UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Transport-on-Demand (ToD) Planner for MaaS - Resources Management

Mário José Martins Teixeira

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Versão pública

Trabalho de Projeto orientado por:
João Carlos Balsa da Silva

2020

Agradecimentos

Não poderia deixar de começar por agradecer àquela que foi como a minha segunda casa nestes últimos anos, que me deu inúmeras amizades e que me fez crescer muito enquanto ser humano, a nossa *mui nobre* Faculdade de Ciências da Universidade de Lisboa.

Em segundo lugar, gostaria de agradecer à Card4B - Systems S.A. e a todos aqueles que fazem parte desta equipa, por me terem dado a possibilidade de realizar este projeto. Em particular, agradeço ao Eng. João Almeida e ao João Zenha pela paciência e por toda a ajuda que me deram.

De seguida, quero agradecer ao meu orientador da Faculdade de Ciências da Universidade de Lisboa, o Prof. Doutor João Carlos Balsa da Silva, por estar sempre disponível para responder a todas as minhas dúvidas e pela sua orientação ao longo destes últimos meses.

Quero deixar um especial agradecimento à Catarina Guerreiro, pelo carinho e amor que me dá, por todo o seu apoio, por ouvir-me e por dar-me sempre os melhores conselhos.

Por último, quero agradecer aos meus pais pela educação e valores que me deram, à minha irmã e cunhado pela amizade e cumplicidade e à minha sobrinha linda por colocar-me um sorriso na cara sempre que estou com ela.

À minha famúlia, namorada e amigos.

Resumo

O presente relatório descreve o trabalho desenvolvido no projeto *Transport-on-Demand (ToD) Planner for MaaS - Resources Management*, na empresa Card4B - Systems, S.A., cujo tema relaciona-se com os transportes públicos e tem o intuito de desenvolver soluções que melhorem as atuais funcionalidades de alguns produtos.

Nos dias de hoje, existem atributos essenciais que procuramos no nosso dia a dia, tais como rapidez, eficiência e opção de escolha. O setor da mobilidade adaptou-se a esta tendência, ao oferecer serviços integrados e inteligentes. Com o passar dos anos, os serviços adaptaram-se às necessidades dos passageiros, e foi então que nasceu o conceito *Mobility as a Service* (MaaS). Os utilizadores têm acesso a um único serviço que resulta na integração de várias opções de transporte, e que, graças ao avanço da tecnologia, é-lhes permitido definir a origem e destino pretendidos, saber previamente o preço que vão pagar pela viagem e a duração da mesma.

Este projeto veio trazer melhorias a produtos já existentes, como é o caso de um *widget web* que permite aos utilizadores consultar trajetos e horários de autocarros de acordo com os seguintes critérios: carreira, origem e destino. É de realçar ainda a introdução de um novo conceito – paragem satélite – que funciona como uma *flag* dessa paragem, permitindo assim ativá-la ou desativá-la. Posteriormente foi desenvolvido um ecrã que permite desenhar o percurso que cada autocarro irá fazer, para mais tarde poderem ser comparados aos trajetos que foram realmente executados.

Finalmente foi necessário substituir uma tecnologia que deixará de ter suporte a partir de 2021, o *Silverlight*. A substituição foi feita em 2 partes: a primeira realizada em *Blazor* e a segunda recorrendo a *Javascript* e *jQuery*.

Palavras-chave: Transport on Demand, Mobility as a Service, Utilizadores, Tecnologia

Abstract

This report describes the work developed in the Transport-on-Demand (ToD) Planner for MaaS - Resources Management project, at Card4B - Systems, S.A., whose theme is related to public transport and aims to develop solutions that improve the current features of some products.

Nowadays, there are essential attributes that we look for in our daily lives, such as quickness, efficiency and flexibility of options. The mobility industry has adapted to this trend by offering integrated and intelligent services. Over the years, services have adapted to the passengers needs and it was then that the Mobility as a Service (MaaS) concept was born. Users have access to a single service that results in the integration of several transport options which, due to the technology advancement, allow them to define their desired origin and destination, previously know the price they will pay for the trip and the duration of it.

This project has brought improvements to existing products, such as a web widget that allows users to consult bus routes and schedules according to the following criteria: career, origin and destination. It is also worth to mention the introduction of a new concept - satellite stop - which acts as a flag for that stop, allowing it to be enabled or disabled. Posteriorly, a screen was developed to allow the drawing of routes that each bus will take, so that later they can be compared to the routes that were executed.

Finally, it was necessary to replace a technology that will no longer be supported from 2021, Silverlight. The replacement was made in 2 parts: the first made in Blazor and the second using Javascript and jQuery.

Keywords: Transport on Demand, Mobility as a Service, Users, Technology

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas

AJAX Asynchronous JavaScript And XML. 25, 35

API Application Programming Interface. xv, 2, 26, 27, 28, 29, 40, 41, 51, 53, 55

CAN-BUS Controller Area Network Bus. 24

Card4B Card4B - Systems, S.A. 3

CIM Comunidade Intermunicipal do Médio Tejo. 6

CSS Cascading Style Sheets. 40, 42

CSV Comma-separated Values. 16

DOM Document Object Model. 39, 41, 45, 47

ELK Elasticsearch, Logstash, and Kibana. 8

ETA Estimate Time of Arrival. 11, 22, 23

GPS Global Positioning System. 11, 17, 19, 25

GTFS General Transit Feed Specification. xv, 31, 32, 33

HTML HyperText Markup Language. 36, 37, 39, 40, 43, 44, 45, 47, 51

ID Identificador. 29, 30

IMT Instituto da Mobilidade e dos Transportes. 7

JSON JavaScript Object Notation. 30

JSX JavaScript XML. 47

KML Keyhole Markup Language. 31

KPI Key Performance Indicator. 5

LAN Local Area Network. 13

MaaS Mobility as a Service. 1, 2, 53

MAN Metropolitan Area Network. 13

MVVM Model-view-viewmodel. 44

SEO Search Engine Optimization. 47

SP Stored Procedure. 16, 28

SQL Structured Query Language. 16, 28, 29

SVG Scalable Vector Graphics. 36

ToD Transport-on-Demand. 1, 2, 5, 53

TPA Terminal de Pagamento Automático. 19

TVDE Transporte individual e remunerado de passageiros em veículos descaracterizados a partir de plataforma eletrónica. 7

UI User Interface. 40, 41, 42, 43, 45

WAN Wide Area Network. 13

WKT Well-known Text. 31, 32, 35

WPF Windows Presentation Foundation. 42, 43

XAML Extensible Application Markup Language. 37, 42

XHTML Extensible Hypertext Markup Language. 45

XML Extensible Markup Language. 18, 42, 45

Capítulo 1

Introdução

Este projeto foi desenvolvido no âmbito da disciplina de Dissertação/Projeto de Engenharia Informática do 2º ano de Mestrado em Engenharia Informática, no ramo de Engenharia de Software, na Faculdade de Ciências da Universidade de Lisboa.

O Transporte a Pedido (*Transport-on-Demand* - ToD) é um tipo de serviço relativamente recente nos serviços públicos de mobilidade. Em vez dos habituais autocarros com horários e trajetos previamente definidos (serviços regulares), o cliente pode organizar a sua viagem de forma independente.

Desde há vários anos que os sistemas ToD estão operacionais para pessoas com deficiência que são fisicamente incapazes de usar o transporte público comum [2]. Atualmente, este tipo de serviço está muito associado ao *smartphone*, pois com poucos cliques é possível chamar um motorista que nos leve ao destino pretendido e escolher onde irá ser o ponto de recolha, através das inúmeras aplicações que existem no mercado.

Com os diferentes modos de transporte e serviços de mobilidade disponíveis, surgiu o conceito de *Mobility as a Service* (MaaS) que combina e fornece esta variedade de serviços e as necessidades de transporte do utilizador numa única interface [3].

1.1 Motivação

Os transportes a pedido distinguem-se dos transportes tradicionais [4], pelas seguintes razões:

- A requisição do transporte é feita do lado da população;
- A disponibilidade do serviço é gerida por uma tabela de rotas e horários construída de acordo com as requisições efetuadas;
- Uma considerável redução de custos para os operadores do transporte, proveniente da redução de quilómetros efetivamente percorridos pelas viaturas.

Em conjunto com os sistemas MaaS, o objetivo de ambos é melhorar o acesso e a mobilidade das pessoas dentro das cidades. Segundo [5], a capital portuguesa está avaliada como a 74ª cidade com trânsito mais caótico. O estudo efetuado analisa vários parâmetros como o número de carros, tempo que os condutores passam ao volante, a qualidade das estradas e dos transportes públicos, a idade dos carros, qualidade do ar, custos de estacionamento, número de acidentes, custos de combustíveis, entre outros. Lisboa destaca-se pelo número de carros *per capita* (0,44) superior a cidades como Barcelona (0,39), Berlim (0,29) ou Estocolmo (0,24). Atenas, a capital grega, apresenta o maior número de automóveis por habitante: 0,77.

Outro dos aspetos a ter em consideração como uma vantagem na implementação de sistemas ToD e MaaS, é a redução da poluição atmosférica. As fontes das principais emissões poluentes, segundo [6], são:

- A queima de combustíveis fósseis na geração de eletricidade, nos transportes, na indústria e nos aglomerados domésticos;
- A utilização de solventes nas indústrias, especialmente a química e a extrativa;
- A agricultura e o tratamento de resíduos.

Há ainda outras causas, fora do domínio da ação humana, que também contribuem. As erupções vulcânicas, as poeiras transportadas pelo vento, a água do mar vaporizada e as emissões de compostos orgânicos das plantas são fontes naturais deste problema. No nosso país, as áreas metropolitanas de Lisboa e do Porto são as mais pressionadas por atividade industrial e, principalmente, por um tráfego urbano intenso. Por consequência disto, a qualidade do ar nestas regiões é muito pobre.

1.2 Objetivos

Este projeto incidiu na melhoria da flexibilidade quanto às dimensões dos serviços (acréscimo/supressão de paragens e alteração de trajetos/horários) e de recursos para a execução dos mesmos. Posto isto, foram definidos os seguintes objetivos:

- Receção de serviços a realizar e alocação de meios, viaturas e motoristas em função dos mesmos;
- Flexibilização das regras de alocação de recursos com base em motores e APIs (*Application Programming Interface*) abertas (como o *OpenStreetMap*) e flexibilidade do *workflow* no *BackOffice*;
- Ver e editar informação dos operadores e motoristas sobre os serviços previstos;
- Iniciação, execução e conclusão de serviços;

- Seleção de uma tecnologia de substituição no *BackOffice*;
- Integração de funcionalidades no *BackOffice*.

1.3 Instituição de acolhimento

O projeto foi desenvolvido na Card4B - Systems, S.A.[1] (Card4B), uma empresa fundada em 2007 que tem vindo a crescer de forma expressiva e consistente a cada ano. A empresa nasceu após a identificação da necessidade de soluções para uma "nova cultura de mobilidade" nos centros urbanos, e está focada no fornecimento de componentes de *software* e serviços especializados de soluções integradas de mobilidade e *city-services*, como transportes públicos, portagens, táxis, entre outros.

1.4 Estrutura do documento

Os capítulos que se seguem estão organizados da seguinte forma:

- Capítulo 2 - Consiste na apresentação de alguns trabalhos e artigos já existentes, relacionados com o conteúdo deste projeto;
- Capítulo 3 - São descritos os sistemas onde este projeto está inserido e também os desafios enfrentados ao longo do trabalho;
- Capítulo 4 - Ilustra as soluções implementadas nos sistemas da empresa Card4B, fazendo referência às tecnologias e estratégias utilizadas;
- Capítulo 5 - Identificação das principais conclusões extraídas da realização deste projeto e considerações sobre trabalho futuro a realizar.

[1] https://www.card4b.pt/pt/about.html

Capítulo 2

Trabalho relacionado

2.1 ToD

Um estudo de Linh N. K. Duong, Lincoln C. Wood, Jason X. Wang e William Y. C. Wang em [7], fornece um caso aprofundado, que teve como objetivo identificar e entender as principais variáveis que têm impacto no desempenho de serviços ToD e examinar como estes podem ser reduzidos. Os problemas enfrentados neste caso envolveram o cumprimento dos níveis de serviço/KPIs (*Key Performance Indicators* - Indicadores Chave de Performance), especificamente reduzindo a incidência de atrasos de serviço. Procurou-se responder às seguintes perguntas:

1. Que fatores contribuem para os atrasos?
2. Como é que estes fatores se relacionam com os pedidos em atraso?
3. Como é que ações de gestão podem ser tomadas para permitir que a empresa cumpra os seus KPIs?

Posto isto, foi proposto um modelo (*"to be" model*) composto por 3 KPIs, de forma a melhorar o desempenho de serviços ToD. Estes 3 *milestones* são:

- *Time-to-Arrival* - o veículo chegou ao local do cliente. Este é o KPI principal e deve ser inferior a 60 minutos;
- *Time-to-Load* - quando o veículo chegou até ao cliente e está pronto a levá-lo ao destino;
- *Total time* - o cliente está entregue e o veículo está pronto para iniciar o próximo serviço.

Os resultados indicaram que uma pequena mudança na gestão e no uso das informações pode oferecer benefícios significativos no desempenho geral da prestação de serviços. Concluiu-se então que:

- Os problemas na localização e na partilha e uso de informação são mais importantes;
- Ter e usar informações precisas sobre os requisitos do consumidor é fundamental, pois nem todos os veículos são iguais;
- Alocar um veículo a um serviço é uma atividade crucial, suportada pela partilha de informações entre empresas.

2.1.1 *Link* - estamos todos ligados

Em Portugal, existe uma nova campanha de Transporte a Pedido chamada "Link - estamos todos ligados", como descrito em [8], onde os utilizadores vão poder circular entre as cidades do Médio Tejo a partir de Dezembro de 2019. O projeto irá conter dois circuitos, o primeiro irá ligar Abrantes - Tomar - Ourém - Fátima e o segundo ligará Abrantes - Entroncamento - Torres Novas - Fátima.

O objetivo da Comunidade Intermunicipal do Médio Tejo (CIM) é bastante claro: melhorar a mobilidade nas ligações entre cidades do Médio Tejo, complementando assim os serviços existentes de transporte regular de passageiros e os serviços de transporte a pedido de âmbito municipal.

Apesar de ainda ser um serviço "piloto", é um projeto muito inovador a nível nacional e uma eventual referência para uma futura replicação em outros locais do país.

2.1.2 Plataforma de Mobilidade Como Serviço do Alentejo - Projeto Piloto "Transporte a Pedido"

No seguimento do "Link - estamos todos ligados", foi apresentado em maio de 2019 a Plataforma de Mobilidade Como Serviço do Alentejo - Projeto Piloto "Transporte a Pedido" [9]. Tal como o anterior, este tem como objetivos garantir uma resposta adequada e complementar às necessidades dos cidadãos e, ainda, disponibilizar uma solução de "Mobilidade Como Serviço do Alentejo".

Os municípios que aderiram a este serviço são: Moura, Reguengos de Monsaraz, Beja, Odemira e Mértola. Futuramente, pretende-se alargar esta plataforma a todos os concelhos da Região do Alentejo.

2.1.3 Uber

A Uber[2] é uma empresa tecnológica que surgiu em Portugal em 2014, e que permite requisitar viagens "com apenas um clique" [10]. O pedido é enviado ao motorista mais próximo e, uma vez aceite a viagem, são dadas algumas informações

[2] https://www.uber.com/pt/pt-pt/

ao utilizador como o tempo que o veículo demorará a chegar, nome do motorista e marca/modelo/matrícula da viatura.

Este tipo de empresas teve um aumento exponencial e, segundo dados do Instituto da Mobilidade e dos Transportes (IMT) [11], a 30 de junho de 2019, existiam no país 4686 empresas de veículos TVDE (Transporte individual e remunerado de passageiros em veículos descaracterizados a partir de plataforma eletrónica) registados com cerca de 13015 motoristas certificados. Representando assim no mercado do transporte individual de passageiros, 32% do universo de empresas e 33,5% dos motoristas.

2.1.3.1 Solução

Apesar do objetivo da Uber passar por manter uma interface simples, por trás dela existem sistemas complexos que permitem mantê-la ativa, lidar com interações difíceis e atender a grandes quantidades de dados [12]. Por este motivo, é uma boa razão para darmos uma especial atenção à maioria das tecnologias usadas na sua implementação.

