UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Long Distance Bus - Yield Management

Anastasiya Zyenina

Mestrado em Engenharia Informática

Especialização em Interação e Conhecimento

Trabalho de projeto orientado por:
Prof. Doutora Maria Isabel Alves Batalha Reis da Gama Nunes

2020

Agradecimentos

O maior agradecimento é para a minha melhor amiga, Renata Roncon, que esteve presente em todos os momentos da escrita desta dissertação, dando apoio, incentivos e conselhos sempre que necessário.

Quero agradecer também ao meu namorado que me deu sempre motivação e um grande empurrão quando necessário para continuar.

Agradeço também à professora Doutora Maria Isabel Gama Nunes, pela enorme ajuda, orientação e disponibilidade ao longo da realização desta dissertação, sem a qual não seria possível a elaboração deste documento.

Gostaria ainda de agradecer à instituição CARD4B, por me ter acolhido e ter dado oportunidade de realizar este trabalho. Além disso quero agradecer à Ana Rita Caçador pela disponibilidade e apoio prestado ao longo de todo o trabalho, ao Maksym Bondarchuk pelos ensinamentos e concelhos dados, e ao resto dos colegas pelo ambiente acolhedor e simpatia.

Por fim agradeço aos meus pais e irmã pelo apoio prestado ao longo da realização da presente dissertação.

Resumo

No âmbito da cadeira de Dissertação do 2º ano de Mestrado em Engenharia Informática, especialização em Interação e Conhecimento, na Faculdade de Ciências da Universidade de Lisboa, foi realizado o presente documento com o objetivo de relatar o trabalho realizado no projeto *Long-Distance Bus – Yield Management* (YM) na empresa Card4B - Systems, S.A.

Este projeto, tal como o nome sugere, está diretamente relacionado com transportes de longo curso, mais especificamente com os autocarros, *YM* e tarifas dinâmicas nos transportes rodoviários de longo curso. Ao longo deste documento cada conceito será detalhadamente estudado e descrito.

YM tem sofrido grandes mudanças ao longo dos últimos anos, sendo que a maioria dos mercados passou a desenvolver e a aplicar técnicas e estratégias cada vez mais sofisticadas. A indústria de autocarros de longo curso apenas recentemente passou a aplicar estratégias de YM mais inteligentes e dinâmicas, existindo por isso poucos estudos e trabalhos disponíveis nesta indústria, daí o objetivo desta dissertação ser estudar bem o meio e soluções, a fim de perceber como aplicar a estratégia e qual a melhor forma de o fazer.

O presente documento faz a análise de todas as etapas do processo de aplicação de YM, desde o estudo do sistema e criação de novas ferramentas de gestão, até à sugestão de novos módulos e de uma nova arquitetura.

De uma forma sucinta o trabalho pode ser dividido em quatro etapas principais. A primeira etapa consiste em estudar o sistema, que atualmente se encontra em funcionamento em empresas de autocarros de longo curso, com o intuito de perceber e identificar quais as suas limitações e quais as alterações que este precisa de enfrentar. A segunda etapa é a de estudo de ferramentas e soluções existentes, com o objetivo de poder criar uma ferramenta (aplicação *Web*) que ajude na modernização e gestão do sistema YM já existente, a fim de poder geri-lo e integra-lo, com maior facilidade, com as novas alterações que se pretendem introduzir. São também realizados o estudo e o levantamento de novos fatores a introduzir no YM, mais especificamente no cálculo de preços, com a finalidade de ter as ofertas mais inteligentes e atualizadas. Por fim, com base no estudo realizado e nas ferramentas criadas, é feita a sugestão de uma nova arquitetura a aplicar no sistema.

Com este trabalho pretende-se, no futuro próximo, integrar todo o conhecimento desenvolvido e adquirido e ferramentas criadas no sistema real de uma empresa de autocarros de longo curso, tornando a gestão mais simples e os preços mais dinâmicos.

Palavras-chave: *Yield Management*, autocarros de longo curso, preços, ferramentas de gestão, arquitetura.

Abstract

Within the scope of the Dissertation module of the 2nd year of the Master in Informatics Engineering, specialization in Interaction and Knowledge, at the Faculty of Sciences of the University of Lisbon, this document reports the work carried out under the Long-Distance Bus project - Yield Management (YM) at Card4B - Systems, SA.

This project, as the name suggests, is directly related to long-distance transports, more specifically buses, YM and dynamic road fares, concepts that will be explored further in detail throughout the report.

YM has undergone major changes over the past few years, most markets have started to develop and apply increasingly sophisticated techniques and strategies. The long-distance bus industry has only recently started to apply more intelligent and dynamic YM strategies, consequently, there are few studies and works available about YM in this industry, hence, the objective of this dissertation is to study the environment and solutions in order to understand how to apply the strategy and what is the best way to do it.

This document goes through an analysis of all stages of the YM application process, from studying the current system and creating new management tools, to suggesting new changes and architecture.

In general, the work can be divided into four main stages. The first step is to study the system, which is currently operating in long-distance bus companies, in order to understand and identify what its limitations are and what changes it needs to face. The second step is to study existing tools and solutions, in order to be able to create a tool (Web application) that helps in the automation of the existing YM system, and to be able to manage it and integrate it more easily with the new changes to be made. The study and survey of new factors to be introduced in YM are also carried out, more specifically in the calculation of prices, to have the most intelligent and updated offers. Finally, based on this study and on tools created, the suggestion of a new architecture to be applied in the system is carried.

This work paves the way to, in a near future, integrate all the developed and acquired knowledge and tools created, in the real system of a long-distance bus company, making management simpler and prices more dynamic.

Keywords: *Yield Management*, long-distance bus, price, management tools, architecture.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O conceito de *Yield Management* (YM) surgiu na indústria de transportes aéreos no final dos anos 70. A maioria das pessoas que viajam sabem que os passageiros do mesmo voo pagam tarifas diferentes. A variação das tarifas depende de diversos fatores, tais como a data de compra, idade do passageiro, tipo de lugar ou até de eventos externos como concertos ou feriados nacionais/internacionais. O cálculo dinâmico de tarifas (*Dynamic Pricing*) surgiu com o objetivo de maximizar os proveitos decorrentes da maior ocupação.

Atualmente, YM é aplicado em diversas indústrias tais como hotelaria, aluguer de carros, linhas de cruzeiros e autocarros de longo curso, entre outras. Neste documento será estudado o caso dos autocarros de longo curso, pois em Portugal a aplicação de tarifas dinâmicas nos transportes rodoviários interurbanos é quase restrita a ocorrências sazonais planeadas a médio prazo e depende pouco da avaliação dinâmica.

1.1 Motivação

Atualmente, *Yield Management* e *Dynamic Pricing* são aplicados em diversas indústrias, não tendo ainda sido muito explorado, no entanto, na indústria dos autocarros de longo curso. Algumas empresas como a FlixBus, que já recorrem a este modelo (Gaggero, Branko Bubalo and Ogrzewalla), estão a ter bastante sucesso e a expandir-se pelo mundo, abrangendo já Portugal.

No entanto, em Portugal as maiores empresas de transportes ainda utilizam métodos mais básicos que apenas se baseiam em informações estáticas, como a idade do passageiro, descontos pontuais que surgem de mudanças de mercado, ou descontos sazonais. As empresas portuguesas dominam o mercado nacional, no entanto é sempre necessário evoluir para fazer frente aos concorrentes.

Daí vem a necessidade de criação de uma ferramenta/algoritmo que seja capaz de calcular e alterar as tarifas de forma dinâmica com base em informações externas, ajudando assim a maximizar os proveitos.

Além disso, é um tema que é sempre atual visto que pode ser aplicado em qualquer indústria e existe pouca informação disponível em relação aos algoritmos aplicados ou à sua funcionalidade.

1.2 Objetivos

Um dos objetivos do trabalho aqui descrito foi o de desenvolver uma ferramenta *front-end* que permite fazer a gestão de regras de preços instanciadas ao transporte de passageiros de longo curso e a sua integração numa ferramenta de *BackOffice* (BO). Esta ferramenta será referida daqui em diante como *BackOffice* ou aplicação *Web*.

Para além deste objetivo, pretendeu-se também estudar a temática de *Dynamic Pricing*, perceber como aplicar esta estratégia na indústria de autocarros de longo curso, de que forma tornar dinâmicas as regras de cálculo de preços e como integrá-las no *BO*.

1.3 Contribuições

Com esta aplicação *BackOffice* é possível tornar rápido, simples e intuitivo o processo de gestão de regras de preços instanciadas ao transporte de passageiros de longo curso. Esta aplicação *Web* veio proporcionar liberdade aos clientes da instituição de acolhimento (as empresas de autocarros de longo curso) na gestão e criação das regras de forma autónoma visto que, até à data, esses clientes eram obrigados a fazer o pedido à empresa fornecedora sempre que era necessário inserir alguma alteração.

A segunda parte da dissertação contribui com o estudo da temática de *Dynamic Pricing* no contexto dos autocarros de longo curso, visto que é um tema pouco estudado nesta área. É feito também o levantamento de metodologias, variáveis e construção de algumas ferramentas que são necessárias para a definição e implementação de regras, de modo a que, numa próxima etapa, seja possível dinamizar as regras de cálculo de preços tornando os preços mais competitivos.

1.4 Enquadramento Institucional

O projeto descrito neste documento está inserido no âmbito da disciplina Dissertação/Projeto de Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa (FCUL), sendo esta disciplina um requisito para a conclusão do Mestrado em Engenharia Informática (MEI).

O projeto foi desenvolvido na empresa Card4B – Systems, empresa fundada em 2007 que nasceu após a identificação da necessidade de soluções para uma "nova cultura de mobilidade" nos centros urbanos. O seu foco comercial é produzir *software* de bilhética *front-end* e *back-end* para empresas de transporte e fornecer serviços especializados para soluções de Mobilidade Integrada e serviços da cidade, como transporte público, rua e estacionamento na rua, pedágios, táxis, partilha de carros, partilha de bicicletas, transporte a pedido, entre outros.

1.5 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2 – Trabalho relacionado, que abrange o contexto histórico em que é inserido o trabalho e análise dos conceitos teóricos necessários para a compreensão do trabalho realizado e da sua importância;
- Capítulo 3 – Análise do sistema do módulo exemplo e das suas limitações;

- Capítulo 4 – Estudo de ferramentas e tecnologias necessárias para a implementação da aplicação *Web* para a gestão estática de rendimentos (*Yield Management*);
- Capítulo 5 – Descrição do trabalho realizado, incluindo uma breve análise da arquitetura da aplicação *Web* implementada, decisões tomadas na implementação e análise das funcionalidades implementadas e por implementar;
- Capítulo 6 – Análise de ferramentas e tecnologias necessárias para transformar o *Yield Management* estático em dinâmico. Sugestão de uma nova arquitetura e de regras a inserir para tornar o processo de gestão de rendimentos dinâmico;
- Capítulo 7 – Considerações finais acerca do trabalho realizado e discussão do trabalho futuro.

Capítulo 2

Trabalho Relacionado

Esta secção aborda a gestão de rendimentos (*Yield Management* - YM) tanto estática como dinâmica. Numa primeira parte é realizada uma pesquisa acerca de YM estático com o intuito de compreender o conceito e o funcionamento do sistema. Numa segunda parte é estudado o *Dynamic Pricing* (DP) com o propósito de compreender como tornar o YM estático num sistema mais automático e dinâmico.

2.1 Contexto Histórico de *Yield Management*

Antes de ser explicado o conceito *Yield Management* (YM) é importante perceber a sua origem.

Tudo começou com a desregulamentação da aviação nos Estados Unidos nos finais dos anos 70, mais precisamente em 1978.

