dirige-se até junto do cliente para efetuar a recolha, e quando chega faz o *scan* às bagagens (evento *SCANNING*) e dirige-se para o centro logístico, onde elas serão armazenadas (evento *STORED*) dando por terminado o processo de recolha (visível na Figura 72 a rosa), ficando este Keeper livre para novos pedidos de serviço.

Para a realização da entrega, é calculado após o evento *STORED* o tempo que o Keeper demora a dirigir-se até ao destino, é somado esse tempo com uma constante de 15 minutos, de modo ao Keeper ser notificado com bastante tempo de antecedência à hora prevista para realizar o serviço de entrega, ao que este aceita (evento *STORAGE_COLLECTING*), dirigindo-se de novo ao centro logístico, onde faz de novo o *scan* às bagagens (evento *STORAGE_SCANNING*) e dirige-se para o local de entrega indicado pelo cliente (evento *STORAGE_DELIVERING*). Assim que o Keeper chega ao local de entrega indica que de facto já chegou (evento *ARRIVED*), espera pela chegada do cliente e entrega-lhe as bagagens de volta, terminando o processo de entrega (que podemos ver na Figura 72 indicado a cor azul) e o serviço (evento *FINISHED*).

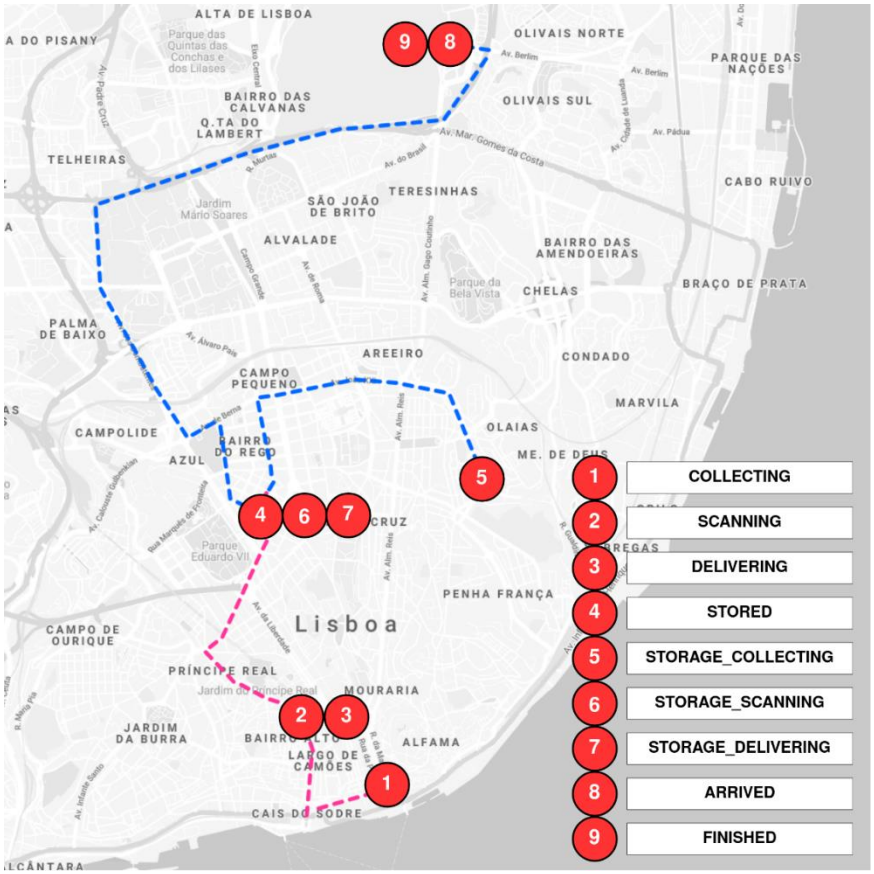

Figura 72 - Eventos com influência dos Keepers numa recolha (a rosa) e entrega (a azul) de um serviço da LUGGit

De seguida os dados do serviço são atualizados na base de dados, onde se regista o Keeper responsável pelo serviço, quanto tempo este demorou a aceitar o serviço, entre outros. O diagrama de interação de processamento poderá ser visto na Figura 73.

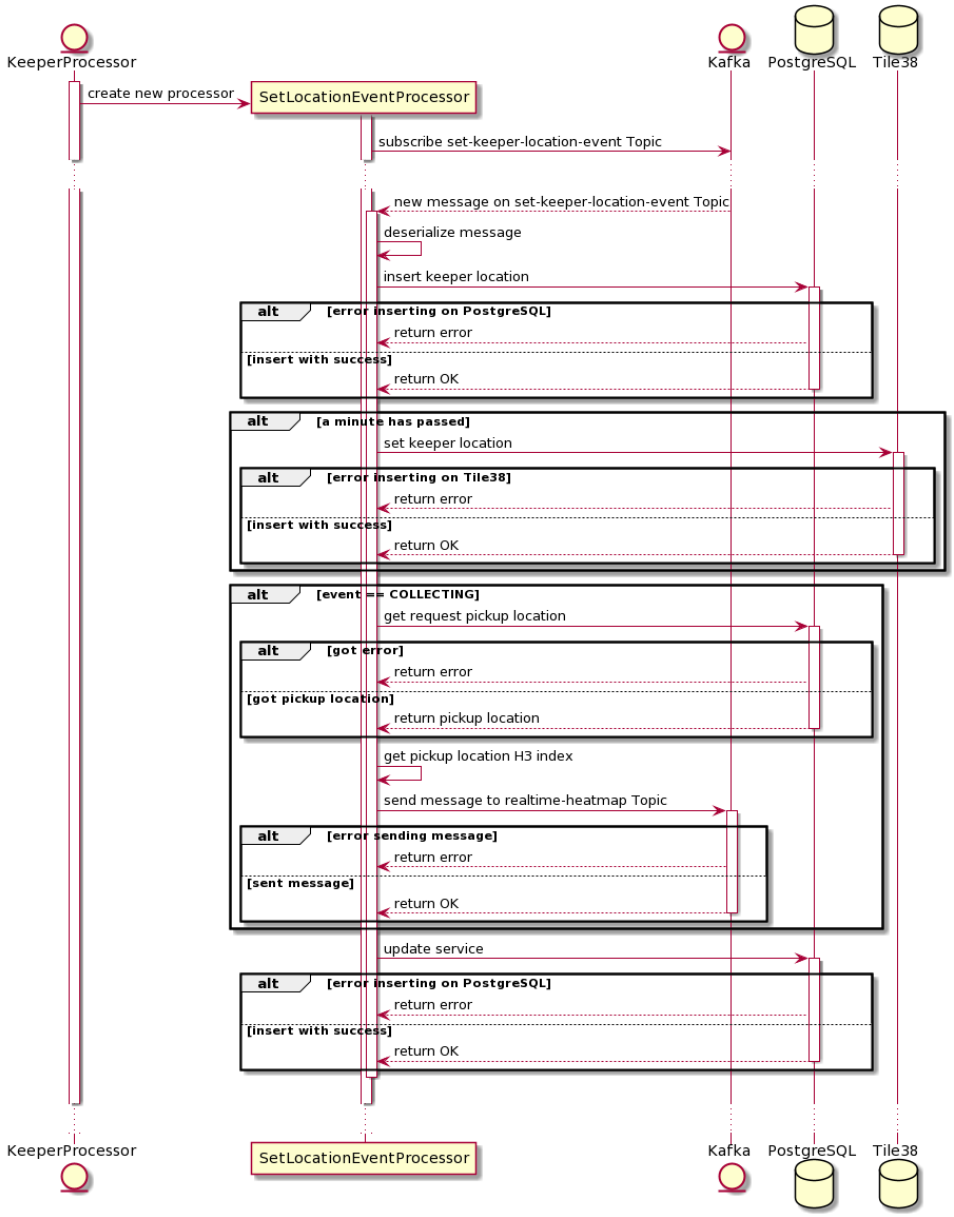

Figura 73 - Diagrama de interação do *endpoint* SetLocationEvent (lado do Keeper Processor)

Ainda no lado do Keeper Processor, e uma vez que foi emitida uma mensagem para o tópico responsável por criar e atualizar o *heatmap* em tempo-real, é necessário processar essa mensagem, enviada pelo SetLocationEvent. O processador do RealtimeHeatmap não só processa as mensagens provenientes desse tópico como também mantém uma Goka *view* que serve de *cache* local, guardando todas as mensagens recebidas na última hora (tempo que se definiu como sendo "tempo-real") e removendo as mensagens expiradas tanto da *group table* no Kafka (que guarda todas as mensagens recebidas pelo *processor group*) como da própria *view*. É através dessa *view* que se constroem os *heatmaps* em tempo-real, pois contém as mensagens da última hora. Quando este

processador é criado, a *view* por ser uma *cache* local encontra-se vazia, sendo que quando esta é criada é feito um pedido ao Kafka para que esta possa ser sincronizada e preenchida com o que existe no momento no Kafka, sendo utilizada uma função de *callback* para esse efeito. Assim constrói-se a *view* com os dados atualizados, sendo que caso estes tenham passado a última hora serão removidos, tal como explicado acima. O processo de funcionamento deste *processor* é mostrado abaixo.

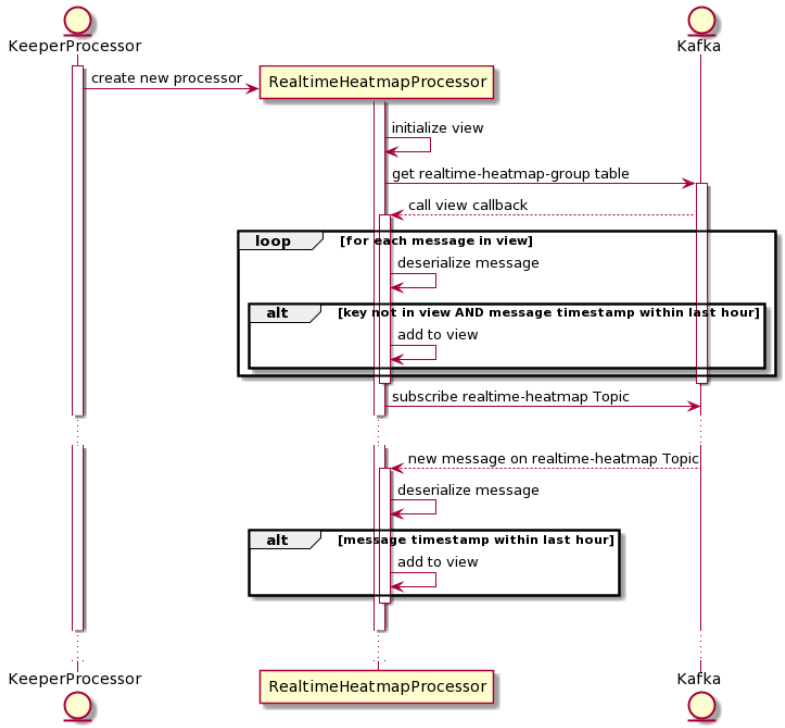

Figura 74 - Diagrama de interação de consumo e processamento do tópico RealtimeHeatmap

4.5.2.3. SetRequestState

O *endpoint* SetRequestState é como o nome indica, responsável por indicar o estado do pedido, quando este não é alterado pelo Keeper, mas sim pelo utilizador, o que acontece quando este cria um novo pedido, quando o confirma através do seu pagamento e quando o cancela. Torna-se necessário ter esta indicação para que as métricas relativas ao serviço como o tempo de confirmação ou de cancelamento do pedido sejam corretamente calculadas.

```
{
    "request_id": "oXnBwbjPz3gnLZ8TqDXf",
    "state": "WAITING"
}
```

Figura 75 - Exemplo de pedido ao *endpoint* SetRequestState

Para efetuar o pedido, é necessário enviar o identificador do pedido (campo *request_id*) bem como o estado deste. Como o ator responsável pela interação é de facto o utilizador, o pedido é feito pela API da LUGGit através das Firebase Cloud Functions.

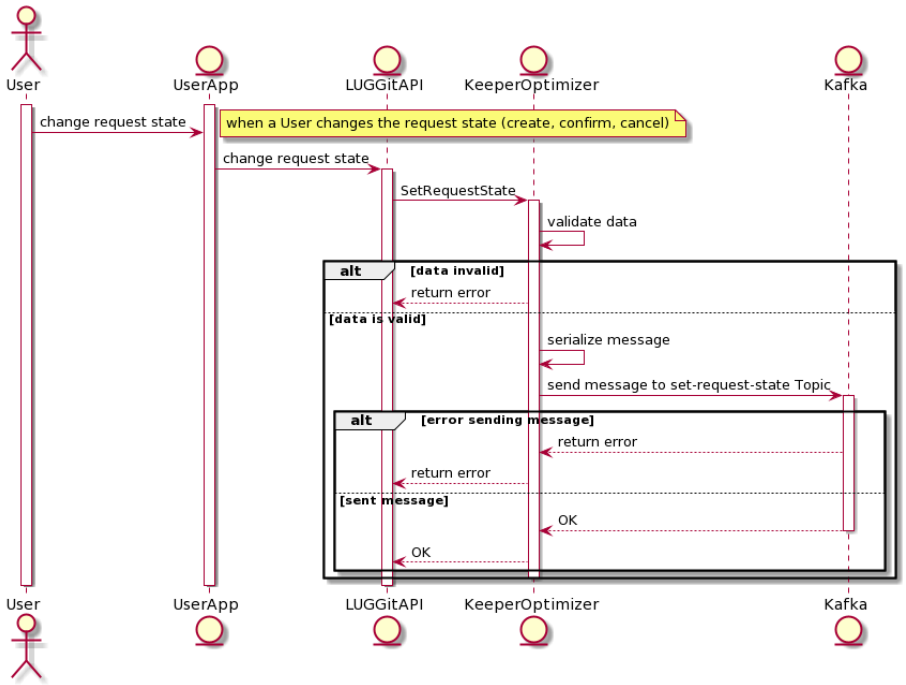

Figura 76 - Diagrama de interação do *endpoint* SetRequestState (lado do Keeper Optimizer)

O processo é semelhante ao já discutido anteriormente nos restantes *endpoints*, sendo que a informação é criada ou atualizada na base de dados quando uma mensagem nova chega ao Keeper Processor, diferenciando-se pelo evento, conforme podemos ver na figura abaixo.

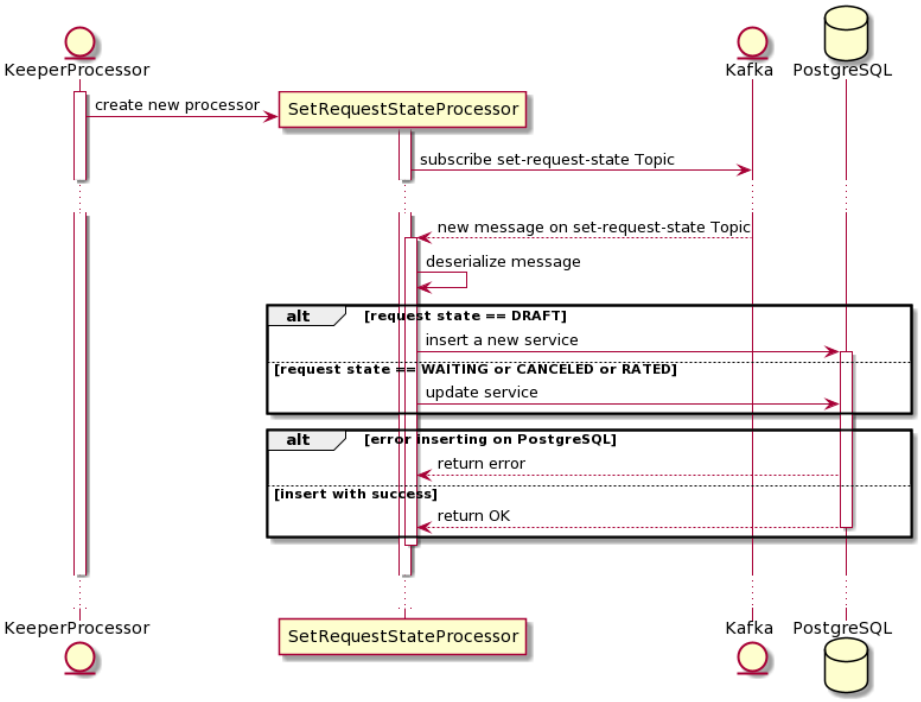

Figura 77 - Diagrama de interação do *endpoint* SetRequestState (lado do Keeper Processor)

4.5.2.4. Heatmaps

Nesta secção será discutida a abordagem de construção e atualização dos heatmaps que este serviço fornece. Como já foi dito, os *heatmaps* são mapas que representam o número de pedidos numa dada região, e é de todo o interesse disponibilizá-los aos Keepers da LUGGit para os auxiliar sugerindo locais tendo em conta duas componentes: a histórica, que se baseia em pedidos feitos no passado, e pedidos em tempo-real, que são úteis para representar pedidos que ocorreram na última hora, como por exemplo em locais onde haja um pico inesperado de pedidos como grandes eventos, entre outros.

Conforme vimos no capítulo 4.5.2.2, os *heatmaps* em tempo-real são criados a partir dos dados recebidos por um tópico Kafka criado especificamente para o efeito (*RealtimeHeatmapTopic*), sendo que os *heatmaps* históricos são criados a partir dos dados históricos guardados armazenados na base de dados. Na Figura 78 podemos ver um diagrama exemplificativo de todo o processo.

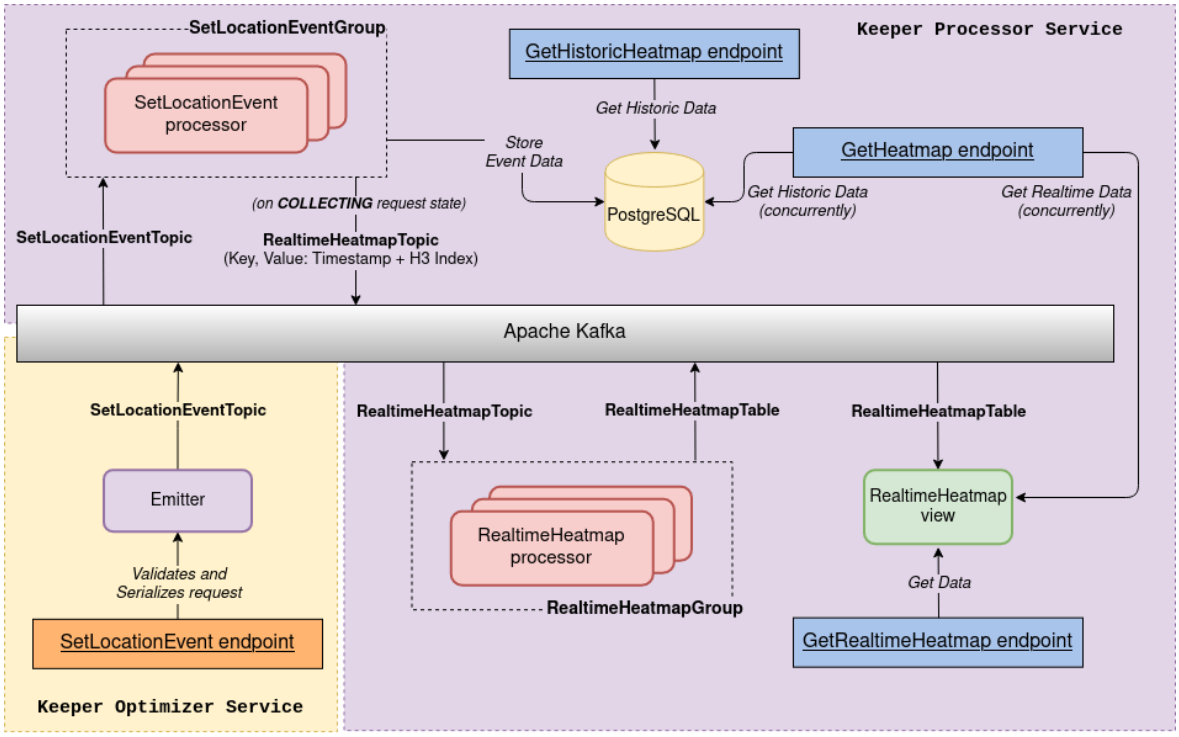

Figura 78 - Vista geral do processamento e fornecimento de *heatmaps*

Assim que o evento *COLLECTING* é enviado, significa que o pedido foi feito pelo cliente e aceite por um Keeper, a localização da recolha é enviada para esse tópico, que por sua vez alimenta uma *group table* (*RealtimeHeatmapTable*) onde constam todas as localizações de recolha da última hora (tempo que se considerou como sendo tempo-real), bem como a base de dados, que é sempre atualizada de todos os estados e atualizações do serviço. Os dados em tempo-real são consultados a partir de uma *view*, uma *cache* local que reúne o estado da *group table*, e estes são verificados a cada minuto removendo eventuais localizações que ultrapassem o tempo máximo considerado como sendo tempo-real. Os dados históricos encontram-se na base de dados, que conforme vimos reúne não só dados relativos ao serviço como outros dados que poderão ser úteis através do uso da *data mart*.