Infraestrutura

O negócio está implementado num modelo de nuvem híbrida (*hybrid cloud model*), através da combinação entre fornecedores de nuvem (*cloud providers*) e vários centros de dados (*data centers*) ativos. Se um centro de dados falhar, as viagens (e todos os serviços que lhes estão associados) são transferidas para outro. As cidades são atribuídas ao centro de dados mais próximo (geograficamente) porém, cada cidade tem *backup* num centro de dados diferente (noutro local). A infraestrutura existe graças a uma mistura entre ferramentas internas e *Terraform*[3].

Armazenamento

Inicialmente existia apenas uma única instância de *PostgreSQL*[4] que era responsável pelo armazenamento dos dados. No entanto, com o crescimento tão rápido que ocorreu, houve uma necessidade de aumentar a capacidade disponível em disco e diminuir os tempos de resposta do sistema. Atualmente é utilizado *Schemaless* (integrado no *MySQL*[5]) para armazenamento de dados de longo prazo e, *Riak*[6] e *Cassandra*[7] que trouxeram velocidade e desempenho. Para distribuir armazenamento e

[3] https://www.terraform.io/
[4] https://www.postgresql.org/
[5] https://www.mysql.com/
[6] https://riak.com/
[7] https://cassandra.apache.org/

analisar dados complexos é utilizada uma *warehouse* de *Hadoop*[8]. Para tratar dos problemas de cache e dos pedidos em fila de espera é utilizado *Redis*[9]. Através de *Twemproxy*[10] obteve-se um algoritmo de *hash* consistente que trouxe escalabilidade à camada da cache. Por fim, com o uso de *Celery workers*[11] foi possível processar operações assíncronas de *workflow* recorrendo a instâncias de *Redis*.

Logging

Os serviços interagem entre si e com dispositivos móveis, o que permite obter informações valiosas para usos internos como *debugging* ou alguns casos de negócio. São utilizados vários *clusters Kafka*[12], para *log*, e os dados são armazenados no *Hadoop* ou num *webservice* para armazenamento em ficheiros. Estes dados também são tratados em tempo real por vários serviços e são indexados numa pilha ELK[13] (*Elasticsearch, Logstash, and Kibana*) para pesquisa e visualização.

App provisioning

São utilizados *containers* de *Docker*[14] no *Mesos*[15] que permitem executar microsserviços com configurações consistentes de forma escalável, tendo ainda a ajuda da ferramenta *Aurora*[16] para serviços de longa duração e *cron jobs*[17].

Routing e *Service Discovery*

Uma arquitetura orientada a serviços (SOA[18]) é o que torna a descoberta e o roteamento de serviços cruciais para o sucesso da Uber. Para que os serviços sejam capazes de comunicar entre eles, na rede complexa que existe, é usada uma combinação de *HAProxy*[19] e *Hyperbahn*[20].

Desenvolvimento e implementação

[8] https://hadoop.apache.org/
[9] https://redis.io/
[10] https://github.com/twitter/twemproxy
[11] https://docs.celeryproject.org/en/stable/userguide/workers.html
[12] https://kafka.apache.org/
[13] https://www.elastic.co/
[14] https://github.com/moby/moby
[15] http://mesos.apache.org/
[16] http://aurora.apache.org/
[17] https://en.wikipedia.org/wiki/Cron
[18] https://en.wikipedia.org/wiki/Service-oriented_architecture
[19] https://www.haproxy.com/
[20] https://github.com/uber-archive/hyperbahn

É utilizado o *Phabricator*[21] que possibilita várias operações internas, desde a revisão de código até à documentação e automatização de processos. A pesquisa no código interno é feita com a ajuda de *OpenGrok*[22]. O desenvolvimento é feito maioritariamente em máquinas virtuais que correm num *cloud provider* ou num computador de um *developer*. Para uma integração contínua é utilizado *Jenkins*[23]. Através da combinação entre *Packer*[24], *Vagrant*[25], *Boto*[26] e *Unison*[27] foram criadas ferramentas de construção, gestão e desenvolvimento em máquinas virtuais. É ainda utilizado *Clusto*[28] para gerir o *stock* durante o desenvolvimento e *Puppet*[29] para gerir a configuração do sistema.

Linguagens

Para baixo nível é utilizado *Python*, *Node.js*, *Go* e *Java*. Inicialmente começou por utilizar-se *Node.js* e *Python* e, posteriormente, para aumentar o desempenho, foi adotado a utilização de *Go*, que trouxe eficiência, simplicidade e velocidade de execução, e *Java*, para tirar vantagem do ecossistema de *open source* e por integrar-se com tecnologias externas como *Hadoop*. O código *Python* foi sendo retirado e substituído à medida que o código base foi dividido em microsserviços (o que aumentou o rendimento). É utilizado *Tornado*[30] com *Python* porém, o facto do *Go* ter um suporte nativo à simultaneidade (execução de tarefas em simultâneo) é ideal para a maioria dos novos serviços de desempenho crítico. Algumas ferramentas são escritas em C e C++ quando necessário (por exemplo, quando é para código de alta eficiência ou alta velocidade ao nível do sistema).

Testes

Foram desenvolvidas duas ferramentas internas para garantir que os serviços conseguem atender aos pedidos que chegam: *Hailstorm* e *uDestroy*. A primeira direciona testes de integração e simula picos de carga fora dos horários onde a procura é maior, e a segunda destrói coisas intencionalmente para melhorar o tratamento de falhas inesperadas.

[21] https://github.com/phacility/phabricator
[22] https://github.com/oracle/opengrok
[23] https://www.jenkins.io/
[24] https://www.packer.io/
[25] https://www.vagrantup.com/
[26] http://docs.pythonboto.org/en/latest/
[27] https://www.cis.upenn.edu/~bcpierce/unison/
[28] https://github.com/clusto
[29] https://puppet.com/
[30] https://www.tornadoweb.org/en/stable/

Fiabilidade

Quando algum código de produção falha, os *developers* responsáveis por esse(s) serviço(s) de *back-end* são avisados. Para tal, é utilizado o sistema de alertas de *Nagios*[31], para monitorização, juntamente com um sistema de alertas para notificações.

Observabilidade

Para garantir que a Uber funciona como um todo e para manter as suas diferentes partes saudáveis, o seu conjunto de sistemas atua como os olhos, os ouvidos e como o seu sistema imunitário ao redor do mundo.

Telemetria

Foi desenvolvido o M3[32], em *Go*, para colecionar e armazenar métricas de todas as partes da Uber (servidores, serviços de *host*, pedaços de código, etc.). Depois dos dados serem recolhidos são procuradas tendências. Com o uso de *Grafana*[33] são construídos painéis e gráficos que permitem contextualizar as informações recebidas de forma mais expressiva.

Deteção de anomalias

Para detetar anomalias é utilizada uma ferramenta interna chamada *Argos* que examina as métricas recebidas e compara-as com modelos de previsão, de acordo com dados históricos, para determinar se os dados atuais estão dentro dos limites esperados.

Métricas

A ferramenta interna *µMonitor* permite visualizar as informações e os limites referidos no ponto anterior, o que ajuda a que se tomem medidas a partir desses dados. Quando um fluxo de dados sai dos limites, é passada uma informação ao *Common Action Gateway* - um sistema de resposta automática - que vai reagir em função do problema.

Utilizar os dados de forma criativa

[31] https://www.nagios.org/
[32] https://youtu.be/89H48IwFeV4
[33] https://github.com/grafana/grafana

Storm[34] e *Spark*[35] transformam fluxos de dados em métricas de negócio úteis. Foram construídas ferramentas que permitem consumir os dados de forma mais visual, convertendo-os em informações claras e sensatas. Estas ferramentas foram implementadas recorrendo a *Javascript* e *React*.

Mapeamento

São priorizados os conjuntos de dados, algoritmos, ferramentas em torno dos dados do mapa, rotas e sistemas de recolha e recomendação de moradas e locais. Os serviços do mapa (*Map Services*) são executados numa pilha baseada principalmente em *Java*. Um dos maiores serviços desta área é o *Gurafu* que fornece um conjunto de ferramentas para trabalhar com dados de rotas e melhora a eficiência e a precisão, o que dá opções de rastreio mais sofisticadas. Tanto os negócios (da empresa) como os clientes dependem de ETAs (*Estimate Time of Arrival*) muito precisos e, por esta mesma razão, os *developers* dos *Map Services* estão constantemente a trabalhar para tornar estes sistemas com melhor qualidade. Toda a tecnologia de *back-end* por trás das pesquisas, tanto na aplicação do cliente como na do motorista, é também potenciada pelos *Map Services*. Estas tecnologias incluem um mecanismo de pesquisa com preenchimento automático (que permite a procura de locais e moradas com alta velocidade e de acordo com a tendência local), um mecanismo de previsões (para prever o destino do passageiro através da combinação entre o histórico do utilizador e outros fatores) e um serviço de geocodificação reversa (*reverse geocoding*) (que determina a localização do utilizador através do GPS - *Global Positioning System*).

2.2 MaaS

2.2.1 Jelbi, *the world's largest MaaS solution*

A aplicação Jelbi[36] incluirá 12 tipos de transporte público e serviços de mobilidade compartilhada para viajar em Berlim, tendo como objetivo reduzir o uso de veículos com apenas um passageiro [13]. Usando esta aplicação, os utilizadores têm acesso ao elétrico, comboio, barco, metro e algumas opções de mobilidade partilhada tais como bicicletas, *e-kick scooters*, *e-scooters*, autocarros, *car-sharing* e táxis. Para o futuro, a empresa pretende chegar a mais países e mais cidades.

[34] http://storm.apache.org/
[35] http://spark.apache.org/
[36] https://www.jelbi.de/en/home/

2.2.2 UbiGo

O serviço da UbiGo[37], desenvolvido em Gotemburgo, Suécia, oferece aos utilizadores um acesso completo a uma variedade de serviços de viagem através de uma interface *web* adaptada a *smartphones* [14].

O objetivo da UbiGo é facilitar a vida quotidiana das famílias urbanas e promover cidades sustentáveis, através de um serviço simples, flexível, confiável e acessível como alternativa ao carro. Este serviço combina transporte público, *car-sharing*, serviço de aluguer de carros, táxis e sistemas de bicicletas, tudo numa aplicação para *smartphone*. Um estudo feito durante 6 meses, a 70 famílias de Gotemburgo, que utilizaram UbiGo, revelou que a maioria dos clientes queriam continuar a usar o serviço mesmo este estando numa versão beta [15].

2.3 Gestão de Recursos (*Resources Management*)

2.3.1 Arquitetura de gestão de recursos para sistemas de *metacomputing*

Metacomputing é uma tecnologia projetada para integrar vários recursos de computação para desenvolver uma variedade de aplicações relacionadas com negócios, gestão, indústria e *software* [16]. Esta tecnologia também é usada para recolher, interpretar e analisar dados de várias bases de dados e dispositivos. O objetivo de um sistema de *metacomputing* é facilitar a heterogeneidade transparente dos recursos e da rede, usando efetivamente todos os recursos dela.

Segundo [17], o ambiente de *metacomputing* apresenta cinco problemas desafiantes de gestão de recursos:

- *Site autonomy*: refere-se ao facto de que, geralmente, os recursos pertencem e são operados por diferentes organizações, em diferentes domínios administrativos;
- *Heterogeneous substrate*: deriva do problema de *site autonomy* e refere-se ao facto de que diferentes *websites* podem usar diferentes sistemas de gestão de recursos locais. Mesmo quando o mesmo sistema é usado em dois *websites*, diferentes configurações e modificações locais levam, geralmente, a diferenças significativas na funcionalidade;
- *Policy extensibility*: surge porque as aplicações de *metacomputing* são extraídas de uma ampla variedade de domínios, cada um com os seus próprios requisitos;

[37] `https://www.ubigo.me/`

- *Co-allocation*: surge porque muitas aplicações têm requisitos de recursos que apenas podem ser satisfeitos usando recursos em vários *websites* simultaneamente;

- *Online control*: surge porque uma negociação substancial pode ser necessária para adaptar os requisitos da aplicação à disponibilidade dos recursos, principalmente quando os requisitos e as características dos recursos mudam durante a execução.

2.3.2 *Mobile Taxi Dispatch System*

Um *mobile taxi dispatch system* [1] pode receber uma solicitação de um táxi por parte de um utilizador, selecionar um ou mais táxis para responder à solicitação e enviar a solicitação a um ou mais desses táxis selecionados. Este sistema pode ainda receber aceitações de solicitação de pelo menos um dos táxis selecionados e, selecionar um deles para realizar o serviço. Além disso, pode também enviar uma confirmação ao utilizador e ao táxi em particular.

Abaixo está um diagrama que ilustra uma visão geral de uma operação de um *mobile taxi dispatch system*, de acordo com a implementação descrita neste artigo.

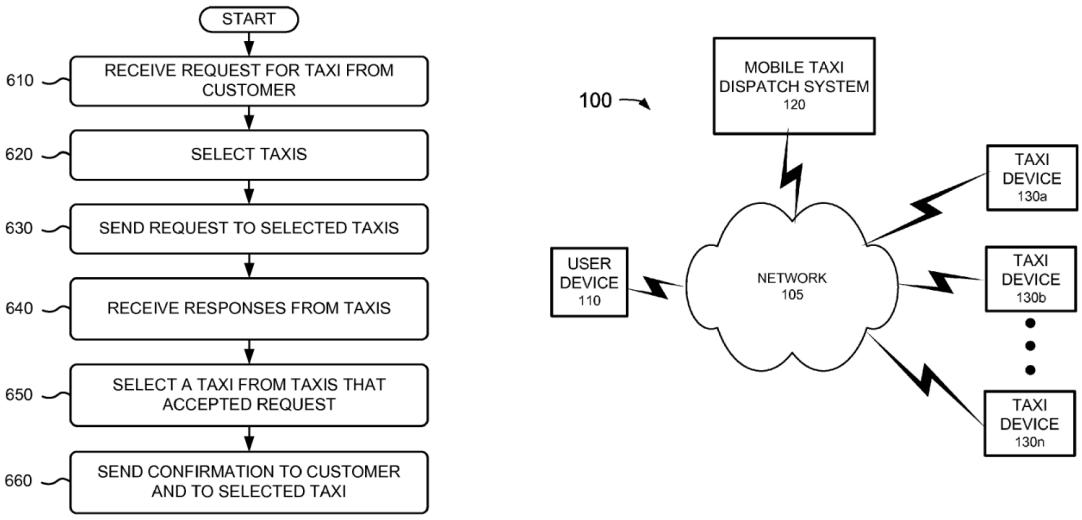


Figura 2.1: Diagrama que ilustra uma visão geral de uma operação de um *mobile taxi dispatch service*, retirado de [1]

Segundo esta solução, um *mobile taxi dispatch service* pode incluir:

- uma rede: por exemplo, uma Rede de Área Local (LAN - *Local Area Network*), uma Rede de Área Ampla (WAN - *Wide Area Network*), uma Rede de Área Metropolitana (MAN - *Metropolitan Area Network*), uma rede *wireless*, etc. Esta rede permite que todos os componentes comuniquem entre si;

- um dispositivo para o utilizador: pode ser qualquer dispositivo de comunicação que permita a um utilizador comunicar com o *mobile taxi dispatch system* e que se possa conetar à rede através de uma conexão com ou sem fio;
- um *mobile taxi dispatch system*;
- um ou mais dispositivos táxis: que pode ser um qualquer dispositivo de comunicação associados a táxis que permitam comunicar com o *mobile taxi dispatch system*.

Capítulo 3

Análise

3.1 Arquitetura

Este projeto incidiu essencialmente sobre dois sistemas já existentes: ***Planeamento*** e ***Localização***. Porém, estes sistemas estão integrados numa plataforma maior, onde todos os constituintes interagem entre si e são responsáveis por uma parte isolada do sistema. O diagrama seguinte ilustra essa interação.