A necessidade desta mudança surgiu devido às dificuldades que o transporte aéreo enfrentava há décadas. Para além da crise mundial do petróleo que existia na altura, o que dificultava ainda mais a situação eram os modelos de gestão das empresas, que estavam bastante ultrapassados. Os preços até então oferecidos, eram calculados normalmente com base na distância percorrida mais uma margem de lucro determinada pelo governo. Assim, não sendo possível fazer uma competição baseada em preços, esta girava em volta do serviço prestado ao passageiro (Monteiro). Além disso, qualquer alteração que se queria introduzir na empresa, tinha que passar pela aprovação dos sindicatos, o que muitas vezes não acontecia, impedindo assim as empresas de se colocarem em situações mais competitivas.

Assim, em 24 de outubro de 1978, o Presidente Jimmy Carter assinou o Airline Deregulation Act que deu início à desregulamentação do transporte aéreo nos Estados Unidos. Desregulamentação é o processo de remover ou reduzir regulamentações do estado, o que levou à remoção de controlo do estado sobre diversas áreas, tais como rotas, tarifas, horários, etc.

O objetivo da desregulamentação implementada foi atribuir autonomia às empresas aéreas para estipular as suas rotas e horários de voos segundo as suas conveniências, visto que, até a data, todo o processo era controlado pelo Civil Aeronautics Board (CAB), que era um processo demorado para qualquer pedido, como por exemplo, pedidos de entrada setorial ou expansão de rotas. Passou assim a existir um único critério de aprovação, que consistia em determinar se a empresa tinha capacidade para prover o novo serviço com competência, segurança e eficiência (Souza and Nunes).

Esta mudança levou de imediato a grandes transformações no setor aéreo, pois as novas regras permitiram que empresas *low-cost* invadissem o mercado, o que levou de imediato a uma descida drástica dos preços, pois, até a data, viajar de avião era considerado um luxo e os preços eram muito elevados.

Esta alteração permitiu também liberdade no desenvolvimento de novos serviços, estratégias de *marketing* e preços, que posteriormente foram introduzidos e aplicados (Schmidt Hahn de Lima, Montenegro de Lima and Agrello Dias).

Relativamente às novas empresas *low-cost* que entraram no mercado, estas ofereciam preços 70% abaixo dos preços oferecidos até a data pelas grandes companhias aéreas, o que levou as grandes empresas a

um impasse, visto que não conseguiam competir com preços tão baixos. Uma redução nos preços deste calibre levaria as companhias a falência, pois não teriam forma de cobrir os custos; por outro lado, se mantivessem as tarifas mais altas do que as dos concorrentes, iriam perder os seus passageiros.

Estas novas condições levaram a uma guerra de preços que prejudicou imenso a margem de lucro das companhias aéreas, forçando-as a concentrarem-se em novas abordagens e técnicas para a gestão da procura. (Fossati Figueiredo, Machado Silva and Pereira Pinto Junior).

Perante esta situação, em 1979, Robert Crandall Cross, na altura presidente e CEO da companhia American Airlines, começou à procura de soluções para o problema da ocupação de lugares nos aviões (*load factor*), pois com a queda de vendas de bilhetes, a venda das tarifas regulares não ultrapassava os 40%. Assim o desafio era perceber como vender os restantes 60% dos bilhetes (Lieberman).

Em resposta à situação, em 1985 a American Airlines (AA) lança "Ultimate Super Saver" como tentativa desesperada de aumentar as suas receitas. A ideia consistia em vender os assentos vazios a tarifas baixas, tal como as companhias aéreas *low-cost*, de forma a atrair passageiros que já fossem clientes da AA, ou até novos, tendo o cuidado, no entanto, para não vender demasiados lugares a esses preços antecipadamente, pois o objetivo era vender o máximo de assentos a tarifas normais (Monteiro). Existiam apenas duas diferenças relativamente às tarifas baixas oferecidas por companhias *low-cost*:

1. Se um passageiro quisesse comprar uma tarifa "Ultimate Super Saver", ele precisava de fazer a reserva pelo menos duas semanas antes da partida e permanecer no seu destino durante uma noite de sábado.
2. O número de assentos que poderiam ser vendidos por preço com desconto foi restrito. Desta forma, a American Airlines poderia reservar lugares com tarifa completa para clientes que fizessem a reserva apenas alguns dias antes da partida.

Estas medidas não eram suficientes para superar o problema na totalidade, por isso Robert Crandall continuou à procura de soluções para conseguir encontrar um equilíbrio entre o *load factor* e a receita média de passageiro por assento-quilómetro (*Yield*) de forma a maximizar a receita de cada voo (Monteiro). Posteriormente, após trabalhar bastante tempo na solução e de observar também o resultado de aplicação do "Ultimate Super Saver", apercebeu-se de que o valor percebido do produto variava de passageiro para passageiro, ou seja, de quanto cada um está disposto a pagar. Desta forma foi possível separar os passageiros em dois tipos: os que viajam por lazer e os que viajam em negócios (Treszl). Cross conjugou então esta nova ideia com a de gestão de assentos disponíveis com desconto e denominou essa nova estratégia de *Yield Management*.

Este é considerado o nascer do *Yield Management*. Com o tempo, cada vez mais companhias aéreas começaram a aplicar este método de gestão e, *a posteriori*, até outras indústrias, tais como a hoteleira e a de aluguer de carros.

2.2 Definição de *Yield Management*

Existem inúmeras definições para *Yield Management* (YM) disponíveis na literatura e na *Internet*. As próprias definições foram mudando com o passar do tempo – houve autores, que tendo atribuído uma definição ao conceito, anos mais tarde reformularam as suas ideias e definições.

Em 1987, Darren Lee-Ross e Nick Johns definiram YM como "o processo que se foca na maximização do lucro, usando informações sobre as vendas e os compradores " (Balkoulis). Sheryl E. Kimes (1989) atribui a seguinte definição "YM é o processo de reservar o tipo certo de capacidade ao tipo certo de cliente, pelo preço certo, a fim de maximizar a receita ou o rendimento." (Kimes, The basics of yield management.).

A definição sugerida por Barry C. Smith, John F. Leimkuhler, e Ross M. Darrow (1992) foi "YM é uma maneira sofisticada de gerir a oferta e a procura, atuando simultaneamente nos preços e na capacidade disponível. É um processo de atribuir o melhor serviço ao melhor cliente, ao melhor preço e no melhor momento" (Selmi and Raphael Dornier). Jauncey, Mitchell e Slamet (1995) viam YM como "uma abordagem integrada, contínua e sistemática para maximizar a receita através da manipulação do preço do produto em resposta a padrões de procura previstos" (Okumus) .

YM foi também definido como "Precificação dinâmica, *overbooking* e reserva de bens perecíveis nos segmentos de mercado, com objetivo de maximizar as receitas de curto prazo para a empresa" (Baker, Collier, 1999) (Balkoulis).

Em 2009, Wang e Bowie usaram uma descrição que, do seu ponto de vista, é o conceito fundamental – "O principal objetivo é maximizar a receita através da gestão eficaz de três áreas principais: estratégia de preços, controlo de *stock* e controlo de disponibilidade" (Wang and David Bowie).

Como foi possível observar, não existe uma definição única ou geral para o conceito de *Yield Management,* diferindo de acordo com as perspetivas de diferentes setores e de diferentes autores.

De uma forma geral, tentando sumarizar todas as ideias, YM consiste em definir o melhor compromisso possível entre os custos e o produto oferecido, ou seja, atribuir preços certos aos produtos ou serviços, a fim de vender no momento mais adequado, aos clientes mais adequados, concentrando-se assim apenas no preço de venda para gerar a maior receita possível a partir de um *stock* limitado e perecível.

Muitas vezes o termo *Yield Management* (YM) é confundido com *Revenue Management* (RM).

RM surgiu da necessidade de incorporar outras atividades inerentes à captação da receita, visto que YM sozinho não era suficiente para aumentar a receita e conseguir mantê-la. Sendo assim, a união de YM com as outras atividades, tais como planeamento, *marketing*, precificação, etc., deu origem ao que hoje chamamos *Revenue Management* (Monteiro).

2.3 Estratégias *Yield Management*

Yield Management (YM) é o termo generalizado para um conjunto de estratégias que permitem que os diferentes setores de serviços, com capacidade de produto oferecido restrita, obtenham a receita ideal das operações (Withiam).

Segundo Sheryl E. Kimes e Richard B. Chase (1998), na definição de YM de Smith, Leimkuhler e Darrow, em 1992 – " oferecer o serviço certo ao cliente certo, na hora certa, pelo preço certo" – encontra-se implícita a noção de capacidade perecível no tempo e, por extensão, a noção de segmentação da capacidade de acordo com quando é reservada, quando e quanto tempo deve ser usada, e de acordo com o cliente que o utiliza (Kimes and Chase, The Strategic Levers of Yield Management). Desta forma, Kimes e Chase sugerem que as alavancas estratégicas de YM podem ser resumidas em função dos quatro Cs (4-C, *calendar*, *clock*, *capacity* e *cost*) : calendário (antecedência com que as reservas são feitas), relógio (hora/momento do dia em que o serviço é prestado), capacidade (capacidade de recursos

disponíveis) e custo (preço do serviço oferecido ao cliente). Estes 4-C estão ligados entre si para gerir um quinto "C": a procura do cliente, de forma a maximizar o lucro.

C1 - Calendário: Certos dias ou estações têm diferente procura em diferentes serviços, tornando-se desta forma mais rentáveis ou menos rentáveis. Para o serviço de autocarros de longo curso, a sua procura aumenta, por exemplo, às sextas-feiras, quando os estudantes que estudam fora querem voltar para casa, feriados ou dias de eventos ou então no verão, em dias de sol, quando as pessoas querem ir à praia. Na indústria de restauração, o Dia dos Namorados e o Dia da Criança são dias que têm uma elevada procura. Desta forma, a gestão do calendário consiste em prever a procura para determinadas datas, quando esta é elevada e quando é baixa. Ao fazer uma previsão correta, o sistema YM torna possível determinar quando abrir ou fechar categorias de preço para períodos de serviço específicos.

C2 - Relógio: Determinados serviços são mais solicitados em determinadas horas do dia. Assim, a maioria das estratégias de YM baseia-se em sistemas orientados a reservas que aplicam barreiras nas tarifas com o objetivo de separar os clientes de acordo com as suas características de procura. Desta forma, analisando os horários, é possível criar táticas para aumentar o número de reservas ou vendas do serviço. Uma das estratégias a ser aplicada poderia ser a de oferecer promoções, por exemplo, nas horas de menor procura, ou então criar descontos; a mesma estratégia pode ser sempre aplicada nas situações opostas, nos horários de pico. Assim sendo, ao ser comunicada uma oferta limitada a um preço específico, o consumidor é incentivado a comprar o serviço no momento.

C3 - Capacidade: YM é aplicado principalmente por empresas que têm capacidade do produto/serviço oferecido limitada. Os autocarros de longas distâncias, por exemplo, têm inicialmente um número limitado de assentos (dependendo das circunstâncias podem ser criados desdobramentos dos serviços, alocando mais recursos como autocarros e motoristas), sendo por isso importante aproveitar ao máximo a capacidade, minimizando o número de assentos não vendidos.

C4 - Custo: É importante que as empresas tenham a capacidade de variar os preços. São estabelecidas classes de preços de acordo com as previsões de procura, com as características e necessidades dos diferentes segmentos. Apesar de YM envolver ajustes nos preços, estes não são um programa de descontos, os preços são ajustados para ir de encontro à procura (Withiam)**.**

2.4 Serviço *Long-Distance Bus*

Esta dissertação é direcionada às estratégias *Yield Management* (YM) e *Dynamic Pricing* (DP) aplicadas ao serviço de autocarros de longo curso, sendo que a segunda estratégia será abordada e definida na segunda parte deste capítulo.