Os *heatmaps* são construídos tendo por base uma resolução H3 máxima e mínima, sendo que a resolução máxima se refere a uma menor área e a resolução mínima ao oposto. Inicialmente, é preenchido todo o polígono referente à área de operação com índices H3 através da função *polyfill* disponibilizada pelo H3, usando a resolução máxima pretendida para o *heatmap*, onde se usou a resolução 9.

H3 Resolution	Average Hexagon Area (km^2)	Average Hexagon Edge Length (km)	Number of unique indexes
0	4,250,546.8477000	1,107.712591000	122
1	607,220.9782429	418.676005500	842
2	86,745.8540347	158.244655800	5,882
3	12,392.2648621	59.810857940	41,162
4	1,770.3235517	22.606379400	288,122
5	252.9033645	8.544408276	2,016,842
6	36.1290521	3.229482772	14,117,882
7	5.1612932	1.220629759	98,825,162
8	0.7373276	0.461354684	691,776,122
9	0.1053325	0.174375668	4,842,432,842
10	0.0150475	0.065907807	33,897,029,882
11	0.0021496	0.024910561	237,279,209,162
12	0.0003071	0.009415526	1,660,954,464,122
13	0.0000439	0.003559893	11,626,681,248,842
14	0.0000063	0.001348575	81,386,768,741,882
15	0.0000009	0.000509713	569,707,381,193,162

Figura 79 - Tabela de resolução do H3 [39]

De seguida são contados os números de pedidos feitos num índice H3, começando pela resolução mínima pretendida a mostrar no *heatmap*, pelo que caso o número de pedidos seja maior que um limite definido de pedidos para um único índice (aqui considerou-se o número 5), este índice é subdividido para a resolução imediatamente acima, e feita a mesma verificação, sucessivamente até à resolução máxima definida, onde se usou a resolução 7.

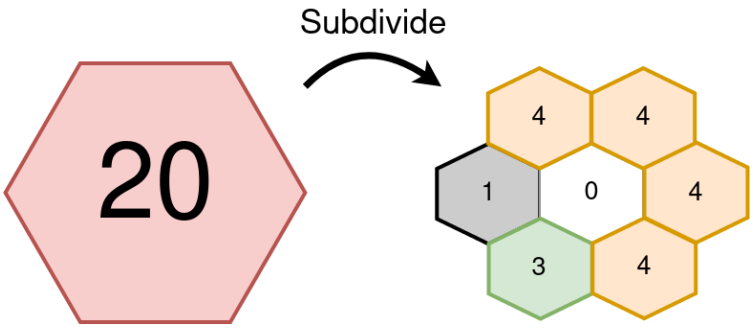

Figura 80 - Subdivisão de hexágonos de acordo com o número de pedidos

O H3 também fornece uma função chamada *h3ToGeoBoundary*, que devolve as fronteiras de um dado índice, o que origina o hexágono. Uma vez que no passo anterior já sabemos qual o número e os índices H3 que irão compor o *heatmap* final já com as subdivisões, cada um dos hexágonos é

tratado como um polígono, que por sua vez é inserido em conjunto como sendo uma *Feature* para construir um objeto GeoJSON que representa todo o *heatmap*, sendo inserido em cada *Feature* uma cor representativa de acordo com o número de pedidos. Na Figura 81 é possível ver zonas com uma cor laranja mais intensa, que significam zonas com mais pedidos e outras com cor menos intensa significando zonas com menos pedidos, bem como zonas a cinzento que significam a inexistência de pedidos.

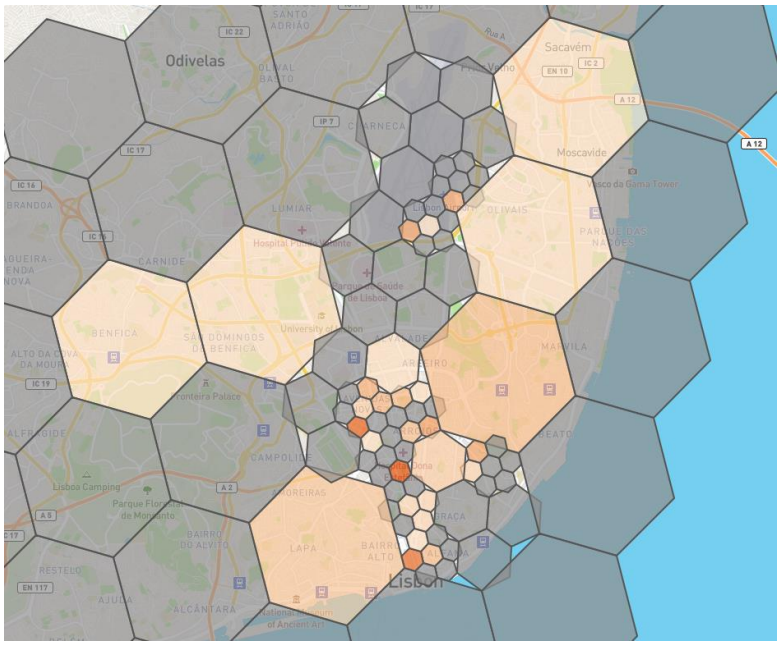

Figura 81 - Exemplo de *heatmap* na cidade de Lisboa

Foram assim criados três *endpoints* distintos no Keeper Processor: o *GetRealtimeHeatmap*, que fornece heatmaps em tempo-real; o *GetHistoricHeatmap*, que fornece heatmaps históricos através dos dados armazenados na base de dados; e o *GetHeatmap*, que combina dados das duas componentes.

4.5.2.4.1. GetRealtimeHeatmap

O *endpoint* GetRealtimeHeatmap é responsável por fornecer um *heatmap* para uma cidade com dados em tempo-real. Uma vez que os dados poderão ser atualizados em tempo-real, este *endpoint* responde com um *stream* de dados unidirecional no lado do servidor, o que é possível através do gRPC. Desta forma garante-se que qualquer cliente integrado seja capaz de receber atualizações a partir do *stream*, recebendo todas e quaisquer alterações relativas ao *heatmap*.

```
message GetRealtimeHeatmapRequest {
  oneof type {
    string city                          = 1 [(validate.rules).string.min_len = 1];
    luggit.common.geo.Point location     = 2 [(validate.rules).message.required = true];
  }
}
```

Figura 82 - Pedido do *endpoint* GetRealtimeHeatmap

Para este pedido, utilizou-se uma das funcionalidades dos Protocol Buffers, o *oneof* [117], que permite oferecer várias possibilidades para campos de entrada. No caso deste pedido é utilizado o *oneof* para escolhermos entre o nome da cidade ou uma localização, campo que depois é verificado na execução do *endpoint*.

```
// check which city to get heatmap
var city string
switch req.GetType().(type) {
case *pb.GetRealtimeHeatmapRequest_City:
    log.Infof("GetRealtimeHeatmap - getting cities from Geofencing Service")
    cts, err := ks.geofencing.GetCities(ctx, &pb_geo.GetCitiesRequest{})
    if err != nil {
        log.Errorf("GetRealtimeHeatmap(req=%+v) - error while getting cities from GeofencingService: %v", req, err)
        return err
    }
    log.Infof("GetRealtimeHeatmap - got cities from Geofencing Service sucessfully")
    for _, c := range cts.GetCities() {
        if req.GetCity() == c.GetSlug() {
            city = req.GetCity()
            break
        }
    }
case *pb.GetRealtimeHeatmapRequest_Location:
    log.Infof("GetRealtimeHeatmap - checking availability from Geofencing Service")
    ca, err := ks.geofencing.CheckAvailability(ctx, &pb_geo.CheckAvailabilityRequest{
        Location: req.GetLocation(),
    })
    if err != nil {
        log.Errorf("GetRealtimeHeatmap(req=%+v) - error while checking availability from GeofencingService: %v", req, err)
        return err
    }
    log.Infof("GetRealtimeHeatmap - checked availability from Geofencing Service sucessfully")
    if !ca.Available {
        log.Errorf("GetRealtimeHeatmap(req=%+v) - location entered is outside our operation areas: %v", req, errNotInOperationArea)
        return errNotInOperationArea
    }
    city = ca.Slug
}
```

Figura 83 - Verificação de campos no *endpoint* GetRealtimeHeatmap utilizando o *oneof* dos Protocol Buffers

Assim, torna-se possível o pedido ter uma localização ou um nome de uma cidade, sendo depois utilizado o serviço de Geofencing para averiguar a existência tanto da cidade (4.4.2.2) como da localização enviada (4.4.2.1), conforme seja o pedido feito.

Figura 84 - Exemplo de pedido ao *endpoint* GetRealtimeHeatmap

A resposta é dada através de um *stream* unidirecional do lado do servidor, em que a resposta nos mostra um *heatmap* no formato GeoJSON, bem como é indicado o momento da última atualização e a cidade. Na Figura 85 podemos ver um exemplo de uma resposta, em que foram removidas algumas linhas do campo *heatmap* para a figura não ser demasiado grande.

Figura 85 - Exemplo de resposta ao *endpoint* GetRealtimeHeatmap

Ao termos um *stream* de dados unidirecional do lado do servidor, cabe ao cliente indicar-lhe a intenção de terminar a escuta por novas mensagens. A interação deste *endpoint* é mostrada na figura abaixo.

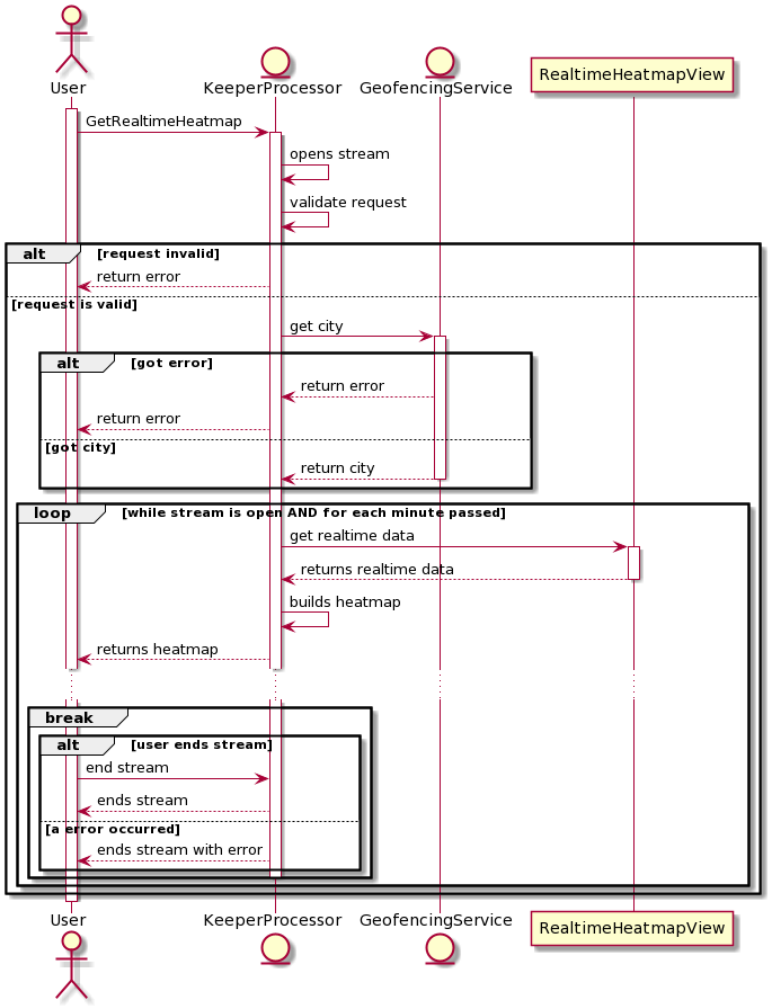

Figura 86 - Diagrama de interação do *endpoint* GetRealtimeHeatmap

4.5.2.4.2. GetHistoricHeatmap

Os *heatmaps* da componente histórica são disponibilizados através do *endpoint* GetHistoricHeatmap, onde é feita uma consulta à base de dados relativamente ao histórico de pedidos efetuados num período de tempo definido pelo utilizador. Neste sentido, usou-se o mesmo conceito do *endpoint* GetRealtimeHeatmap relativamente à localização, utilizando o *oneof*. Também foi usado o *oneof* para referenciar o período temporal pretendido para os *heatmaps*, sendo que foi disponibilizado um enumerado com espaços temporais predefinidos:

- Hoje – *TODAY*
- Ontem – *YESTERDAY*
- Semana atual - *THIS_WEEK*

- Semana passada - *LAST_WEEK*
- Mês atual - *THIS_MONTH*
- Mês passado - *LAST_MONTH*
- Trimestre atual - *THIS_QUARTER*
- Trimestre passado - *LAST_QUARTER*
- Ano atual - *THIS_YEAR*
- Ano passado - *LAST_YEAR*

Para além de se poder referenciar um dos períodos temporais predefinidos é também possível a especificação de um intervalo de datas. A resposta a este pedido, quando bem-sucedida, tem o mesmo formato que a mostrada na Figura 85.

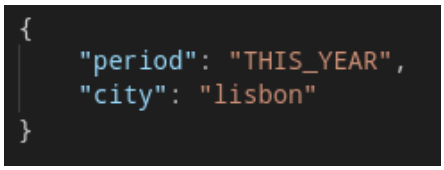

Figura 87 - Exemplo de pedido ao *endpoint* GetHistoricHeatmap

O diagrama de interação deste *endpoint* encontra-se abaixo:

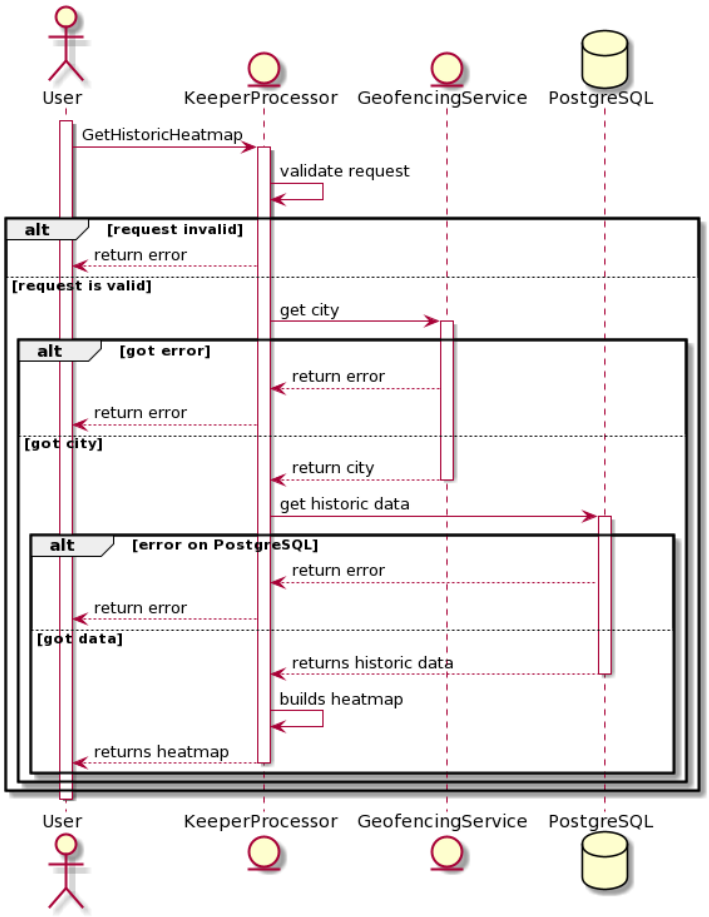

Figura 88 - Diagrama de interação do *endpoint* GetHistoricHeatmap

4.5.2.4.3. GetHeatmap

O *endpoint* GetHeatmap foi criado para se fornecer um *heatmap* que pudesse de algum modo integrar tanto a componente histórica, importante para perceber locais onde por norma existe maior possibilidade de haver pedidos, como a componente em tempo-real, que permite obter atualizações frequentes do estado atual de uma área de operação. A cada uma das componentes é dado um peso em percentagem, ao que se escolheu fornecer por defeito um peso de 60% à componente histórica e 40% à componente de tempo-real, sendo que o valor de pedidos dado a cada índice H3 consiste no cálculo desse peso.

Figura 89 - Exemplo de *heatmap* histórico (à esquerda), *heatmap* em tempo-real (ao meio), combinação de ambos os *heatmaps* (à direita)

Tanto o pedido como a resposta a este *endpoint* são semelhantes ao GetRealtimeHeatmap. A diferença neste caso é que o *endpoint* funciona através de um *stream* bidirecional entre o cliente e o servidor, sendo que a qualquer momento o Keeper poderá enviar dados para o servidor referente à sua localização à medida que este esteja a deslocar-se. Já do lado do servidor como já referido no caso do *heatmap* em tempo-real, os dados são enviados para o cliente frequentemente para que este tenha o *heatmap* o mais atualizado possível.

Neste contexto o uso de *goroutines* e de *channels* disponibilizados pela linguagem Go torna-se imperativo. Na realização deste pedido, os dados referentes aos dois tipos de *heatmaps* são pedidos em simultâneo por duas *goroutines*, sendo que existe uma terceira *goroutine* que aguarda pelas restantes, junta os dois resultados das tarefas, constrói o *heatmap* resultante e envia-o para o cliente. De referir que os pesos dados a cada uma das componentes são estáticos mas configuráveis, sendo possível modificá-los a qualquer instante, caso se queira dar uma maior importância a uma das componentes num determinado contexto. Abaixo podemos ver como a interação é feita neste *endpoint*.