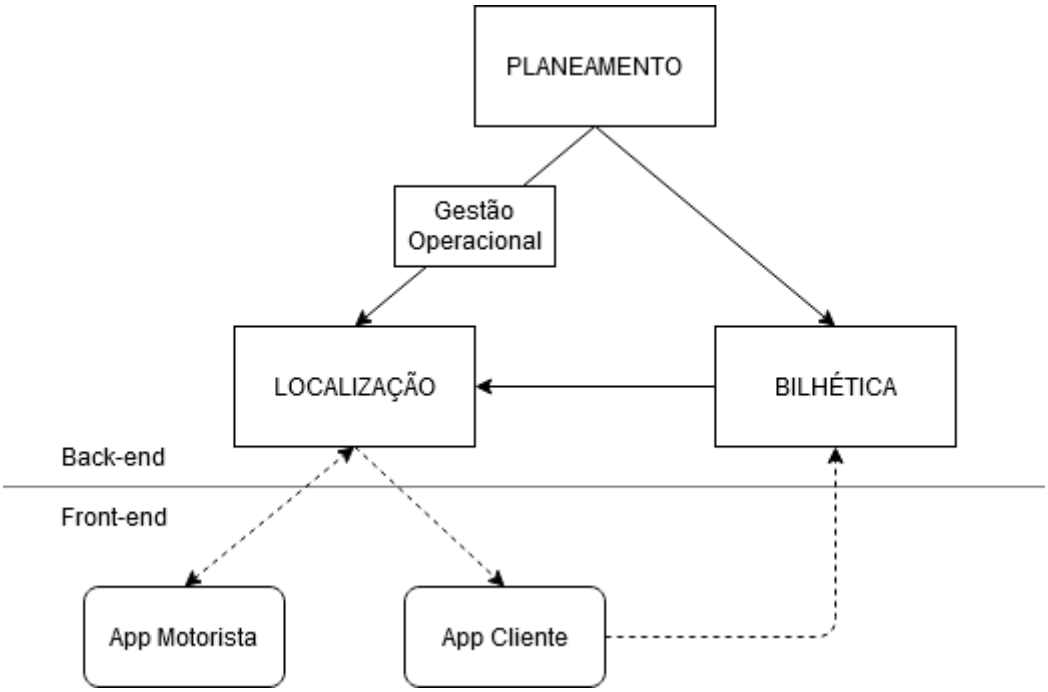


Figura 3.1: Diagrama dos sistemas existentes

Apesar deste projeto não ter tido impacto em todos estes sistemas, foi necessário conhecer um pouco sobre cada um, dando claro um maior destaque ao ***Planeamento*** e ao ***Localização***, nomeadamente em termos de arquitetura e tecnologias utilizadas.

3.1.1 *Planeamento*

O ***Planeamento*** é responsável pela gestão das operações de transporte público, contém dados como carreiras, paragens e horários e é onde é possível agendar serviços, planear diariamente veículos e motoristas e verificar custos.

Existem duas formas de adicionar dados à base de dados do ***Planeamento***: através da importação de ficheiros CSV (*Comma-separated Values*), por parte da empresa cliente; ou através da própria interface *web* do ***Planeamento***.

Esta interface permite ao utilizador adicionar, remover, alterar e consultar dados, presentes na base de dados, sobre:

- Carreiras
- Circulações
- Épocas
- Horários
- Motoristas
- Paragens
- Planeamento de operações
- Serviços ocasionais
- Tipos de dia
- Unidades
- Viaturas

Uma carreira tem várias circulações que podem decorrer ao longo do dia. Sendo que uma circulação pertence a uma dada época e tipo de dia, tem um motorista e uma viatura associados para realizar o serviço e contém um conjunto de paragens (desde a inicial à final), com os seus respetivos horários de paragem, que definem o percurso/trajeto que o autocarro irá cumprir.

O ***Planeamento*** pode ser visto como uma aplicação de 3 camadas (*3-Tier Application*): Camada de dados (*Data tier*), Camada de lógica (*Logical tier*) e Camada de apresentação (*Presentation tier*).

A camada de dados (camada inferior) é responsável por retirar, adicionar e atualizar informação da base de dados. Através do servidor SQL (*Structured Query Language*) são definidas todas as tabelas, *views*, SPs[38] (*Stored Procedures*), etc.

[38] `https://www.w3schools.com/sql/sql_stored_procedures.asp`

Por outro lado, a camada de lógica (camada intermédia) é quem comanda a aplicação. É aqui que se realizam comandos, tomam-se decisões, fazem-se avaliações e executam-se cálculos. Além disso também movimenta e processa dados entres as outras duas camadas.

Por último, a camada de apresentação (camada superior) tem a responsabilidade de comunicar com o utilizador e traduzir-lhe tarefas e resultados em algo que este possa visualizar.

A camada de lógica e de apresentação foram produzidas através da utilização das tecnologias C# e .NET da *Microsoft*. Recorreu-se ainda a *Javascript*, na camada de apresentação, para complementar a forma como os dados são apresentados ao utilizador. A figura 3.2 ilustra toda esta interação.

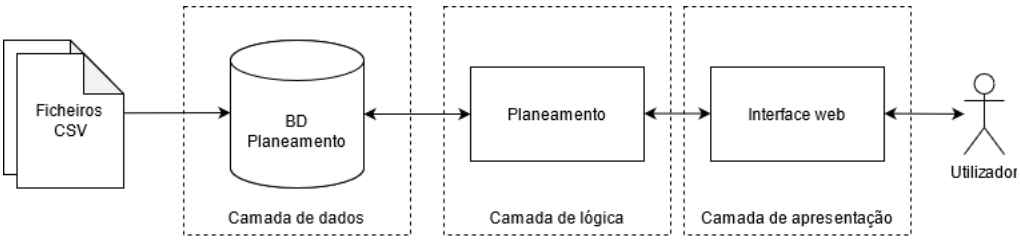


Figura 3.2: Visão geral do sistema ***Planeamento***

3.1.2 *Localização*

O ***Localização*** utiliza o GPS dos veículos para obter informações, em tempo real, sobre o cumprimento de rotas e horários dos serviços prestados.

Os dados presentes neste módulo provêm da informação do ***Planeamento***. No final de cada dia, por volta da meia-noite, há um executável que é responsável por carregar os dados para o próprio dia e para o dia seguinte.

O ***Localização*** também contém um interface *web*, porém, esta não permite editar quaisquer dados presentes na base dados, apenas consultá-los.

Tal como acontece no ***Planeamento***, o ***Localização*** também pode ser visto como uma aplicação de 3 camadas pelas mesmas razões. A arquitetura do sistema está organizada com a mesma lógica e as tecnologias utilizadas também são as mesmas. Na figura 3.3 encontra-se a representação geral deste sistema.

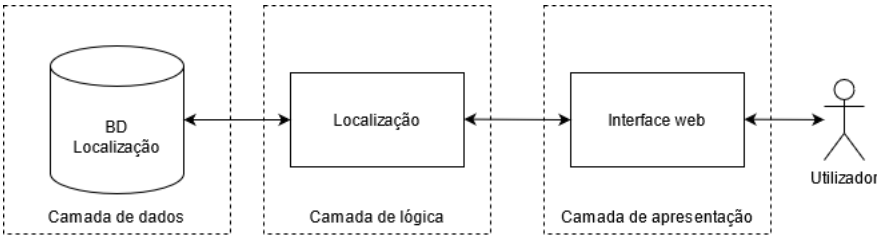


Figura 3.3: Visão geral do sistema ***Localização***

3.1.3 *Gestão Operacional*

Gestão Operacional é uma plataforma que permite gerir recursos para serviços de transporte inteligentes. Este sistema dá suporte a uma série de operações como manutenção, reparação, lavagem e estacionamento de veículos.

3.1.4 *Bilhética*

O módulo ***Bilhética*** está focado em operações *back-end* que permitem a compra e venda de bilhetes, carregamento de passes e, ainda, consulta de preços, tarifas e descontos.

Tal como no ***Planeamento***, os dados no ***Bilhética*** são importados de duas formas: pela empresa cliente através de um ficheiro XML (*Extensible Markup Language*) denominado de EOD; ou através da sua interface *web*.

Caso os dados sejam inseridos através do ficheiro EOD, é necessário convertê-lo num ficheiro TKD, para que o ***Localização*** o consiga interpretar e de forma a que possa ser importado para a base de dados.

Este módulo também contém uma interface *web* que permite adicionar, alterar e remover dados da base de dados.

A figura abaixo permite uma melhor visualização das interações neste sistema.

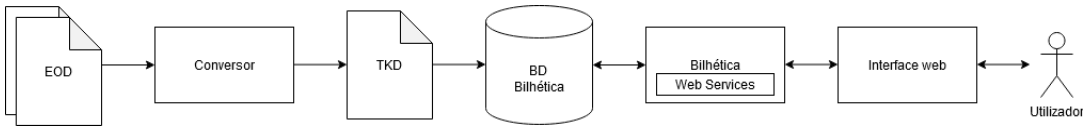


Figura 3.4: Visão geral do sistema ***Bilhética***

3.1.5 *App Motorista*

App Motorista é uma aplicação móvel que se encontra instalada no *tablet* ou no *smartphone* presente no autocarro, e é o sistema utilizado pelos condutores dos veículos que permite aos motoristas realizarem os seus serviços. Por esta razão, este módulo não contém uma interface *web*, contrariamente aos que foram referidos antes,

pois a sua informação é apresentada através de um dispositivo móvel. A figura 3.5 ilustra estas interações.

Este módulo contém funcionalidades de bilhética do ***Bilhética***, pois permite uma interação com o cliente final através de um Terminal de Pagamento Automático (TPA) onde é possível: imprimir bilhetes e faturas, ler/validar cartões e realizar pagamentos com cartão bancário. Há um módulo dedicado para ajudar no completo funcionamento de TPAs mas que não será abordado neste relatório.

Além disso, trata da gestão da rota e das paragens do autocarro com a ajuda do ***Localização***, que lhe envia as coordenadas GPS da localização atual do autocarro e é calculado, do lado da ***App Motorista*** em tempo de execução, a primeira/próxima/última paragem.

A sincronização com as bases de dados do ***Localização*** e do ***Bilhética*** é feita no momento do *login* do utilizador. Desta forma, evita-se uma constante ligação ao servidor.

Esta aplicação requer uma ligação à internet para receber e enviar dados ao servidor, uma ligação *bluetooth* para comunicar com o TPA e utiliza o serviço de localização do telemóvel.

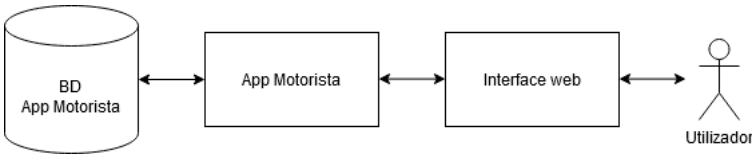


Figura 3.5: Visão geral do sistema ***App Motorista***

3.1.6 *App Cliente*

App Cliente é o sistema que funciona como base das aplicações móveis focadas para os clientes. Estão disponíveis para os diversos transportes públicos existentes e, através delas, é possível obter informações sobre os mesmos.

3.2 Desafios

Além da integração no seio de uma equipa e das adaptações aos produtos já desenvolvidos pela empresa, este projeto implicou ter de encarar alguns desafios durante o desenrolar das tarefas planeadas, tais como:

- Aprendizagem das linguagens de programação C# e .NET;
- Correções e melhorias ao *widget web* do ***Planeamento***;
- Adição de novas funcionalidades aos sistemas ***Planeamento*** e ***Localização***;

- Estudo da tecnologia *Silverlight*, da *Microsoft*, e de outras que consigam realizar as mesmas (ou mais) funções que esta;
- Implemetação de plataformas que sirvam de base a uma nova versão do ***Planeamento***, o ***Planeamento*** V2.

Capítulo 4

Trabalho desenvolvido

Foram identificados vários aspetos a melhorar nos sistemas ***Planeamento*** e ***Localização***.

Desde logo, foram realizadas alterações ao *widget web* - um produto que está contido no sistema ***Planeamento*** - e ao *dashboard* do ***Localização***. Assim como foi também projetada a introdução de um novo mecanismo no ***Planeamento*** - *Paragens satélites*. Posteriormente foi adicionado um ecrã ao ***Planeamento*** que permite desenhar percursos que os autocarros irão fazer, para mais tarde serem comparados com os percursos realmente executados (através do *dashboard* do ***Localização***). Por último, foi efetuado um conjunto de alterações que permitiram substituir a tecnologia *Silverlight* de alguns ecrãs do ***Planeamento***.

Este capítulo vem apresentar as soluções implementadas e, ainda, explicar que tecnologias foram utilizadas.

4.1 *Widget web*

O ***Planeamento*** contém um *widget* que tem como principais objetivos permitir a pesquisa de trajetos de carreiras e a consulta de horários de paragens dos autocarros (inclusive em tempo real) e dos preços dos bilhetes. A pesquisa é realizada de acordo com os seguintes atributos: Carreira, Origem e Destino. Estes atributos variam consoante o operador ao qual se quer efetuar a pesquisa, porém a base do *widget* é a mesma para todos, com o intuito de se ter um produto simples e organizado.

Isto possibilita dois modos de pesquisa: um mais simples, apenas usando a origem e o destino, em que o resultado da pesquisa indica-nos todas as circulações, de todas as carreiras, que circulem entre estas duas paragens; e outro mais complexo, onde se especifica uma carreira, e se pode realizar uma pesquisa selecionando uma origem e/ou destino entre as paragens pertencentes à carreira selecionada.

No início foi necessário realizar um estudo sobre o funcionamento e as funcionalidades existentes no *widget*. Essencialmente, existem 2 ecrãs:

- a página inicial, onde nos é apresentado, do lado esquerdo, as *dropdowns* que permitem filtrar a pesquisa (carreira, origem, destino, dia e hora) e, do lado direito, o mapa (centrado na zona onde o operador incide as suas funções) com todas as paragens que abrange sinalizadas. Ao selecionar-se uma origem/destino, essas paragens ficam sinalizadas no mapa a vermelho/azul, respetivamente;

- a página com o resultado da pesquisa, que está organizado sob a forma de tabela. Cada linha da tabela corresponde a uma circulação, dentro dos filtros selecionados pelo utilizador, e cada coluna da tabela contém uma informação sobre essa circulação (como a hora de partida/chegada, a tarifa da viagem ou até mesmo a rota a ser efetuada).

No ecrã inicial, após selecionar-se uma origem, é possível pesquisar pelos próximos autocarros que irão passar nessa paragem, através do botão "Próximos Horários". Através deste botão, surge um novo "ecrã" na forma de um *popup*. Este ecrã é denominado de "ecrã de ETA" e contém um mapa com a paragem sinalizada e, mais abaixo, uma tabela com as informações sobre as horas previstas de paragem do autocarro e a carreira à qual pertence.

Caso haja algum horário com o fundo laranja, isto significa que o autocarro já iniciou o seu trajeto. Ao clicar sobre essa hora, o mapa fica com um *zoom* centrado entre a paragem e o autocarro onde é possível visualizar o seu percurso, em tempo real, com constantes atualizações geográficas.