Em primeiro lugar, é necessário perceber em que consiste o serviço de autocarros de longo curso. Este serviço é chamado também de autocarros expressos, e atende às necessidades de transporte de passageiros envolvendo distâncias significativas, entre cidades ou até países. Para serem consideradas longas, as distâncias têm de ser de mais de 80 quilómetros entre o ponto de partida e o destino. As viagens podem ser linhas diretas entre origem e destino, ou com diversas paragens pelo meio. Estas viagens costumam ser oferecidas em autocarros com grande conforto.

Desde 2017 que em Portugal já se encontram presentes, no mercado rodoviário, empresas estrangeiras de autocarros expressos, como a FlixBus, que disponibiliza 20 linhas internacionais diretas que ligam as 20 maiores cidades portuguesas a 50 cidades europeias.

No início os trajetos eram somente internacionais, tendo as operadoras portuguesas a exclusividade nas ligações por autocarros entre cidades portuguesas. No entanto, no dia 4 de dezembro de 2019, entrou em vigor a nova legislação, aprovada no conselho de Ministros a 22 de agosto de 2019, que deu início à liberalização do transporte rodoviário em Portugal.

Esta liberalização permitiu a entrada de qualquer operador no mercado nacional para viagens entre cidades portuguesas, o que fez com que as maiores operadoras deste tipo de viagens em Portugal, perdessem a sua exclusividade e ganhassem novos concorrentes, nomeadamente a grande operadora FlixBus[1], que já recorre aos modelos de YM e DP.

As maiores empresas portuguesas de transporte de passageiros ainda utilizam métodos mais simples, com os quais é difícil fazer frente às novas empresas estrangeiras para continuarem a liderar o mercado. Daí a necessidade de criação de uma ferramenta/algoritmo que seja capaz de calcular e alterar os preços de forma dinâmica com base em informações internas e externas, aumentando desta forma a venda de lugares pelo melhor preço.

2.5 *Yield Management* Estático no contexto de *Long-Distance Bus*

A designação *Yield Management estático* tem na base o facto de que o preço que é calculado e atribuído ao serviço oferecido é estático, ou seja, permanece o mesmo enquanto alguém responsável na empresa não o alterar; toda a informação, promoções e variáveis utilizadas para o cálculo são sempre introduzidas manualmente.

Os dados recolhidos que servem de tomada de decisão para alterar os preços ou fazer descontos baseiam-se em informação já conhecida, que foi adquirida com a experiência no negócio. Um exemplo são as sextas-feiras ou feriados com os quais é possível fazer a "ponte", que são dias de maior procura – nestes dias faz sentido subir um pouco os preços.

Outro exemplo são os finais e os inícios dos dias em que os preços também costumam ser diferentes, pois são partes dos dias em que os autocarros têm maior ocupação, visto que são os horários mais convenientes em que as pessoas se deslocam para o trabalho ou para casa.

Mais uma situação conhecida, bastante comum atualmente, é o decorrer de eventos em certas cidades em que os próprios representantes do evento contactam as empresas de autocarros para que baixem os preços entre determinada origem-destino recebendo em troca, por exemplo, a indicação sobre o patrocínio nos cartazes do evento e outros meios de divulgação. Como se pode ver, são tudo alterações no momento, sem dinâmica alguma.

De uma forma sucinta, todo o cálculo é feito de forma estática e simples, ou seja, existe um preço fixo para cada origem-destino que é calculado com base nos quilómetros percorridos, sobre o qual são posteriormente aplicados descontos ou promoções pontuais.

Neste trabalho vão ser discutidas e analisadas estratégias estáticas de cálculo de preços; serão também estudadas as variáveis e informação que são utilizadas como dados de entrada. Desta forma será possível perceber como melhorar as estratégias e torná-las dinâmicas, como discutido a seguir.

[1] https://www.flixbus.pt/

2.6 Contexto Histórico de *Dynamic Pricing* (DP)

Desde a desregulamentação do setor aéreo em 1978, deu-se o início às práticas que lidam com decisões complexas de preço e gestão de procura, que se expandiram para diferentes setores tais como hotelaria, cruzeiros, aluguer de carros, comunicação, entre outros (Besbes and Zeevi).

As empresas que operavam em setores altamente competitivos e dinâmicos, viram-se na necessidade de recorrer a estratégias mais inteligentes de gestão e de *marketing*, a fim de não ficar para trás e preservar ou até aumentar o lucro obtido nas vendas do produto (Jallat and Ancarani).

O setor da aviação foi o primeiro a tornar-se altamente competitivo. Daí, tal como referido numa das secções anteriores, surgiu a estratégia *Yield Management* (YM) usada inicialmente por grandes companhias aéreas, que passaram a introduzir diversas técnicas de discriminação de preços com base em diferentes classes tarifárias, sistemas complexos de descontos com acesso limitado, esquemas de fidelidade do cliente e técnicas de *overbooking* (Malighetti, Paleari and Redondi). As operadoras *low-cost*, acabadas de entrar no mercado, optaram por uma estratégia bastante diferente que consistia em alterar os preços com muita frequência, variando-os desde valores muito baixos para atrair os clientes, até valores consideravelmente altos. Neste contexto surgiu a primeira aplicação da estratégia que hoje é denominada por *Dynamic Pricing* (DP).

Analisando a história um pouco antes da desregulamentação, é possível verificar que DP já se tinha tornado interessante do ponto de vista teórico no início da década de 1970. Os primeiros trabalhos de investigação surgiram em 1971, de Marvin Rothstein, e em 1972, de Ken Littlewood. Ambos os trabalhos analisavam as possibilidades de aplicação da precificação dinâmica (*Dynamic Pricing*) nos setores de companhias aéreas e hoteleiras. No entanto, apenas após a desregulamentação do setor aéreo surgiu a necessidade de aprofundar e pôr em prática técnicas de DP (Deksnyte and Lydeka).

Desde a primeira abordagem e definição de DP, durante todos estes anos até a data de hoje, o assunto continuou a ser estudado e a evoluir e a própria definição de DP também acompanhava a evolução. Todos os estudos realizados tinham como objetivo a modelação do DP em diferentes indústrias/serviços, e perceber que fatores tinham importância na sua formação. Além disso, a maioria dos investigadores tentavam redefinir ou adaptar a definição de DP de acordo com o setor que estudavam.

2.6.1 O que motiva as empresas a usar precificação dinâmica

Existem diversos objetivos que motivam as empresas a usar precificação dinâmica, uns mais óbvios do que os outros, e o estudo realizado por Weatherford e Bodily em 1992 [(Santos, Feder Mayer and Bezerra Marques), (Weatherford and Bodily)], propõe os objetivos que consideram ser os mais relevantes:

- **Maximização de lucro/contribuição** – lucro e contribuição da empresa encontram-se relacionados. A contribuição em termos de custos fixos é definida como a receita menos o custo variável. Para obter o lucro, deve-se subtrair os custos fixos da contribuição, e acredita-se que se a contribuição for maximizada, o lucro será maximizado;
- **Maximização da capacidade utilizada** – o objetivo é vender o máximo de unidades de produto disponível, mesmo causando algum prejuízo em preço obtido nalgum item;
- **Maximização da receita média / cliente** – procura-se um equilíbrio entre o número de clientes e a receita, de modo a evitar que apenas um cliente consuma toda a capacidade;

- **Maximização da receita** – o lado dos custos é ignorado pela empresa, porque os custos são insignificantes ou essencialmente fixos e não é uma questão relevante para a decisão;
- **Minimização da perda de credibilidade do cliente** – uma empresa pode decidir que não irá oferecer os descontos de preços praticados no mercado. É um objetivo difícil de ver como único objetivo operacional, no entanto pode ser utilizado como secundário a outros objetivos;
- **Maximização do valor líquido presente** – num horizonte temporal curto, a empresa pode descontar os fluxos de caixa recebidos em diferentes períodos;
- **Extração do preço máximo de cada cliente** – é um ótimo objetivo no entanto difícil de se atingir, visto que envolveria negociação com cada cliente de forma individual.

Estes objetivos são identificados no estudo realizado pelos autores referidos, contudo dependendo do objetivo, do setor e dos investigadores, os objetivos podem mudar e adaptar-se de diversas formas, visto que é um tema bastante flexível.

De uma forma geral, o tema continua a ser estudado, constantemente a mudar e a evoluir. As estratégias de preços dinâmicos são cada vez mais robustas e aplicadas cada vez mais em diversos setores e indústrias.

2.6.2 Utilização de ferramentas de precificação dinâmica

Nas últimas quatro décadas foi possível observar uma implantação universal de *Internet* e um rápido desenvolvimento das tecnologias de informação. Com o surgimento de canais de vendas *online* foi possível criar etiquetas de preços digitais que permitiram variar o preço com frequência, sem esforço e custo adicional, automatizando desta forma a variação de preços.

O preço dos produtos ou serviços oferecidos podia então ser calculado de acordo com as características e o número dos clientes, com o tempo e com os preços dos concorrentes, que seriam constantemente monitorizados e combinados com os preços oferecidos pela própria empresa (Pupavac).

Estas mudanças permitiram que diversas indústrias adotassem ferramentas de precificação dinâmica nos sistemas de gestão de receitas. Em (Bandalouski, Kovalyov and Pesch), os autores elencam os principais fatores que levaram ao uso cada vez mais frequente de ferramentas de precificação dinâmica:

- Processamento digital de dados que permite recolher e usar eficientemente informações valiosas sobre a procura, *stock* disponível, e preços dos concorrentes, com a possibilidade de posteriormente processar estas informações em tempo real;
- Os custos de imprimir etiquetas de preços e de informar os clientes sobre mudanças de preços cada vez diminuem mais e são menos recorrentes;
- Fácil acompanhamento de mudanças de preço.

Um exemplo atual de vendas dinâmicas *online* é o da Amazon, que atualiza os preços dos produtos em média a cada dez minutos, com base em diversos fatores e dados que está constantemente a recolher (Pupavac). Atualmente existem milhares de lojas de vendas *online* que recorrem ao mesmo sistema.

2.7 Definição de *Dynamic Pricing*

Definir *Dynamic Pricing* (DP) com exatidão não é uma tarefa fácil, uma vez que a sua definição e aplicação variam de acordo com diversos fatores, dos quais os mais relevantes são*: (i)* variação da interpretação pelos representantes de diferentes campos científicos; *(ii)* orientação dos investigadores de DP para diferentes ramos académicos; *(iii)* definição do problema de DP (Deksnyte and Lydeka).

Abaixo seguem algumas definições atribuídas a estratégia de DP em diferentes trabalhos:

- No trabalho realizado por Guillermo Gallego e Garrett van Ryzin são sugeridas duas alternativas para a definição de DP: *(i)* "uma tentativa de ajustar os preços para compensar as alterações estatísticas "normais" na procura"; *(ii)* "uma tentativa de "sintetizar" um conjunto de preços ótimos a partir de um pequeno conjunto estático de preços em resposta a uma função de procura variável" (Ryzin and Gallego);
- Em outubro de 2000, Paul Krugman publicou um artigo, na revista *The New York Times*, acerca dos preços justos, no qual descreve o DP da seguinte forma: "A precificação dinâmica é a nova versão de uma prática antiga, a discriminação de preços. É usada a impressão digital de um potencial consumidor – registo das suas compras anteriores, o seu endereço, talvez outros sites que ele tenha visitado – para avaliar a probabilidade de ele hesitar se o preço for alto. Se o cliente parece sensível ao preço, irá conseguir uma promoção/valor mais baixo. Se não for, pagará um valor superior pelo produto." (Jallat and Ancarani);
- No estudo realizado por Rama Yelkur e Maria Manuela Nêveda DaCosta em 2001 foi usada a seguinte definição: "preços diferenciados (dinâmicos) permitem personalização adicional por cliente-alvo e aprimora ainda mais os preços tradicionais segmentados ou diferenciados" (Jallat and Ancarani);
- No trabalho realizado por Dimicco et al. DP foi referido como "venda dos produtos a preços personalizados de acordo com a procura dos compradores, o ambiente do mercado e a oferta do vendedor no momento da transação." (Dimicco, Maes and Greenwald);
- A definição que é considerada como a mais comumente empregue é a de R. Preston McAfee: "A precificação dinâmica, também conhecida como gestão de rendimento ou gestão de receita, é um conjunto de estratégias de precificação destinadas a aumentar os lucros." (McAfee and Vera te Velde);
- V. den Boer no seu trabalho de pesquisa, começa a sua introdução com a descrição do termo da seguinte forma: "Preços dinâmicos é o estudo da determinação de preços ótimos de venda de produtos ou serviços, num ambiente em que os preços podem ser ajustados com facilidade e frequência" (Boer);
- Um dos trabalhos mais recentes, realizado por Luiz Claudio de Freitas Filho, refere o DP como "uma estratégia de precificação em que os preços não são fixos ou constantes. Em vez disso, eles são continuamente reajustados de acordo com a procura percebida e os fatores que influenciam a conveniência do produto" (Filho).