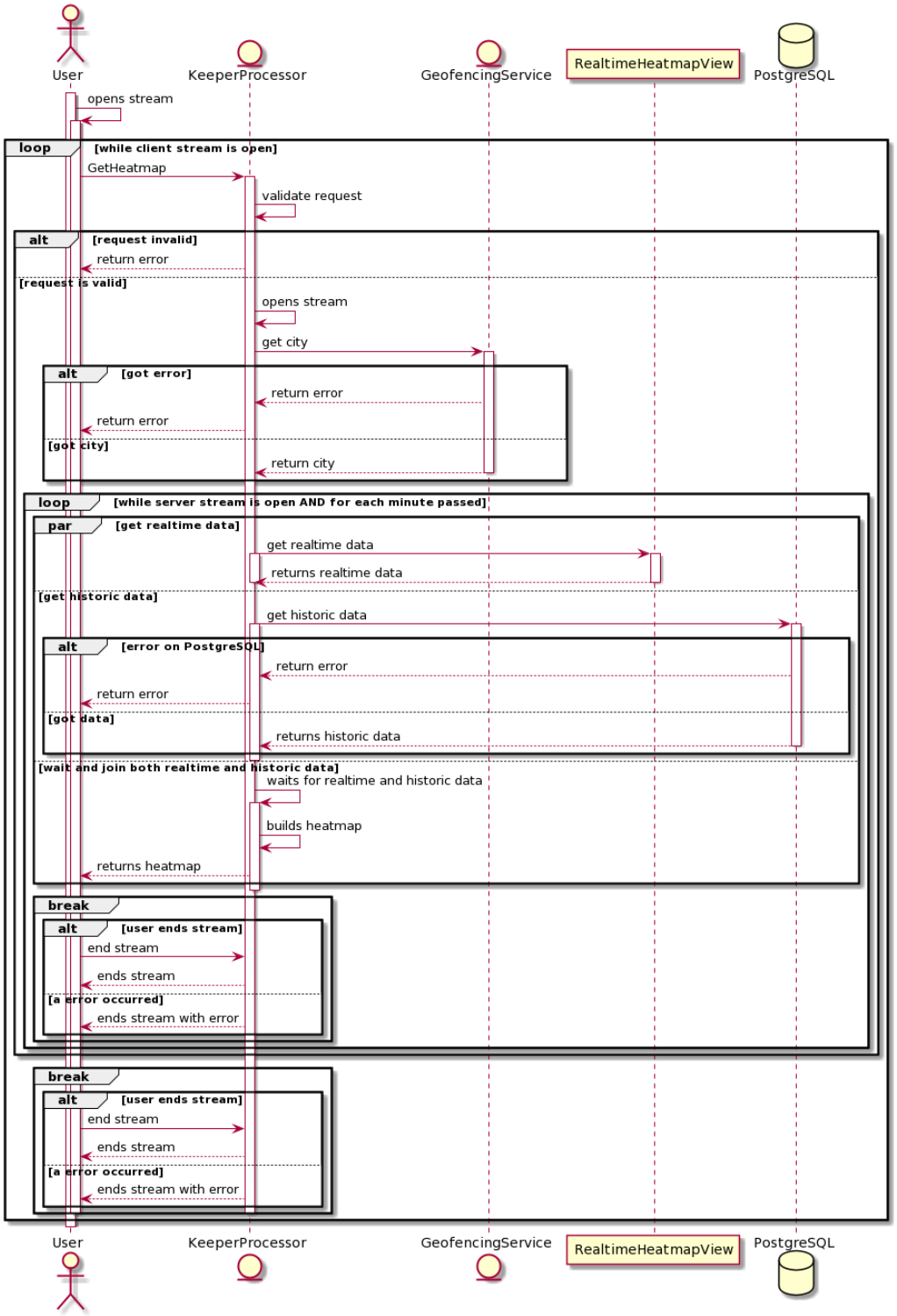

Figura 90 - Diagrama de interação do *endpoint* GetHeatmap

4.5.2.5. GetNearKeepers

À medida que este projeto foi sendo realizado e ao terem-se aprofundado conhecimentos nas áreas e tecnologias abordadas acima, chegou-se à conclusão que era possível criar um melhor sistema de atribuição de Keepers a um novo pedido de serviço comparando com o sistema existente até à data na empresa, e desta forma surgiu o *endpoint* GetNearKeepers, que é responsável por fornecer os Keepers mais próximos de uma dada localização, permitindo efetuar a seleção dos Keepers a realizar um novo serviço. A localização atual dos Keepers é conhecida através do envio frequente da localização à medida que estes usam a sua aplicação, sendo que a base de dados PostgreSQL é atualizada tanto da sua localização, bem como do seu estado *online*/*offline*. Ao ser realizado um novo pedido de serviço da LUGGit, é enviada para a rede de Keepers existente um alerta de um novo pedido, para que estes o aceitem. No início do desenvolvimento do produto, a LUGGit enviava esse mesmo pedido para todos os Keepers, e o pedido era escolhido através de uma abordagem *first come, first served* independentemente do seu estado e da sua localização. Este *endpoint* torna-se assim fulcral para indicar quais os Keepers que se encontram melhor posicionados para efetuar os serviços, assim como seguir uma abordagem de notificação sequencial.

Propõem-se para este *endpoint* duas versões distintas: uma utilizando o H3 para a procura de vizinhos diretos a partir da localização dos Keepers e da localização de recolha de um serviço, e uma outra onde se sugere o uso de regiões de procura definidas por *isochrones*. Para ambas as versões, o pedido é o mesmo: a localização do local de recolha, conforme mostrado na Figura 84. As respostas variam de acordo com a versão, o que será mostrado adiante.

4.5.2.5.1. Versão 1

Para a primeira versão, propôs-se o seguinte método: obter uma lista de Keepers situados em níveis de proximidade dados pelo H3, o que pode ser visto na Figura 91, sendo que o índice inicial a analisar é dado pela localização de recolha de um serviço. Um índice do H3 é capaz de agregar várias localizações e apresenta funcionalidades hierárquicas utilizando a função *hexRing* para fornecer os índices vizinhos, em anel. Deste modo, o índice inicial da procura (Nível 1) corresponde ao índice da localização de recolha do pedido. O Nível 2 é obtido através da execução da função *hexRing* para a vizinhança imediatamente acima, e o Nível 3 contém os vizinhos seguintes. Para cada um destes níveis obtidos são verificados os Keepers cujas localizações estão inseridas no mesmo, bem como o

seu estado (*online*/*offline*). Criou-se também um Nível 4, que é composto por todos os Keepers situados na área de operação do pedido que não tenham sido selecionados nos níveis anteriores.

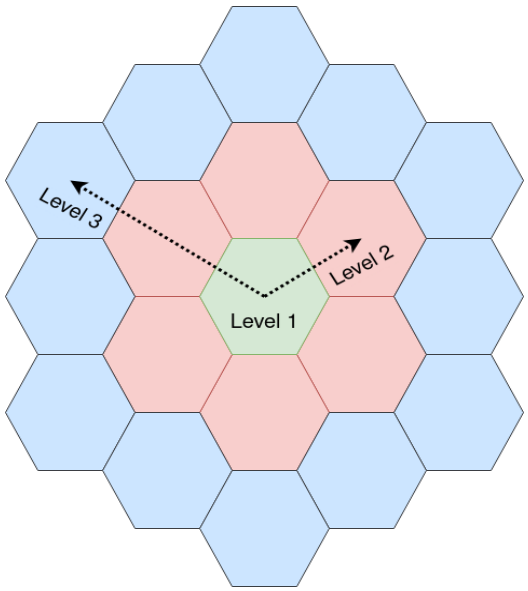

Figura 91 - Níveis de procura de Keepers no *endpoint* GetNearKeepers (Versão 1) utilizando o H3

Na resposta, são indicados para cada nível os Keepers nele encontrados, e para cada Keeper encontrado é mostrado o seu identificador, se este se encontra *online* e quando foi a última vez que este esteve *online*, este último um dado importante que permite perceber se o Keeper ficou inativo recentemente (o que poderá indicar que a localização onde se encontra é relativamente perto da localização anteriormente enviada).


```
{
  "levels": [
    {
      "keepers": [
        {
          "keeper_id": "BrGzzNK42qUyyuwaYwi1FMA3YZd2",
          "online": false,
          "timestamp": {
            "seconds": "1590171919",
            "nanos": 836000000
          }
        }
      ],
      "level": 4
    }
  ]
}
```


Figura 92 - Exemplo de resposta ao *endpoint* GetNearKeepers (Versão 1)

Assim, o diagrama de interação é o seguinte:

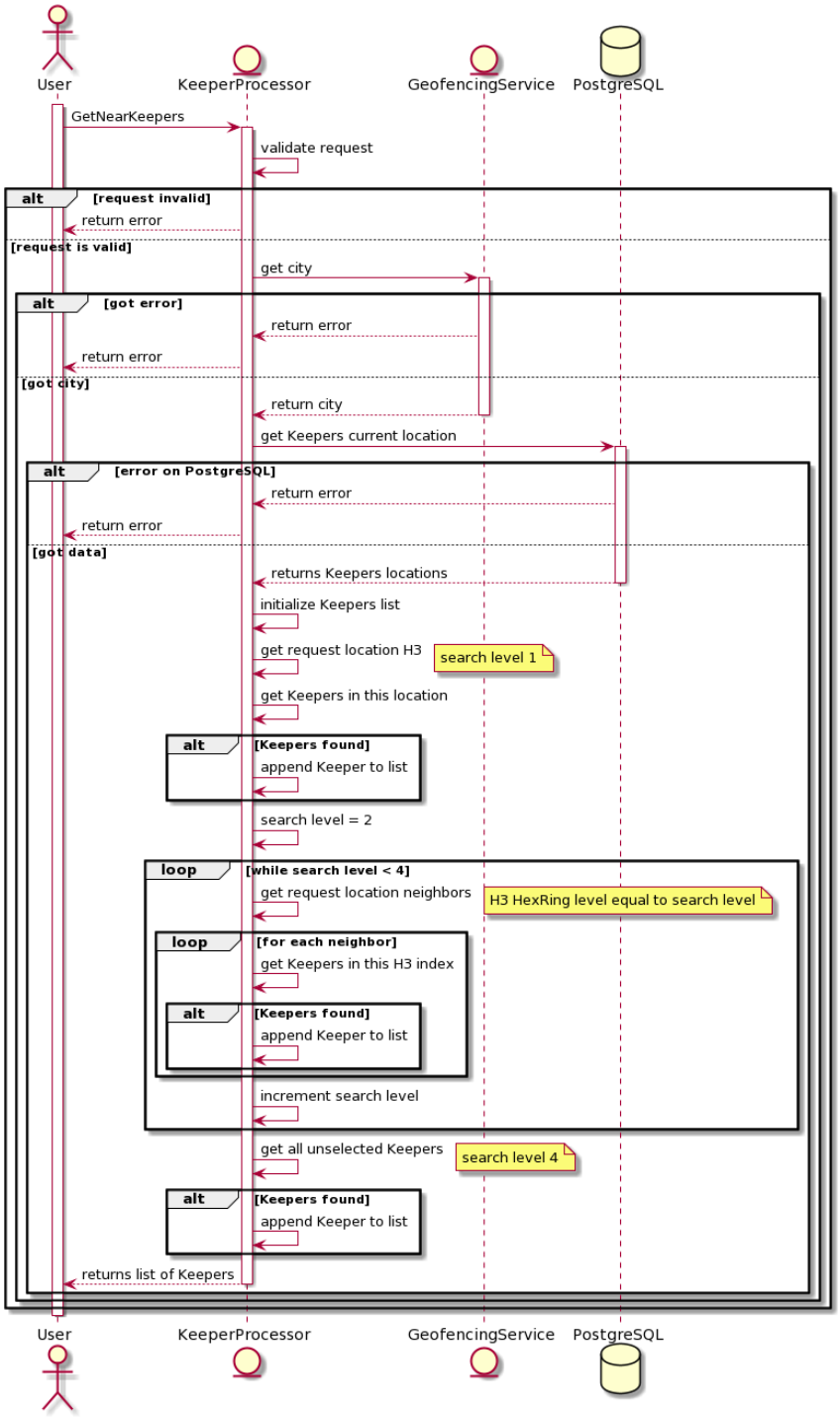

Figura 93 - Diagrama de interação do *endpoint* GetNearKeepers (Versão 1)

4.5.2.5.2. Versão 2

As *isochrones* fornecidas pelo serviço de Routing fornecem regiões que representam a distância máxima a que se pode chegar até a um certo limite temporal. Estas regiões têm assim uma particularidade que a Versão 1 deste *endpoint* não tem, pois estas são calculadas pelo Valhalla tendo em conta o conhecimento que este tem relativamente às estradas e aos possíveis trajetos a efetuar numa área de operação. O que aqui se propõe é utilizar os polígonos obtidos das *isochrones* como os níveis de procura de Keepers, uma vez que neste caso se sabe o tempo que estes demoram a chegar à localização pretendida - o local de recolha de novos pedidos.

Para tal, utiliza-se a função *polyfill* do H3 para preencher cada um dos polígonos obtidos com índices H3, e efetua-se a procura da mesma forma que na Versão 1, obtendo para cada nível de procura os Keepers nele situados. Conforme vimos no capítulo 4.3.2.5, as *isochrones* apenas permitem o cálculo de 4 períodos de tempo distintos, pelo que teremos neste caso 5 níveis de procura, sendo os primeiros 4 correspondentes às regiões calculadas pelas *isochrones* e o último nível de procura correspondente aos Keepers restantes na área de operação, tal como acontece na Versão 1. Na Figura 94 podemos ver um exemplo de uma *isochrone* calculada a partir do Miradouro de Santa Luzia em Alfama para 10, 15, 20 e 30 minutos, sendo que na figura correspondem aos polígonos de cor verde, amarelo, laranja e vermelho respetivamente.

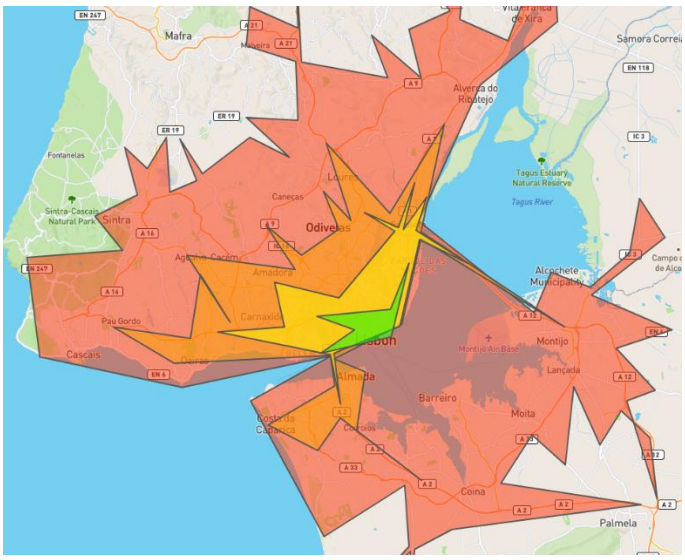

Figura 94 - *Isochrone* calculada pelo serviço de Routing a partir de Alfama (Lisboa)

Na resposta, tal como acontece com a Versão 1, é indicado para cada nível de procura os Keepers encontrados e para cada um deles é indicado se este se encontra *online* bem como o seu identificador. Neste caso, uma vez que se sabe à partida quanto tempo cada um deles demora a partir do cálculo das *isochrones* é também enviado o tempo expectável de chegada.

```
{
  "levels": [
    {
      "keepers": [
        {
          "keeper_id": "mneoR3cIOoYoXRHsHSnsbEZJqQJ3",
          "online": false
        },
        {
          "keeper_id": "BrGzzNK42qUyyuwaYwilFMA3YZd2",
          "online": false
        },
        {
          "keeper_id": "fidtPtxYtFdvi6dAEgmlBVKPTx03",
          "online": false
        }
      ],
      "level": 2,
      "time_expected": 15
    }
  ]
}
```


Figura 95 - Exemplo de resposta ao *endpoint* GetNearKeepers (Versão 2)

O diagrama de interação deste *endpoint* encontra-se abaixo:

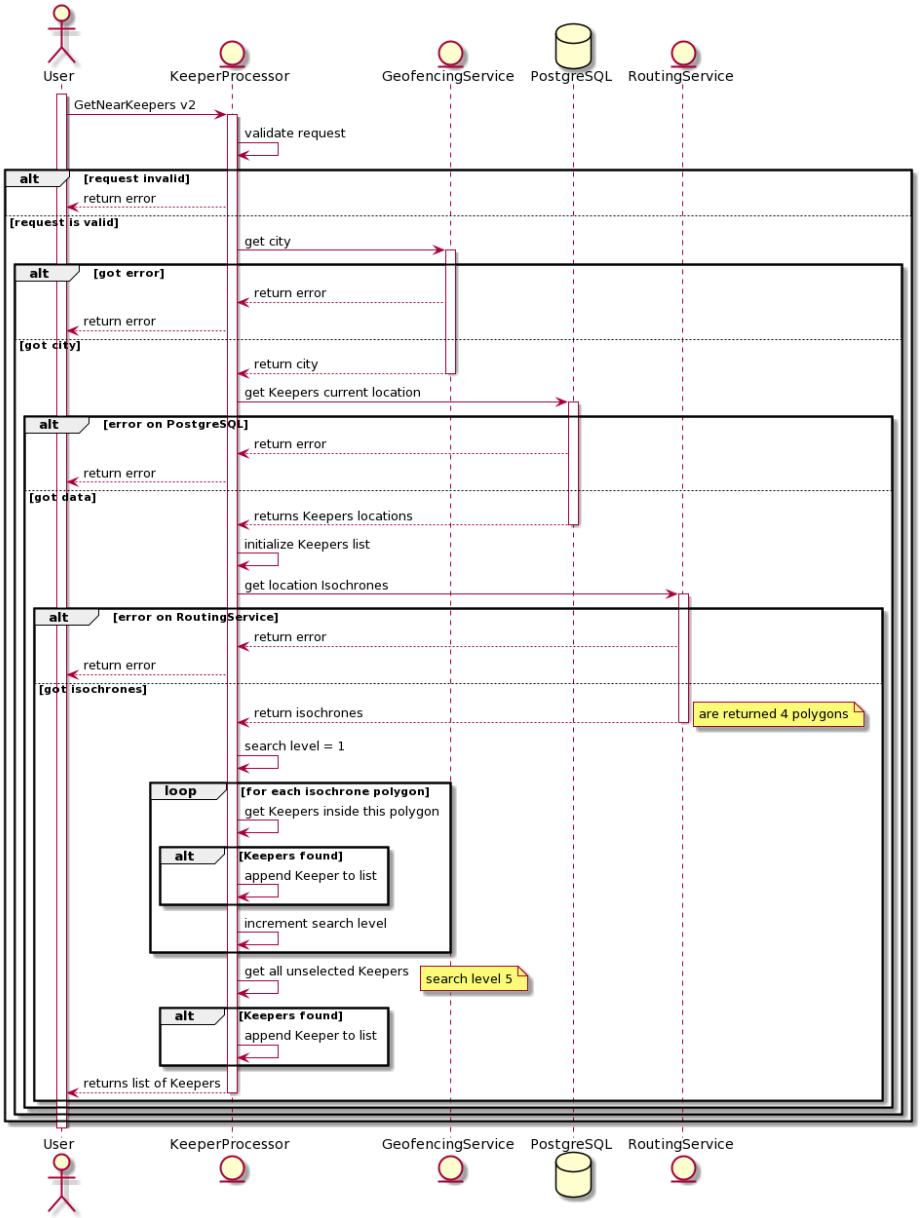

Figura 96 - Diagrama de interação do *endpoint* GetNearKeepers (Versão 2)

4.6. Previsão da Duração de Viagem entre dois locais

A duração de uma viagem entre dois locais é um dos fatores importantes a analisar no âmbito desta dissertação, ao propor-se um método de seleção do melhor Keeper a efetuar um novo serviço. Conforme mostrado no capítulo 2.3.1.2, o Valhalla já nos indica a duração de uma viagem, contudo os seus resultados não são afetados por fatores como o estado do tempo, o trânsito, entre outros, tornando-se impraticável num contexto real de utilização saber quanto tempo cada Keeper demora a dirigir-se até a um determinado local, através de pedidos ao Valhalla. Neste sentido, foram

utilizados dados reais de condução obtidos na cidade do Porto para treinar um modelo cujo objetivo é prever as durações de próximas viagens tendo como base este tipo de dados, e assim validar-se o método de seleção de Keepers utilizando o H3 como ferramenta de seleção.