A lista seguinte representa o conjuntos de alterações que realizei ao *widget web*:

1. Quando se clica sobre uma paragem, fazer zoom ao mapa na região da paragem;

2. Quando se seleciona uma origem/destino, fazer zoom ao mapa na região da origem/destino;

3. No ecrã de ETA, maximizar o ícone para a paragem em causa;

4. Alteraração do termo "Linha" para passar a chamar-se "Carreira";

5. Ícones iguais aos da Aplicação Móvel para as paragens (apenas no caso do operador Rodoviária de Lisboa);

6. Troca da ordem dos botões "Pesquisa" com "Próximos Horários";

7. Quando se seleciona uma carreira, e não se seleciona uma origem/destino, mostrar o *link* "Ver todos os horários desta carreira";

8. Quando se clica em "voltar", quer seja através de "Ver todos os horários desta carreira", quer seja através de "Pesquisar", recuperar o zoom que estava no mapa;

9. Quando se seleciona uma paragem na parte do ecrã "Rota" no menu "Pesquisar", fazer zoom ao mapa na região da paragem;

10. Ao selecionar uma carreira, depois uma origem e, em seguida, um destino, o destino selecionado já aparece no mapa e já é feito um zoom entre as duas paragens selecionadas;

11. Quando não há uma carreira selecionada e se seleciona origens e destinos, o mapa "ajusta-se" a essas paragens;

12. Quando não há uma carreira selecionada é possível desseleccionar origem/destino, sem que fiquem os seus pontos vermelho/azul no mapa;

13. Ícones origem (azul) e destino (vermelho) tiveram o seu tamanho duplicado, tanto em largura como comprimento;

14. Alteração do ícone de paragem na parte do ecrã "Rota" no menu "Pesquisar";

15. Ao selecionar uma carreira, as paragens aparecem numeradas;

16. Quando há uma carreira selecionada, ao selecionar uma origem/destino, o seu respetivo número aparece a preto, ao lado do ícone (azul/vermelho);

17. No ecrã de ETA, é feito um zoom centrado entre os dois pontos (paragem, autocarro);

18. No ecrã de ETA, aparece o nome da paragem ao clicar no ícone da mesma;

19. No ecrã "Rota" do menu "Pesquisar", é feito um zoom ao mapa entre as paragens;

20. Ao selecionar uma origem e destino, sem ter uma carreira selecionada, o botão "Próximos Horários" já não fica bloqueado;

21. Na *tab* "Detalhes" do ecrã "Rota" do menu "Pesquisar", passam a constar também as localidades onde estão inseridas as paragens físicas (apenas no caso do operador Mafrense);

22. Tendo uma carreira selecionada, quando se tenta pesquisar sem preencher a origem e o destino, surge uma mensagem de alerta de que pelo menos um desses campos deve ser preenchido, para realizar a pesquisa;

23. Na página com os resultados da pesquisa, o corpo da tabela é ajustado ao ecrã;

24. Todas as tabelas passaram a ter apenas o seu corpo *scrollable*, ficando apenas o cabeçalho fixo;

25. Todos os *popups* tiveram o seu tamanho aumentado para que ocupem 80% do ecrã;

26. Os cabeçalhos dos resultados da pesquisa passaram a ter a informação apenas numa só linha.

Todas as alterações efetuadas foram executadas usando as linguagens de programação C#, .NET e *Javascript*.

Estas alterações vieram tornar o *widget* num produto melhor, principalmente em termos de aparência, acessibilidade e visualização para o utilizador, pois é ele que vai tirar partido deste sistema e o que se pretende sempre é fornecer a melhor experiência de utilização possível.

4.2 *Dashboard* de CAN-BUS v2

Neste *dashboard* de CAN-BUS (*Controller Area Network Bus*), presente no ***Localização***, é possível visualizar, através de um mapa, os autocarros em circulação e os seus respetivos dados, que surgem sobre a forma de um *popup* após clicar-lhes (tais como motorista, velocidade atual, etc.).

Dentro deste *popup*, existe a possibilidade de visualizar uma informação mais detalhada sobre o autocarro, através de um botão que se encontra no canto inferior direito e que faz surgir outro *popup* maior por cima deste. Foram feitas algumas alterações a este segundo *popup*, passando a conter mais informação e uma melhor visualização dos dados pertencentes a um autocarro em tempo real. Primeiramente, foi alterado o seu tamanho para que ficasse muito maior e ocupasse a maioria do ecrã. Para isto, foi usado um *plugin* de *Bootstrap*[39], denominado de *Modal*[40], o que permitiu criar um *modal popup*. Foi ainda adicionada outra classe a este popup, designada de "modal-xl", que tem como objetivo manter a maior largura possível. Por fim, os atributos a apresentar foram divididos em 2 colunas dentro do "corpo" do *popup*, para que houvesse mais organização e foram identificadas por "leftModalBody" e "rightModalBody", respetivamente.

Depois do *popup* já ter ficado com a sua estrutura e tamanho definidos, foi-lhe adicionado mais campos para ser possível apresentar informação mais detalhada sobre o autocarro em questão. Os campos adicionados foram:

[39] https://getbootstrap.com/
[40] https://getbootstrap.com/docs/4.0/components/modal/

- Quilometragem
- Coordenada Latitude
- Coordenada Longitude
- Altitude
- Distância total do veículo
- *Total Engine Hours*
- Total de combustível usado
- Início da viagem
- Fim da viagem

O preenchimento deste segundo *popup* é feito dinamicamente, recorrendo a *Javascript*. Sempre que é carregado no botão do primeiro *popup*, é chamada a função "OpenSensors_Click". Esta, por sua vez, faz um pedido AJAX (*Asynchronous JavaScript And XML*) para um *WebMethod* em C#, denominado de "GetSensorValues" que, recorrendo aos dados presentes na base de dados do ***Localização***, devolve os últimos dados acerca dos sensores do autocarro em questão.

É de realçar ainda que as coordenadas GPS do autocarro são enviadas para o ***Localização*** de 30 em 30 segundos e, apenas são atualizadas na base de dados caso sejam diferentes dos últimos dados pois, caso contrário, pode ser um indicativo de que o autocarro se encontra parado.

4.3 Paragens satélites

4.3.1 O que é uma paragem satélite?

Foi introduzido no ***Planeamento*** a criação de um novo conceito: **paragem satélite**. Uma paragem satélite pode ser definida como uma paragem que normalmente pertence a uma carreira, mas que para uma dada circulação, o autocarro não irá passar por ela, ou seja, encontra-se "desativada" para essa circulação. Como a paragem pertence a essa carreira, pode então ser "ativada" para um dia e hora em específico, desde que avisado previamente pelo utilizador. Com esta adição, um serviço deixa de ser "regular" e passa a ser "flexível".

Por exemplo, na carreira 727 da Carris[41] (ver anexo A) que faz o percurso entre a Estação Roma-Areeiro e Restelo, é possível verificar que, aos sábados à tarde,

[41] https://www.carris.pt/

domingos e feriados, o autocarro apenas circula entre Entrecampos e Restelo - Av. Descobertas. Com a adição desta funcionalidade, será possível fazer um pedido para que, por exemplo, num sábado à tarde, o autocarro termine a circulação em Fonte Caselas ao invés de Restelo - Av. Descobertas, como estava programado.

4.3.2 Implementação de paragens satélites

Para que os pedidos de ativação de paragens satélites, por parte dos utilizadores, cheguem até aos motoristas, em tempo real, foi necessário trabalhar nas APIs tanto do ***Planeamento*** como do ***Localização***. Os pedidos serão feitos pelos clientes, através da ***App Cliente***. Esta, por sua vez, comunica com a API do ***Planeamento***, enviando-lhe os dados do pedido. O ***Planeamento*** verifica que a paragem em questão é uma paragem satélite e, em caso afirmativo, encaminha o pedido para a API do ***Localização***. Por fim, o ***Localização*** é responsável por: primeiro, verificar se o dia para o qual está a ser feito o pedido corresponde a uma data igual ou posterior à de execução do serviço, pois caso seja anterior o pedido é negado; segundo, verificar se a hora do pedido é igual ou posterior à hora prevista de início do serviço e igual ou anterior à hora prevista de fim do serviço e em caso afirmativo, o pedido é aceite. Antes de enviar uma mensagem de resposta ao utilizador informando-o se o seu pedido foi ou não bem sucedido, o ***Localização*** verifica se já existia um pedido anterior para a mesma paragem e para a mesma circulação. Em caso afirmativo a mensagem enviada informa o cliente de que o pedido foi aceite, porém para uma hora diferente da selecionada pois já existia um pedido prévio para a mesma paragem e circulação. Em caso negativo, o cliente é informado de que o seu pedido foi aceite (para a hora por ele selecionada).

O diagrama da figura 4.1 representa o fluxo de mensagens efetuado durante um pedido de ativação de paragem satélite.

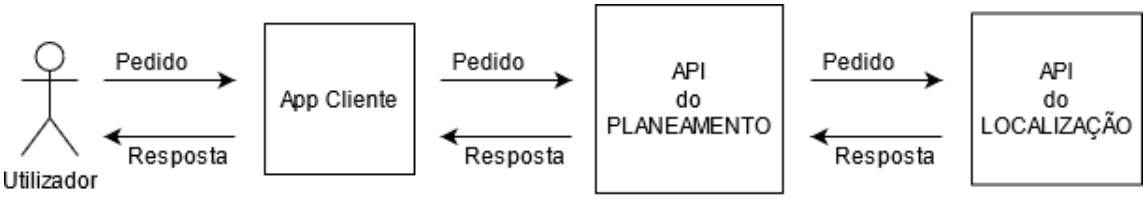


Figura 4.1: Fluxo de mensagens durante um serviço a pedido

Desta forma, foi realizado um conjunto de alterações de modo a permitir que se realizem pedidos, para uma dada circulação, num certo dia, a uma certa hora, para que o autocarro faça o desvio na sua trajetória habitual e passe então por aquela paragem.

4.3.2.1 Adição de paragens satélites aos serviços regulares

Primeiramente foi adicionada uma coluna, denominada de *isStopOnDemand*, a uma tabela da base de dados do ***Planeamento***, que servirá como uma *flag* que sinaliza se a paragem em questão é uma paragem satélite ou não. Se for uma paragem satélite o valor desta coluna é de 1 (em *byte*), caso contrário, é 0 (em *byte*). Desta forma, enquanto se está a planear os horários de passagem nas paragens dos autocarros, nas circulações, passa a existir mais uma opção:

- Não para
- Para
- Para sem hora definida - serviço ocasional
- Não para (exceto se for ativada), sem hora definida - serviço a pedido **(nova adição)**

4.3.2.2 Receção de pedidos de ativação de paragens satélites (*Planeamento*)

Posteriormente, foi adicionado um *webservice* à API do ***Planeamento*** que permite receber pedidos de ativação de paragens satélites. Esta API é do tipo *RESTful*, ou seja, utiliza os pedidos HTTP GET, PUT, POST e DELETE e contém 4 entidades às quais se podem fazer estes pedidos: *Driver*, *DriverService*, *Stop* e *StopService*.

Este *webservice* em questão é do tipo POST, foi associado à entidade *StopService* e recebe cinco parâmetros: a identificação da unidade em que o autocarro opera, a direção que a rota do autocarro tem de fazer (seja ida, volta ou circular), o código público da carreira, o código público da paragem, para que seja possível identificá-la, e um objeto do tipo *DateTime*, que representa o dia e a hora do pedido. Na tabela 4.1 representa-se a API relativa a cada funcionalidade associada às entidades referidas e, no anexo C.1, encontra-se a representação de um serviço a pedido - *StopServiceRequest*.

	GET	**PUT**	**POST**	**DELETE**
Driver	/unit/unitId/driver/id	/unit/unitId/driver/id	/unit/unitId/driver	/unit/unitId/driver/id
DriverService			/api/service/driver/action	
Stop	/stop/operatorID/publicCode	/stop/operatorID/publicCode		
StopService			/api/service/stop/StopOnDemand	

Tabela 4.1: API do ***Planeamento***

Depois dos dados recebidos através de um *StopServiceRequest* serem validados, o ***Planeamento*** verifica se a paragem em questão é uma paragem satélite, ou seja, se pode receber pedidos de ativação (em tempo real) e, em caso afirmativo, reencaminha os dados para o ***Localização***.

4.3.2.3 Receção de pedidos de ativação de paragens satélites (*Localização*)

A API do lado do ***Localização*** também é do tipo *RESTful* e, neste caso, o *web-service* em questão também é do tipo HTTP POST e está associado à entidade *TrackingService*. Este recebe como parâmetro um objeto do tipo *StopOnDemand*. Na tabela 4.2 está representada a API relativa à entidade *TrackingService* e, no anexo C.2, encontra-se a representação de um objeto *StopOnDemand*.

	POST
TrackingService	TrackingService/Tracking/StopOnDemand

Tabela 4.2: API do ***Localização***

Depois de receber os dados do pedido, o ***Localização*** executa um SP, um procedimento SQL armazenado na sua base de dados.

Este SP verifica se existe algum serviço disponível para o dia e hora selecionado pelo utilizador e devolve sempre 2 variáveis como resposta:

- A primeira - **@result** - que sinaliza se a operação de ativação de uma paragem satélite foi ou não bem sucedida. Sendo então o resultado do tipo booleano (*true* ou *false*).
- A segunda - **@result_date** - do tipo *DateTime*, que indica o horário para o qual ficou atribuído o pedido. Este resultado pode ter 3 possibilidades:
  - Vazio, ou seja, "" - que acontece quando a operação não foi bem sucedida;
  - Igual à hora pedida pelo utilizador - que acontece quando a operação é bem sucedida e ainda não existia um pedido de ativação prévio para a mesma paragem e circulação;
  - Diferente da hora pedida pelo utilizador - que acontece quando a operação é bem sucedida, porém já existia um pedido prévio para a mesma paragem na mesma circulação, feito por outro utilizador. Neste caso a hora retornada corresponde a essa hora já existente.

A resposta do SP segue de volta para a API do ***Planeamento*** e, por sua vez, para a ***App Cliente*** (utilizada pelos clientes). Desta forma, com estas 2 variáveis, é possível dar um *feedback* com mais informação ao utilizador que faz o pedido.

4.4 Percurso de viagens planeadas

Foi criado, no ***Planeamento***, um novo ecrã denominado de "Editar Percursos" que permite para cada circulação, de cada carreira, editar/definir o trajeto que o

autocarro irá fazer para essa mesma circulação. O título desta nova página além de conter a designação da carreira em questão, contém uma data de versão que corresponde à última data em que se definiu ou alterou um percurso de pelo menos uma circulação desta carreira. A forma como foi implementado utiliza um mapa, que é carregado através da API do *OpenStreetMap*[42] e, com a ajuda de uma biblioteca de *JavaScript* denominada de *OpenLayers*[43], são carregados os pontos correspondentes a cada paragem.

Uma carreira tem várias circulações que variam de acordo com alguns fatores:

- Época: Anual, Escolar, Carnaval, etc;
- Tipo de dia: Sábados, Dias úteis, Só se efetua aos feriados, Domingos, etc;
- Direção: Ida, Volta e Circular;
- Horário: 08:25 - 08:50, 09:15 - 10:00, etc.

O anexo C.3 ilustra o Modelo Entidade-Associação entre as entidades referidas anteriormente.

Então, para se obter as paragens que pertencem a uma dada circulação é necessário saber todos estes fatores. Para tal, adicionou-se à página 4 listas *dropdown*.

As *dropdowns* foram implementadas de forma a que se siga uma ordem de preenchimento. Primeiro é necessário preencher a época. Após esta *label* ser preenchida, a *dropdown* sobre o tipo de dia é desbloqueada. Depois de se selecionar o tipo de dia, é desbloqueada a *dropdown* referente à direção. Por fim, após esta última ter sido preenchida, passa a ser possível visualizar (na última *dropdown*) os horários das circulações que estão disponíveis para a carreira em questão.

O preenchimento das *dropdowns* segue esta ordem porque, para se obter os horários de uma carreira, são necessários todos estes dados e só assim é possível garantir que estão devidamente preenchidos. Quando é selecionada a penúltima *dropdown* (sobre o tipo de dia), é feita uma *query* com 3 SQL JOINS[44] de modo a obter-se os horários das circulações em questão.

Para cada circulação, fica associado um Identificador (ID) "escondido" ao utilizador, para que se possa identificar qual foi o horário escolhido. O que o utilizador pode ver é guardado no campo *DataTextField*, o que é usado para identificar o que foi selecionado na *dropdown* é guardado no campo *DataValueField* - como está representado na tabela 4.3.

[42] https://wiki.openstreetmap.org/wiki/Main_Page
[43] https://wiki.openstreetmap.org/wiki/OpenLayers
[44] https://www.w3schools.com/sql/sql_join.asp

DataTextField	DataValueField
Horário	ID

Tabela 4.3: Atributos de cada circulação - *dropdown* horário

4.4.1 Carregar paragens no mapa

Depois de se selecionar uma circulação, toda a informação acerca das paragens que o autocarro faz nesse horário é obtida e, com esta informação, é feita uma chamada a um método *Javascript*, do lado do C#, de modo a representar essas paragens no mapa. Para tal, toda a informação de uma paragem é formatada em JSON (*JavaScript Object Notation*), como está representado no anexo B.1.

Neste contexto, o ID da paragem corresponde à ordem em que aparece no trajeto, isto é, a primeira paragem tem ID 1, a segunda tem ID 2 e assim sucessivamente até à última.