Como é possível verificar em todos estes trabalhos, não existe uma única definição geral para o termo *Dynamic Pricing* (DP). Indre Deksnyte, no seu trabalho de pesquisa, tentou sintetizar e agrupar as definições de diferentes autores em diferentes categorias tendo identificado quatro grupos: *(i)* "definições de DP que são feitas de forma precisa como um meio relacionado com a fixação de um preço ótimo para maximizar a receita"; *(ii)* "definições orientadas para a procura, onde a maior atenção é dada aos parâmetros de procura e sua contribuição para a formação de DP"; *(iii)* "definições voltadas

para a oferta, mais comuns na área de pesquisas operacionais"; *(iv)* "definições orientadas para o equilíbrio entre oferta e procura, que enfatizam que a maximização dos preços pode ser alcançada se DP controlar o desequilíbrio entre oferta e procura" (Deksnyte and Lydeka).

2.8 Estratégias de *Yield Management* dinâmico

Nesta secção será realizada uma análise dos seguintes tópicos: *(i)* ideias a ter em conta antes de aplicar DP; *(ii)* estratégias que podem ser aplicadas para implementar DP; *(iii)* algoritmos para aquisição das informações necessárias para o cálculo de DP; *(iv)* arquiteturas de *software* para o cálculo do DP final.

2.8.1 Ideias a ter em conta antes de implementar DP no contexto de *Long-Distance Bus*

Sabendo que o preço é um fator muito importante na tomada de decisão do cliente, a capacidade da empresa de fornecer esquemas de preços mais inteligentes, competitivos e específicos só trará vantagens para o negócio. Assim, quando é dito que é necessário atribuir o preço certo aos produtos ou serviços, isso não significa que exista apenas um valor certo ou que esse tem de ser estático; o valor certo tem alturas certas para existir, ou seja, tem de ser definido com diferentes regras e ser ajustado ao longo do tempo em resposta à mudança da procura ou outros fatores importantes definidos pela empresa. Assim, usando o *Dynamic Pricing* (DP) é possível recalcular e otimizar os preços em tempo real, sempre que necessário, para maximizar a receita e aumentar a sua margem.

Para implementar a estratégia de DP, antes de tudo é necessário ter noção de algumas ideias importantes na tomada de decisão como, por exemplo, que variáveis ou estratégias utilizar.

As ideias a ter em conta antes de aplicar a estratégia de DP, descritas abaixo, foram escritas com base nas ideias do *blog* Turnit[2]. Turnit[3] é uma empresa de tecnologia de viagens com mais de 20 anos de experiência na indústria de transporte terrestre de passageiros. O produto principal oferecido pela empresa é o *software* Turnit Ride, que é uma solução para a reserva e gestão de *stock*; esta encontra-se direcionada para as operadoras de transporte que exigem gestão de logística complexa, preços dinâmicos e capacidade de resposta flexível para a situação do mercado, maximizando a sua receita e simplificando os seus processos de negócios. Devido aos anos de experiência da empresa e ao seu conhecimento na área de interesse desta dissertação (transportes de passageiros e DP), o seu *blog* foi considerado como fiável, a fim de se poder usar como base para os pontos descritos abaixo.

- ***Real-time inventory management system*** – Para começar, se queremos ter um sistema de preços dinâmicos devemos implementar um *real-time inventory management system* eficiente. Resumidamente, é um conjunto de bens que uma empresa possui e usa com o objetivo final de vender e/ou ganhar dinheiro com isso. O principal objetivo de gerir um *stock* é controlá-lo e supervisioná-lo para garantir a sua disponibilidade.

  O caso de estudo desta dissertação são autocarros com rotas de longa distância, neste caso o *stock* refere-se ao número de lugares disponíveis entre determinada origem e destino; assim,

[2] https://blog.turnit.com/6-steps-to-implementing-dynamic-pricing-in-bus-company

[3] https://turnit.com/

inventory management seria um processo de controlo de disponibilidade dos lugares dos autocarros de modo a maximizar a receita por partida.

O processo é bastante complexo visto que, à partida, estão associados vários pares origem-destino devido a paragens realizadas ao longo da viagem. Com o aumento do número de paragens aumenta o número de segmentos origem-destino, o que torna a gestão mais complexa, tendo em conta que uma gestão de *stock* eficiente tem de prever a disponibilidade de cada lugar em cada segmento da viagem.

Portanto, para se poder aplicar DP é importante ter uma completa noção em tempo real da quantidade de lugares disponíveis e vendidos. Esta abordagem resolve também o problema de sobrelotação ou existência de lugares não vendidos quando os bilhetes são vendidos em diferentes plataformas. Para automatizar todo o processo e planear recursos com mais eficiência deve-se recorrer a *machine learning* e inteligência artificial;

- **Boa tecnologia** – Outro ponto bastante importante é investir em boa tecnologia porque é necessário garantir um bom desempenho do sistema, mesmo em horas de sobrecarga nas plataformas como, por exemplo, durante campanhas de venda;
- **Análise de dados** – Analisar os dados é de crucial importância, pois para além de dados simples e estáticos como o tipo de tarifa, horários de compra, idade do cliente, etc., teremos de recolher dados externos tais como preços dos concorrentes, estudar os hábitos dos clientes, tentar perceber quanto é que estão dispostos a pagar num determinado momento, e através disso definir a faixa de preços;
- **Canal de vendas** – Escolher bem em que canal de vendas aplicar o DP. Em pontos de venda físicos como venda a bordo ou venda na caixa na estação, normalmente são aplicados preços fixos, pois são os canais onde os clientes estão dispostos a pagar mais (normalmente estão à espera de comprar pelo preço disponível). Por outro lado, é nas vendas *online* que os clientes passam algum tempo a comparar os preços entre diferentes empresas ou meios de transporte antes de comprarem o bilhete, o que as torna um alvo perfeito para aplicar o DP, de modo a ajustar os preços de acordo com o mercado, para não ter preços excessivamente altos e os potenciais clientes não passarem para a concorrência;
- **Segmentar os clientes** – Perceber melhor os clientes de modo a poder segmentá-los em diferentes categorias, o que ajudará a criar ofertas que correspondam às suas expectativas e se encaixem melhor nos seus perfis;
- **Discriminação de preços** – Por fim, ter bastante cuidado com a discriminação dos preços. Quando são realizadas campanhas ou a definição das tarifas, estas podem ser baseadas em dados demográficos, no entanto, é necessário ter bastante atenção na escolha dos dados, como, por exemplo, não usar os dados como o tipo de dispositivos através do qual o passageiro está a efetuar a reserva ou o endereço IP, pois os clientes ao aperceberem-se destas táticas, podem mudar para a concorrência.

2.8.2 Modelos matemáticos desenvolvidos para DP

Existem inúmeros estudos e trabalhos realizados acerca de estratégias e modelos possíveis de utilizar no Dynamic Pricing. As estratégias variam de acordo com diversos fatores, tais como os objetivos da empresa, se o seu stock é estático ou não, se existe data limite para venda dos produtos, a procura, se é um único produto ou são vários, se os clientes são sensíveis a variação dos preços ou não, entre outros. Além disso, para o mesmo problema existem várias soluções. O mundo de preços dinâmicos, e da gestão de rendimentos no geral, é muito vasto e requer conhecimentos de diversas áreas científicas, tais como:

gestão, probabilidades, estatística, *marketing*, ciências da computação, sociologia, entre outros. Todo este conhecimento é requerido para poder evoluir as estratégias de gestão de rendimentos, que é no fundo o que traz lucro para a empresa.

No trabalho realizado por Boer são referidos diferentes trabalhos de diversos autores; estes trabalhos encontram-se organizados de acordo com as características do *stock* (Boer).

Para esta dissertação são de interesse modelos baseados em *stock* finito sem possibilidade de reposição e com o tempo finito para a sua venda, visto que a temática da dissertação se encontra direcionada para autocarros de longo curso, e estes têm um número fixo de lugares/bilhetes que têm de ser vendidos até uma determinada data.

Assim sendo, um dos trabalhos referidos, que é um dos primeiros a ser publicado, é o do Lazear (1986) que considera um modelo simples em que a empresa divide o período de venda de um produto em dois períodos. Caso o produto não seja vendido no primeiro período, ou seja, vendido em quantidade inferior à esperada, a empresa altera o preço de venda e tenta vender o produto no segundo período. Lazear mostra que o lucro esperado aumenta com a estratégia de dois períodos em vez de um.

Em 2004, Aviv e Pazgal, realizaram uma pesquisa direcionada à precificação dinâmica em que vendem um produto num período fixo. O seu modelo incorpora três tipos de incerteza: *(i)* a frequência e o tempo de chegada do cliente; *(ii)* o preço que cada cliente pagaria pelo produto; *(iii)* e o sucesso do produto no mercado.

No seu trabalho eles argumentam que as estratégias de preços dinâmicos são críticas em ambientes de mercado com incerteza inicial elevada, no entanto este problema é resolvido com a estratégia de vendas informada, ou seja, é necessário reunir e integrar informações de procura de forma inteligente. Deste modo, para atingir o objetivo de maximizar as receitas, é necessário que a empresa defina os preços de modo a que seja alcançado o equilíbrio entre os ganhos atuais e futuros (Aviv and Pazgal).

Na pesquisa realizada por (Ryzin and Gallego), é proposto o modelo que envolve um processo contínuo e dinâmico de procura, que permite decisões dinâmicas de preços ao longo do período. Com os resultados dos estudos, com a maior certeza, foi possível concluir que o uso de políticas simples de preço fixo, funcionam bem em bastantes casos. Segundo os autores, é um resultado bastante encorajador uma vez que as políticas dinâmicas ideais são muito instáveis e exigem constantes ajustes de preços.

No artigo (Braden and Oren) é investigado o problema de preços dinâmicos não lineares. São apresentadas diferentes técnicas através das quais as empresas conseguem determinar os custos implícitos para resolver a incerteza sobre a procura, por meio da sua política de preços. Além disso é mostrado que a rapidez com que a incerteza é resolvida depende exclusivamente do conjunto de compradores excluídos do mercado ou agrupados de acordo com quantidades de compra positivas. A perceção da ideia apresentada, e um dos principais resultados da análise realizada no artigo, pode ser interpretada como "principio da separação": O "problema de estimativa" de aprender os parâmetros que caracterizam a heterogeneidade do cliente pode ser separado do "problema de controlo" de determinar descontos por volume apropriado, para explorar essa heterogeneidade.