Os dados reais de condução são provenientes de 442 taxistas em circulação na cidade do Porto, que permitiram desenvolver um modelo de aprendizagem automática utilizando o método de regressão Random Forest, executado através da ferramenta gratuita Google Colab. Para tal é necessário carregar e tratar os dados, procedendo ao seu pré-processamento de modo a poder-se obter o tempo médio de viagem entre duas localizações, o que torna possível poder treinar o modelo. Para a validação do modelo, este será testado e comparado diretamente com os resultados de duração de viagem obtidos pelo Valhalla, o que nos permitirá determinar qual a diferença que os dados reais representam comparando aos dados estáticos fornecidos pelo OSM para a construção das *tiles* do Valhalla. Finalmente, para se validar o H3 como uma ferramenta de seleção de Keepers neste contexto será também feita uma análise com os mesmos dados utilizando os níveis de procura do método de seleção do capítulo 4.5.2.5.1, o que permitirá verificar se a utilização deste método é válida para selecionar um Keeper que possa responder a um novo serviço em menos de 15 minutos.

4.6.1. Tecnologias Usadas

O Google Colab permite a execução de um ambiente interativo Python em servidores alojados pela Google utilizando um documento denominado de *Colab notebook*[111], baseado no projeto Jupyter Notebook[112] que permite editar, e executar instruções de código bem como visualizar imagens, entre outros.

Para o tratamento e manipulação dos dados foi utilizada a biblioteca pandas, que permitiu carregar os dados através de um ficheiro CSV, bem como aplicar funções a linhas e colunas.

Uma vez que se pretende a criação de um modelo de *machine learning* foi usada a biblioteca scikit-learn, que reúne funcionalidades e funções que permitem realizar o treino, teste e validação de modelos de aprendizagem automática.

[111] https://colab.research.google.com/notebooks/intro.ipynb
[112] https://jupyter.org/

4.6.2. Desenvolvimento do Modelo de *Machine Learning*

A fonte de dados utilizada para o desenvolvimento deste modelo fornece dois conjuntos de dados distintos no formato CSV: um referente ao período entre Julho de 2013 até Junho de 2014, sendo indicado para o treino do modelo; e um outro que consiste em dados entre Julho e Dezembro de 2014, sendo este indicado para teste. Pretende-se prever a duração de viagem entre dois locais, sendo que o desenvolvimento deste modelo foi baseado numa solução a um problema semelhante, cujo objetivo era prever o destino de uma viagem[113].

O primeiro passo foi carregar os dados utilizando o pandas, de modo a estes poderem ser manipulados por esta mesma ferramenta, através de um *data frame*, um conceito muito semelhante às tabelas de uma base de dados. De facto, o pandas permite-nos depois aplicar funções personalizadas a um *data frame*, seja nas suas linhas como nas colunas, inserir ou remover campos, entre outros.

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE	MISSING_DATA	POLYLINE
1099635	1393303842620000239	B	NaN	36.0	20000239	1393303842	A	False	[[-8.649657, 41.154201], [-8.65026, 41.154183]...
110336	1374648340620000051	C	NaN	NaN	20000051	1374648340	A	False	[[-8.613648, 41.141385], [-8.61363, 41.141457]...
382296	1379983771620000649	A	35587.0	NaN	20000649	1379983771	A	False	[[-8.572428, 41.180121], [-8.574759, 41.180598...
1387248	1398764552620000030	B	NaN	33.0	20000030	1398764552	A	False	[[-8.600013, 41.182704], [-8.600229, 41.182731...
87994	1374228831620000633	B	NaN	27.0	20000633	1374228831	A	False	[[-8.608806, 41.147739], [-8.608644, 41.147721...

Figura 97 - *Data frame* original após carregamento do conjunto de dados de treino

Após os dados terem sido carregados, e uma vez que se trata de 1 milhão e 700 mil linhas únicas de dados, passou-se a uma seleção dos mesmos, por intermédio de uma amostragem aleatória para reduzir o número de dados a analisar. Na Figura 97 podemos ver um exemplo dos dados após carregamento e amostragem, onde se considerou 40% do conjunto de dados original. Os campos, que após terem sido carregados correspondem a colunas no *data frame* referem-se por ordem ao número da linha, ao identificador da viagem (*TRIP_ID*), ao tipo de chamada (*CALL_TYPE*) (A - pedido enviado através da central; B - pedido direto ao taxista; C - outros casos), os identificadores do telefone (*ORIGIN_CALL*) e da praça de táxis onde o pedido de viagem foi feito (*ORIGIN_STAND*), o identificador do táxi (*TAXI_ID*), a marca temporal que indica o início do pedido (*TIMESTAMP*), o tipo de dia onde ocorreu a viagem (*DAY_TYPE*) (A - dia normal; B - feriado; C - véspera de feriado), uma indicação de que os dados estão incompletos (*MISSING_DATA*), e um

[113] https://www.kaggle.com/urayukitaka/prediction-taxi-trajectory

conjunto de localizações obtidas durante uma viagem sendo que cada uma foi enviada com 15 segundos de distância entre elas (*POLYLINE*).

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 684268 entries, 1099635 to 436407
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   TRIP_ID       684268 non-null  int64
 1   CALL_TYPE     684268 non-null  object
 2   ORIGIN_CALL   145598 non-null  float64
 3   ORIGIN_STAND  322538 non-null  float64
 4   TAXI_ID       684268 non-null  int64
 5   TIMESTAMP     684268 non-null  int64
 6   DAY_TYPE      684268 non-null  object
 7   MISSING_DATA  684268 non-null  bool
 8   POLYLINE      684268 non-null  object
dtypes: bool(1), float64(2), int64(3), object(3)
memory usage: 47.6+ MB
```

Figura 98 - Campos originais após carregamento do ficheiro CSV

De seguida preparou-se os dados, começando por filtrar todos os dados onde a coluna *MISSING_DATA* fosse verdadeira, o que indica que existem dados incompletos ou em falta. Também foram filtrados dados onde o campo *POLYLINE* estivesse vazio. Podemos ver desde já que existem colunas que não são úteis para o desenvolvimento do modelo. Por exemplo, as colunas *TRIP_ID*, *CALL_TYPE*, *ORIGIN_CALL*, *ORIGIN_STAND*, *TAXI_ID* e *MISSING_DATA* são referentes ao contexto das viagens de táxi. Depois, foi extraída a localização inicial e final a partir da coluna *POLYLINE*, e obtida a duração total da viagem multiplicando o número de localizações recebidas por 15 segundos, sendo a duração o que se pretende prever através deste modelo.

Foram também criados novos campos derivados do momento em que o pedido foi feito, através da coluna *TIMESTAMP* como o dia, o mês, o ano, a hora, os minutos e o dia da semana, onde podemos visualizar a distribuição de dados através da Figura 99. Podemos desde já afirmar num primeiro ponto de vista que a grande maioria dos pedidos são feitos entre as 8 e as 17 horas, e é na sexta-feira o dia em que ocorrem maior número de pedidos.

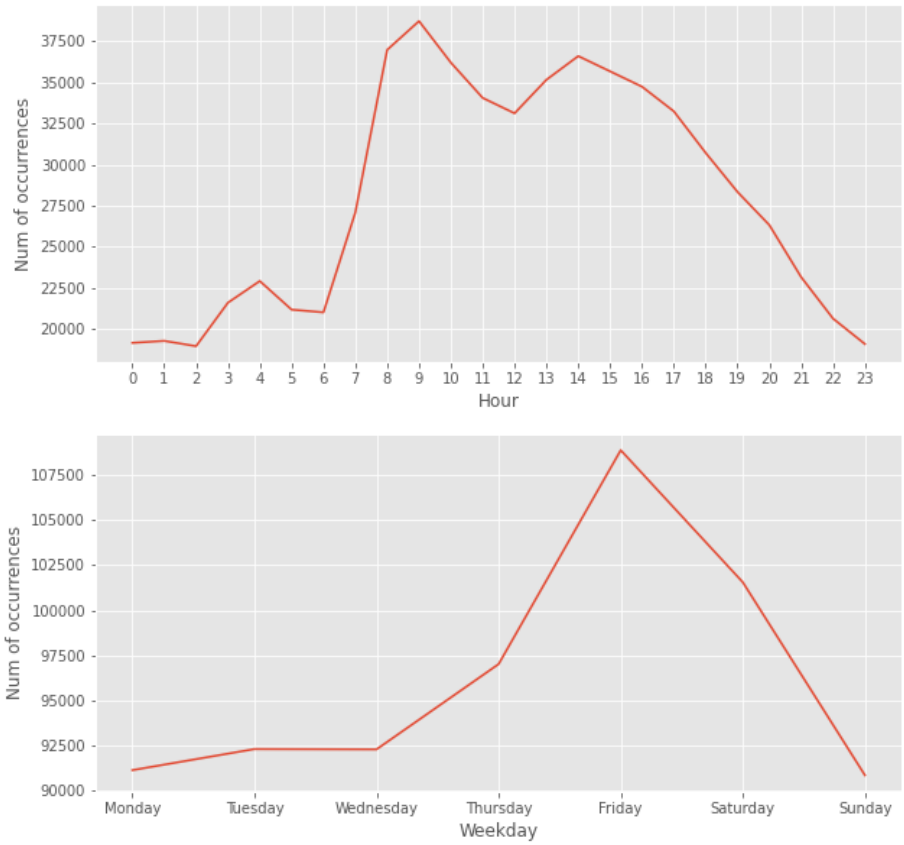

Figura 99 - Distribuição de horas e dia da semana nos dados obtidos

De seguida, foi feita uma filtragem de modo a garantir que as localizações de origem e de chegada estejam dentro da área de operação da LUGGit no Porto, e para tal utilizou-se o H3, tanto para a filtragem como para visualização dos dados no mapa, o que podemos ver na Figura 100. Foi utilizada a resolução 8 para cada índice, ao que corresponde a uma área de aproximadamente 0.73 km^2, pelo facto desta representar a melhor relação precisão-ocupação relativamente à área de operação em questão (resolução mais precisa que melhor cobriu a área de operação).

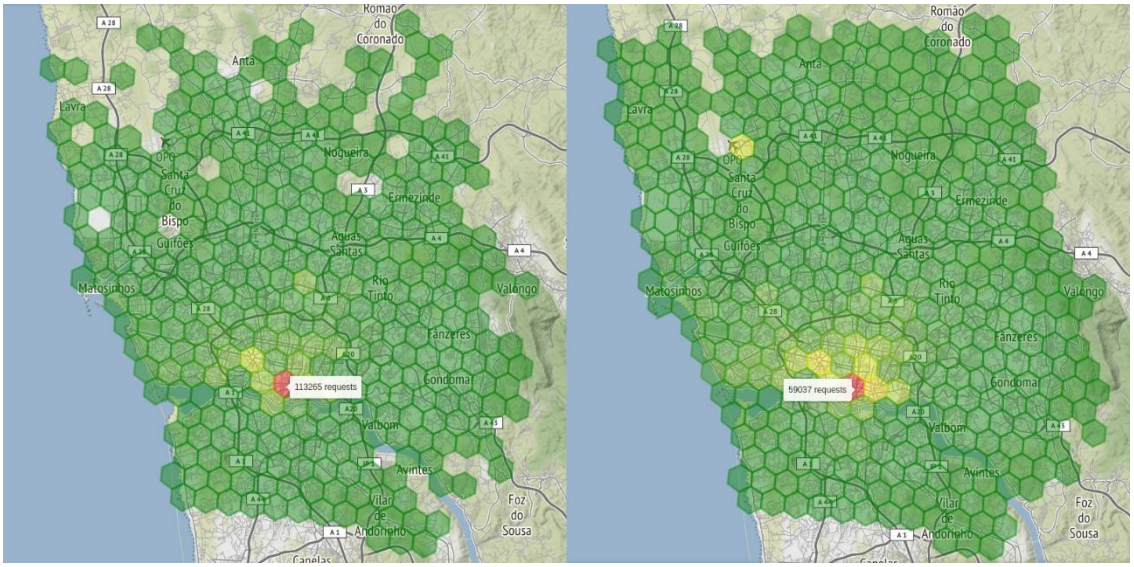

Figura 100 - Visualização das localizações de origem (à esquerda) e de destino (à direita) nas viagens de táxi utilizando o H3

Procedeu-se depois ao cálculo das diferenças entre a latitude e longitude dos locais de origem e destino, de modo a melhor se compreender a distribuição entre estes dois locais, verificando-se que existe uma maior possibilidade de as viagens serem realizadas entre sítios relativamente próximos entre si. Também foi calculada a distância entre as duas localizações utilizando a fórmula de Haversine [135], de modo a obter-se mais um atributo relevante para o desenvolvimento do modelo.

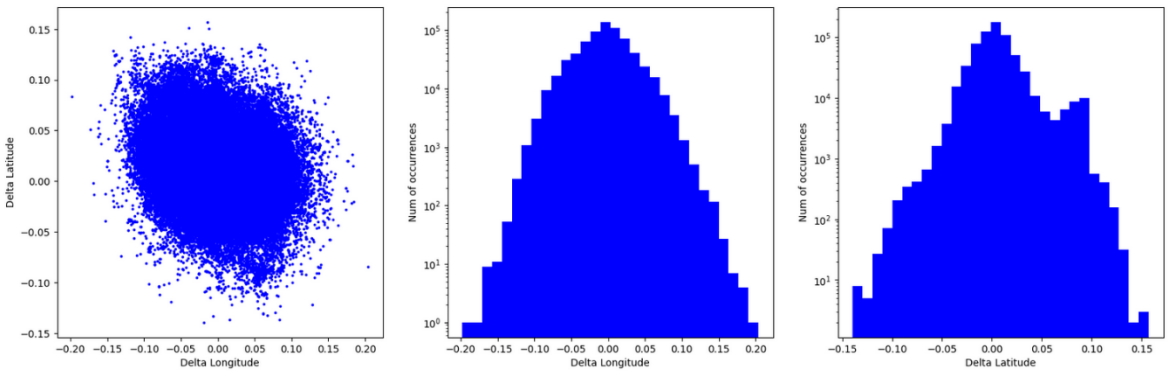

Figura 101 - Distribuição das diferenças entre a latitude e longitude de origem e destino

Desta forma obteve-se o conjunto de dados já processado e manipulado, conforme mostrado na Figura 102, constando nele as colunas calculadas anteriormente. Para o treino do modelo, optou-

se por selecionar a coluna *duration* como sendo a classe da previsão, sendo as restantes, exceto a coluna *TRIP_ID* escolhidas como os atributos para o treino do modelo.

	TRIP_ID	origin_lat	origin_lon	destination_lat	destination_lon	duration	delta_lat	delta_lon	distance	year	month	day	hour	min	weekday
1677101	1403582831620000154	41.146200	-8.618895	41.166963	-8.620245	585	0.020763	-0.001350	2.312158	2014	6	24	4	7	1
1564452	1401702999620000030	41.182578	-8.599887	41.158458	-8.604540	585	-0.024120	-0.004653	2.710918	2014	6	2	9	56	0
322086	1378985015620000296	41.149818	-8.605071	41.147271	-8.621955	210	-0.002547	-0.016884	1.442202	2013	9	12	11	23	3
121730	1374850319620000198	41.168277	-8.571987	41.160942	-8.597565	540	-0.007335	-0.025578	2.291865	2013	7	26	14	51	4
59428	1373675680620000600	41.146191	-8.617635	41.142933	-8.638362	855	-0.003258	-0.020727	1.773494	2013	7	13	0	34	5

Figura 102 - Conjunto de dados final, pronto para a construção do modelo

Uma vez que se pretende fazer a previsão de um valor numérico utilizando Random Forests, o método de aprendizagem a utilizar é a regressão, utilizando o *regressor* disponível na biblioteca scikit-learn, *RandomForestRegressor*[114]. Este *regressor* permite a configuração de vários parâmetros, que especificam o processo de aprendizagem, tais como o número de árvores de decisão a serem utilizadas, o número de atributos máximo a ser utilizado por cada nó-folha, entre outros. Estes são conhecidos como *hyperparameters*, e uma seleção adequada dos mesmos poderá fazer com que o modelo seja mais eficaz a fazer uma previsão. Assim, foi usada a função *RandomizedSearchCV*[115], que efetua uma escolha aleatória de *hyperparameters* num método de aprendizagem como o *RandomForestRegressor*, treinando um modelo e utilizando de seguida o método *cross validation*[116] para validar e avaliar a generalização do modelo, devolvendo no final o melhor conjunto de parâmetros encontrados bem como o melhor modelo dentro desse conjunto de parâmetros. Utilizaram-se diversos parâmetros diferentes de modo a que estes quando combinados pudessem ser usados por este método de escolha de parâmetros, para definir o melhor conjunto encontrado para treino do modelo, conforme podem ser vistos na Figura 103.

[114] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[115] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
[116] https://scikit-learn.org/stable/modules/cross_validation.html

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 1000, num = 5)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 6)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

Figura 103 - Lista de parâmetros usados na função *RandomizedSearchCV*

De seguida, foram definidos o número de subdivisões (*folds*) a efetuar no conjunto de dados de treino para o método *cross validation* usado por esta função, onde se escolheram 3 subdivisões (*3-fold cross validation*), de modo a encurtar o tempo de execução, uma vez que é limitado pelo Google Colab. A validação cruzada permite subdividir o conjunto de dados em 3, ficando $\frac{2}{3}$ dos dados usados para treino e $\frac{1}{3}$ para o teste e validação do modelo, conforme podemos ver na Figura 104 no caso generalizado. Finalmente, foi escolhido o número de iterações a efetuar na execução da função *RandomizedSearchCV*, sendo que se optou por escolher 3 *folds* a efetuar por cada uma das 10 iterações, perfazendo no total cerca de 30 testes a serem efetuados.

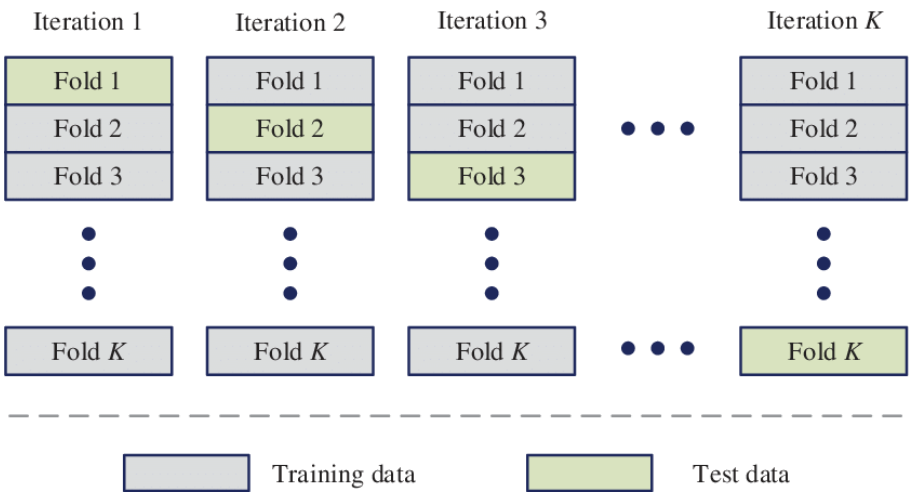

Figura 104 - Método de *cross validation* com K subdivisões (*k-fold cross validation*) [136]

4.6.3. Resultados e Validação

Tendo em conta o objetivo do desenvolvimento do modelo, pretende-se treinar o conjunto de dados de modo a que o modelo desenvolvido consiga prever as durações de viagem obtidas por base nos dados reais, pré-processados anteriormente. Depois, será criado um conjunto de dados de teste, em que as durações serão obtidas pelo Valhalla, o que permitirá fazer uma comparação direta entre os dois tipos de dados. Aqui iremos analisar e discutir os resultados obtidos do desenvolvimento do modelo e proceder à sua validação e teste, já com os dados das durações do Valhalla.

Após a execução do método *RandomizedSearchCV*, determinando os melhores parâmetros a utilizar no *RandomForestRegressor*, foram obtidos os resultados para os 10 casos aleatórios executados, os quais se apresentam na Tabela 4. Como dito anteriormente, foram feitos 3 *folds* para cada um dos casos aqui apresentados, sendo que o que diferencia cada uma das *folds* no resultado é a pontuação (*score*) obtida, bem como o tempo de execução da mesma, sendo que foi calculada a média dos 3 resultados obtidos. O método de pontuação utilizado pelo *regressor* foi o R^2 [117], um coeficiente de determinação que define o quão ajustado está o modelo aos dados observados (dados de treino), em que uma pontuação de 0 indica que não existe nenhuma relação linear entre os dados de treino e a previsão feita, e uma pontuação de 1 indica que as previsões são idênticas aos dados de treino [137].

[117] https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score

Caso	Nº de árvores usadas	Nº mínimo de amostras por divisão	Nº mínimo de amostras por folha	Máximo de atributos por divisão	Profundidade máxima da árvore	Amostragem feita em cada árvore?	Pontuação média	Tempo médio de execução (min)
A	400	2	1	Raíz quadrada	110	Não	0.229	11.5
B	200	5	1	Raíz quadrada	50	Não	0.230	5.2
C	200	5	1	Raíz quadrada	110	Não	0.229	5.4
D	200	2	2	Automático	110	Não	-0.283	18.5
E	1000	5	1	Raíz quadrada	90	Não	0.232	27.7
F	400	2	4	Automático	30	Sim	0.213	21.7
G	400	2	2	Automático	110	Sim	0.203	23.6
H	800	5	2	Raíz quadrada	-	Não	0.236	20.9
I	200	2	2	Raíz quadrada	70	Sim	0.230	3.4
J	600	10	4	Raíz quadrada	10	Sim	0.176	4.4

Tabela 4 - Resultados da execução da função *RandomizedSearchCV*

Ao analisarmos a tabela, podemos ver que no geral a pontuação média é relativamente idêntica em todos os casos, exceto nos casos D (em que a pontuação é negativa, o que nos indica que o modelo não se ajustou aos dados), e J (em que a pontuação ficou ligeiramente abaixo da média entre os restantes casos). O caso onde se obteve melhor resultado neste conjunto de parâmetros foi o caso H, com uma pontuação média de 0.236 sendo o caso escolhido para analisarmos adiante. Este é também o único caso onde a profundidade máxima da árvore não é especificada, onde segundo a documentação significa que todos os nós-folha são expandidos por cada árvore até que todas as folhas tenham menos que o número mínimo de amostras por divisão, o que neste caso tem o valor 5. Também neste caso não foi feita amostragem de dados (escolha de uma parte dos dados) em cada árvore usada, significando que foram usados todos os dados de treino para a construção e divisão de todos os nós da árvore.

Com o treino deste modelo, podemos obter qual a importância que cada atributo dos dados teve para a construção do mesmo, onde vimos que o atributo mais importante é a distância de Haversine entre as duas localizações, o que se compreende dado que quanto mais distantes duas localizações estiverem, maior será a duração da viagem entre elas, e vice-versa. Outros fatores importantes são as diferenças entre a latitude e a longitude (*delta_lat* e *delta_lon* respetivamente), que também estão relacionadas com a distância, em que quanto maior for a diferença entre a latitude e a longitude, mais distante as localizações se encontram, tendo impacto na previsão da duração. Relativamente aos atributos temporais, o que maior importância tem é a hora da viagem, com uma importância de 6.28%, o que é explicado pelo facto de a hora na qual uma viagem é feita ser uma hora em que por exemplo o trânsito está intenso como as horas de ponta, entre outros.

	importance
distance	0.212563
delta_lat	0.105040
delta_lon	0.101489
destination_lat	0.093544
origin_lat	0.092496
destination_lon	0.091271
origin_lon	0.089961
hour	0.062882
min	0.050274
day	0.040545
month	0.033332
weekday	0.026602

Figura 105 - Importância de cada atributo do conjunto de dados final na construção do modelo

Para que possamos comparar e validar o modelo desenvolvido, foi construído um conjunto de dados auxiliar tendo por base o mesmo conjunto de dados de treino, com a diferença de a duração (o que se pretende prever através do modelo) para cada uma das localizações ser obtida pelo Valhalla. Para que não fosse feito um número exagerado de pedidos, o que iria ser um processo demorado pois as coordenadas das localizações de origem e destino de uma viagem poderão ser distintas, existindo diversas possibilidades diferentes. Assim, utilizou-se o H3 para agregar tanto as localizações de origem como as de destino, através da função *h3ToGeo* que obtém o centroide (ponto central) de um índice. Foi utilizada a mesma resolução H3 mostrada na Figura 100, a

resolução 8, que conforme vimos anteriormente distribui-se por praticamente toda a área analisada. À medida que o conjunto de dados auxiliar ia sendo criado, foi-se armazenando uma lista auxiliar com os resultados obtidos a partir do Valhalla, para que para cada combinação origem-destino fosse guardada e só calculada uma vez, reduzindo o número de pedidos feitos ao Valhalla ao estritamente necessário.

Após a construção do conjunto de dados auxiliar, este foi considerado como sendo o conjunto de dados a testar comparativamente com o modelo desenvolvido. Para tal, foi utilizada uma métrica de avaliação de modelos de regressão, nomeadamente a métrica MAE (*Mean Absolute Error*), que nos indica a diferença média (erro) entre os valores estimados e os valores de teste [137], sendo de facto a métrica que se pretende obter com a construção deste modelo. O cálculo desta métrica já é fornecido pelo scikit-learn[118], sendo necessário para o seu cálculo indicar as classes de treino como o conjunto de classes previsto pelo modelo.