Antes de ser enviado, o objeto que contém os dados sobre todas as paragens é serializado através da biblioteca *JsonConvert*, para que fique tudo em formato de texto, numa única *string*. A função *JavaScript*, que trata de colocar os pontos no mapa, designa-se de "drawMarkersFromJSON". Existe um objeto do tipo *Map*, da biblioteca *OpenLayers*, que representa o mapa que é visto na página e é nele que os pontos são adicionados. Cada ponto representa uma variável do tipo *Vector* - "OpenLayers.Feature.Vector" - que é composta por três atributos:

- Uma variável do tipo *Point* - "OpenLayers.Geometry.Point" - que contém a longitude e latitude da paragem;
- Uma descrição da paragem com o ID e o nome da mesma;
- Um *externalGraphic* que representa o estilo do ícone que simboliza a paragem.

A figura 4.2 ilustra melhor esta associação.

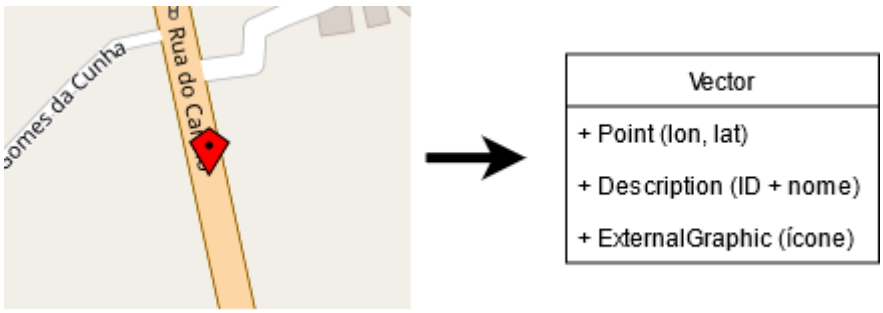


Figura 4.2: Representação de um ponto no mapa - Editar Percursos

4.4.2 Desenhar percurso no mapa

Depois das localizações das paragens terem sido carregadas para o mapa, é então possível, também com a ajuda da biblioteca *OpenLayers*, desenhar pontos no mapa que se ligam, formando um trajeto.

4.4.3 Guardar percursos desenhados

Através de um botão "Guardar Rota", é possível guardar os trajetos que são desenhados. Para isso, foi adiconada uma coluna nova chamada "Wkt", do tipo *text*, a uma tabela da base de dados do ***Planeamento*** para que deste modo fique associado um trajeto por circulação. O nome dado à coluna deve-se ao próprio *Well-known Text* (WKT), uma linguagem de marcação de texto que é utilizada para representar objetos geométricos vetoriais num mapa.

Depois do trajeto ser guardado na base de dados, é então atualizada a data de versão da carreira.

4.4.4 Exportar percursos

Foi ainda adicionada a possibilidade de extrair o percurso desenhado através da geração de um ficheiro KML (*Keyhole Markup Language*), utilizando o botão "Export to KML".

4.5 Percurso executado face ao planeado

Para se obter informações sobre as carreiras e, para que seja possível visualizar no ***Localização*** o percurso planeado para uma dada circulação, proveniente do ***Planeamento***, é necessário exportar esses dados via GTFS[45] (*General Transit Feed Specification*).

GTFS, em Português, Especificação Geral sobre *Feeds* de Transporte Público, define um formato comum para os horários de transporte público juntamente com as informações geográficas relacionadas.

4.5.1 O que é um *Feed* GTFS?

Um *Feed* GTFS é composto por um conjunto de ficheiros de texto (.txt) agrupados num ficheiro ZIP. Cada ficheiro aborda um aspeto específico das informações sobre o transporte público: paragens, percursos, viagens, horários, etc.

[45] https://developers.google.com/transit/gtfs/

4.5.2 Geração de GTFS

Como já foi referido no capítulo 3, no final de cada dia há um executável que gera um GTFS a partir da informação do ***Planeamento***, para o próprio dia e para o dia seguinte. Posteriormente, do lado do ***Localização***, há um processo que vai interpretar esse GTFS, transformando-o em serviços.

Foi então necessário conseguir exportar os WKT, que são gerados quando é desenhado um trajeto para uma circulação, no formato do GTFS. De acordo com a especificação suportada pelo GTFS, a forma de o fazer é através de *Shapes*[46]. A diferença é que, no WKT, a informação sobre toda a circulação está numa só *string*, enquanto que no *Shapes* é preciso distribuir essa informação colocando cada par (latitude, longitude) por linha.

Segundo esta especificação, a informação sobre as circulações é dividida por 12 ficheiros, sendo depois tudo guardado (e compactado) num ficheiro denominado de "google_transit.zip" - o GTFS. Os 12 ficheiros são:

- agency.txt - empresa de transporte público à qual este conjunto de dados pertence;
- calendar.txt - especifica as datas de início e de fim dos serviços prestados;
- calendar_dates.txt - define algumas exceções para os serviços definifos no calendar.txt;
- delays.txt - corresponde aos valores de desvio permitidos para cada circulação;
- lines.txt - especifica, para cada circulação, a sequência/ordem das paragens em função da direção que a circulação toma (ida, volta, etc.);
- routes.txt - representa circulações de transporte público, sendo que uma circulação é um conjunto de viagens que é apresentado aos passageiros como um único serviço;
- service_trips.txt - indica, para cada serviço, a paragem inicial e final com as respetivas horas previstas de início e de fim, mais a distância total (em km);
- services.txt - refere que motorista está associado a um certo serviço, para um certo dia e hora;
- shapes.txt - conjunto de regras que permitem mapear as circulações;
- stop_times.txt - contém as horas de chegada e de partida dos veículos nas paragens de cada circulação;

[46] https://developers.google.com/transit/gtfs/reference#shapestxt

- stops.txt - define as paragens onde se recebe/deixa passageiros;
- trips.txt - representa as viagens que existem para cada circulação.

Para esta tarefa apenas é de realçar os dados contidos no ficheiro "shapes.txt", que contém informação dos pontos geográficos associados a cada trajeto, e do ficheiro "trips.txt", que contém informação associada a cada trajeto. Toda esta informação está representada, mais detalhadamente, na tabela 4.4.

Trip	**Shape**
string Id	string Id
string RouteId	double Latitude
string ServiceId	double Longitude
DirectionType Direction	int Sequence
string ShortName	double DistanceTravelled
string ShapeId	

Tabela 4.4: Entidades GTFS

A informação no ficheiro "shapes.txt" é representada pela seguinte forma:
shape_id, shape_pt_lat, shape_pt_lon, shape_pt_sequence, shape_dist_traveled

Já no ficheiro "trips.txt", a informação está organizada desta maneira:
route_id, service_id, trip_id, direction_id, trip_short_name, shape_id

A figura 4.3 esquematiza todos estes dados.

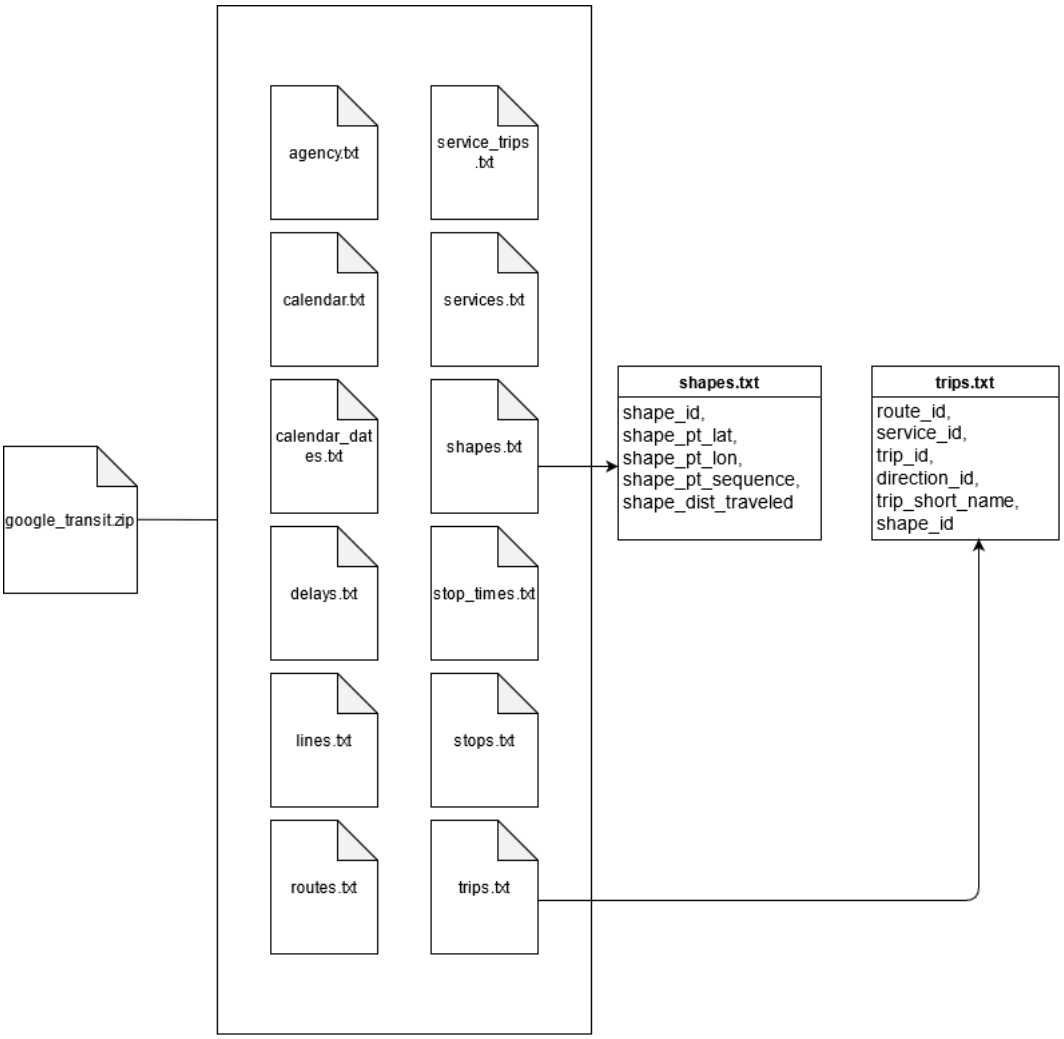


Figura 4.3: Ficheiros GTFS

Como é possível verificar, existe um campo em comum entre os dois ficheiros, **shape_id**, o que permite facilmente descobrir qual é o conjunto de pontos geográficos pertencentes a cada circulação.

4.5.3 Carregar o percurso planeado para o mapa

Depois de já se ter conseguido exportar os dados provenientes do ***Planeamento***, foi necessário carregar estes percursos planeados das carreiras para o mapa que existe no *dashboard* do ***Localização***. Desta forma, é possível verificar e comparar o percurso executado face ao planeado.

Neste *dashboard* já existia uma *dropdown* contendo todas as carreiras onde, ao selecionar uma, aparecia o percurso da mesma (em espinha) por baixo da *dropdown*, porém, esta informação não era visível no mapa. Para tal, foi adicionada outra *dropdown* que permite ao utilizador selecionar um horário de circulação para

cada carreira. Desta forma, ao selecionar uma carreira e depois a circulação que se pretende visualizar, apenas é verificado se já existe um trajeto associado a essa circulação e, em caso afirmativo, representá-lo no mapa.

Os horários das circulações são preenchidos dinamicamente. Sempre que se seleciona uma carreira, é ativado uma função *JavaScript* "change" que deteta essa mudança e faz um pedido AJAX para um *WebMethod* em C# que, recorrendo aos dados presentes na base de dados do ***Localização***, devolve os horários das circulações para a carreira em questão.

Como se pode verificar, na *dropdown* referente aos horários das circulações, é também ocultado um "value" ao utilizador, para se poder identificar qual foi o horário escolhido. O que o utilizador vê é guardado no campo "text", o que é usado para identificar o que foi selecionado na *DropDownList* é guardado no campo "value". Neste caso, o que fica guardado neste parâmetro "oculto" é o valor da componente WKT. Na tabela 4.5 pode ver-se esta associação entre horário e WKT.

DataTextField	**DataValueField**
Horário	WKT

Tabela 4.5: Atributos de cada circulação - *dropdown* horário - *Dashboard* do ***Localização***

Por fim, para se visualizar no mapa a circulação pretendida, deve carregar-se no botão "Desenhar no Mapa". Quando este botão é clicado, faz uma chamada a uma função *JavaScript* de modo a obter o "value" associado para o horário selecionado e, assim, adquirir o WKT para esta circulação. Sendo que o objeto que representa o mapa do *dashboard* também utiliza a biblioteca *OpenLayers*, o que este método faz é adicionar uma *feature*, com o WKT, ao mapa, que neste contexto funciona como uma "camada" sobreposta, representando o percurso planeado para o autocarro.

4.6 Evolução de tecnologia no *Planeamento*

Existe no ***Planeamento*** um conjunto de ecrãs que utilizam *Silverlight*[47] "por trás" da sua execução. O *Silverlight* é uma ferramenta que permite criar experiências interativas e atraentes aos utilizadores, para *web* e aplicações móveis. No entanto, há um problema quanto à utilização desta ferramenta, a *Microsoft* deixará de dar-lhe suporte a partir de 2021 [18]. Atualmente, o *Silverlight* apenas é suportado no Internet Explorer 10 e 11, isto é, já não existe suporte para Chrome, Firefox ou qualquer outro *browser*.

[47] `https://www.microsoft.com/silverlight/`

4.6.1 Ecrãs que utilizam *Silverlight*

No ***Planeamento*** existe um total de 5 ecrãs que necessitam de sofrer esta alteração tecnológica, nomeadamente:

1. Ecrã Épocas;
2. Ecrã Tipos de dia;
3. Ecrã Calendário;
4. Ecrã Planeamento Semanal;
5. Ecrã Graficagem;

4.6.2 Tecnologias para substituir o *Silverlight*

Com isto, foi então necessário encontrar uma alternativa para substituir o *Silverlight*. Foi iniciado um estudo com o intuito de encontrar a tecnologia ideal para realizar esta substituição.

4.6.2.1 HTML5

Cinco funcionalidades que eram usadas em *Silverlight* e que, atualmente já conseguem ser feitas com HTML5, segundo [19]:

1. Vídeo/áudio
   - O HTML (*HyperText Markup Language*) apresenta duas novas *tags*, <audio> e <video>, para trabalhar com multimédia.
2. Gráficos
   - O HTML teve vários sistemas de gráficos vetoriais no passado e, atualmente é usado o padrão SVG (*Scalable Vector Graphics*) que tem um bom suporte. Esta alternativa aos gráficos é feita através do uso da *tag* <canvas>.
3. Melhorias nos *input widget*
   - O HTML5 fez melhorias aos vários *input widget* o que permitiu ter um melhor controlo sobre estes. Por exemplo, o atributo "type" na *tag* <input> permite informar o *browser* que tipo de dados está a entrar num *text field*, o que, por sua vez, permite que o *browser* faça uma melhor apresentação do *widget* ao utilizador.

4. Operações assíncronas

    - Os processos assíncronos são fundamentais para o desenvolvimento de aplicações. O HTML5 atende a esta necessidade através do uso de *Web Workers*.

5. Comunicações

    - Através do uso de *WebSockets* é possível definir comunicações bidirecionais entre o *browser* e um *host*.

C#/XAML para HTML5 segundo [20]:

A ferramenta vem como uma extensão do *Visual Studio* no *Visual Studio Marketplace*, e promete criar aplicações HTML5, utilizando apenas C# e XAML (*Extensible Application Markup Language*), ou migrar aplicações *Silverlight* existentes para a *web*.

Não é necessário nenhum conhecimento de HTML5 ou *Javascript* para usar a extensão porque os ficheiros HTML5 e *JavaScript* são compilados pela própria extensão. É então possível criar aplicações na *web* com *static typing* e todos os pontos fortes de C# e XAML e, posteriormente, a extensão verifica se o código está pronto quando o *WebAssembly* é lançado.

O *WebAssembly* é uma tecnologia que permite aos *developers* escreverem código tipo *Assembly*, de baixo nível, para o *browser* em linguagens *non-Javascript* como C, C++ e até mesmo linguagens .NET como C#, para melhorar o desempenho sobre *Javascript*.

Até o *WebAssembly* ser totalmente suportado na *web*, o CSHTML5 pode ser visto como uma alternativa para *developers* centrados em .NET.

Os *developers* poderão tirar proveito do ganho de desempenho fornecido pelo *WebAssembly* em ficheiros C#, já os ficheiros XAML serão compilados pelo CSHTML5 até que haja um compilador XAML para *WebAssembly*.