2.8.3 Tipos de estratégias de *Dynamic Pricing*

Existem diferentes tipos de estratégias dinâmicas que podem ser aplicadas no cálculo de preços. O tipo de estratégia aplicada difere de empresa para empresa, a escolha é baseada nas necessidades do negócio. Sendo assim, no processo de escolha de uma estratégia é necessário ter em atenção quais os objetivos da empresa e quais os seus clientes-alvo.

Abaixo segue uma breve descrição sobre os tipos de estratégias mais aplicadas.

- **Preços Segmentados** – a estratégia consiste em dividir os clientes em diferentes grupos. A divisão ocorre com base no que cada grupo/tipo de cliente é capaz ou está disposto a pagar pelo produto ou serviço oferecido pela empresa. Com esta estratégia a empresa passa a oferecer preços diferentes para grupos de clientes diferentes. Tendo como exemplo os bilhetes de aviões, existem bilhetes para o mesmo voo que diferem bastante entre si em termos de preço, dependendo do tipo de classe que for escolhida – classes executivas com maior conforto são mais caras;
- **Preços de pico** – esta estratégia consiste em subir os preços no período de maior procura (horários de pico). Por exemplo, no caso de autocarros de longo curso, nos feriados ou horários de maior movimento, a tarifa pode subir e logo a seguir voltar à normalidade;
- **Preço baseado no tempo** – estratégia que consiste em cobrar mais dinheiro por serviços mais rápidos. Diferentes indústrias utilizam esta estratégia para cobrar preços mais altos pelos serviços mais rápidos que são oferecidos no mesmo dia. Por exemplo, no caso de autocarros de longo curso, podem existir duas rotas que têm a mesma origem e destino e que partem a uma hora bastante semelhante, no entanto, uma das rotas pode não ter paragens pelo caminho e ser muito mais rápida do que a outra. Neste caso se um cliente precisar de chegar mais cedo a algum destino irá optar pelo serviço mais rápido mesmo que este tenha um preço um pouco mais elevado;
- **Preço de penetração** – é uma estratégia que consiste em atribuir um preço ao produto ou serviço, inferior ao nível do mercado, com o objetivo de aumentar a procura. Com esta estratégia, ao diminuir significativamente os preços, é possível atrair clientes novos e atingir uma maior parte do mercado. Muitas vezes a estratégia é usada nos novos produtos/serviços lançados, com o objetivo de se dar a conhecer e atrair clientes; após isso, de uma forma regular, voltar a subir os preços até ao nível do mercado. É possível observar a aplicação desta estratégia na empresa de autocarros de longo curso FlixBus. Esta empresa alemã arrancou com ligações nacionais em Portugal a 30 de julho do ano corrente (2020). Anteriormente a esta data, as ligações nacionais eram apenas realizadas por empresas portuguesas. Por esse motivo, a FlixBus, para chamar a atenção dos potencias clientes às novas linhas de serviço expresso lançadas, atribuiu o preço de um euro a determinadas viagens;
- **Condições de mudança** – estratégia usada quando as condições do mercado são incertas e encontram-se em constantes alterações. O objetivo é reduzir os preços à medida que as vendas começam a cair. Quando a procura aumenta, os preços voltam a subir.

2.8.4 Tipos de algoritmos para obter os dados para o cálculo dos preços

Para criar preços personalizados e dinâmicos, é necessário introduzir novos fatores/variáveis para o cálculo de preço, fatores estes que, por exemplo, podem identificar características do ambiente de compra ou o perfil do cliente e os seus comportamentos que afetam a sua disposição de pagar.

Atualmente, com o grande avanço da tecnologia e conhecimento, não é necessário reinventar algoritmos ou ferramentas, encontrando-se a maioria disponível na *Internet* com livre acesso. *Internet*, *Big Data* e digitalização permitem às empresas incorporar tecnicamente informações e novos fatores nas suas definições de preços, sendo estas informações obtidas com a ajuda de algoritmos. Entre eles encontram-se os algoritmos que podem obter a seguinte informação (Krämer and Kalka):

- **Preços com base no tempo** – os preços aumentam sistematicamente quando o aumento da procura é previsível e diminuem quando a redução da procura é prevista. Eventos, clima, férias escolares, podem ser fatores de influência;
- **Preços com base na concorrência** – as mudanças nos preços dos concorrentes podem ou não influenciar a própria política de preços;
- **Preços com base na distância** – distância a que o cliente se encontra do próximo ponto de venda;
- **Preços com base no histórico** – o histórico de pesquisa do cliente fornece informação da sua disposição de pagar;
- **Preços com base no comportamento passado** – as transações do cliente e a sua lealdade no passado, ou seja, informação sobre o produto comprado anteriormente e o preço pago, determinam o preço atual;
- **Preço com base em dispositivos** – o tipo do dispositivo através do qual está a ser realizado o acesso, por exemplo, tipo de *smartphone*, PC, portátil, *tablet*, etc.;
- **Preço com base demográfica** – a idade e o sexo do cliente, permitem construir uma estimativa da sua disposição a pagar. De uma forma geral, se for um adolescente, por exemplo, este tem o poder de compra inferior à de um adulto já formado e trabalhador.

2.8.5 Tipos de arquiteturas de *software* para DP

Em termos de arquiteturas de *software*, para o cálculo de *Dynamic Pricing* (DP) existem duas soluções disponíveis: *(i)* sistema baseado em regras (*rule-based*); *(ii) software* com tecnologia de aprendizagem automática *(machine learning).*

- **Sistema baseado em regras** – é uma abordagem básica de precificação dinâmica. Os sistemas baseados em regras implementam regras escritas com base no conhecimento do domínio, para atender às necessidades de negócios de uma empresa. As regras normalmente são apresentadas na forma de declarações "se-então". Quando o *software* deteta um padrão nos dados é então definida uma relação entre as regras e os factos conhecidos. Em seguida, uma regra apropriada é executada. À medida que surgem alterações na empresa como, por exemplo, alterações no *stock* de assentos disponíveis, ou novos itens adicionados ao negócio, é necessário adicionar novas regras para gerir as vendas ou então modificar as que já existem. Além disso é necessário que, após as alterações inseridas tudo funcione conforme previsto e esteja alinhado com os objetivos de negócio. Assim sendo, com o aumento do número de regras e o número de ações, torna-se cada vez mais difícil a gestão do sistema. No entanto, são sistemas eficazes e menos arriscados, pois a atualização das regras definidas é feita de modo controlado; deste modo, os sistemas criados dependem exclusivamente de conhecimento "embutido" para reagir às alterações no ambiente em que trabalham;

- **Aprendizagem automática** – tem por base uma camada de regras definidas, tentando, no entanto, simular a inteligência humana, a fim de se tornar capaz de interpretar, categorizar e desenvolver a capacidade de aprender regras novas e descartar as que não forem mais necessárias. Quantos mais dados o sistema recebe, mais ele aprende e melhora o seu desempenho. Além disso, ao combinar aprendizagem automática com algoritmos de previsão avançados, é possível modelar a curva de procura de preços para cada produto, o que ajuda a medir a recetividade dos clientes.

Ambas as técnicas continuam a ser utilizadas nos dias de hoje e têm os seus prós e contras. Em relação aos sistemas baseados em regras, estes são fáceis de definir e implementar, conhecendo o domínio onde se quer integrá-las. Além disso, existe o controlo total sobre cada decisão tomada e regra escrita. No entanto, o acréscimo de novos produtos ou alteração de lógica de venda, implicam alterações nas regras existentes ou acréscimo de novas.

No que diz respeito a aprendizagem automática, para criar um bom sistema é necessário gastar bastante tempo no início com a implementação, no entanto todo o esforço acaba por ser compensado a longo prazo, visto que se obtém um sistema autónomo e sempre atualizado com as decisões mais recentes e cada vez mais acertadas.

2.9 YM Dinâmico no contexto de *Long-Distance Bus*

Yield Management Dinâmico, no contexto desta dissertação, refere-se a gestão dinâmica da precificação. Tal como já foi referido numa das secções anteriores, atualmente a estratégia de precificação dinâmica encontra-se aplicada em diversos setores. O setor de transporte e viagem, mais especificamente transporte aéreo de passageiros, parece ser o setor que da melhor forma empregou a estratégia (Filho). No que toca ao setor de autocarros de longo curso, a abordagem é bastante recente, poucas empresas utilizam as novas e modernas tecnologias na gestão de rendimentos, e existem poucos trabalhos de pesquisa e estudos relativamente ao *Yield Management* e *Dynamic Pricing* neste setor. No entanto, sendo a gestão de rendimentos e precificação das áreas de transporte aéreo e de transporte ferroviário semelhantes aos autocarros de longo curso, é possível consultar os trabalhos realizados nestas áreas para extrair algumas ideias, tais como: regras a aplicar, variáveis e algoritmos a utilizar.

Um aspeto em comum estudado em todas as áreas é o comportamento da escolha do cliente, que é importante para estimar a procura dos clientes, e com base nesta informação melhorar os níveis de serviço oferecido. Na maior parte da literatura em que é realizado o estudo do impacto do comportamento de escolha, os fatores que influenciam o comportamento são divididos em duas categorias: atributos pessoais e atributos de viagem (Qin, Zeng and Yang).

Outra questão estudada neste contexto é a alocação de assentos. Um método de ajuste dinâmico da alocação de assentos, no setor de transporte ferroviário, foi proposto por Jiang et al. No estudo, o autor tenta resolver problemas de alocação determinística com base no bilhete fixo, no entanto esta abordagem não resolve o problema da procura dos passageiros (Qin, Zeng and Yang).

Ao consultar os estudos realizados para diferentes setores, é necessário ter em atenção algumas diferenças, como por exemplo, as viagens de companhias aéreas são realizadas ponto-a-ponto, e os preços dos bilhetes entre os diferentes pares origem-destino, na sua maioria, são independentes entre si, ou seja, os preços podem ser determinados separadamente. No que toca aos autocarros de longo curso, existem percursos diretos, no entanto, a maioria das viagens é realizada com várias paragens, o que pode gerar procura entre muitos pares origem-destino diferentes, ou seja, se o objetivo é viajar do ponto A ao ponto C e ao longo da viagem existir uma paragem B, o percurso passa a poder ser dividido em três pares origem-destino: A-B, A-C e B-C, que por conseguinte podem ter procura diferente. Isso leva à correlação dos preços dos bilhetes entre os diferentes pares origem-destino. Esta diferença torna o problema de gestão de preços muito mais complexo do que o dos aviões, visto que é necessário determinar o preço para cada par origem-destino, e estes podem ser demasiados.

Outro ponto a ter em conta é que o uso de preços dinâmicos no transporte de passageiros pode criar situações em que, embora seja usado o mesmo serviço itinerário, os clientes não pagam exatamente o mesmo valor, o que tem como consequência para os clientes os preços tornarem-se imprevisíveis (Filho).

Capítulo 3

Análise do Sistema YM de um módulo exemplo

O sistema que será analisado é um sistema bem-adaptado para o período anterior à liberalização. Antes de 2017 havia muito pouca concorrência, o mercado era dominado na totalidade por operadoras portuguesas, não havendo necessidade de sistemas complexos de *Yield Management* (YM). Configurar um desconto simples e pontual era mais que suficiente.

No entanto, após a liberalização, com a entrada de operadoras estrangeiras com inovadores métodos de gestão de rendimentos, os utentes de autocarros começaram a ser atraídos para os seus preços e serviços interessantes, tornando o mercado mais competitivo. Por conseguinte, para poder continuar a manter-se competitivas no mercado, e fazer frente às novas operadoras, as empresas portuguesas viram-se obrigadas a fazer evoluir os seus métodos e estratégias de gestão de rendimentos (YM).

De seguida descreve-se o sistema de um módulo exemplo simples e motiva-se, a partir das suas limitações e problemas, a necessidade de criação de um novo sistema de gestão.