```
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor

X = df[['origin_lat', 'origin_lon', 'destination_lat', 'destination_lon',
        'delta_lat', 'delta_lon', 'distance', 'month', 'day', 'hour', 'min', 'weekday']]
y = df[['duration']]

f = RandomForestRegressor(n_estimators=800, min_samples_leaf=2, min_samples_split=5,
                          max_features='sqrt', max_depth=None, bootstrap=False)
f = f.fit(X, y.values.ravel())

X_test = test_df[['origin_lat', 'origin_lon', 'destination_lat', 'destination_lon',
                  'delta_lat', 'delta_lon', 'distance', 'month', 'day', 'hour', 'min', 'weekday']]
y_test = test_df[['duration']]

# Predict
y_train_pred = f.predict(X)
y_test_pred = f.predict(X_test)

print('\nMean Absolute Error train:', metrics.mean_absolute_error(y_train, y_train_pred))
print('\nMean Absolute Error test:', metrics.mean_absolute_error(y_test, y_test_pred))
```

```
Mean Absolute Error train: 62.23330557222015
Mean Absolute Error test: 285.0189285754873
```

Figura 106 - Treino e validação do modelo

Conforme vimos na Figura 106 foram calculados dois valores para o MAE: um que compara a diferença entre os resultados estimados, obtidos a partir do modelo já treinado, com os dados reais de condução dos táxis; e o outro valor que compara os resultados estimados com os dados escolhidos para teste, sendo neste caso os dados da duração fornecidos pelo Valhalla. A partir do primeiro valor podemos afirmar que o modelo desenvolvido tem um erro médio de cerca de 62.23

[118] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html

segundos, ou seja, em média o modelo prevê a duração da viagem entre dois locais com uma margem de erro de aproximadamente 1 minuto o que é bastante aceitável, tendo em conta que não foi utilizada a totalidade dos dados para o desenvolvimento do modelo. Já no caso do segundo valor obtido, trata-se da diferença média entre os valores estimados e os dados obtidos a partir do Valhalla, que é cerca de 285.01 segundos, o que corresponde a aproximadamente 5 minutos. Uma vez que o Valhalla não fornece dados de trânsito por si só na sua execução, o efeito que a margem de erro poderá ter no âmbito da seleção dos Keepers e no contexto de realização de um serviço poderá ser importante quando o objetivo é otimizar o tempo de resposta a um novo pedido de serviço, reduzindo-o ao máximo possível. Se neste caso também acrescentarmos o tempo que se demora a efetuar cada pedido feito ao Valhalla em comparação com a obtenção de uma previsão a partir de um modelo já calculado, torna-se mais vantajosa a sua utilização.

```
durations = df.groupby(['origin_h3', 'destination_h3'])['duration'].mean().reset_index()

def findOutNeighbors(row):
  cnt = 1
  if row['origin_h3'] == row['destination_h3']:
    return 0

  while True:
    if row['destination_h3'] in h3.hex_ring(row['origin_h3'], cnt):
      return cnt
    cnt += 1

durations['hex_distance'] = durations.progress_apply(findOutNeighbors, axis=1)
durations
```

Figura 107 - Uniformização e cálculo da distância entre índices origem e destino

Finalmente, e tendo em conta o método de seleção de Keepers proposto no capítulo 4.5.2.5.1, vamos verificar, com base neste conjunto de dados, se é de facto possível selecionar um Keeper para responder a um novo pedido de serviço em menos de 15 minutos. Como já mostrado, sugeriu-se a utilização de 3 níveis de procura, sendo que o melhor caso deste método é selecionar um Keeper que esteja num destes níveis para realizar o serviço. Assim, e tendo em conta os dados já obtidos, primeiro uniformizaram-se os dados, dado que poderão haver várias durações tendo em conta o mesmo local de origem e de destino, e deste modo calculou-se a média dessas durações. Criou-se desta forma uma nova coluna no conjunto de dados com a distância entre os índices H3. Deste modo sabemos não só a duração da viagem bem como o número de índices, em anel, entre a origem e o

destino, o que podemos ver na Figura 107. De seguida, consultou-se a duração média de viagem nos casos em que a distância é 3, sendo estes os níveis de procura.

```
durations.loc[durations['hex_distance'] <= 3]['duration'].value_counts(bins=10)
```

```
(6.643999999999999, 850.5]    2816
(850.5, 1686.0]                305
(1686.0, 2521.5]                46
(2521.5, 3357.0]                22
(3357.0, 4192.5]                10
(4192.5, 5028.0]                 6
(6699.0, 7534.5]                 2
(5028.0, 5863.5]                 2
(7534.5, 8370.0]                 1
(5863.5, 6699.0]                 0
Name: duration, dtype: int64
```

```
hex_durations = durations.loc[durations['hex_distance'] <= 3]['duration']
print(hex_durations[hex_durations < (15*60)].shape[0] / hex_durations.shape[0])
```

```
0.8947040498442368
```

Figura 108 - Duração média de viagem nos 3 níveis de procura, e percentagem de durações de viagem inferiores a 15 minutos

Como podemos verificar na Figura 108, a maior duração de viagem foi entre 7534.5 e 8370 segundos, isto é, aproximadamente 125.5 e 139.5 minutos, respetivamente. Este é um dos casos considerados falsos positivos, dado que a distância efetiva entre a origem e o destino está dentro dos níveis de procura, o que à partida seria considerado como próximo em termos de índices do H3 mas como podemos ver no mapa da Figura 100, estes índices poderão estar separados por um rio, o que implica no seu trajeto atravessar uma ponte para que se possa chegar ao destino. No entanto, verificou-se que de acordo com estes dados, em 89.4% das vezes é possível efetuar uma viagem em menos de 15 minutos, o que torna possível utilizar o H3 nesta área de operação como uma ferramenta válida de seleção de Keepers tendo em conta o método já apresentado anteriormente.

4.7. Testes de Qualidade

Todos os componentes de *software* desenvolvidos no âmbito desta dissertação foram alvo de testes de qualidade, tanto através de testes unitários de *software*, como através de testes de desempenho. Nesta secção serão abordados esses conceitos, tal como será mostrado como foi feito o seu desenvolvimento e integração ao longo desta dissertação.

4.7.1. Testes Unitários

Os testes unitários permitem a verificação automática de erros à medida que novos componentes e funcionalidades são acrescentadas ao sistema. A sua utilização leva a um melhoramento geral na qualidade do código à medida que este é desenvolvido e integrado no sistema, dado que é utilizado o conceito de CI/CD, já referido no capítulo 4.1.4.

Todos os componentes de *software* desenvolvidos no âmbito desta dissertação foram alvo de testes unitários à medida que um novo *endpoint* era desenvolvido. No caso da linguagem Go, os testes de *software* são implementados nativamente através do package *testing*[119], que fornece ferramentas que permitem testar a aplicação sem a necessidade de usar uma ferramenta externa.

```go
func TestSetRequestState(t *testing.T) {
    mockCtrl := gomock.NewController(t)
    defer mockCtrl.Finish()

    svc := mocks.NewMockKeeperOptimizerServiceClient(mockCtrl)

    tt := []struct {
        name     string
        request  *pb.SetRequestStateRequest
        response *empty.Empty
        err      error
    }{
        {
            name: "Normal request",
            request: &pb.SetRequestStateRequest{
                RequestId: "abcdefgh",
                State:     pb.EventType_DRAFT,
            },
            response: &empty.Empty{},
            err:      nil,
        },
        {
            name: "Request without RequestID",
            request: &pb.SetRequestStateRequest{
                State: pb.EventType_DRAFT,
            },
            response: nil,
            err:      errRequestIDInvalid,
        },
    }

    for _, tc := range tt {
        t.Run(tc.name, func(t *testing.T) {
            svc.EXPECT().SetRequestState(
                gomock.Any(),
                tc.request,
            ).Return(tc.response, tc.err).Times(1)

            ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
            defer cancel()

            r, err := svc.SetRequestState(ctx, tc.request)
            if !reflect.DeepEqual(r, tc.response) {
                t.Errorf("%s test FAILED, got %v; result should be %v", tc.name, r, tc.response)
            }
            if !errors.Is(err, tc.err) {
                t.Errorf("%s test FAILED, got error %v; error should be %v", tc.name, err, tc.err)
            }
        })
    }
}
```

Figura 109 - Exemplo de um teste unitário

[119] https://pkg.go.dev/testing

Para simular uma ligação feita ao servidor gRPC de um serviço, foi utilizada a ferramenta gomock[120], que simula o funcionamento de um servidor ao gerar um *mock object* que implementa os métodos criados no ficheiro Protocol Buffers do próprio serviço. Para construirmos testes unitários cria-se inicialmente um controlador, que será o ponto de ligação entre o *mock object* gerado e a implementação do serviço, é criada uma tabela de testes para serem executados, onde cada teste tem um nome para posterior identificação nos resultados, bem como o pedido, e a resposta e erro esperados. De seguida os testes são executados sendo os resultados verificados de acordo com o que consta na seguinte figura, onde podemos ver cada teste individual e o que resultou de cada execução.