Segundo a empresa, atualmente o CSHTML5 suporta 99% dos recursos de C#, juntamente com 75% de XAML e 55% de .NET Core e, é compatível com o *Visual Studio* 2012, 2013, 2015 e 2017.

4.6.2.2 Angular

Razões da migração de *Silverlight* para *AngularJS* segundo [21]:

1. Melhor manutenção

- As aplicações *web* com a estrutura *Angular* têm melhor capacidade de manutenção, apesar da sua grande base de dados. Uma aplicação *web* precisa de constantes atualizações e correções de *bugs* e *glitches*. Para isso, é necessário que esteja configurada de tal forma que essas pequenas mudanças sejam fáceis de fazer. O *Angular* utiliza o *model* (ng-model[48]) como base o que incentiva um *design* orientado a objetos para o lado do cliente. É através destes princípios de *design* orientado a objetos que a manutenção de aplicações da *web* é simplificada.

2. Facilidade na construção de uma aplicação *web*

   - A estrutura do *Angular* facilita os *web developers* a criar aplicações *web* que sejam totalmente responsivas, juntamente com uma *user interface* impressionante. Esta estrutura vem com vários recursos embutidos que aceleram o processo de desenvolvimento, com diretivas que abrangem muitas das operações comuns no *jQuery*. Desta forma gasta-se menos tempo a produzir código e mais tempo a trabalhar no *website* para aprimorar a sua aparência, produzindo uma melhor experiência ao utilizador.

3. Manipulação de dependências

   - O *Angular* lida facilmente com dependências. Pode dividir-se a aplicação *web* em módulos lógicos, conectados entre si através de dependências. Estas dependências podem ser inicializadas separadamente. Desta forma, pode carregar-se módulos *Javascript* dinamicamente quando necessário. Assim melhora-se o desempenho e o desenvolvimento de aplicações *web* torna-se mais fácil.

4. Simplicidade nos testes

   - Devido à facilidade de manusear dependências, é possível dividir a aplicação *web* em vários módulos e testá-los separadamente. Isto permite uma maneira clara de testar e ajuda a detetar erros mais rapidamente. Além disso, é mais fácil identificar problemas e procurar zonas onde há espaço para melhorias. Tudo isto, por sua vez, melhora o desempenho do *websites*, pois uma testabilidade aprimorada permite aos *web developers* aperfeiçoar regularmente a aplicação *web*.

Razões da migração de *Silverlight* para *Angular* de acordo com [22]:

[48] `https://docs.angularjs.org/api/ng/directive/ngModel`

Porquê migrar do *Silverlight* para o *Angular*?

As duas razões principais é porque o *Angular* é viável e possui muitos recursos/*features*.

Vantagens e *features* do *Angular*:

1. *Angular* é uma ferramenta para desenvolvimento de aplicações *web open-source*;
2. É gerido e mantido pela *Google* e por uma grande comunidade de *coders*;
3. É suportado pelo Visual Studio .NET IDE e IntelliJ IDE;
4. Fornece recursos alargados como *routing*, injeção de dependências, animações e muito mais;
5. Permite que os *developers* realizem algo em particular, usando enumeras abordagens através do fornecimento de diferentes tarefas e estilos de desenvolvimento;
6. Permite aos *developers* um desenvolvimento paralelo intensivo;
7. Em *Angular*, a página DOM (*Document Object Model* - Modelo do objeto do documento) é alinhada diretamente em vez de ser adicionada ao código HTML, tornando assim o sistema mais rápido;
8. Com *Angular*, as dependências são facilmente trabalhadas;
9. A maioria dos problemas enfrentados durante o desenvolvimento de aplicações são abordados pelo *Angular*. Por ser baseado em *TypeScript*, traz muitos benefícios principalmente a detecção precoce de erros;
10. Existe uma consistência geral ao usar o *Angular* no desenvolvimento de aplicações *web* pois a estrutura base de um componente parece sempre a mesma;
11. Os serviços do *Angular* são consistentes: permitem escolher entre *factories*, *services*, *providers*, *values* e *constants* quando existe código necessário para ser reutilizado numa aplicação;
12. Através desta grande consistência, obtém-se benefício na produtividade;
13. Organizar facilmente o código por *"buckets"* - tudo o que é criado pode ser organizado num ou mais *"buckets"*.

Benefícios desta migração:

- *Angular*, em comparação com o *Silverlight*, é uma plataforma de *web* moderna. Não só oferece formulários mais inteligentes como também uma experiência de utilizador rica em recursos, com benefícios adicionais de *design* de *responsive web* que suporta vários dispositivos como telemóveis e tablets.

- Se se tiver uma aplicação para computador, criada com *Silverlight*, mas que se percebe que é mais adequada como aplicação móvel, com *Angular* é possível criar uma aplicação móvel *cross-platform*. Desta forma, é possível aprimorar os serviços de dados existentes com uma nova aplicação móvel ou criar uma camada de dados "moderna" que utilize a base de dados existente.

- Muitas aplicações *Silverlight* destinam-se apenas a ser aplicações do *Windows*. Isto limita muito os "horizontes" quando comparado com aplicações móveis *cross-platform*. Com *Angular*, essa limitação é eliminada pois é fornecido diferentes soluções.

- Outros benefícios:

  - *Thread-like form operations*;
  - APIs com suporte à geolocalização;
  - Fornecimento de bases de dados do lado do cliente;
  - Opções para incorporar vídeos e áudios;
  - Melhores soluções de *responsive design*;
  - Personalizável, concentrando-se nas necessidades de aplicações da *web*.

4.6.2.3 Blazor

De acordo com [23]:

- UI (*User Interface*) interativa da *web* com C#:

  - O *Blazor* permite criar UIs interativas da *web* usando C# em vez de *Javascript*. As aplicações *Blazor* são compostas por componentes reutilizáveis da UI da *web* implementados com C#, HTML e CSS (*Cascading Style Sheets*). O código do cliente e do servidor é escrito em C#, permitindo a partilha de código e bibliotecas.

- É executável no *WebAssembly* ou no servidor:

  - O *Blazor* pode executar o código C# do lado do cliente diretamente no *browser*, usando o *WebAssembly*. Por ser uma execução real de .NET no *WebAssembly*, é possível reutilizar código e bibliotecas de partes da aplicação do lado do servidor.
  - Como alternativa, o *Blazor* pode executar a lógica do cliente no servidor. Os eventos da UI do cliente são enviados de volta ao servidor através de *SignalR* - uma estrutura de mensagens em tempo real. Depois da execução estar concluída, as alterações necessárias na UI do cliente são-lhe enviadas e fundidas no DOM.

- Construído em padrões *open web*:
  - O *Blazor* usa padrões abertos de *open web* sem *plugins* ou transpilação de código (transpilar é uma mistura de compilar e traduzir; por outras palavras é uma ferramenta para gerar uma nova versão de um dado código). Funciona em todos os *browsers* modernos, incluindo *browsers* móveis.
  - O código em execução no navegador é executado na mesma *sandbox* de segurança que as ferramentas *Javascript*. O código *Blazor* em execução no servidor tem a flexibilidade de fazer qualquer coisa que seria possível fazer normalmente num servidor, como conectar diretamente a uma base de dados.

- Compartilhar código e bibliotecas:
  - As aplicações *Blazor* podem usar bibliotecas .NET existentes, graças ao .NET Standard - uma especificação formal das APIs .NET que são comuns em todas as implementações .NET.
  - O .NET Standard permite que o mesmo código e bibliotecas sejam usados no servidor, no navegador ou em qualquer lugar onde se escreva código .NET.

- Interoperabilidade *Javascript*:
  - O código C# pode facilmente chamar APIs e bibliotecas *Javascript*. É possível continuar a usar o grande ecossistema de bibliotecas *Javascript* existentes para a UI do lado do cliente enquanto se escreve a lógica em C#.
  - Ao correr código do lado do servidor, o *Blazor* encarrega-se de executar sem problemas qualquer código *Javascript* no cliente.

- Ferramentas gratuitas para todos os sistemas operativos:

  - O *Visual Studio* e o *Visual Studio Code* fornecem uma excelente experiência no desenvolvimento de *Blazor* em *Windows*, Linux e macOS.

4.6.2.4 WPF

De acordo com [24]:

- O que é o WPF (*Windows Presentation Foundation*)?
  - O WPF é um componente do *Microsoft .NET Framework* 3.5. É considerado a próxima geração de UI, que permite a criação de aplicações com um grau de personalização extremamente elevado, criando assim uma experiência única ao utilizador.
  - O WPF suporta interfaces de aplicação, gráficos 2D e 3D, documentos, aceleração de *hardware*, gráficos vetoriais, visualização de dados interativos e multimédia numa única estrutura.
  - Utiliza a aceleração de *hardware* das novas placas gráficas, permitindo assim tornar a interface mais rápida, escalável e com resolução independente.
- Separação entre a aparência e comportamento:
  - O WPF separa a aparência da interface do utilizador do comportamento.
  - A aparência no WPF é criada geralmente no XAML, o desempenho/comportamento é implementado numa linguagem de programação, como por exemplo o C#.
  - As duas partes estão ligadas entre si pelos eventos de ligação a dados e comandos.
- A separação da aparência e comportamento tem como vantagem:
  - A personalização dos modelos e das aplicações - As ferramentas de *design* gráfico podem trabalhar em documentos XML "simples" em vez de ter de analisar o código.
- Extremamente personalizável:
  - Devido à separação da aparência do comportamento, é extremamente simples alterar a aparência de um botão, por exemplo.
  - O conceito de estilos permite ter um controlo da aparência idêntico ao CSS.
- Interface sem dependência da resolução:

  - Todas as medidas em WPF são unidades lógicas - não pixeis. Uma unidade lógica equivale a 1/96 polegadas. Se se aumentar a resolução do ecrã, a interface do utilizador fica do mesmo tamanho - só que fica mais nítida. Com o WPF a interface "acompanha" o aumento da resolução.

4.6.2.5 Vue

De acordo com [25]:

- O que é o *Vue.js*?
  - É uma ferramenta progressiva de *Javascript* para criar uma UI e uma das mais populares para simplificar o desenvolvimento na *web*. O *Vue* trabalha principalmente com a camada de apresentação, ou seja, foca-se na parte da *view*. O seu desenvolvimento também é capaz de possibilitar uma *Single Page Application* quando usado em conjunto com bibliotecas modernas de ferramentas e suporte. É fácil integrar-se em grandes projetos para o desenvolvimento de *front-end*.
- Vantagens:
  - Tamanho muito pequeno
    - Quanto menor for o tamanho de uma ferramenta *Javascript*, mais será usada e maior será o seu sucesso. Uma das maiores vantagens do *Vue* é o seu tamanho pequeno (de 18 a 21 KB).
  - Flexibilidade
    - Permite que o utilizador escreva um *template* num ficheiro HTML, ficheiro *Javascript* e ficheiro *Javascript* puro, usando nós virtuais.
  - Abordagem simplista
    - Uma das razões para a popularidade desta ferramenta é que é fácil de entender. O utilizador pode facilmente adicionar o *Vue* a um projeto seu existente devido à sua estrutura simples. Basta estar familiarizado com HTML e *Javascript* para começar rapidamente. É isto que torna o *Vue* mais vantajoso para um ambiente de desenvolvimento rápido, pois facilita muito a mudança.
  - Foco
    - *Vue* é uma estrutura relativamente nova e tem muito menos conteúdo. Esse minimalismo e ausência de "geração" na sua estrutura é um dos seus atributos mais fortes, graças ao qual o *Vue* tem um caso de uso bastante restrito. Este foco permite ignorar a grandeza de outras ferramentas.

  - Documentação detalhada
    * Os *developers* gostam de usar ferramentas com documentação detalhada porque facilita no desenvolvimento da primeira aplicação. A documentação do *Vue* é tão abrangente que qualquer utilizador que conheça um pouco de *Javascript* e HTML pode desenvolver a sua própria aplicação ou página da *web*.
  - Reatividade
    * Significa que, a qualquer momento, quando as variáveis mudam, os *peers* são informados.
  - Copiar concorrentes
    * O *Vue* traz eficiência de outras ferramentas. É uma biblioteca baseada em fluxo para gestão de estados e foi inspirada em Elm[49].
  - Comunicação bidirecional
    * O *Vue* facilita a comunicação bidirecional devido à sua arquitetura MVVM (*Model-view-viewmodel*), que facilita o movimento de blocos HTML. Neste aspeto, é muito parecido com o *Angular*.

- Desvantagens:
  - Um desenvolvimento comunitário fechado
    * Ao contrário de outras ferramentas, o *Vue* ainda não tem um amplo suporte pois não é tão popular quanto *React* ou *Angular*. Existem muito mais recursos para *React* do que para *Vue*.
  - Falta de apoio a grandes projetos
    * O *Vue* é uma linguagem relativamente nova, que não é suportada por uma grande comunidade de programadores ativos. Da mesma forma, a equipa de desenvolvimento do *Vue* também é pequena.
  - Ser muito flexível pode ser problemático
    * Às vezes, a flexibilidade pode causar problemas para os *developers*. O *Vue*, por ser tão flexível, pode ser integrado em projetos de larga escala, mas por não haver experiência com soluções possíveis é difícil ter a perceção de saber qual o próximo passo a seguir.
  - Candidatos futuros
    * Por ter uma pequena equipa de desenvolvimento, a falta de uma pesquisa ativa pode fazer com que o interesse se perca para uma

[49] `https://elm-lang.org/`

nova ferramenta. Com isto, o desenvolvimento é afetado, tornando-se mais lento e pode até mesmo estagnar.

- Barreiras linguísticas
  - O criador desta ferramenta é Chinês-americano e apoia muito a comunidade de desenvolvimento chinesa. Os principais utilizadores não falam Inglês, que talvez seja um dos maiores problemas desta ferramenta.

No geral, o *Vue.js* é uma estrutura poderosa para o desenvolvimento de UIs. O seu pequeno tamanho e ferramentas de personalização tornam-no numa escolha sólida para *developers* e é uma ferramenta *user-friendly* para o desenvolvimento de *web*. A maioria das suas desvantagens podem ser facilmente corrigidas, mantendo os *developers* otimistas de que continuará a melhorar no futuro.

4.6.2.6 React

De acordo com [26]:

- O que é o *ReactJS*?
  - O *ReactJS* é uma biblioteca *Javascript* usada no desenvolvimento de *web* que permite criar elementos interativos nos *websites*.

Prós e contras do *React* segundo [27]:

- Prós do *React*:
  - O DOM virtual no *React* melhora a experiência do utilizador e permite ao *developer* trabalhar mais rápido
    - DOM é uma estrutura lógica de documentos nos formatos HTML, XHTML *Extensible Hypertext Markup Language* ou XML. A equipa por trás do *React* conseguiu aumentar a velocidade das atualizações usando o DOM virtual. Diferente de outras ferramentas que funcionam com o DOM real, o *React* usa uma cópia abstrata - o DOM virtual. Deste modo, até as alterações minimalistas aplicadas pelo utilizador são atualizadas, não afetando outras partes da interface. Isto também é possível graças ao isolamento dos componentes do *React* e ao uso de uma estrutura de dados especial na biblioteca. A abordagem permitiu que os *developers* trabalhassem com objetos da UI mais rapidamente e usassem *hot reloading* (aplicando assim alterações em tempo real). Além de aumentar o desempenho, também torna a programação mais rápida.