3.1 Fatores que contribuem para o cálculo dos preços

Abaixo referem-se apenas as variáveis mais relevantes e interessantes do ponto de vista de gestão de *Yield Management*. Todas as variáveis mencionadas são os *inputs* que são utilizados, no módulo exemplo, para criar descontos e promoções, com o objetivo de tornar *Yield Management* o mais diversificado possível.

- **Perfil do passageiro** – esta é talvez a variável mais óbvia. Como é sabido, o preço pago por um jovem não é o mesmo pago por um utente sénior ou um utente normal. Já há muito tempo que esta estratégia é aplicada, especialmente nas indústrias de transporte de passageiros, pois o poder de compra de jovens ou de idosos é mais reduzido que o de um passageiro trabalhador normal, além de que, normalmente, têm horários de pico mais específicos;
- **Origem e destino** – por vezes é útil criar um desconto entre uma determinada origem e destino, como, por exemplo, quando ocorre um evento numa determinada cidade, quando se pretende atrair pessoas a um determinado destino de forma a aumentar o turismo numa área, ou então, simplesmente para aumentar vendas para um determinado trecho;
- **Datas –** estas variáveis ajudam a definir quando é que se pretende que seja aplicada uma determinada regra de promoção. Podem ser **datas de partida**, **datas de regresso**, **datas de venda**. Com a data de partida podemos definir o intervalo de dias de partida válidos para a promoção, o mesmo se aplicando para datas de regresso. Quanto a datas de venda, é possível definir um intervalo de tempo em que a regra criada se encontra válida;
- **Registo** – também denominado de *loyalty card* (gratuito), existe em bastantes negócios, desde o serviço de óticas até lojas de roupa, e permite criar regras com ofertas exclusivas para os clientes com registo ou cartão de fidelidade. O serviço de autocarros de longo curso não é exceção. Normalmente criam-se cartões de adesão, que conferem ofertas e preços exclusivos de forma a reter clientes e a torná-los em clientes habituais. Desta forma mantém-se sempre o

contacto com o cliente através de *emails* ou notificações, que contêm geralmente informação sobre ofertas que o cliente pode achar tentadoras e talvez até aproveitar;

- **Pontos de venda** – os bilhetes podem encontrar-se disponíveis em diferentes pontos de venda**,** tais como plataformas *online*, *sites* e aplicações, podendo estes ser da própria empresa ou *sites* independentes que oferecem o serviço de venda de bilhetes de diferentes operadoras, para que o cliente possa comparar os preços e escolher a melhor oferta; existe também a venda em quiosque, que é a venda nas caixas; é interessante ter a possibilidade de criar preços mais competitivos adaptados a cada ponto de venda, ou poder estabelecer acordos com as próprias plataformas;
- **Frequência** – um *input* semelhante ao *input* de datas, também é uma variável que ajuda a definir o desconto no tempo, no entanto de forma diferente, mais precisa. É possível definir com que frequência por semana e a que horas será válida a regra. Existem dias de pico de vendas como sexta-feira, ou então horários, como o final do dia, que é a hora de regresso para casa após trabalho, por isso é importante ter a oportunidade de tornar os preços mais flexíveis nesses horários;
- **Serviço / Expresso** – é possível também definir regras para cada serviço individualmente. Podem ser aplicados descontos ou restrições de viagem para casos pontuais;
- **Lugares**– este *input* é uma introdução bastante recente. É possível definir preços mais baixos para alguns lugares. Essa é uma ótima estratégia visto que os lugares que costumam ser os menos desejados podem ter um preço ligeiramente mais baixo o que atrai os clientes aumentando assim a lotação do autocarro que é o objetivo de *Yield Management* – aumentar o rendimento, vendendo o máximo de lugares pelo melhor preço.

Ao cruzar todos estes *inputs*, é possível criar inúmeras regras e descontos. Para que o resultado traga lucro, é necessário ter uma ideia fixa do que se quer, ter uma visão global do negócio e ter a informação sempre atualizada para poder tomar as melhores decisões nos melhores momentos.

3.2 Organização da informação

No sistema do módulo exemplo, o processo de *Yield Management* encontra-se numa forma bastante básica. Todo o sistema gerido localiza-se na base de dados (BD), na qual existem diversas tabelas que suportam as diversas configurações e variáveis de *input* para cálculo de preços diferenciados.

O modelo de dados consiste em tabelas principais e tabelas auxiliares. As tabelas principais são usadas para configurar as regras de descontos/promoções, sendo a configuração realizada através do preenchimento de campos das tabelas. Estas tabelas são compostas por campos como, por exemplo, o identificador, e configurações base como, por exemplo, se o bilhete comprado com esta regra permite anulações, reembolsos, etc., com quantos dias de antecedência é necessário comprar a viagem para que o desconto seja aplicado, entre outras.

As outras tabelas que se encontram no modelo de dados, ou são tabelas de ligações, que têm o papel de interligar tabelas diferentes através dos seus identificadores / chaves, ou são tabelas que representam variáveis. As tabelas que representam variáveis, ao serem ligadas através da tabela de ligação à tabela principal, permitem formar regras mais complexas e diversificadas, pois as regras ficam com variáveis extra a que foram ligadas, para além das configurações base. Deste modo, ao serem preenchidas as tabelas necessárias, de acordo com as necessidades, formam-se regras de descontos/promoções. Ao aplicar os descontos/promoções sobre os preços, introduzimos variações nos preços fixos.

Como exemplo de configuração de uma regra no modelo de dados, suponhamos que queremos agora configurar um desconto para uma determinada origem. Para fazer isto seria necessário recorrer a várias tabelas do modelo de dados. A primeira tabela a preencher seria a tabela principal; nesta tabela seria necessário criar uma nova entrada, que corresponderia a uma nova regra, e após isso preencher os campos necessários, alguns deles referidos mais em cima. De seguida, seria necessário consultar uma tabela onde se encontram localidades (origens e destinos) possíveis para consultar os códigos de localidades para os quais pretendemos configurar a regra. Depois de termos os códigos, seria necessário recorrer a uma outra tabela para criar o item correspondente à origem (item que será ligado à tabela principal), onde teríamos que criar um campo com o código da origem desejada e configurar mais alguns campos, como a descrição, e validar se é só para a origem ou também para o destino. Para terminar e ligar essa restrição (item criado) à regra principal, é necessário recorrer à quarta tabela em que se criaria uma linha com o código do item e o código da regra principal, para ligar as duas.

Desta forma ficaríamos com um desconto configurado para apenas uma determinada origem. O processo seria igual caso fosse necessário configurar a regra para um determinado perfil de cliente ou determinado tipo de local de venda (loja física, *site*, quiosque, etc).

3.3 Limitações do sistema

Na manipulação e organização dos dados descritas em cima, todas as regras já criadas, bem como descontos para seniores ou descontos para jovens, foram feitas com base em observações do mercado e das vendas ao longo dos anos. Foram decisões pontuais que, uma vez criadas, continuam a ser utilizadas com ligeiras modificações em termos das suas configurações. Por outro lado, as regras que vão sendo adicionadas são por norma temporárias e surgem das necessidades do momento.

O processo de manutenção/alteração de descontos já criados ou inserção de novos, é um processo bastante demorado, visto que tem de ser feito manualmente. Têm de ser preenchidos diversos campos de diversas tabelas para que as ligações estejam todas corretas e a regra possa ser ativada. É necessário conhecer as regras que já existem na base de dados, pois pode haver regras semelhantes e acontecer sobreposição, ou falta de coerência, por isso é necessário ter as regras bem ordenadas e à vontade na manipulação dos dados.

Outra limitação é que quem manipula os dados e insere as alterações tem de ter conhecimento da ferramenta Microsoft SQL Server, um sistema de gestão de base de dados relacional, que modela os dados de uma forma intuitiva e direta, para que estes sejam percebidos pelo utilizador como tabelas. Além disso, para poder usar a ferramenta, é necessário ter o conhecimento de Linguagem de Consulta Estruturada, em inglês *Structured Query Language* (SQL), que é a linguagem de pesquisa declarativa padrão para as bases de dados relacionais.

Destas limitações surgiu a necessidade de criar a aplicação *BackOffice*, uma ferramenta que permita gerir todos os rendimentos de uma forma fácil e intuitiva e, mais importante, rápida. Com esta ferramenta, os próprios utilizadores podem introduzir todas as alterações necessárias a qualquer momento, necessitando apenas de ter conhecimento do seu negócio.

Capítulo 4

Estudo preliminar sobre *frameworks* a utilizar no desenvolvimento da aplicação *BackOffice*

Neste capítulo é realizado o levantamento de requisitos, para a nova aplicação *BackOffice*, e uma pesquisa com o objetivo de encontrar as ferramentas mais adequadas para o seu desenvolvimento.

Na abordagem anterior, os dados encontram-se armazenados em tabelas, tendo a sua gestão sido sempre feita neste formato. Considera-se, por isso, que a melhor abordagem será a de criar um *interface* semelhante, ou seja, no formato de tabela de dados, pois a melhor forma de configurar uma regra é ter todos os campos necessários ao preenchimento visíveis.

Para a criação de um *interface*, no formato de tabela de dados, optou-se primeiro por realizar um estudo para perceber que *frameworks* de componentes gráficos existem para o desenho de *interfaces* e se realmente vale a pena a sua utilização.

Existem diversas *frameworks* criadas para diferentes necessidades, mas todas têm uma característica em comum: capacidade de reutilizar componentes. Uma *framework* tem como principal objetivo permitir ao desenvolvedor focar os seus esforços na resolução do problema e não na reescrita do *software*, permitindo um desenvolvimento rápido e seguro.

À primeira vista, a utilização de uma *framework* pouparia bastante tempo na construção de um *interface* gráfico se comparado com a implementação completa de todas as funcionalidades e componentes, visto que o *interface* que é necessário desenvolver será utilizado para configurar regras de descontos/promoções e terá de ser bastante rico e suportar diferentes componentes gráficos.

4.1 Requisitos funcionais para o *interface*

Como já foi referido, a melhor abordagem será organizar os dados num formato tabular, em que cada elemento corresponderá a uma regra de desconto/promoção e os atributos da linha permitirão a configuração da regra. Tendo isso em conta, a seguir listam-se as funcionalidades que o *interface* necessita de ter para permitir a configuração das regras.

- Operações CRUD (*Create*, *Read*, *Update* e *Delete*), para alterar e manipular as linhas (regras) da tabela;
- Seleção de múltiplos objetos;
- Edição *inCell/inLine* intuitiva e rápida;
- *Checkbox* para valores *boolean;*
- Ordenação de objetos por coluna – por exemplo, ordenar todas as regras da tabela de acordo com a informação de uma determinada coluna, que pode ser de vários tipos, tais como: data, número, palavra;

- Duplicar a linha/elemento – como a tabela consiste de linhas/elementos que são regras, a possibilidade de duplicar a linha/elemento e de seguida alterar algum atributo que seja necessário, facilita a criação de novas regras;
- Paginação – como a lista de regras existentes é demasiado extensa, a paginação facilita a navegação;
- Configurar número de regras por página – ajusta e facilita a visualização da tabela;
- Filtragem, para facilitar a procura;
- Pesquisa por palavra-chave para encontrar mais rapidamente o que se procura;
- Dados em cascata, pois são manipuladas diferentes tabelas e existem dados encadeados que precisam de ser mostrados;
- *Drag and drop* – poder arrastar os componentes ajuda na ordenação e organização (neste caso será o arrastar das linhas/elementos para organizar as regras de acordo com a sua prioridade);
- Aspeto intuitivo, "user-friendly", moderno e agradável.

4.2 Primeira abordagem – usar uma *front-end framework* paga

Com base nos requisitos identificados, foi realizado um estudo para encontrar a ferramenta mais adequada. Abaixo apresenta-se a comparação das seis *front-end frameworks* mais adequadas e complexas, em termos de componentes mais relevantes.