```
[cvagos@warrior keeper-optimizer]$ go test -v ./...
?       luggit.app/keeper-optimizer     [no test files]
?       luggit.app/keeper-optimizer/benchmark   [no test files]
=== RUN   TestSetRequestState
=== RUN   TestSetRequestState/Normal_request
=== RUN   TestSetRequestState/Request_without_RequestID
=== RUN   TestSetRequestState/Request_with_wrong_state
--- PASS: TestSetRequestState (0.00s)
    --- PASS: TestSetRequestState/Normal_request (0.00s)
    --- PASS: TestSetRequestState/Request_without_RequestID (0.00s)
    --- PASS: TestSetRequestState/Request_with_wrong_state (0.00s)
=== RUN   TestSetKeeperLocation
=== RUN   TestSetKeeperLocation/Normal_request_-_not_on_request
=== RUN   TestSetKeeperLocation/Normal_request_-_on_a_request
=== RUN   TestSetKeeperLocation/Request_without_Keeper_ID
=== RUN   TestSetKeeperLocation/Request_without_location
=== RUN   TestSetKeeperLocation/Request_without_timestamp
--- PASS: TestSetKeeperLocation (0.00s)
    --- PASS: TestSetKeeperLocation/Normal_request_-_not_on_request (0.00s)
    --- PASS: TestSetKeeperLocation/Normal_request_-_on_a_request (0.00s)
    --- PASS: TestSetKeeperLocation/Request_without_Keeper_ID (0.00s)
    --- PASS: TestSetKeeperLocation/Request_without_location (0.00s)
    --- PASS: TestSetKeeperLocation/Request_without_timestamp (0.00s)
=== RUN   TestSetKeeperLocationEvent
=== RUN   TestSetKeeperLocationEvent/Normal_request
=== RUN   TestSetKeeperLocationEvent/Request_without_Keeper_ID
=== RUN   TestSetKeeperLocationEvent/Request_without_location
=== RUN   TestSetKeeperLocationEvent/Request_without_timestamp
=== RUN   TestSetKeeperLocationEvent/Request_without_request_ID
=== RUN   TestSetKeeperLocationEvent/Request_without_event
--- PASS: TestSetKeeperLocationEvent (0.00s)
    --- PASS: TestSetKeeperLocationEvent/Normal_request (0.00s)
    --- PASS: TestSetKeeperLocationEvent/Request_without_Keeper_ID (0.00s)
    --- PASS: TestSetKeeperLocationEvent/Request_without_location (0.00s)
    --- PASS: TestSetKeeperLocationEvent/Request_without_timestamp (0.00s)
    --- PASS: TestSetKeeperLocationEvent/Request_without_request_ID (0.00s)
    --- PASS: TestSetKeeperLocationEvent/Request_without_event (0.00s)
PASS
ok      luggit.app/keeper-optimizer/server      0.012s
```

Figura 110 - Resultados de um teste unitário

4.7.2. Testes de Desempenho

Os testes de desempenho permitem testar os limites de cada componente de *software* instalado, para verificar e analisar casos de utilização onde o sistema começa a falhar, sendo

[120] https://github.com/golang/mock

necessário escalar para que este possa responder aos utilizadores, o que poderá acontecer em picos de utilização. Para podermos perceber até que ponto o sistema poderá ser utilizado sem erros de resposta foram efetuados testes de carga.

Os testes de carga foram desenvolvidos utilizando a linguagem Go e a ferramenta de *benchmark* ghz[121], criada para testar serviços gRPC. O ghz fornece parâmetros que permitem a configuração de um teste, especificando o número de pedidos, a concorrência utilizada nos pedidos, tempo limite de teste, entre outros. O serviço escolhido para teste foi o Keeper Optimizer, que foi instalado na Google Cloud Platform através de um container Docker, executado pelo Kubernetes. Para este teste foi utilizada apenas uma instância do serviço bem como o *endpoint* SetKeeperLocation, um dos *endpoints* mais utilizados pois irá receber frequentemente as localizações dos Keepers.

É importante referir que o Kubernetes é executado de uma forma distribuída, isto é, em diversos nós de execução, onde são distribuídos os diversos *containers* a serem executados. Cada execução de um *container* tem um conjunto de recursos que são requisitados aos nós de execução, que também podem ser distribuídos pelo Kubernetes. No ficheiro YAML de configuração de um *Deployment* é possível definir limites de utilização na execução de cada *container*, tanto ao processador como à memória sendo que no caso de um desses limites ser atingido em *runtime*, o Kubernetes termina a execução do *container*. No caso do teste executado, os limites impostos foram de 500 *Megabytes* de memória e 0.1 unidades de CPU (medida utilizada pelo Kubernetes, em que 1 CPU corresponde na Google Cloud Platform a 1 *core* GCP)[138]. Na execução dos testes estes limites não foram atingidos, significando que a execução do serviço não foi interrompida pelo Kubernetes.

Foram executados dois conjuntos de pedidos: pedidos sequenciais durante um minuto, isto é, o número de pedidos total foi distribuído por um minuto; e pedidos concorrentes, sendo um conjunto de pedidos enviados num segundo, de uma vez. Os resultados podem ser vistos na Tabela 5.

[121] https://ghz.sh/

	Pedidos sequenciais (intervalo de um minuto)				*Pedidos concorrentes*			
Nº Total de Pedidos	Nº de Pedidos Médio por segundo	Tempo médio de resposta (s)	Taxa de sucesso (%)	Taxa de erro (%)	Nº de Pedidos Médio por segundo	Tempo médio de resposta (s)	Taxa de sucesso (%)	Taxa de erro (%)
50	1	0.243	100.00	0.00	50	0.994	6.00	94.00
250	4.1	0.373	100.00	0.00	250	0.997	0.00	100.00
500	8.3	0.523	100.00	0.00	500	0.998	0.00	100.00
1000	16.6	0.845	100.00	0.00	1000	0.996	0.00	100.00
2500	41.6	1.79	97.02	2.98	2500	0.972	0.00	100.00
5000	83.3	3.05	94.92	5.08	5000	0.939	0.00	100.00

Tabela 5 - Resultados do teste de carga efetuado ao *endpoint* SetKeeperLocation (1 instância do serviço)

Com estes resultados consegue-se perceber que o *endpoint* começa a responder com erros aos 2500 pedidos sequenciais, onde foram efetuados em média cerca de 41.6 pedidos a cada segundo, correspondendo a 2.98% de taxa de erros. Em contrapartida verifica-se que até 1000 pedidos sequenciais não existem quaisquer erros. Uma vez que a carga vai aumentando através do número de pedidos, o tempo médio de resposta é maior dado que o servidor ainda está a responder a pedidos anteriores. Já no que diz respeito ao número de pedidos concorrentes, quando o número de pedidos é curto ainda se consegue obter 6% de taxa de sucesso o que já é muito pouco, mas o *endpoint* torna-se praticamente inutilizável a partir desse caso, significando que o serviço não é capaz de responder a qualquer pedido. Por estes resultados podemos afirmar que uma só instância do serviço consegue lidar, para este *endpoint*, com menos de 2500 pedidos distribuídos num minuto, e no caso de pedidos concorrentes um número inferior a 50 pedidos por segundo.

De modo a permitir mais pedidos, sejam eles com concorrência ou não, deverá aumentar-se o número de instâncias do serviço, escalando-o horizontalmente. Para o fazer, deverá ser alterado o número de réplicas na configuração do *Deployment* Kubernetes referente ao serviço [139]. Uma vez que a API do Keeper Optimizer é exposta para o público através de um *Service* do Kubernetes [140], o acesso à mesma será balanceado entre todas as réplicas através de um balanceador de carga (*load balancer*), algo que o Kubernetes já disponibiliza internamente (sendo no entanto possível o uso de outros *load balancers* externos), o que fará com que os pedidos sejam distribuídos entre as várias

réplicas. Então fez-se o mesmo teste de desempenho mas com 4 réplicas, ou seja com 4 instâncias do serviço em funcionamento, cujos resultados poderão ser vistos na Tabela 6.

	Pedidos sequenciais (intervalo de um minuto)				*Pedidos concorrentes*			
Nº Total de Pedidos	Nº de Pedidos Médio por segundo	Tempo médio de resposta (s)	Taxa de sucesso (%)	Taxa de erro (%)	Nº de Pedidos Médio por segundo	Tempo médio de resposta (s)	Taxa de sucesso (%)	Taxa de erro (%)
50	1	0.165	100.00	0.00	50	0.972	12.00	88.00
250	4.1	0.299	100.00	0.00	250	0.998	0.00	100.00
500	8.3	0.393	100.00	0.00	500	1.000	0.00	100.00
1000	16.6	0.583	100.00	0.00	1000	0.995	0.00	100.00
2500	41.6	1.04	98.76	1.24	2500	0.936	0.00	100.00
5000	83.3	1.74	97.09	2.91	5000	0.968	0.00	100.00

Tabela 6 - Resultados do teste de carga efetuado ao *endpoint* SetKeeperLocation (4 instâncias do serviço)

Com o aumento do número de instâncias garantiu-se uma diminuição no tempo médio de resposta no caso dos pedidos sequenciais, bem como a taxa de erro nos casos de 2500 e de 5000 pedidos sequenciais. Já no caso dos pedidos concorrentes, a taxa de sucesso na ocorrência de 50 pedidos num segundo aumentou, passando de 6% para 12%. Estes resultados demonstram que a capacidade de várias instâncias poderem responder aos diversos pedidos entre si tem um efeito benéfico na utilização do serviço. Para que o serviço pudesse ser capaz de responder a um maior número de pedidos que no caso deste serviço fará com que este seja capaz de atender a vários pedidos, o que será útil tanto para casos em que ocorram picos de utilização, bem como no futuro à medida que é experienciado o crescimento do número de Keepers.

5. Conclusão

O uso de plataformas tecnológicas de mobilidade é cada vez mais comum na sociedade em contexto urbano, levando o conceito de transporte *on-demand* e *on-site* como parte integrante do dia-a-dia. O grande desafio destas plataformas é garantir e otimizar a sua operação, e neste campo os condutores são os maiores responsáveis por fazer questão que todo o processo ocorra o mais rápido e eficaz possível. Para que estes consigam realizar e responder a novos serviços de modo a melhorar a satisfação do cliente, que pretende que o serviço seja completado o quanto antes, o seu posicionamento é um dos fatores mais importantes.

Em suma, todos os objetivos desta dissertação foram atingidos, tendo sido criado um sistema que corrige o posicionamento dos condutores (Keepers) da LUGGit, sugerindo locais estratégicos utilizando *heatmaps* como ferramentas auxiliares de visualização. Para além disso, criaram-se outras funcionalidades e ferramentas dentro deste contexto, que à medida que o desenvolvimento, estudo e implementação da solução foram decorrendo fizeram sentido em ser desenvolvidas e integradas. De seguida foi também possível criar um sistema que permite selecionar os Keepers capazes de responder a um novo pedido de serviço em menos de 15 minutos, o que veio melhorar a capacidade operacional da empresa, que não possuía até à data um sistema de seleção de Keepers que permitisse otimizar o seu tempo de resposta.

Foi criado um serviço de Routing, que permite obter trajetos utilizando e tendo em conta os dados do OpenStreetMap, bem como pontos de interesse tendo por base localizações, onde neste âmbito foram configuradas e instaladas ferramentas auxiliares. Estes dados em combinação com dados de contexto geoespacial relativos às áreas de operação, fornecidos pelo serviço de Geofencing, que também foi criado neste trabalho, permitiram fornecer locais para o condutor se dirigir na realização de um serviço e características extra importantes para a realização de um serviço. Para que se efetue uma análise a este tipo de dados, oriundos de todos os serviços realizados desenvolveu-se também um sistema que não só efetua a sua recolha, como também fornece métricas úteis para a compreensão do comportamento dos Keepers em contexto real de condução, correlacionando esses mesmos dados com dados meteorológicos, partidas e chegadas de voos em aeroportos e feriados. Desta forma, conseguiu-se fornecer *heatmaps*, que são uma ferramenta visual que ajuda os condutores a saber quais as localizações ideais para estes se posicionarem de modo a estes responderem a novos serviços o mais rápido possível.

Todos os serviços desenvolvidos no âmbito desta dissertação encontram-se já instalados e em funcionamento em produção, bem como todas as funcionalidades que foram abordadas ao longo desta dissertação (estando algumas ainda numa fase embrionária no ambiente de produção). O trabalho desenvolvido revelou-se muito importante para os Keepers da LUGGit no seu quotidiano, onde se obteve *feedback* muito positivo no que aos métodos de seleção desenvolvidos diz respeito, que segundo os próprios veio promover maior justiça comparativamente ao método de seleção anteriormente utilizado. O desenvolvimento deste sistema traduziu-se também num decréscimo do tempo médio de resposta a novos serviços, sendo que o objetivo de garantir que a resposta a um novo serviço é feita em menos de 15 minutos também foi superado, e assim todos os objetivos propostos foram cumpridos.

5.1. Discussão e Trabalho Futuro

Durante a realização deste trabalho foram surgindo alguns assuntos que merecem ser alvo de discussão e futura análise, sendo eles úteis para trabalho futuro e para uma melhor compreensão do problema, que levou a esta solução aqui apresentada.

Relativamente ao serviço de Routing, verificou-se que ao ser utilizado o Valhalla como ferramenta auxiliar, apesar de este fornecer as funcionalidades necessárias para os requisitos definidos e não exigir maiores recursos que as restantes ferramentas apresentadas, a não-utilização de dados de trânsito coloca em causa a sua utilização num caso de uso exigente a nível temporal, como o aqui apresentado. No caso do *endpoint* Directions do serviço de Routing, o tempo estimado de viagem é por vezes demasiado otimista, o que poderá induzir o Keeper em erro na disponibilização do tempo estimado de viagem. Com o uso do *endpoint* Isochrone e conforme visto na Figura 94, verificamos que segundo o Valhalla é possível ir até Cascais a partir da zona de Alfama em 30 minutos, sendo este resultado constante durante todo o dia, ao contrário do que é indicado na Figura 111, uma vez que a não-existência de dados de trânsito não têm qualquer influência no resultado, sendo as *tiles* construídas apenas a partir dos dados do OSM. Se tivermos em conta a utilização das *isochrones* para fornecer os Keepers mais próximos (no capítulo 4.5.2.5.2), indicar o tempo estimado para chegada dos Keepers através de um resultado demasiado otimista poderá ter um efeito indesejado na seleção dos Keepers do próximo pedido de serviço, uma vez que se pretende que o Keeper chegue num período de tempo inferior a 15 minutos. Com o desenvolvimento do modelo de *machine learning* conseguiu-se determinar que na cidade do Porto o Valhalla tem em

média 5 minutos de diferença comparativamente a dados reais de condução, o que numa primeira fase se poderá assumir este erro para próximos cálculos de trajeto. No entanto, uma possível solução para uma correção dos cálculos seria a implementação de métodos que permitam integrar custos dinâmicos no Valhalla [141], sendo estes alimentados pelos dados recolhidos pela *data warehouse* desenvolvida.

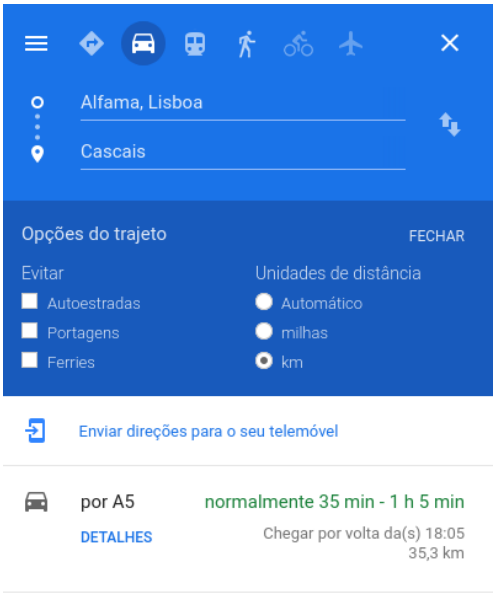

Figura 111 - Cálculo do trajeto Alfama-Cascais a uma segunda-feira às 17h, segundo o Google Maps [142]

As funcionalidades de *geocoding* fornecidas no serviço de Routing, apesar de não serem requeridas para esta dissertação, são úteis em contextos de análise de dados para fornecer características provenientes dos próprios locais de serviço. Contudo, verificou-se que ao serem utilizadas ferramentas auxiliares de *geocoding* que usem dados provenientes do OSM e da Wikidata (o caso do Nominatim e do Photon), não são fornecidos tantos locais e pontos de interesse comparativamente com uma solução como por exemplo a Google Places, o que torna essas funcionalidades ineficazes num contexto de utilização de procura de locais com maior especificidade. Apesar destas questões é justo dizer-se que o serviço de Routing oferece funcionalidades que permitem substituir por completo uma solução paga, contudo, fornecendo informação com menor eficácia e detalhe, o que também se compreende.

No serviço de Geofencing, as localizações são um assunto chave que permite fornecer o máximo de informação possível a um Keeper no âmbito da realização de um serviço. Ao desenvolver este serviço deu-se particular importância às áreas de operação bem como a locais como os

aeroportos, e eventos, devido ao facto de estes serem locais que historicamente representaram desafios no desenvolvimento do produto, sendo no âmbito do posicionamento dos Keepers uma necessidade. No entanto, esta informação deverá ser recolhida diretamente a partir do terreno, de modo a que esta seja integrada para ajudar ainda mais o Keeper na realização de serviços. Por exemplo, o serviço ser capaz de fornecer zonas e locais de estacionamento grátis e permitidos numa área de operação, tendo por base a proximidade com o ponto de encontro, o que poderá ser útil no caso de atrasos por parte do cliente na entrega. Desta forma o Keeper não fica sujeito a uma possível multa por parte das autoridades, enquanto aguarda pela chegada do cliente. Outra funcionalidade interessante é a integração de *geofences* dinâmicas, já suportada pelo Tile38, de modo a que cada interveniente no serviço saiba que está próximo um do outro, o que irá aumentar não só a experiência de utilização como facilitar o encontro entre eles.