- Reutilizar componentes de *React* economiza significativamente o tempo
  - Outra vantagem que o *Facebook* introduziu com o *React* é a capacidade de reutilizar componentes de código de um nível diferente a qualquer momento, outro efeito significativo de economia de tempo. Gerir atualizações é então fácil para os *developers*, porque todos os componentes do *React* são isolados e a alteração de um não afeta a de outros. Isto permite reutilizar componentes que não produzem alterações por si só de modo a tornar a programação mais precisa, segura e confortável para os *developers*.
- O fluxo de dados unidirecional no *React* fornece um código estável
  - O *React* permite o trabalho direto com componentes e usa ligação de dados descendente para garantir que as alterações nas estruturas "filho" não afetem os "pais". Isto torna o código estável. Os sistemas mais complexos de *view-model* têm uma desvantagem significativa, mas compreensível - a estrutura do fluxo de dados. Num sistema *view-model*, os elementos "filhos" podem afetar o "pai", se alterados. O *Facebook* removeu estes problemas através do *React*, tornando-o apenas num sistema de *view*. Para alterar um objeto, é necessário modificar o seu estado e aplicar atualizações. Desta forma, apenas os componentes permitidos serão atualizados.
- Uma biblioteca *open-source* do *Facebook*: em constante desenvolvimento e aberta à comunidade
  - O *React* foi um dos primeiros projetos *Javascript open-source* lançados pelo *Facebook*.
- *Redux*: *convenient state container*
  - O *Redux* simplifica o armazenamento e a gestão de estados de componentes em grandes aplicações com muitos elementos dinâmicos. O estado da aplicação é armazenado num único objeto e permite que todos os componentes acedam ao estado da aplicação sem lidar com componentes "filhos" ou usar *callbacks*. O *Redux* fornece um objeto de armazenamento de dados centralizado para permitir que os componentes o acedam diretamente. Além disso, à medida que os estados se tornam mais manuseáveis, a aplicação fica mais fácil de testar e de registar alterações de dados. Por fim, a ferramenta permite ainda *hot reloading*.
- Conjunto de ferramentas *React* e *Redux*
  - O *React* e o *Redux* trazem um conjunto de ferramentas relacionadas que ajudam os *developers*. Por exemplo, a extensão *React Developer*

Tools para Chrome permite examinar hierarquias de componentes no DOM virtual e editar estados e propriedades.

- Contras do *React*:
  - Alto ritmo de desenvolvimento
    - O ambiente muda constantemente e é necessário reaprender regularmente as novas formas de execução. Como está sempre tudo a evoluir, alguns *developers* não se sentem confortáveis em acompanhar este ritmo.
  - Documentação fraca
    - O problema com a documentação remonta a lançamentos constantes de novas ferramentas. Diferentes e novas bibliotecas como *Redux* e *Reflux* prometem acelerar o trabalho de uma biblioteca ou melhorar todo o ecossistema *React*. Como as tecnologias *React* estão a atualizar e a acelerar de forma tão rápida não há tempo para escrever as instruções apropriadas. Para resolver isto, os *developers* escrevem a sua própria documentação para ferramentas específicas usadas por eles nos seus projetos atuais.
  - "HTML no meu *Javascript*!" - JSX (*JavaScript XML*) como barreira
    - O *React* usa JSX. É uma extensão de sintaxe, que permite misturar HTML com *Javascript*. O JSX tem os seus próprios benefícios (por exemplo, proteger o código contra injeções). Porém, muitos *Developers* e *designers* reclamam da complexidade do JSX e da consequente curva de aprendizagem acentuada.
  - Incómodo adicional de SEO (*Search Engine Optimization*)
    - Há preocupações de que o *Google* e outros mecanismos de pesquisa não consigam indexar ou indexem mal páginas dinâmicas da *web*, com a renderização do lado do cliente.

4.6.3 Substituição

A substituição do *Silverlight* foi realizada em 2 partes/versões. Inicialmente, a tecnologia escolhida para realizar esta substituição foi o *Blazor* e o ecrã ao qual se começou por efetuar as alterações foi o ecrã da Graficagem[50], justamente por ser o ecrã mais complexo entre todos. Esta foi a tecnologia escolhida porque como o código do cliente e do servidor são escritos em C#, isto iria facilitar bastante o desenvolvimento dos "novos" ecrãs que previamente já eram escritos em C#. Acontece

[50] Graficagem é uma das operações disponíveis no ***Planeamento***

que esta primeira tentativa de evolução tecnológica não correu tão bem porque não foi possível integrar uma aplicação *Blazor* (ASP .NET Core), que funciona muito à base de módulos simples, dentro do sistema do ***Planeamento*** (ASP .NET), que tem um alto nível de complexidade (principalmente no ecrã da Graficagem). Esta primeira versão levou cerca de 2 meses, desde o fim de março até ao fim de maio. Ao fim destes 2 meses chegou-se à conclusão que não se estava a atingir os objetivos desejados e decidiu-se procurar uma alternativa para combater o tempo perdido. Foi a partir desta altura que surgiu a segunda versão. Como um dos requisitos principais era encontrar alguma "forma" de conseguir realizar um evento de *drag and drop*, devido ao ecrã da Graficagem (ver subsecção 4.6.3.1), decidiu-se simplificar e começar por tentar fazê-lo através de *Javascript*, que já ofereçe a possibilidade de arrastar itens, e tentando tirar partido de alguns *plugins open-source* de *jQuery*. Esta segunda tentativa já correu bem melhor.

4.6.3.1 Ecrã Graficagem

O ecrã da Graficagem permite definir uma folha de serviços na qual vão ser feitas várias viagens. Esta folha de serviços é designada de chapa e é o planeamento daquilo que algum motorista poderá vir a fazer naquele dia. Este ecrã permite ter várias chapas definidas por dia, consoante a hora, arrastando serviços para as chapas específicas (o uso de *Silverlight* era fundamental neste contexto, visto que era o que permitia realizar este efeito de *drag and drop* que é essencial para associar um serviço a uma folha de serviços). Depois de uma chapa estar preenchida, fica então definido um trajeto que será posteriormente atribuído a alguém, ou seja, não fica atribuído a nenhum motorista nem a nenhuma viatura (isso é feito noutro ecrã).

É possível ter versões da mesma chapa. Por exemplo, é criada uma chapa para o dia de hoje, porém durante o dia repara-se que esta não pode estar assim porque, por exemplo, o autocarro está com problemas e então é necessário mudá-la. Desta forma, consegue-se criar uma nova versão da chapa e aquilo que estava criado antes, fica guardado em histórico. Assim, consegue-se saber qual foi a alteração efetuada e saber as mudanças que foram feitas ao longo do dia.

O trajeto definido tem que fazer sentido, isto é, tem de ser coerente no sentido em que, por exemplo, não pode haver um autocarro que faz o percurso de Lixa (cidade do concelho de Felgueiras) a S. Simão (freguesia do concelho de Amarante) e depois tem um serviço do Porto a Lisboa porque há uma falha, entre S. Simão e Porto, que não está definido supostamente o que o autocarro devia fazer (ilustração na figura 4.4, com a falha entre S. Simão e Porto representada a vermelho).

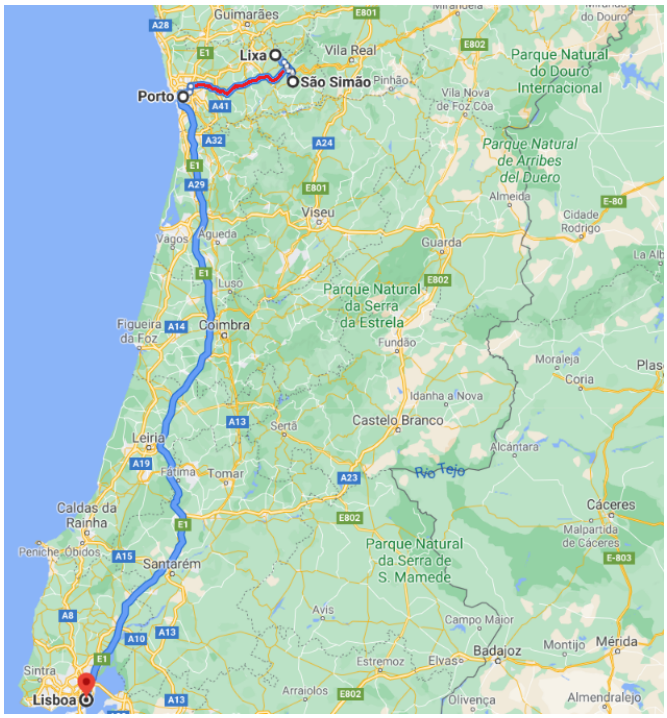


Figura 4.4: Exemplo de trajeto - Graficagem

Outro exemplo, é ter um motorista cujo local de serviço é em Braga e está-se a atribuir uma chapa em que o serviço se inicia em Guimarães, isto vai dar um erro. Além disso, tem de se atribuir pausas aos motoristas, sejam elas, pausas livres, de refeição ou de serviço.

Quando se clica numa circulação, é possível obter mais informações sobre a mesma, como as viagens que faz, distância, horários de passagens, época, etc.

Existem vários "serviços", localizados na parte superior do ecrã, que permitem preencher os buracos dos trajetos que estão a ser definidos:

- Vazio: o autocarro faz uma circulação sem serviço;
- Boleia: é oferecido pela empresa. Permite fazer a boleia do condutor até ao destino ou ir à boleia de outra pessoa que esteja a fazer uma circulação;
- Desdobramento: permite que se faça uma divisão de uma circulação, dividindo-a em dois. Por exemplo, quando em certas alturas do dia em que um autocarro não é suficiente para a circulação (devido a excesso de passageiros, por exemplo na "hora de ponta"). Isto indica então que naquela hora, aquele serviço vai ser feito por dois autocarros;
- Viatura própria: deslocação do condutor na sua viatura própria, até ao destino. Por exemplo, no cenário em que o serviço começa em Braga e o condutor é

de Guimarães, pode-se definir que o condutor usa a viatura dele para ir de Guimarães a Braga para depois realizar o serviço;

- Troca de viatura: é possível fazer a troca de autocarros por causa da quilometragem entre outras medidas de manutenção;
- Dividir: é semelhante ao desdobramento, mas atua de forma diferente;
- Serviço ocasional: é um serviço específico que pode ser definido na hora. É feito a pedido de um cliente;
- Pausa livre;
- Pausa de refeição;
- Pausa de serviço;
- Disponibilidade: o condutor estar disponível para fazer um serviço, mas não tem nada atribuído. Funciona como um descanso pois é uma pausa enquanto está a trabalhar, mas é diferente de uma pausa de serviço em que o objetivo desta é o condutor não poder estar mais de x horas seguidas a conduzir;
- Preparação de serviço: tem uma duração de 5 minutos para se preparar para fazer o próprio serviço;
- Reserva: Igual ao serviço ocasional, mas com hora definida. Tem uma hora específica para ser executado.

Na parte inferior do ecrã, é onde estão configurados todos os serviços - também designados de polígonos - que apenas podem ser realizados dentro da hora que está definida. Estes polígonos contêm o código do serviço, a época, os tipos de dia em que este serviço é executado e a distância entra a origem e o destino, em quilómetros. Cada serviço tem uma cor diferente, para ter uma melhor visualização.

Implementação

De forma a conseguir realizar esta substituição, para este ecrã em específico, foi utilizado um *plugin* de *jQuery* denominado de "sked-tape"[51]. Este *plugin* permite realizar um planeamento diário sob a forma de um calendário tal como é pretendido para o ecrã da Graficagem. Além disso, o uso deste *plugin* trouxe uma nova vantagem para este ecrã: a capacidade de ser *zoomable*, isto é, é possível fazer *zoom* na zona dos serviços para que se consiga visualizar melhor as suas informações.

[51] `https://github.com/lexkrstn/jquery-sked-tape`

Através da API *Drag and Drop* de HTML[52] obteve-se o efeito de arrastar. Por fim, apenas foi necessário "converter" a implementação anterior em *Silverlight* para puro *Javascript*, C# e *jQuery*.

4.6.3.2 Ecrãs Calendário e Planeamento Semanal

Os ecrãs Calendário e Planeamento Semanal não são tão complexos quanto o ecrã da Graficagem porém, também são baseados em calendarização. O ecrã Calendário dá-nos uma visão sobre como se encontra o planeamento dos serviços ao longo dos anos, dos meses e dos dias para uma unidade em questão.

Já o ecrã do Planeamento Semanal permite-nos definir e ter uma perspetiva sobre como está planeada a semana de trabalho dos motoristas para a unidade selecionada. Tal como acontece no ecrã da Graficagem, a forma de associar uma operação a um dia de trabalho de um motorista é através de arrasto. Estas operações servem para complementar os dias em que o motorista não vai trabalhar, especificando o motivo da sua ausência. As operações que estão disponíveis são:

- Férias;
- Greve;
- Baixa;
- Licença parental;
- Licença de maternidade;
- Suspensão;
- Casamento;
- Folga compensatória;
- Descanso semanal;
- Descanso complementar;
- Falta injustificada;
- Falta justificada.

Obviamente que nos dias em que não há nenhuma das operações acima referidas associada, significa que esse é um dia de trabalho normal e, como tal, o seu

[52] https://www.w3schools.com/html/html5_draganddrop.asp

planeamento encontra-se no ecrã da Graficagem (para esse dia em específico) e é facilmente acedido clicando no espaço (retângulo) reservado para esse dia da semana. Nos dias em que o motorista trabalha mas que ainda não tem o planeamento para esse dia definido, um aviso de erro surge no espaço reservado a esse dia, sinalizado a vermelho com a mensagem "Vazio!".

4.6.3.3 Ecrãs Épocas e Tipos de dia

Estes dois últimos ecrãs foram mais simples e, consequentemente, mais rápidos de realizar a substituição do *Silverlight* justamente por serem muito menos complexos face aos referidos anteriormente.

O ecrã Épocas possibilita-nos visualizar quais são as épocas que pertencem à unidade selecionada. Uma época pode ter um ou mais períodos de época associados simbolizando assim em que alturas do ano, para esta unidade em questão, esta época é abrangida. Posto isto, foi necessário adicionar duas tabelas a este ecrã, uma que mostre as épocas e outra que, após selecionar uma época, mostre quais são os períodos dessa época. Cada tabela tem 2 colunas. Na tabela Épocas é possível visualizar o código e o nome da Época. Na tabela Períodos da época é possível visualizar a data de início e a data de fim de cada período. Foi ainda adicionado uma zona com fundo cinzento, do lado direito do ecrã, que serve para visualizar os dias que um período abrange. Após se selecionar um período de época, os dias são carregados para um calendário triplo que se encontra nesta tal zona.

Na implementação deste ecrã, o maior desafio foi realizar a substituição do calendário triplo. Para tal, foi utilizado um *plugin* de *jQuery* denominado de "Multi-DatesPicker"[53] que permitiu representar um calendário triplo referente ao período de época selecionado.

Por outro lado, o ecrã Tipos de dia permite saber quais são os tipos de dia que estão relacionados com todas as unidades do operador selecionado. Para tal, foi adicionado uma tabela com 4 colunas: código, nome, ordem e bilhética. Um tipo de dia que tenha, por exemplo, o código "2346" significa que atua às segundas, terças, quartas e sextas-feiras. Esta tabela permite listar todos os tipos de dia do operador e, ao clicar-se num, surge, do lado direito do ecrã, uma tabela que ilustra através de *checkboxes* os dias pertencentes a esse tipo de dia, desde domingo a sábado, por linha e as propriedades do tipo de dia - todos, feriados, véspera de feriados, pós-feriados e, restantes - por coluna.

[53] https://dubrox.github.io/Multiple-Dates-Picker-for-jQuery-UI/

Capítulo 5

Conclusão

Neste último capítulo são abordados os resultados obtidos, os desafios superados e as dificuldades enfrentadas ao longo do projeto. É ainda feita uma proposta para trabalho futuro.

5.1 Discussão

Este projeto incidiu essencialmente na melhoria de produtos que existem nos sistemas ***Planeamento*** e ***Localização***. As alterações efetuadas no sistema ***Planeamento*** tiveram impacto na forma como se planeiam e se gerem as operações de transportes públicos, para cada operador, enquanto que as alterações ao sistema ***Localização*** tiveram em vista uma aprimoração da informação que é utilizada em contexto real.

Em termos temáticos este projeto enquadra-se em três áreas distintas: *Transport-on-Demand*, *Mobility as a Service* e Gestão de recursos. Todas elas requereram uma pesquisa com o objetivo de encontrar produtos e soluções que pudessem, de certa forma, ajudar no desenvolvimento deste projeto, tal como foi apresentado no Capítulo 2.