4.2.1 Análise de *frameworks*

Framework	Preço ($)	Múltipla – Seleção	Date	Edição – InCell	Filtrar	Ordenar	Duplicar	Paginação	Dados em Cascata	Pesquisa
KendoReact	899	✔	✔	✔	✔	✔	—	✔	✔	✔
DataTables	109	✔	✔	✔	✔	✔	✔	✔	—	✔
DevExtreme	499	✔	✔	✔	✔	✔	—	✔	✔	✔
ag-Grid	750	✔	✔	✔	✔	✔	—	✔	✔	✔
Webi	849	✔	✔	✔	✔	✔	—	✔	—	✔
DHTMLX	600	✔	✔	✔	✔	✔	—	✔	+/-	✔

Tabela 1 - Análise de frameworks

Apesar de, à primeira vista, todas terem a maioria dos componentes necessários para a apresentação dos dados, as *frameworks* diferem bastante entre si, desde a forma de representação dos componentes até ao seu funcionamento; por exemplo, na representação de dados em cascata, DevExtreme tem uma boa apresentação e é rica em opções, enquanto que DataTables é bastante pobre e simples.

Abaixo segue uma análise um pouco mais detalhada das ferramentas em termos do seu aspeto, riqueza dos componentes e organização da documentação. Este último aspeto é bastante importante pois, mesmo que a ferramenta seja muito boa, uma má organização da documentação e poucos exemplos torna-a de difícil uso e aplicação.

Aspeto

Em termos de aspeto, de entre as frameworks apresentadas, destaca-se KendoReact[4] e DevExtreme[5]. Ambas têm uma aparência moderna, simples e agradável. Em termos do design dos componentes, são um pouco semelhantes, pois ambas recorrem às mesmas *front-end open-source frameworks*, Material-UI e Bootstrap. DHTMLX[6] e DataTables[7] não ficam muito atrás, também têm componentes esteticamente bonitos, mas um pouco mais simples.

DataTables tem algumas semelhanças com as duas primeiras *frameworks*, visto que um dos temas/estilos que disponibiliza é também o Bootstrap. Em relação ao Webix[8] e ag-Grid[9] têm componentes visualmente bastante simples, pobres e um pouco antiquados.

Riqueza dos componentes

Neste aspeto podem-se destacar quatro *frameworks (*KendoReact, DevExtreme, DHTMLX e DataTables*)*, que têm uma variedade bastante extensa de componentes e ricos em diversidades. Têm diferentes tipos de filtragem de colunas, arrumação da representação dos dados em cascata e no geral uma grande variedade e possibilidade de personalizar as tabelas de diversas formas. Em comparação, ag-Grid e Webix apesar de terem as mesmas funcionalidades base, são bastante mais simples e primitivas.

Compatibilidade com bibliotecas JavaScript

Todas as frameworks para além de DataTables são compatíveis com jQuery, Angular, React e Vue. DataTables tem apenas uma biblioteca com a qual tem compatibilidade e porventura, dependência para funcionar - jQuery.

4 https://www.telerik.com/kendo-react-ui/

5 https://js.devexpress.com/

6 https://dhtmlx.com/

7 https://editor.datatables.net/

8 https://webix.com/

9 https://www.ag-grid.com/

Documentação

Apesar de praticamente todas terem compatibilidade com diferentes bibliotecas JavaScript, apenas KendoReact e DevExtreme disponibilizam documentação relativamente completa, com muitos exemplos visuais e de implementação para cada biblioteca e para todos os componentes, o que facilita bastante a sua aplicação e implementação. DHTMLX, ag-Grid e Webix também disponibilizam alguns exemplos de códigos, mas para muito poucos componentes, o que no processo de implementação vai suscitar bastantes dúvidas.

Organização da documentação

Organização é um aspeto bastante importante porque facilita a procura, poupa tempo e torna o processo de implementação menos exaustivo. Aqui destacam-se KendoReact, DevExtreme, DataTables e ag-Grid, que têm uma organização bastante simples, percetível e de fácil consulta.

4.2.2 Resultado da análise

Após a análise de todas as características, é possível verificar que os dois melhores candidatos para a construção do *interface* são KendoReact e DevExtreme – são bastante semelhantes entre si em todos os aspetos, divergindo somente no preço.

A *framework* escolhida foi então a DevExtreme com integração ReactJS, uma biblioteca JavaScript *open-source* usada para a construção de *interfaces* de utilizador.

4.3 Análise de bibliotecas, ferramentas e soluções utilizadas na implementação do *interface*

Nesta secção serão abordadas as ferramentas, soluções e bibliotecas que foram escolhidas para a implementação da aplicação *BackOffice*, e apresentada uma breve análise explicando o porquê de cada decisão.

4.3.1 Biblioteca escolhida para implementação do *interface*

Na secção anterior referiu-se que a *framework* escolhida para a implementação do *BackOffice* foi a DevExtreme com integração ReactJS[10]. Abaixo segue uma breve descrição e análise das características da biblioteca JavaScript escolhida.

Em 2011, com o aumento do número de anúncios e componentes no Facebook, o código tornou-se difícil de gerir e manter, aumentando também o número de atualizações em cascata que os engenheiros do Facebook já não conseguiam acompanhar, nem mesmo com o reforço de mais membros na equipa. Surgiu então a necessidade de tornar o código mais eficiente e melhorar a experiência dos utilizadores. Assim, Jordan Walke, na altura engenheiro de *software* na empresa Facebook, criou um protótipo que tornou o processo mais eficiente, protótipo este que deu origem ao que hoje se chama ReactJS.

[10] https://reactjs.org/

O que nos levou a escolher esta biblioteca JavaScript foi o facto de ser simples, fácil de aprender, rápida, com componentes reutilizáveis e flexível. É também uma abordagem moderna e usada por muitos desenvolvedores *front-end.* Abaixo segue uma descrição mais detalhada das características do ReactJS já mencionadas.

- **Simples e fácil de aprender** – é simples porque é baseada em componentes, ciclo de vida bem definido e uso de JavaScript simplificado. É fácil de aprender por qualquer pessoa com algum conhecimento básico de programação (se tiver conhecimento de JavaScript ainda melhor);
- **Componentes reutilizáveis** – React fornece uma estrutura baseada em componentes. Começa-se sempre com um componente simples como um botão, e vai-se construindo um invólucro à sua volta criando componentes cada vez mais complexos. É uma espécie de um puzzle do qual se pode reutilizar futuramente qualquer parte que for necessária. É possível decidir como cada componente será "renderizado", pois cada um tem a sua própria lógica interna. Desta forma é possível reutilizar componentes em qualquer lugar, ou seja, podem ser reutilizados em classes distintas e projetos. Reutilização do código facilita a sua manutenção e torna o desenvolvimento da aplicação mais fácil.
- **Abordagem nativa** – React não é uma biblioteca "escreve uma vez, corre em todo o lado", mas, como os próprios criadores dizem: "aprende uma vez, escreve em todo o lado". Como suporta uma reutilização extensa de código, é possível criar ao mesmo tempo aplicações IOS, Android e *Web* usando, não o mesmo código, mas a mesma metodologia, arquitetura e componentes.

4.3.2 Ferramentas para desenvolvimento de aplicação *Web*

As aplicações do tipo da *BackOffice* são tipicamente constituídas por duas partes, *front-end* (FE) que é a parte *client-side,* e *back-end* (BE), parte *server-side*. Para auxiliar na escrita do código da primeira parte foi utilizada a ferramenta Microsoft Visual Studio Code (VSC), versão de agosto de 2019, e para a segunda o Microsoft Visual Studio 2019 (VS).

- Ferramenta e linguagem usadas no *front-end*

  VSC é essencialmente um editor de texto/código avançado que fornece destaque de sintaxe, tem complementação simples de código, e tem fácil função de compilação e *debug* de aplicações *Web* e nuvem. Para além disso possui uma boa capacidade para a escrita de HTML, CSS e JavaScript, e um simples suporte de TypeScript e *C#.* VSC também suporta diversos *plugins*, o que amplia bastante as suas habilidades.

  A linguagem de programação utilizada no FE foi JavaScript visto que a *framework* escolhida para a construção foi DevExtreme com integração ReactJS.

- Ferramenta e linguagem usadas no *back-end*

  No BE é usado o VS, que é considerado por muitos o melhor ambiente de desenvolvimento integrado (IDE), inclui um conjunto de ferramentas de desenvolvimento de *software* baseadas em componentes e outras tecnologias para a criação de aplicações poderosas e de alto desempenho. Normalmente escrito em C++, C# e WPF. A linguagem de programação utilizada neste projeto foi C#.

4.3.3 *Web Framework* para desenvolvimento de aplicações *Web*

No *server-side* foi necessário escolher uma *Web framework* que será utilizada no ambiente de desenvolvimento integrado (IDE) do VS. *Web framework* é uma estrutura de *software* projetada para suportar a criação, desenvolvimento e publicação de aplicações *Web* e serviços *Web*. Oferece ferramentas e bibliotecas que simplificam tarefas comuns de desenvolvimento da *Web*, incluindo: roteamento de URLs, que permite definir as URLs que são semanticamente significativas para os utilizadores e que podem ajudar com otimização do mecanismo de pesquisa; interação simplificada com base de dados, suporte a sessões e autorização de utilizadores, formatação de saída (por exemplo, HTML, JSON, XML) e melhoria da segurança contra os ataques na *Web*.

Neste projeto será utilizada a *Web framework* ASP.NET[11], que permite criar de uma forma rápida *sites* baseados em HTML, CSS e JavaScript, escalá-los para serem utilizados com muitos utilizadores e adicionar com facilidade recursos mais complexos, como APIs da *Web*, formulários sobre dados ou comunicações em tempo real. Para a programação do BE será utilizada a linguagem C#, visto que é a linguagem projetada para a plataforma .NET.

4.3.4 Servidor *Web*

Servidor *Web* pode referir-se a um *software* ou a um *hardware* que executa um *software*, para atender solicitações dos clientes tais como armazenar e disponibilizar conteúdos como sites e páginas HTML na *World Wide Web*. Os dados transferidos de um lado para o outro durante os pedidos devem estar em conformidade com um protocolo específico, chamado Protocolo de Transferência de Hipertexto (HTTP), para garantir que todas as páginas e servidores comuniquem de forma eficiente e sem erros.

O servidor *Web* usado neste projeto é o IIS (Internet Information Services) criado pela Microsoft. Este é o servidor mais adequado para se usar neste projeto visto que permite às aplicações *Web* aproveitarem na totalidade os poderosos recursos e extensões de ASP.NET, utilizadas pelo *back-end*.

O modo como funcionará o esquema é o seguinte:

(i) o utilizador executa uma ação que desencadeia uma solicitação;
(ii) esta chega da *Web* ao servidor IIS;
(iii) este, por sua vez, envia a solicitação à aplicação ASP.NET;
(iv) esta última processa a solicitação e envia a sua resposta de volta ao servidor IIS e ao cliente que originou a solicitação.

4.3.5 Base de dados

Neste projeto foi usada uma base de dados relacional, e os dados fornecidos encontram-se em formato de tabelas simples relacionadas entre si. A BD utilizada foi SQL Server Management Studio[12] (SSMS), que é um ambiente integrado para gerir qualquer infraestrutura SQL.

O SSMS é usado para aceder, configurar, gerir, administrar e desenvolver todos os componentes do Servidor SQL, Base de Dados SQL do Azure e Dados SQL de Warehouse. O SSMS fornece uma única

[11] https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet

[12] https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15

utilidade abrangente que combina um amplo grupo de ferramentas gráficas com vários editores de *scripts* avançados para fornecer acesso ao Servidor SQL para desenvolvedores e administradores de base de dados de todos os níveis.