Uma questão que surgiu no âmbito do desenvolvimento dos *heatmaps* prende-se com o facto de a empresa pretender expandir o número de cidades disponíveis, levando à inexistência de dados históricos relativamente a serviços nesses locais. Este aspeto coloca em causa o fornecimento de *heatmaps* aos Keepers que estejam a trabalhar nessas áreas de operação. Como serão sugeridas as localizações possíveis de novos pedidos de serviço nesses casos é uma das questões que importa abordar no futuro. Outro aspeto prende-se com as resoluções do H3 a fornecer nos *heatmaps*, dado que no âmbito desta dissertação foram usadas resoluções estáticas, não sendo facilmente adaptáveis por exemplo pelo próprio Keeper, o que seria uma boa funcionalidade a oferecer. No fornecimento de heatmaps que combinem tanto o contexto em tempo-real como o histórico torna-se também importante para cada área de operação ajustar o peso dado a cada componente de uma forma dinâmica e otimizada de acordo com o estado atual da operação, sendo que no caso da solução implementada foi utilizado um peso estático para cada uma das componentes. Os métodos de procura de Keepers utilizados foram construídos tendo por base métodos distintos, que utilizam sobretudo a localização como o fator principal de procura. A criação da Versão 1 faz com que seja mais fácil encontrar Keepers utilizando os índices H3 vizinhos a partir de uma dada localização, o que tem uma maior performance dado que o H3 é uma biblioteca de indexação hierárquica, sendo fácil a obtenção desses vizinhos. Contudo, a escolha da resolução indicada para a procura poderá ter influência na forma como os Keepers são encontrados, podendo ser pequena ou grande demais para o proposto, o que afeta o tamanho e o número de Keepers encontrado a cada nível de procura. Quanto à Versão 2, apesar de já ter sido discutido um dos problemas na construção dos níveis de procura, o conjunto de durações que definem cada uma das regiões de procura a encontrar também

poderá ter influência nos resultados, sendo também importante referir que esta versão apresenta um pior nível de performance que a Versão 1 uma vez que requer executar um pedido de rede ao serviço de Routing, e por sua vez ao Valhalla. Para este caso, uma abordagem que passe pela aprendizagem e adaptação dos níveis de procura tendo em conta as condições da própria área de operação seria também interessante implementar neste sistema, com vista à otimização da seleção dos Keepers.

No que à recolha e armazenamento de dados na *data warehouse* construída diz respeito, torna-se interessante para o futuro a integração de mais fatores que possam ter influência nos próprios serviços, bem como a implementação de um sistema que de facto suporte e recolha as métricas de uma forma automatizada, através de uma análise de dados mais profunda que permita prever próximos acontecimentos de antemão, numa altura em que a análise de dados tem cada vez mais interesse para as empresas, dando-lhes vantagem competitiva. Outro aspeto importante a abordar no futuro é a otimização de localizações, tanto a partir de próprias fontes como através de fontes externas, de forma a que se possa combater o problema já identificado da integração e oferta de mais cidades e áreas de operação que não reúnam dados históricos, permitindo correlacionar dados de forma a prever próximas localizações de serviços.

O desenvolvimento do modelo de aprendizagem automática acabou por ficar limitado pelo uso gratuito da ferramenta Google Colab, que tem um limite de 25 *Gigabytes* de memória, o que impediu a seleção e análise à totalidade dos dados reais obtidos, tendo sido feita uma amostragem a partir dos mesmos, o que influenciou o resultado final. Durante o desenvolvimento foi utilizado o H3 como forma de agregar várias localizações, e até se chegou a utilizar os índices H3 para representar a origem e destino da viagem como atributos para o treino do modelo, contudo estes não tiveram qualquer influência no mesmo, e acabaram por ser removidos da lista de atributos de aprendizagem aqui mostrada. Uma das razões poderá estar ligada com o facto do algoritmo não perceber como deverá processar um índice H3, não sendo facilmente descoberto um teste que permita construir uma decisão, como por exemplo decidir se o índice H3 é maior ou menor que um determinado número. Outro aspeto também interessante neste âmbito seria o desenvolvimento e análise de modelos utilizando outros métodos de aprendizagem automática como por exemplo o *Gradient Boosting*, o que permitiria comparar diversos métodos com o método utilizado nesta dissertação. A validação do H3 como um método válido de seleção de Keepers tornou-se promissor, sendo que na grande maioria um Keeper situado nos níveis de procura indicados pelo método de

procura consegue de facto responder a um novo serviço num tempo inferior a 15 minutos. De facto, a impossibilidade provocada pelo COVID-19 de realizar testes e validação a este método utilizando os dados recolhidos pelo Serviço de Correção de Posicionamento de Condutores tornou apenas possível efetuar validação na cidade do Porto, utilizando um conjunto de dados externo.

5.2. Dificuldades Sentidas

Durante o desenvolvimento do projeto foram surgindo diversas dificuldades que tiveram de ser resolvidas e ultrapassadas, sendo que também me motivaram a pesquisar e a aprender mais sobre os conceitos abordados.

A primeira dificuldade passou pela aprendizagem de uma nova linguagem de programação, o Go, sendo uma linguagem que não tinha ainda experimentado ou trabalhado até então, apesar de haver uma extensa comunidade e ser uma das linguagens que mais tem crescido a nível de popularidade nos últimos anos [143], principalmente no desenvolvimento *backend*. A implementação dos microserviços, bem como o uso de *goroutines* e de *channels* foi também algo que motivou e ao mesmo tempo impôs alguma dificuldade inicial. Outro desafio foi aprender como funciona o gRPC bem como os Protocol Buffers em contexto de um microserviço, sendo tecnologias modernas e de grande potencial, que se revelaram interessantes no ponto de vista da comunicação entre serviços, dada a sua performance. A documentação já existente bem como a adoção das mesmas pela própria empresa fez com que a aprendizagem fosse facilitada.

Um aspeto que também provocou dificuldades foi o uso do Kubernetes em conjunto com o Istio em ambiente de produção, uma vez que são necessárias configurações extra que fazem com que o *software* instalado esteja protegido através de políticas de segurança e de acesso, sendo também necessário um estudo prévio e maior cuidado na resolução de erros e em manter os serviços seguros, dado que estarão disponíveis para o exterior em ambiente *Cloud*. Após a pesquisa das diversas ferramentas de Routing e de *geocoding*, houve a necessidade de as configurar e instalar, o que também constituiu um desafio pois era necessário fazer uma investigação sobre as mesmas, testá-las e de seguida escolher qual a mais indicada para o problema. A maior parte destas ferramentas não apresentava suporte nem documentação para instalações utilizando o Docker, sendo que são ferramentas sobretudo utilizadas em servidores *bare metal*, isto é diretamente num servidor físico sem a utilização de ambientes virtualizados para a sua gestão, o que fez com que

tivessem de ser construídas *Dockerfiles* que tornassem possível a execução destas ferramentas. Ainda sobre estas ferramentas, houve também dificuldade em compreender como poderiam ser feitas as atualizações de dados sem provocar *downtime*, não existindo documentação nem suporte para o mesmo, e onde se acabou por resolver o problema pela utilização de *cron jobs* para o efeito, gerando novos dados para volumes de dados e de seguida procedendo-se ao reinício do serviço, fazendo com que os dados atualizados sejam carregados.

Ao desenvolver-se a arquitetura especificada na Figura 78 houve obstáculos que impediram um desenvolvimento mais simples e facilitado, passando principalmente pelo uso da biblioteca Goka, que abstrai muitos conceitos internos do Kafka, tendo o seu próprio meio de funcionamento. Apesar de haver diversa documentação na sua página, bem como exemplos práticos, foi também um desafio poder saber como obter dados a partir do Kafka utilizando as *views* do Goka para a construção de *heatmaps* em tempo-real, o que levou a várias horas de desenvolvimento e testes de modo a garantir que o seu funcionamento estava conforme o esperado e pretendido.

Finalmente, outra das dificuldades passou pelo estudo e implementação de um modelo utilizando métodos de aprendizagem automática, o que no contexto desta dissertação reunia bastante interesse por intermédio da análise de dados, permitindo provar que a análise de dados reais de condução oferece uma previsão de duração de viagem com contexto real de utilização, o que não acontece ao usar uma ferramenta estática como o Valhalla. Primeiro, tentou-se analisar dados localmente, mas cedo verificou-se que não seria viável por via da existência de recursos computacionais insuficientes para o fazer. Após ter-se feito o mesmo utilizando o Google Colab verificou-se de igual forma que os recursos computacionais não eram suficientes para a execução do modelo, onde se escolheu reduzir a quantidade de dados de treino, bem como a separação do desenvolvimento em várias fases de modo a não exceder os recursos de uso gratuito que esta ferramenta oferece neste âmbito. A investigação e aprendizagem desta área também foi difícil, no entanto desafiante e motivada pela extensa comunidade e documentação existente, o que permitiu um desenvolvimento e compreensão dos conceitos mais facilitado.

6. Referências

[1] European Commission, “Developments and Forecasts on Continuing Urbanisation | Knowledge for policy,” 2019. https://ec.europa.eu/knowledge4policy/foresight/topic/continuing-urbanisation/developments-and-forecasts-on-continuing-urbanisation_en (accessed Oct. 01, 2019).

[2] “Urban mobility at a tipping point | McKinsey.” https://www.mckinsey.com/business-functions/sustainability/our-insights/urban-mobility-at-a-tipping-point (accessed Oct. 01, 2019).

[3] “Vehicles in use - Europe 2017 | ACEA - European Automobile Manufacturers’ Association,” 2017. https://www.acea.be/statistics/article/Report-Vehicles-in-Use (accessed Dec. 01, 2019).

[4] European Parliment, “What is carbon neutrality and how can it be achieved by 2050? | News | European Parliament,” *European Parliment*, 2019. https://www.europarl.europa.eu/news/en/headlines/society/20190926STO62270/what-is-carbon-neutrality-and-how-can-it-be-achieved-by-2050 (accessed Oct. 03, 2019).

[5] N. Winton, “World Car Sales Will Fall More Than 4 Million In 2019; Report,” *Forbes*, 2019. https://www.forbes.com/sites/neilwinton/2019/06/12/world-car-sales-will-fall-more-than-4-million-in-2019-report/#f56bc0022632 (accessed Oct. 29, 2020).

[6] J. Tapper, “Invasion of the electric scooter: can our cities cope?,” *The Guardian*, 2019. https://www.theguardian.com/cities/2019/jul/15/invasion-electric-scooter-backlash (accessed Oct. 29, 2019).

[7] E. C. Strategy, “EU Cycling Strategy - Recommnedations for Delivering Green Grown and an Effective Mobility System in 2030 - Summary,” 2010. Accessed: Oct. 29, 2019. [Online]. Available: https://ecf.com/eu_cycling_strategy.

[8] SITA, “2019 Baggage IT insights,” 2019. Accessed: May 06, 2020. [Online]. Available: https://www.sita.aero/resources/type/surveys-reports/baggage-it-insights-2019.

[9] “Global Luggage Market Size, Share, Trends and Forecast.” https://www.mordorintelligence.com/industry-reports/luggage-market (accessed Sep. 27, 2019).

[10] UNWTO, “Growth in international tourist arrivals continues to outpace the economy,” *World Tour. Baromoter*, vol. 18, no. 1, pp. 1–6, 2020, Accessed: May 06, 2020. [Online]. Available: https://www.unwto.org/international-tourism-growth-continues-to-outpace-the-economy.

[11] Peerbits, “How are apps transforming the travel industry?,” 2019. https://www.peerbits.com/blog/mobile?app?transforming?travel?industry.html (accessed Sep. 27, 2019).

[12] M. Christian, “Mobile Application Development in the Tourism Industry and its Impact on On-Site Travel Behavior,” 2015.

[13] “Como funciona a Uber?” Accessed: Sep. 27, 2019. [Online]. Available: https://help.uber.com/pt-PT/riders/article/como-funciona-a-uber?nodeId=738d1ff7-5fe0-4383-b34c-4a2480efd71e.

[14] M. Iqbal, “Uber Revenue and Usage Statistics (2020) - Business of Apps,” *Business of Apps*, 2019. https://www.businessofapps.com/data/uber-statistics/ (accessed Sep. 27, 2019).

[15] M. Iqbal, “Lyft Revenue and Usage Statistics (2020) - Business of Apps,” *Business of Apps*, 2019. https://www.businessofapps.com/data/lyft-statistics/ (accessed Sep. 27, 2019).

[16] H. Baskas, “Luggage storage apps can solve the ‘where do I stash my bags’ problem when traveling,” *CNBC*, 2019. https://www.cnbc.com/2019/03/31/luggage-storage-apps-can-solve-the-where-do-i-stash-my-bags-problem.html (accessed Sep. 27, 2019).

[17] J. Settembre, “Too lazy to carry your bag around? There’s now an app for that,” *MarketWatch*. https://www.marketwatch.com/story/too-lazy-to-carry-your-bag-around-theres-now-an-app-for-that-2018-08-16-1088454 (accessed Sep. 30, 2019).

[18] “How drivers and passengers are paired – Lyft Help.” https://help.lyft.com/hc/en-us/articles/115012926847-How-drivers-and-passengers-are-paired (accessed May 27, 2020).

[19] “Matching | Uber.” https://marketplace.uber.com/matching (accessed Sep. 30, 2019).

[20] AdC, “Relatório sobre Concorrência e Regulação no Transporte de Passageiros em Veículos Ligeiros Dezembro 2016,” 2016. [Online]. Available: www.concorrencia.pt.

[21] “Uber surge map last night (3.5x+) : houston,” *Reddit*, 2017. https://www.reddit.com/r/houston/comments/5sfx2p/uber_surge_map_last_night_35x/ (accessed Nov. 29, 2019).

[22] “Hot spots: Clearer demand, clearer earnings - The Hub,” *Lyft*. https://www.lyft.com/hub/posts/hot-spots-feature (accessed Sep. 30, 2019).

[23] “Quando e onde há maior número de utilizadores? | Motoristas e Estafetas de Entrega - Ajuda Uber,” *Uber*. https://help.uber.com/driving-and-delivering/article/quando-e-onde-há-maior-número-de-utilizadores?nodeId=456fcc51-39ad-4b7d-999d-6c78c3a388bf (accessed Sep. 30, 2019).

[24] Ivan Zhou, “AI-Powered Dynamic Pricing Is Everywhere – SyncedReview – Medium,” 2018. Accessed: Sep. 30, 2019. [Online]. Available: https://medium.com/syncedreview/ai-powered-dynamic-pricing-is-everywhere-4271a9939d11.

[25] “How surge pricing works,” *Uber*. https://www.uber.com/us/en/drive/partner-app/how-surge-works/ (accessed Oct. 26, 2019).

[26] S. Guo *et al.*, “ROD-Revenue: Seeking Strategies Analysis and Revenue Prediction in Ride-on-demand Service Using Multi-source Urban Data,” *IEEE Trans. Mob. Comput.*, pp. 1–1, Jun. 2019, doi: 10.1109/tmc.2019.2921959.

[27] L. De Sousa, R. Sousa, F. Nery, and J. Matos, “Grelhas Geodésicas,” in *IV Conferência Nacional de Cartografia e Geodesia*, 2007, Accessed: Nov. 30, 2019. [Online]. Available: https://www.researchgate.net/publication/307575486_GRELHAS_GEODESICAS.

[28] K. Sahr, D. White, and A. J. Kimerling, “Geodesic Discrete Global Grid Systems Discrete Global Grid Systems: Basic Definitions Discrete Global Grid,” 2003.

[29] X. Zhang and Z. Du, “Spatial Indexing,” *Geogr. Inf. Sci. Technol. Body Knowl.*, no. 4th Quarter 2017

Edition, 2017, doi: 10.22224/gistbok/2017.4.12.

[30] “Overview | S2Geometry.” http://s2geometry.io/about/overview (accessed Oct. 27, 2019).

[31] “S2 Cells | S2Geometry,” Accessed: Oct. 27, 2019. [Online]. Available: https://s2geometry.io/devguide/s2cell_hierarchy.

[32] “S2 Geometry | S2Geometry.” http://s2geometry.io/ (accessed Oct. 27, 2019).

[33] “S2 Cell Statistics | S2Geometry.” http://s2geometry.io/resources/s2cell_statistics (accessed Oct. 27, 2019).

[34] J. Miranda, “Uber Unveils its Realtime Market Platform,” *InfoQ*, 2015. https://www.infoq.com/news/2015/03/uber-realtime-market-platform/ (accessed Oct. 27, 2019).

[35] I. Brodsky, “H3: Uber’s Hexagonal Hierarchical Spatial Index,” pp. 1–10, 2019, Accessed: Oct. 30, 2019. [Online]. Available: https://eng.uber.com/h3/.

[36] “Dymaxion World Map by Richard Buckminster Fuller (327CA),” *Atlas of Places*, 2018. https://atlasofplaces.com/cartography/dymaxion-world-map/ (accessed Oct. 28, 2019).

[37] “H3 - Overview of the H3 Geospatial Indexing System.” https://h3geo.org/docs/core-library/overview (accessed May 07, 2020).

[38] “H3 - Why H3? Use Cases.” https://h3geo.org/docs/usecases (accessed May 07, 2020).

[39] “H3 - Table of Cell Areas for H3 Resolutions.” https://h3geo.org/docs/core-library/restable (accessed May 07, 2020).

[40] “H3 - H3 Index Representations.” https://h3geo.org/docs/core-library/h3indexing (accessed May 07, 2020).

[41] J. P. Snyder, “An equal-area map projection for polyhedral globes,” *Cartographica*, vol. 29, no. 1, pp. 10–21, Oct. 1992, doi: 10.3138/27H7-8K88-4882-1752.

[42] “open-eaggr/Software Design Document.pdf at master · riskaware-ltd/open-eaggr.” https://github.com/riskaware-ltd/open-eaggr/blob/master/Documents/Software Design Document.pdf (accessed May 10, 2020).

[43] B. Bondaruk, S. A. Roberts, and C. Robertson, “Assessing the state of the art in Discrete Global Grid Systems: OGC criteria and present functionality,” *Geomatica*, pp. 1–22, May 2020, doi: 10.1139/geomat-2019-0015.

[44] M. B. J. Purss, R. Gibb, F. Samavati, P. Peterson, and J. Ben, “The OGC® Discrete Global Grid System core standard: A framework for rapid geospatial integration,” in *International Geoscience and Remote Sensing Symposium (IGARSS)*, Nov. 2016, vol. 2016-November, pp. 3610–3613, doi: 10.1109/IGARSS.2016.7729935.

[45] “A Complete Guide to Heatmaps | Tutorial by Chartio.” https://chartio.com/learn/charts/heatmap-complete-guide/ (accessed Jun. 04, 2020).

[46] G. B. Dantzig and J. H. Ramser, “The Truck Dispatching Problem,” *Manage. Sci.*, vol. 6, no. 1, pp. 80–91, Oct. 1959, doi: 10.1287/mnsc.6.1.80.

[47] P. Toth and D. Vigo, “1. An Overview of Vehicle Routing Problems,” in *The Vehicle Routing Problem*,

Society for Industrial and Applied Mathematics, 2002, pp. 1–26.

[48] S. Irnich, P. Toth, and D. Vigo, “Chapter 1: The Family of Vehicle Routing Problems,” in *Vehicle Routing*, Society for Industrial and Applied Mathematics, 2014, pp. 1–33.

[49] Y. Dumas, J. Desrosiers, and F. Soumis, “The pickup and delivery problem with time windows,” *Eur. J. Oper. Res.*, vol. 54, no. 1, pp. 7–22, Sep. 1991, doi: 10.1016/0377-2217(91)90319-Q.

[50] J. F. Cordeau and G. Laporte, “The dial-a-ride problem: Models and algorithms,” *Ann. Oper. Res.*, vol. 153, no. 1, pp. 29–46, Sep. 2007, doi: 10.1007/s10479-007-0170-8.