Quanto aos objetivos traçados inicialmente, todos foram atingidos na íntegra. As tarefas realizadas durante o desenrolar do projeto permitiram este cumprimento. Um exemplo disso foi a adição do termo "paragens satélites" ao ***Planeamento***, que consentiu atingir vários objetivos previstos, como a receção de serviços (pedidos de ativação de paragens satélites) e alocação de meios para os realizar (atribuir um motorista e uma viatura para satisfazer o pedido), a visualização e edição de serviços previstos (sempre que um serviço a pedido é aceite, um serviço regular é alterado) e a integração de funcionalidades no *BackOffice*. Outro exemplo foi a criação do ecrã "Editar percursos" no ***Planeamento***, que proporcionou a utilização de recursos provenientes de APIs abertas, como foi o caso do *OpenStreetMap*, e permitiu observar a iniciação, execução e conclusão de serviços, através da adição de percursos

planeados ao mapa presente no *dashboard* do ***Localização***. Por último, a evolução tecnológica realizada na secção 4.6 levou a que fossem selecionadas tecnologias de substituição em alguns ecrãs do ***Planeamento***.

Para realizar todo o trabalho desenvolvido foi necessário enfrentar um conjunto de desafios, desafios estes que me fizeram crescer a nível profissional e que trouxeram-me alguns benefícios como aprender a trabalhar com linguagens de programação não familiarizadas, estudar o funcionamento dos sistemas que existiam atualmente na empresa e aplicar-lhes alterações com vista a torná-los em plataformas mais completas e eficientes.

Como em qualquer desafio, existiram adversidades que vieram dificultar o desenvolvimento das tarefas previamente definidas. Para além da pandemia mundial causada pelo COVID-19 que, de certa forma, obrigou-nos a arranjar uma nova forma de comunicar entre equipa, surgiram dificuldades a nível de calendário e de aprendizagem. Relativamente à calendarização houve tarefas que não foram realizadas nos prazos traçados inicialmente o que, consequentemente, levou a um atraso no plano. Por outro lado, uma aprendizagem mais demorada sobre o funcionamento dos sistemas e sobre as tecnologias por eles utilizadas, como foi o caso do *Silverlight*, levaram ao incumprimento de algumas tarefas nas datas previstas. Uma das maiores consequências destes atrasos foi mesmo a remoção de uma tarefa do plano. Esta tarefa pertencia à secção 4.6 pois consistia no mesmo objetivo porém, com o uso de outra tecnologia de substituição, nomeadamente, *OutSystems*[54]. O objetivo de ter duas tarefas "iguais" com tecnologias diferentes era a de criar um estudo entre ambas para analisar e compará-las em termos de tempo e custo. O problema é que, tal como foi explicado anteriormente, começou por tentar substituir-se o *Silverlight* por *Blazor*, o que não foi bem sucedido. Após 2 meses, foi testada outra alternativa com mais sucesso e que acabou por permanecer até hoje. Como se perdeu muito tempo na primeira tentativa, decidiu-se então por não prosseguir para outra. Desta forma não foi possível realizar a tal comparação entre diferentes soluções, que seria bastante interessante e enriquecedora para este trabalho.

Por fim, concluo que este projeto foi realizado com sucesso e que teve resultados bastantes positivos tanto para a empresa como para mim. A empresa passou a ter novas soluções nos seus sistemas, destacando-se a introdução das paragens satélites que trouxe uma inovação na forma como se lida com os serviços regulares dos transportes públicos. E, pessoalmente, penso que este trabalho ajudou-me a crescer muito a nível profissional pois desenvolvi competências tecnológicas que ajudar-me-ão futuramente.

[54] `https://www.outsystems.com/`

5.2 Trabalho futuro

Obviamente não existem soluções perfeitas e este projeto não é exceção disso. Há ainda algum trabalho pela frente por desenvolver de forma a aprimorar o que foi implementado.

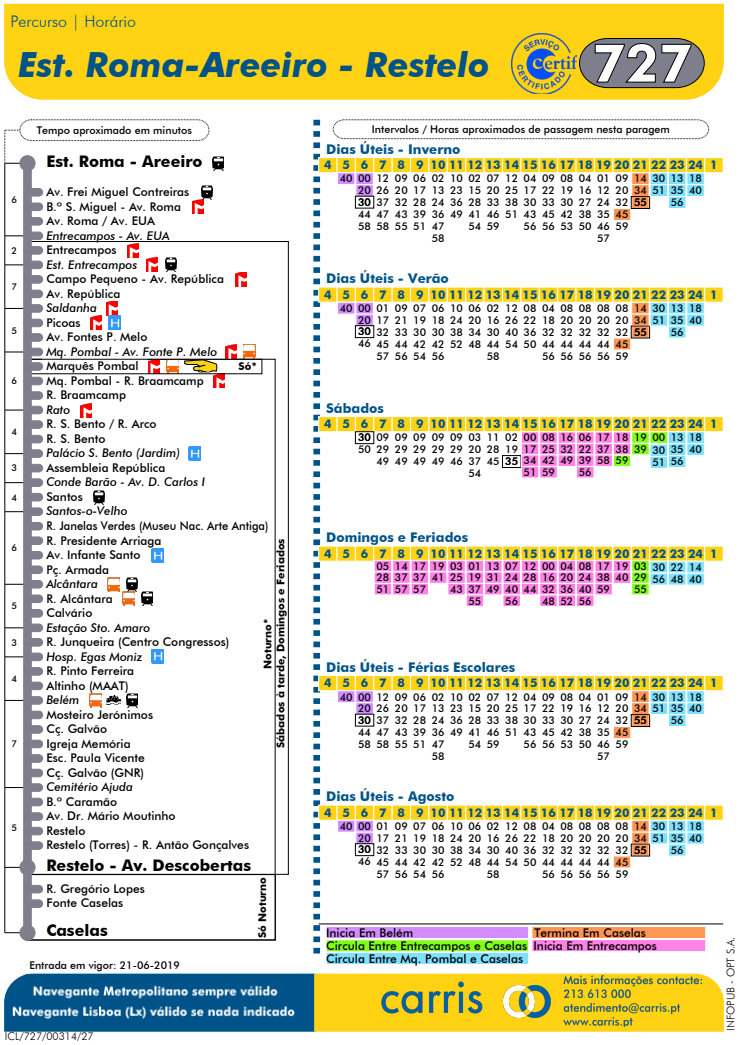
O *widget web*, por exemplo, está constantemente a sofrer alterações quer sejam de manutenção quer sejam para adição de novas funcionalidades e é por esta razão que a lista de modificações mencionada na secção 4.1 foi aumentando durante o desenrolar deste projeto.

Por outro lado, na implementação das paragens satélites, ainda falta criar o mecanismo de enviar solicitações de ativação de serviços a pedido do lado do cliente, ou seja, do lado da ***App Cliente***. Apesar do ***Planeamento*** já ter na sua API a capacidade para receber tais pedidos e de se terem efetuado testes que os simulem e que comprovam o seu funcionamento, será importante ver como se comporta em contexto real e todos os problemas que daí surgirão.

Para terminar, a evolução tecnológica ocorrida no ***Planeamento*** ainda precisa de sofrer mais testes que comprovem o seu correto funcionamento bem como um mecanismo que melhore o seu desempenho, nomeadamente no ecrã Graficagem que lida com uma quantidade enorme de dados. Para este caso em particular, propunha a utilização de algumas tecnologias que ajudassem na diminuição dos tempos de resposta do sistema, nos problemas de cache e no tratamento dos pedidos em fila de espera, como por exemplo as que foram referidas na sub-subsecção 2.1.3.1.

Apêndice A

Carreira 727 - Carris



Apêndice B

Implementações em código

Os anexos aqui apresentados representam as soluções utilizadas nas implementações das tarefas ao longo do trabalho desenvolvido.

B.1 Representação de uma paragem no mapa

```
{
  "Id" : "",
  "Nome" : "",
  "Latitude" : "",
  "Longitude" : ""
}
```

Listing B.1: Representação de cada paragem no mapa - ***Planeamento***

Apêndice C

Modelos de dados

Estes anexos têm o objetivo de clarificar a interação entre diferentes componentes dos sistemas ***Planeamento*** e ***Localização***.

C.1 Serviço a pedido

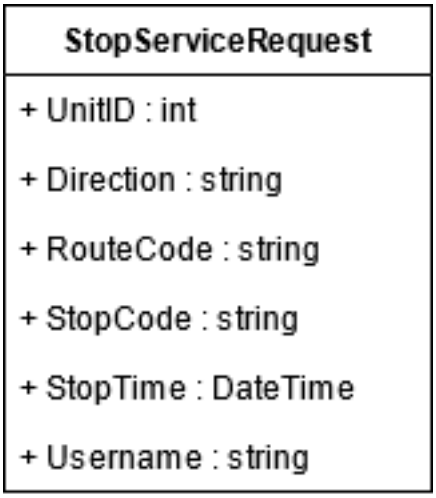


Figura C.1: Diagrama de classes para um serviço a pedido (***Planeamento***)

C.2 *StopOnDemand*

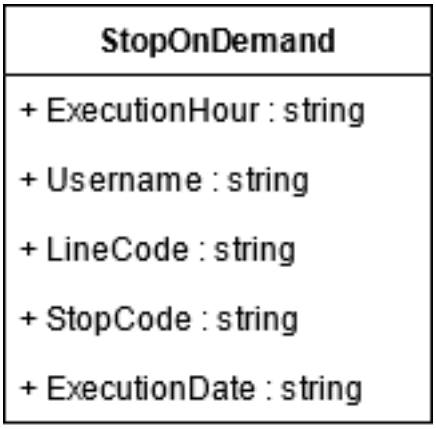


Figura C.2: Diagrama de classes para um objeto *StopOnDemand*

C.3 Entidades de um percurso

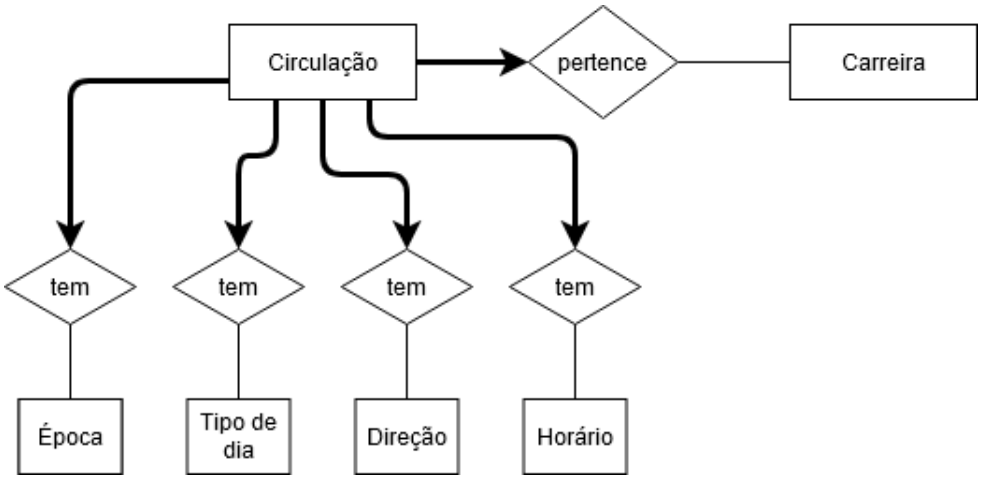


Figura C.3: Modelo Entidade-Associação das entidades de um percurso

Bibliografia

[1] Michelle Felt, Nader Gharachorloo, and Afshin Moshrefi. Mobile taxi dispatch system, April 28 2011. US Patent App. 12/607,782.

[2] M. Stemerding and C. Van Eldik. Market research to design a system for transport on demand: The case of the hague region. https://trid.trb.org/view/659094. [Acedido em 09/12/2019].

[3] MaaS Alliance. What is maas? https://maas-alliance.eu/homepage/what-is-maas/. [Acedido em 09/12/2019].

[4] Transportes em Revista Online. Transportes a pedido. http://www.transportesemrevista.com/Default.aspx?tabid=210&language=pt-PT&id=1845. [Acedido em 11/12/2019].

[5] Expresso. Lisboa entre as cidades com trânsito mais caótico do mundo. https://expresso.pt/economia/2019-11-14-Lisboa-entre-as-cidades-com-transito-mais-caotico-do-mundo. [Acedido em 11/12/2019].

[6] DECO PROTESTE. Poluição do ar em portugal: retrato negativo. https://www.deco.proteste.pt/saude/doencas/noticias/poluicao-do-ar-em-portugal-retrato-negativo. [Acedido em 11/12/2019].

[7] Linh NK Duong, Lincoln C Wood, Jason X Wang, and William YC Wang. Transport on-demand in a service supply chain experiencing seasonal demand: Managing persistent backlogs. *International Journal of Operations Research*, 14(3):121–138, 2017.

[8] Mais Ribatejo. Transporte a pedido liga cidades do médio tejo - "Link, estamos todos ligados". https://maisribatejo.pt/2019/11/13/transporte-a-pedido-liga-cidades-do-medio-tejo-link-estamos-todos-ligados/. [Acedido em 09/12/2019].

[9] CCDR Alentejo. Projeto piloto - transporte a pedido. https://www.ccdr-a.gov.pt/index.php/28-noticias/886-projeto-piloto-transporte-a-pedido. [Acedido em 09/12/2019].

[10] Uber. Como funciona a uber? https://help.uber.com/pt-PT/riders/article/como-funciona-a-uber?nodeId=738d1ff7-5fe0-4383-b34c-4a2480efd71e. [Acedido em 09/12/2019].

[11] Expresso. Já há mais veículos uber em lisboa, porto e faro que táxis. https://expresso.pt/revista-de-imprensa/2019-07-16-Ja-ha-mais-veiculos-Uber-em-Lisboa-Porto-e-Faro-que-taxis. [Acedido em 11/12/2019].

[12] Lucie Lozinski. The uber engineering tech stack, part i: The foundation. https://eng.uber.com/tech-stack-part-one-foundation/. [Acedido em 18/10/2020].

[13] Intelligent Transport. Berlin launches the world's largest maas solution. https://www.intelligenttransport.com/transport-news/88799/berlin-launches-the-worlds-largest-maas-solution/. [Acedido em 10/12/2019].

[14] Jana L Sochor, Helena Strömberg, and Marianne Karlsson. Travelers' motives for adopting a new, innovative travel service: Insights from the ubigo field operational test in gothenburg, sweden. In *21st World Congress on Intelligent Transport Systems, Detroit, September 7-11, 2014*, 2014.

[15] Civitas. Ubigo. https://civitas.eu/tool-inventory/ubigo. [Acedido em 10/12/2019].

[16] Techopedia. Metacomputing. https://www.techopedia.com/definition/25143/metacomputing. [Acedido em 10/12/2019].

[17] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer, 1998.

[18] Microsoft. Fim do suporte do silverlight. https://support.microsoft.com/pt-pt/help/4511036/silverlight-end-of-support. [Acedido em 06/12/2019].

[19] Justin James. How to replace flash and silverlight with html5. https://www.techrepublic.com/blog/web-designer/how-to-replace-flash-and-silverlight-with-html5/. [Acedido em 06/12/2019].

[20] David Ramel. Silverlight reborn? check out 'c#/xaml for html5'. https://visualstudiomagazine.com/articles/2018/02/26/cshtml5.aspx. [Acedido em 06/12/2019].

[21] Spruha Pandya. Why is it finally time to let angularjs replace silverlight? https://medium.com/@spruha.pandya/why-is-it-finally-time-to-let-angularjs-replace-silverlight-b0c0f2d51c5d. [Acedido em 06/12/2019].

[22] CG-VAK Software & Exports Ltd. Microsoft silverlight to angular migration - it is time to say goodbye to the legacy technology. https://www.cgvakindia.com/blog/microsoft-silverlight-to-angular-migration-it-is-time-to-say-goodbye-to-the-legacy-technology/. [Acedido em 06/12/2019].

[23] Microsoft. Blazor - build client web apps with c#. https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor. [Acedido em 06/12/2019].

[24] Hélio Moreira. Tutorial c# - o que é o wpf (windows presentation foundation). https://pplware.sapo.pt/tutoriais/tutorial-c-o-que-e-o-wpf-windows-presentation-foundation/. [Acedido em 26/11/2019].

[25] Mobio Solutions. What is vue.js and what are its advantages and disadvantages. https://mobiosolutions.com/what-is-vue-js-and-what-are-its-advantages-and-disadvantages/. [Acedido em 29/11/2019].

[26] Scott Morris. Tech 101: What is react js? https://skillcrush.com/2019/05/14/what-is-react-js/. [Acedido em 29/11/2019].

[27] AltexSoft. The good and the bad of reactjs and react native. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/. [Acedido em 29/11/2019].