[51] M. Battarra, J.-F. Cordeau, and M. Iori, “Chapter 6: Pickup-and-Delivery Problems for Goods Transportation,” in *Vehicle Routing*, Society for Industrial and Applied Mathematics, 2014, pp. 161–191.

[52] M. M. Solomon, “ALGORITHMS FOR THE VEHICLE ROUTING AND SCHEDULING PROBLEMS WITH TIME WINDOW CONSTRAINTS.,” *Oper. Res.*, vol. 35, no. 2, pp. 254–265, Apr. 1987, doi: 10.1287/opre.35.2.254.

[53] O. Bräysy and M. Gendreau, “Vehicle routing problem with time windows, Part I: Route construction and local search algorithms,” *Transp. Sci.*, vol. 39, no. 1, pp. 104–118, 2005, doi: 10.1287/trsc.1030.0056.

[54] O. Bräysy and M. Gendreau, “Vehicle routing problem with time windows, Part II: Metaheuristics,” *Transp. Sci.*, vol. 39, no. 1, pp. 119–139, 2005, doi: 10.1287/trsc.1030.0057.

[55] A. Dixit, A. Mishra, and A. Shukla, “Vehicle routing problem with time windows using meta-heuristic algorithms: A survey,” in *Advances in Intelligent Systems and Computing*, 2019, vol. 741, pp. 539–546, doi: 10.1007/978-981-13-0761-4_52.

[56] D. Luxen and C. Vetter, “Real-time routing with OpenStreetMap data,” in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS ’11*, 2011, p. 513, doi: 10.1145/2093973.2094062.

[57] “mapbox/mapbox-gl-directions: Directions plugin for mapbox-gl-js using Mapbox Directions API.” https://github.com/mapbox/mapbox-gl-directions (accessed Oct. 13, 2019).

[58] “Directions | How Mapbox works | Help | Mapbox.” https://docs.mapbox.com/help/how-mapbox-works/directions/ (accessed Jun. 04, 2020).

[59] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, “Contraction hierarchies: Faster and simpler hierarchical routing in road networks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5038 LNCS, pp. 319–333, doi: 10.1007/978-3-540-68552-4_24.

[60] J. Hamme, J. Dibbelt, and M. Baum, “Customizable Route Planning in External Memory,” 2013. Accessed: May 27, 2020. [Online]. Available: www.kit.edu.

[61] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: 10.1007/BF01386390.

[62] “Project-OSRM/osrm-backend Open Source Routing Machine - C++ backend.”

https://github.com/Project-OSRM/osrm-backend (accessed Oct. 12, 2019).

[63] “Shouldn’t CH be much faster than MLD? · Issue #5500 · Project-OSRM/osrm-backend.” https://github.com/Project-OSRM/osrm-backend/issues/5500 (accessed Oct. 12, 2019).

[64] “Traffic · Project-OSRM/osrm-backend Wiki.” https://github.com/Project-OSRM/osrm-backend/wiki/Traffic (accessed Oct. 12, 2019).

[65] B. Miller, “Linux Foundation Revives Mapzen, an Alternative to Giants,” *Government Technology*, 2019. https://www.govtech.com/biz/Linux-Foundation-Revives-Mapzen-an-Alternative-to-Giants.html (accessed Nov. 02, 2019).

[66] J. Shieber, “Mapbox makes another acquisition to bolster its navigation toolkits | TechCrunch,” *TechCrunch*, 2018. https://techcrunch.com/2018/01/04/mapbox-adds-makes-another-acquisition-to-bolster-its-navigation-toolkits/?guccounter=1 (accessed Nov. 19, 2019).

[67] Google Maps API, “Encoded Polyline Algorithm Format | Google Maps APIs | Google Developers,” 2016. https://developers.google.com/maps/documentation/utilities/polylinealgorithm (accessed Nov. 19, 2019).

[68] “Valhalla Documentation.” https://valhalla.readthedocs.io/en/latest/ (accessed Nov. 19, 2019).

[69] “Why tiles? - Valhalla.” https://valhalla.readthedocs.io/en/latest/mjolnir/why_tiles/ (accessed Nov. 19, 2019).

[70] “Tile structure - Valhalla.” https://valhalla.readthedocs.io/en/latest/tiles/ (accessed Nov. 19, 2019).

[71] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968, doi: 10.1109/TSSC.1968.300136.

[72] “Overview of how routes are computed - Valhalla.” https://valhalla.readthedocs.io/en/latest/route_overview/ (accessed Nov. 19, 2019).

[73] “Dynamic costing - Valhalla.” https://valhalla.readthedocs.io/en/latest/sif/dynamic-costing/ (accessed Nov. 18, 2019).

[74] A. Wright and D. Dara-Adams, “World Bank + Mapzen + partners = open traffic data · Mapzen,” *Mapzen*. https://www.mapzen.com/blog/announcing-open-traffic/ (accessed Dec. 01, 2019).

[75] World Bank, “The World Bank Launches New Open Transport Partnership to Improve Transportation through Open Data,” pp. 2016–2018, 2016, Accessed: May 27, 2020. [Online]. Available: https://www.worldbank.org/en/news/press-release/2016/12/19/the-world-bank-launches-new-open-transport-partnership-to-improve-transportation-through-open-data.

[76] “When Traffic Influenced Routing will be available in Valhalla? · Issue #1846 · valhalla/valhalla.” https://github.com/valhalla/valhalla/issues/1846 (accessed Nov. 19, 2019).

[77] “V3Tiles: Remove OSMLR support and embedded traffic segments · Issue #1594 · valhalla/valhalla.” https://github.com/valhalla/valhalla/issues/1594 (accessed Dec. 01, 2019).

[78] D. B. Johnson, “Efficient Algorithms for Shortest Paths in Sparse Networks,” *J. ACM*, vol. 24, no. 1, pp. 1–13, Jan. 1977, doi: 10.1145/321992.321993.

[79] R. W. Floyd, “Algorithm 97: Shortest path,” *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962, doi: 10.1145/367766.368168.

[80] S. Warshall, “A Theorem on Boolean Matrices,” *J. ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962, doi: 10.1145/321105.321107.

[81] “Node - OpenStreetMap Wiki.” https://wiki.openstreetmap.org/wiki/Node (accessed Nov. 19, 2019).

[82] “History - OpenTripPlanner.” http://docs.opentripplanner.org/en/latest/History/ (accessed Nov. 19, 2019).

[83] “Deployments - OpenTripPlanner.” http://docs.opentripplanner.org/en/latest/Deployments/ (accessed Nov. 19, 2019).

[84] “OpenTripPlanner - OpenStreetMap Wiki.” https://wiki.openstreetmap.org/wiki/OpenTripPlanner (accessed Nov. 19, 2019).

[85] “Bibliography - OpenTripPlanner.” http://docs.opentripplanner.org/en/latest/Bibliography/ (accessed Nov. 19, 2019).

[86] M. Dornhofer, W. Bischof, and E. Krajnc, “Comparison of Open Source routing services with OpenStreetMap Data for blind pedestrians PgRouting, OpenTripPlanner and OpenSourceRoutingMaschine,” 2014.

[87] X. Liu, “open-source-spec/osrm-vs-valhalla.md at master · Telenav/open-source-spec.” https://github.com/Telenav/open-source-spec/blob/master/osrm/doc/osrm-vs-valhalla.md (accessed May 11, 2020).

[88] M. Goodfellow, “This is what taxi drivers know that you don’t | The Independent,” *The Independent*. https://www.independent.co.uk/news/business/analysis-and-features/this-is-what-taxi-drivers-know-that-you-dont-a6713906.html (accessed May 08, 2020).

[89] T. Michael, “What is The Knowledge taxi test and why is the exam taken by London’s black cab drivers so tough?,” *The Sun*. https://www.thesun.co.uk/news/3307245/the-knowledge-taxi-test-london-black-cab-drivers-exam/ (accessed May 08, 2020).

[90] M. Yang, Y. Liu, and Z. You, “The reliability of travel time forecasting,” *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 1, pp. 162–171, Mar. 2010, doi: 10.1109/TITS.2009.2037136.

[91] W. Y. Loh, “Classification and regression trees,” *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 1, no. 1, pp. 14–23, Jan. 2011, doi: 10.1002/widm.8.

[92] R. S. Brid, “Decision Trees — A simple way to visualize a decision,” *Medium*, 2018. https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb (accessed May 08, 2020).

[93] D. H. Moore, “Classification and regression trees,” *Cytometry*, vol. 8, no. 5, pp. 534–535, Sep. 1987, doi: 10.1002/cyto.990080516.

[94] T. K. Ho, “Random decision forests,” in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 1995, vol. 1, pp. 278–282, doi: 10.1109/ICDAR.1995.598994.

[95] M. Elhenawy, H. Chen, and H. Rakha, “Random forest travel time prediction algorithm using

spatiotemporal speed measurements," *21st World Congr. Intell. Transp. Syst. ITSWC 2014 Reinventing Transp. Our Connect. World*, 2014.

[96] Y. Zhang and A. Haghani, "A gradient boosting method to improve travel time prediction," *Transp. Res. Part C Emerg. Technol.*, vol. 58, pp. 308–324, Sep. 2015, doi: 10.1016/j.trc.2015.02.019.

[97] W. Song and Y. Zhou, "Road Travel Time Prediction Method Based on Random Forest Model," in *Smart Innovation, Systems and Technologies*, 2020, vol. 165, pp. 155–163, doi: 10.1007/978-981-15-0077-0_17.

[98] "Modern Car Navigation Systems and Their Features | Automotive Industry | Infopulse." https://www.infopulse.com/blog/modern-car-navigation-systems-and-their-features/ (accessed Nov. 19, 2019).

[99] "Overview - Nominatim 3.4.2." http://nominatim.org/release-docs/latest/develop/overview/ (accessed Nov. 19, 2019).

[100] L. W. Carbaugh and R. W. Marx, "The TIGER system: A census bureau innovation serving data analysts," *Gov. Inf. Q.*, vol. 7, no. 3, pp. 285–306, Jan. 1990, doi: 10.1016/0740-624X(90)90026-K.

[101] "GB Postcodes - Nominatim 3.4.2." http://nominatim.org/release-docs/latest/data-sources/GB-Postcodes/ (accessed Nov. 19, 2019).

[102] "Wikipedia & Wikidata - Nominatim 3.4.2." http://nominatim.org/release-docs/latest/data-sources/Wikipedia-Wikidata/ (accessed Nov. 19, 2019).

[103] "komoot/photon: an open source geocoder for openstreetmap data." https://github.com/komoot/photon (accessed May 27, 2020).

[104] N. Diakopoulos, "How Uber surge pricing really works," *The Washington Post*. https://www.washingtonpost.com/news/wonk/wp/2015/04/17/how-uber-surge-pricing-really-works/ (accessed May 27, 2020).

[105] "Agile, Scrum, Kanban: What Are They? | Toptal." https://www.toptal.com/project-managers/technical/agile-scrum-kanban-what-do-they-mean (accessed Apr. 09, 2020).

[106] "O que é container Linux e quais as vantagens," *RedHat*. https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container (accessed Apr. 02, 2020).

[107] Docker Inc, "What is a Container? | App Containerization | Docker," *Docker.Com*, 2020. https://www.docker.com/resources/what-container (accessed May 27, 2020).

[108] Docker, "Docker overview | Docker Documentation," *Docker.Com*, 2018. https://docs.docker.com/get-started/overview/#docker-objects (accessed Apr. 02, 2020).

[109] GOLANG, "Frequently Asked Questions (FAQ) - The Go Programming Language," 2013. https://golang.org/doc/faq#history (accessed May 27, 2020).

[110] S. Nilsson, "Why Go? – Key advantages you may have overlooked." https://yourbasic.org/golang/advantages-over-java-python/ (accessed May 14, 2020).

[111] K. Rogovoy, "Here are some amazing advantages of Go that you don't hear much about," *FreeCodeCamp*, 2018. https://www.freecodecamp.org/news/here-are-some-amazing-advantages-of-

go-that-you-dont-hear-much-about-1af99de3b23a/ (accessed May 14, 2020).

[112] M. McGranaghan, “Go by Example: Goroutines,” *Go by Example*, 2017. https://gobyexample.com/goroutines (accessed May 27, 2020).

[113] M. McGranaghan, “Go by Example: Channels,” *Go by Example*. https://gobyexample.com/channels (accessed May 27, 2020).

[114] L. Ryan, “The gRPC Blog – gRPC Motivation and Design Principles.” https://grpc.io/blog/principles/ (accessed May 27, 2020).

[115] “gRPC – Guides.” https://grpc.io/docs/guides/ (accessed May 27, 2020).

[116] Google Development, “Frequently Asked Questions - Protocol Buffers — Google Developers.” https://developers.google.com/protocol-buffers/docs/faq (accessed May 27, 2020).

[117] “Language Guide (proto3) | Protocol Buffers | Google Developers.” https://developers.google.com/protocol-buffers/docs/proto3 (accessed May 27, 2020).

[118] Microsoft, “Compare gRPC services with HTTP APIs | Microsoft Docs,” 2019. https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.1 (accessed May 14, 2020).

[119] “grpc-ecosystem/grpc-gateway: gRPC to JSON proxy generator following the gRPC HTTP spec.” https://github.com/grpc-ecosystem/grpc-gateway (accessed May 27, 2020).

[120] C. Vagos, “Generating gRPC/Protobuf definitions to Golang and Android,” *LUGGit*, 2020. https://medium.com/luggit/generating-grpc-protobuf-definitions-to-golang-and-android-85d5ba9bddc3 (accessed May 14, 2020).

[121] “Replication - Tile38.” https://tile38.com/topics/replication/ (accessed May 27, 2020).

[122] “Geofencing - Tile38.” https://tile38.com/topics/geofencing/ (accessed May 27, 2020).

[123] “Roaming Geofences - Tile38.” https://tile38.com/topics/roaming-geofences/ (accessed May 27, 2020).

[124] “Commands - Tile38.” https://tile38.com/commands/#search (accessed May 27, 2020).

[125] Microsoft Corporation, “Publisher-Subscriber pattern - Azure Architecture Center | Microsoft Docs,” 2018. https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber (accessed May 27, 2020).

[126] J. Kreps, “Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines) | LinkedIn Engineering.” https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines (accessed May 27, 2020).

[127] “Apache Kafka.” https://kafka.apache.org/intro (accessed May 27, 2020).

[128] W. (AQR) McKinney, “Pandas : a Python Data Analysis Library,” *New York*, pp. 1–26, 2009, Accessed: May 27, 2020. [Online]. Available: https://pandas.pydata.org/about/.

[129] “Valhalla Turn-by-Turn Navigation · Mapzen.” https://www.mapzen.com/products/mobility/turn-by-turn/ (accessed Jun. 05, 2020).

[130] “Running Automated Tasks with a CronJob - Kubernetes.”

https://kubernetes.io/docs/tasks/job/automated-tasks-with-cron-jobs/ (accessed May 27, 2020).

[131] R. Fielding *et al.*, "http/1.1: Method Definitions." 1999, Accessed: May 27, 2020. [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

[132] "Isochrones - Valhalla." https://valhalla.readthedocs.io/en/latest/thor/isochrones/ (accessed May 27, 2020).

[133] ISO, "ISO 639-1:2002 Codes for the Representation of Names of Languages - Part 1: Alpha-2 code." 2002, Accessed: May 27, 2020. [Online]. Available: https://www.iso.org/standard/22109.html.

[134] International Organization for Standardization, "ISO - ISO 8601 Date and time format," 2019.

[135] "Distance on a sphere: The Haversine Formula | GeoNet, The Esri Community | GIS and Geospatial Professional Community." https://community.esri.com/groups/coordinate-reference-systems/blog/2017/10/05/haversine-formula (accessed May 27, 2020).

[136] "K-fold cross-validation method. | Download Scientific Diagram." https://www.researchgate.net/figure/K-fold-cross-validation-method_fig2_331209203 (accessed Jun. 03, 2020).

[137] "What is R-Squared? | Displayr." https://www.displayr.com/what-is-r-squared/ (accessed Jun. 03, 2020).

[138] "Assign CPU Resources to Containers and Pods - Kubernetes." https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/#cpu-units (accessed May 28, 2020).

[139] Kubernetes, "Deployments - Kubernetes," 2020. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#scaling-a-deployment (accessed May 28, 2020).

[140] Linux Foundation, "Service - Kubernetes," 2019. https://kubernetes.io/docs/concepts/services-networking/service/ (accessed May 28, 2020).

[141] "Dynamic Costing - Valhalla." https://valhalla.readthedocs.io/en/latest/sif/dynamic-costing/ (accessed Jun. 05, 2020).

[142] "Google Maps." https://www.google.com/maps/ (accessed Jun. 05, 2020).

[143] StackOverflow, "Stack Overflow Developer Survey 2019," *Stack Overflow Insights*, 2019. https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages (accessed Jun. 01, 2020).

Anexo A

Os seguintes esquemas de dados foram implementados de modo a auxiliar a sua modelação numa base de dados relacional.

Serviço de Geofencing

Para este serviço, optou-se por realizar a modelação de dados de modo a poderem ser armazenadas as áreas de operação, bem como locais estratégicos a elas associados como estações de comboio/metro, aeroportos e seus terminais, parceiros (empresas que sugerem aos seus clientes a utilização do serviço) e eventos. Nestes dois últimos foram também modelados locais de ponto de encontro para os Keepers de modo a estes poderem ser sugeridos caso ocorram constrangimentos de circulação (locais proibidos a veículos, entre outros). No caso dos parceiros, por exemplo empresas no ramo do alojamento, existe a possibilidade de estas terem diversos estabelecimentos em várias áreas de operação ou até mesmo mais que um estabelecimento numa área de operação, pelo que também houve a necessidade de efetuar a sua modelação. Todas as entidades aqui modeladas poderão ter nomenclaturas diferentes de acordo com a linguagem, e desse modo também foram acrescentadas tabelas para tradução de texto.

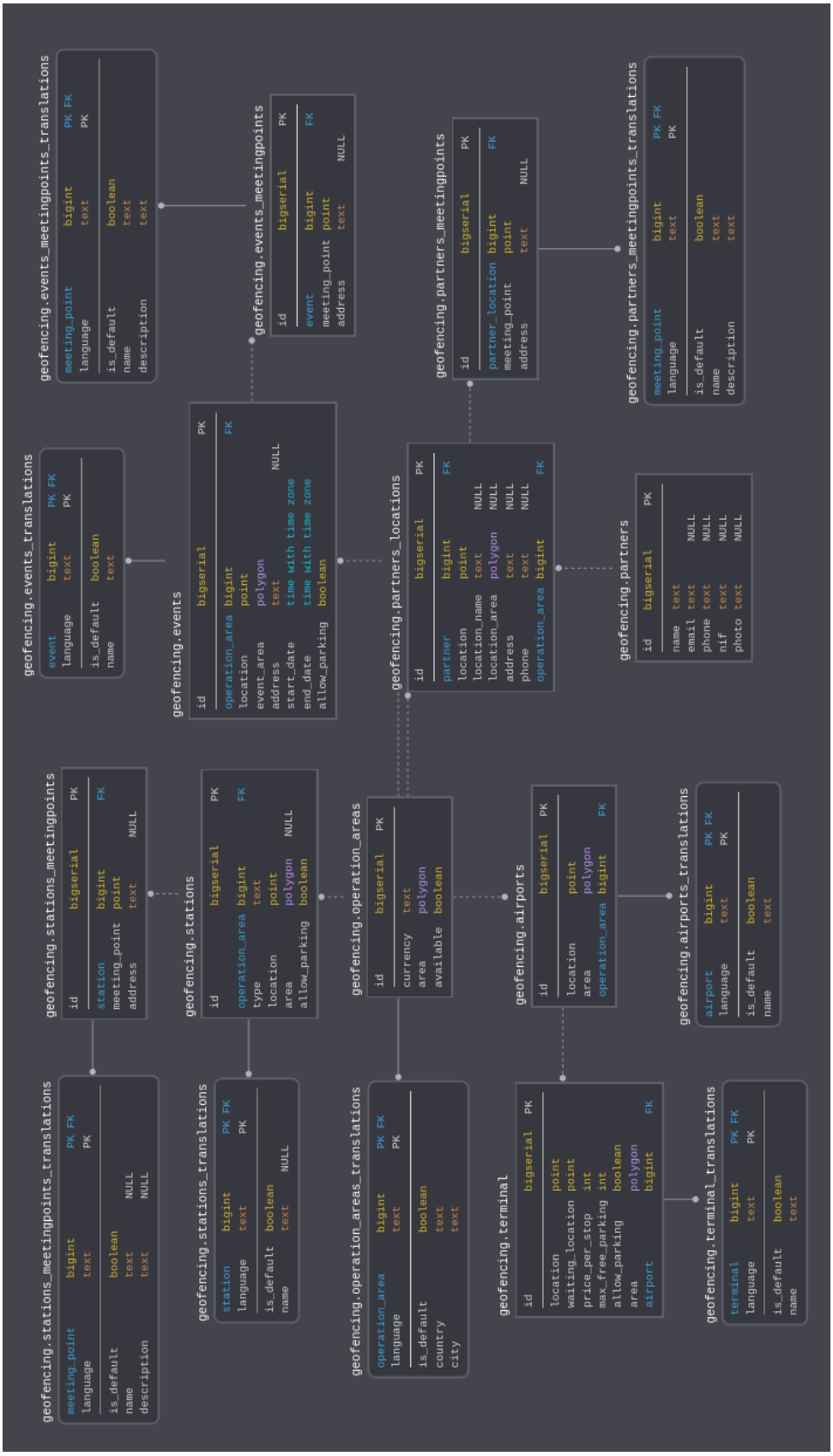

Figura 112 - Esquema da base de dados do serviço de Geofencing

Serviço de Correção de Posicionamento de Condutores

Para modelar os dados deste serviço foram criadas tabelas de factos (na Figura 113 indicadas com o sufixo "*_ft*"), nomeadamente para os serviços e para os eventos, bem como várias dimensões (na Figura 113 indicadas com o sufixo "*_dm*"). Estes em conjunto fazem com que os dados sejam úteis para análise e processamento, o que irá ser útil tanto para os desafios e objetivos propostos por esta dissertação, bem como para aspetos de gestão da operação, onde seja importante recolher métricas que permitam avaliar KPIs relativamente aos Keepers e aos próprios serviços. Também serão úteis para potenciais estratégias de marketing que a empresa venha a realizar, no sentido de, por exemplo, poder-se associar uma data e hora em conjunto com chegadas/partidas de voos numa área de operação, uma vez que os aeroportos são um dos locais onde existe maior probabilidade de haver um pedido de serviço. Outro caso de uso é por exemplo poder relacionar o número de pedidos que ocorrem quando está mau tempo.

Escolheu-se assim utilizar o modelo em estrela para o esquema de dados, o que fez com que fosse possível recolher métricas relativamente aos serviços, fornecendo indicadores que permitem medir a sua performance, sendo dados que oferecem apoio à decisão.

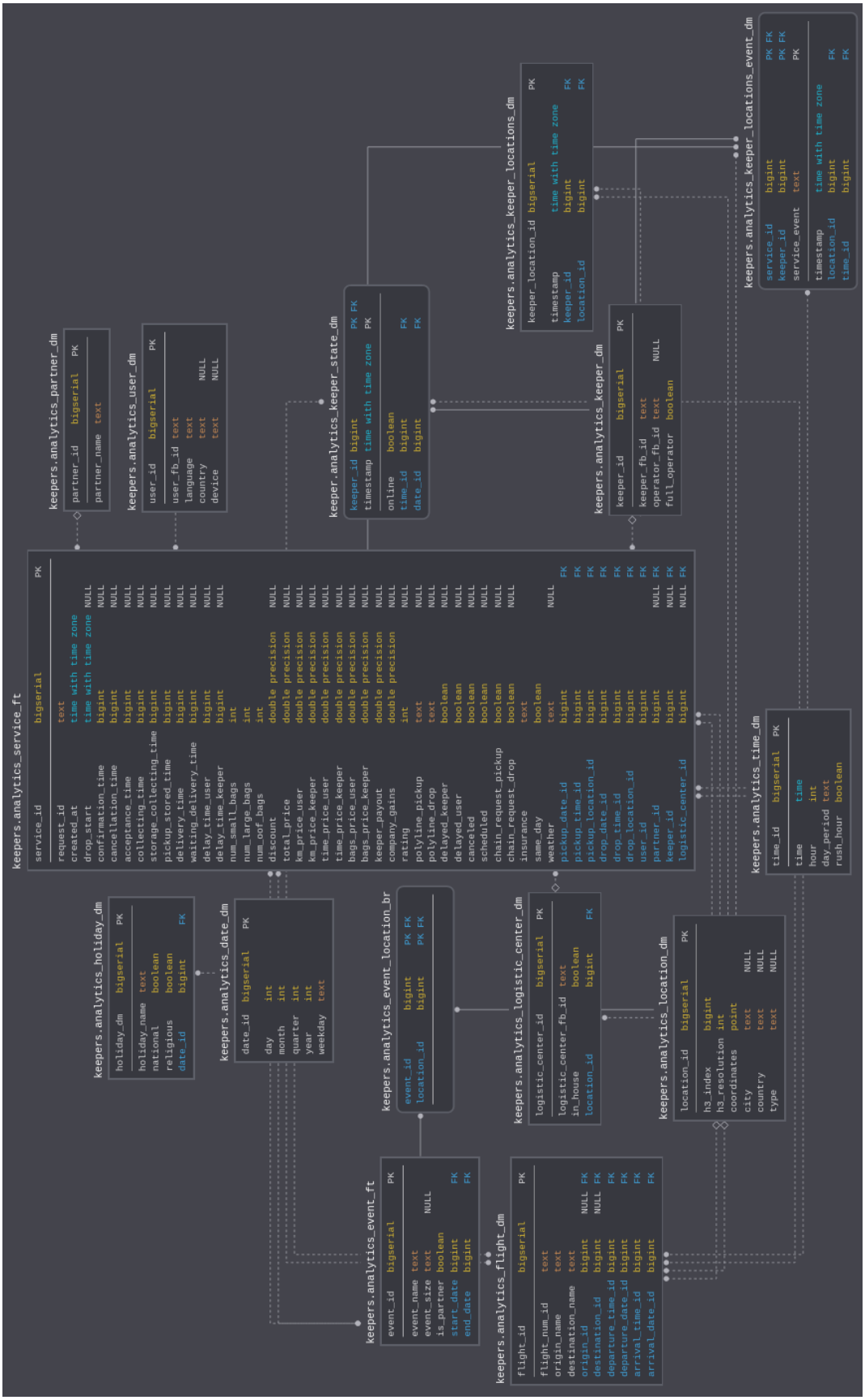

Figura 113 - Esquema da base de dados do Sistema de Correção de Posicionamento de Condutores