Para a implementação de *proof of concept* com a *framework* selecionada foi necessário preparar o ambiente de testes. O primeiro passo foi criar uma base de dados local para ter uma área de trabalho semelhante à usada na empresa, de modo a que no futuro a transição do ambiente de testes para o real seja o mais simples possível. Para a criação das tabelas foram fornecidos *scripts* que fazem a inserção de dados em todas as tabelas.

4.3.6 Transformação dos dados tabulares em ficheiros JSON

Os dados que vão ser usados para a construção do *interface* encontram-se na BD no formato de tabelas, no entanto a *framework* escolhida para construção trata os dados no formato JSON (JavaScript Object Notation). Sendo assim surge a necessidade de fazer a transformação de tabelas SQL para ficheiros JSON.

Para transformar as tabelas recorreu-se à ferramenta PowerShell, desenvolvida pela Microsoft, para ajudar a automatizar e resolver rapidamente muitas tarefas de administração tediosas, juntamente com alguns comandos apropriados.

No PowerShell[13] uma maneira fácil de criar um ficheiro JSON a partir dos dados do SQL Server é usar o comando "Invoke-SqlCmd" que, seguido do comando SQL "SELECT * FROM SampleText", devolve os registos da tabela que posteriormente são passados para "ConvertTo-Json cmdlet", e deste para "Out-File", que produz o ficheiro real com o nome e caminho definidos.

No ambiente de Windows PowerShell "cmdlet"[14] é um comando leve que é invocado no tempo de execução, dentro do contexto de *scripts* de automação que são fornecidos na linha de comando. Este comando pode ser código .NET compilável, reutilizável ou uma função avançada, ou pode ser um fluxo de trabalho que normalmente executa uma tarefa específica. Para o nosso propósito foi usado o comando "cmdlet" chamado "ConvertTo-Json".

Esta abordagem de transformar os dados em ficheiros JSON foi usada durante todo o processo de desenvolvimento da aplicação. Apenas após terminar a implementação é que foi implementada a funcionalidade de conexão à base de dados através do *back-end.*

4.3.7 Junção de todas as partes para construção do *interface*

Após ter as partes todas descritas e preparadas, falta juntar o "puzzle". Temos então a base de dados com todos os dados necessários para a manipulação neste projeto. Temos os nossos ficheiros JSON com os dados das tabelas, que serão usados para exibir os dados no *interface*. Os ficheiros serão enviados pelo nosso *back-end* para o *front-end*. O *front-end* recebe os ficheiros através do URL para o qual foram enviados e, através de uma função, transforma os ficheiros em objetos JSON.

---

[13] https://docs.microsoft.com/pt-pt/powershell/

[14] https://docs.microsoft.com/pt-pt/powershell/scripting/developer/cmdlet/cmdlet-overview?view=powershell-7

A seguir, recorrendo a esses objetos, com a ajuda da *framework* selecionada, DevExtreme, será montado o *interface*. Todas as alterações efetuadas na aplicação *Web* serão recebidas pelo *front-end*, aplicadas nos objetos JSON, e de seguida logo enviados de volta ao *back-end*. O *back-end* recebe os objetos alterados e, através de um método, escreve os dados neles contidos para os ficheiros JSON correspondentes, atualizando a informação nos ficheiros. Por fim, o *back-end* irá enviar os ficheiros atualizados para o *front-end*, para este exibir os dados no *interface* já com as alterações aplicadas. Este processo torna a aplicação *Web* responsiva, aplicação que reage às necessidades do utilizador instantaneamente.

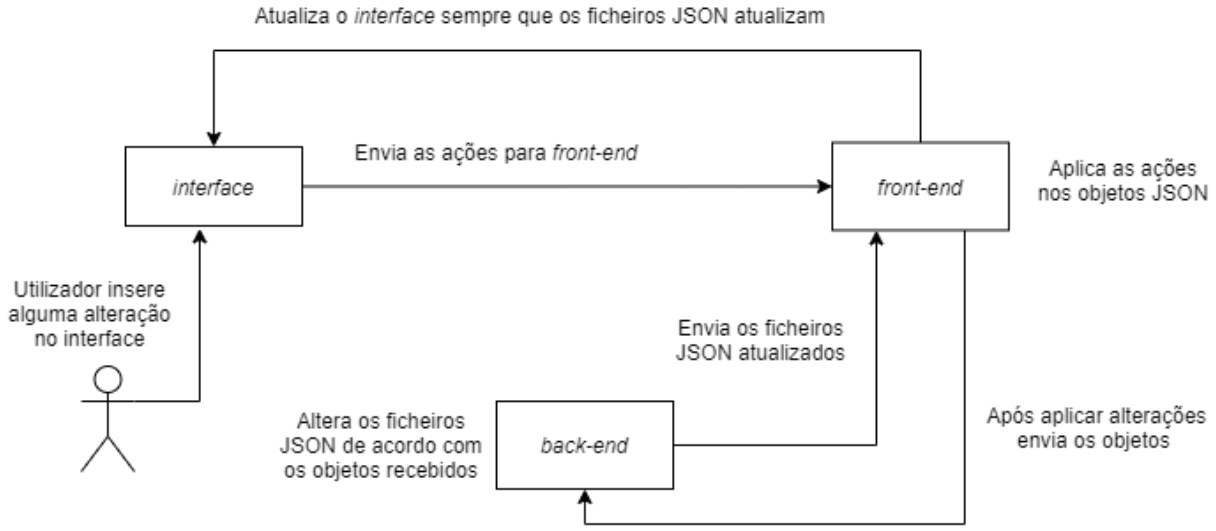

Figura 1- Esquema do caso de uso da aplicação Web

Após aplicar o esquema tal como descrito, foi relativamente fácil visualizar os dados no formato de tabela na página *Web*. O componente da *framework* usado foi o "DataGrid" que permite a edição dos dados *inCell*, ou seja, uma visualização simples dos dados em que é possível editar cada campo (*cell*) individualmente ao clicar em cima dele.

No entanto, à medida que se tentava avançar com a implementação, apanhar eventos dos botões, inserir alterações no componente, guardar as alterações, foi identificado um grande problema, que impedia a continuação da implementação da aplicação com esta *framework*.

Apesar de existir documentação com os componentes necessários disponíveis para montar o *interface* e respetivos exemplos base, isso revelou-se não ser suficiente, pois, mesmo completando com bastante pesquisa adicional na *Internet*, não foram encontradas explicações claras de como personalizar e adaptar os componentes (por exemplo, como apanhar os eventos dos botões, como associar-lhes diferentes ações ou como alterá-los). Após vários dias de pesquisa exaustiva, a ideia de usar uma *framework* paga foi abandonada, tendo em conta que não foi encontrada informação suficiente disponível para continuar com a implementação. Além disso é impensável pagar por um produto cujo propósito é acelerar e facilitar a implementação, mas que no fim de contas apenas causa problemas e impedimentos em continuar.

4.4 Abordagem final – usar uma *front-end framework* grátis

Para esta abordagem mantiveram-se todas as escolhas feitas na secção anterior para além de *front-end framework*, visto que continuam a ser as melhores e as mais adequadas soluções para este projeto. Sendo assim, a seguir descreve-se apenas a nova *framework* escolhida e analisam-se as características que levaram à sua escolha.

4.4.1 Análise da *framework* escolhida para implementação

Optou-se então por montar o *front-end* com a ajuda de Material-UI[15], um projeto *open-source* que disponibiliza componentes React para implementação mais rápida e fácil de páginas *Web*.

Material-UI surgiu com o propósito de unir React e Material Design. Material Design foi desenvolvido pela Google em 2014, enquanto que o Material-UI foi desenvolvido por uma equipa pequena e dedicada em 2017. Foi dada preferência a esta biblioteca *open-source* devido às seguintes características:

- **Bem documentada** – a documentação que se encontra no *site* oficial é bem organizada e facilmente navegável. Quanto mais fácil a utilização da biblioteca, maior a sua popularidade, consequentemente maior número de exemplos disponíveis na *Web*. Também existe a possibilidade de aceder ao StackOverflow[16] para consultar as perguntas e respostas técnicas dos desenvolvedores do Material-UI, ou até fazer perguntas caso haja alguma dúvida ainda não tratada;
- **Atualizações regulares** – Material-UI não parece[17] acabar assim tão cedo. No *site* oficial é prometida uma versão principal a cada doze meses, um a três lançamentos menores para cada versão principal e uma versão *patch* a cada semana ou sempre que exista um *bugfix* urgente. Atualmente está em vigor a versão 4, que foi lançada em maio de 2019, e já se encontra publicada a previsão da publicação da versão 5 para finais de dezembro de 2020. Além disso, no *blog*[18] são feitas publicações semanais acerca dos novos recursos e objetivos futuros;
- **Liberdade criativa** – por norma, todas as bibliotecas têm uma aparência consistente, mas aqui o que se tenta realçar é a vastidão da biblioteca e o número de opções disponíveis. Podemos usar os estilos padrão oferecidos nos exemplos ou criar com facilidade componentes completamente diferentes e únicos dando personalidade à nossa aplicação, desde a alteração da cor e forma até à fonte;
- **Os componentes funcionam isoladamente** – todos os componentes são autossuficientes e injetam apenas estilos necessários para a exibição. Podemos usar apenas um determinado botão ou componente isoladamente. Uma vantagem de poder usar apenas um único componente é evitar sobrecarga, o que leva a um melhor desempenho da implementação;
- **Grande comunidade de utilizadores** – existe um elevado número de utilizadores que fornecem suporte e exemplos adicionais de implementação.

15 https://material-ui.com/pt/

16 https://pt.stackoverflow.com/

17 https://material-ui.com/versions/

18 https://medium.com/material-ui

A montagem com esta solução é mais demorada, visto que temos de montar o ambiente peça a peça, no entanto temos todo o controlo sobre cada ação que fazemos, sobre cada botão que colocamos, cada linha que inserimos, e cada componente usado.

Capítulo 5

Ferramenta de gestão de YM Estático (*BackOffice*)

Como já foi referido, todas as indústrias de transportes estão a evoluir e a melhorar o seu *Yield Management* (YM) uma vez que, com o aumento dos concorrentes, surge a necessidade de ter uma melhor gestão do rendimento, criando sistemas cada vez mais eficazes e menos manuais, para poder beneficiar do cliente certo, na altura certa, ao preço certo, no segmento certo e através do canal mais adequado.

Sendo assim, o primeiro passo é tornar o processo de YM estático menos manual e exaustivo. Para isso, é necessário passar da gestão de todo o processo na base de dados para a gestão através de uma ferramenta de fácil utilização (uma aplicação *Web*/*BackOffice*), de modo a poder oferecer um serviço aos clientes que lhes permita serem mais autónomos e gerir o rendimento livre e instantaneamente.

No capítulo anterior já foram referidos os requisitos funcionais necessários para a aplicação *Web*. Neste capítulo é feita uma descrição geral da arquitetura da aplicação, uma breve descrição das abordagens usadas na implementação do *interface*, e descritas as funcionalidades mais relevantes.

5.1 Arquitetura

Nesta secção é dada uma visão breve e global da arquitetura da aplicação *BackOffice*.

O sistema divide-se em três camadas: camada de apresentação, camada de lógica e camada de acesso aos dados.

- **A camada de apresentação** – é a camada responsável pela apresentação e interação com o utilizador; no contexto do projeto é a parte relativa ao *Web interface* do *BackOffice*;
- **A camada de lógica** – é a camada que serve de intermediária entre o *Web interface* e a base de dados – permite obter os dados para apresentar no *interface*, com recurso a *queries* à base de dados, e alterar os dados existentes na base de dados de acordo com as operações executadas pelo utilizador na aplicação;
- **A camada de acesso aos dados** – neste projeto refere-se à base de dados, que contém todos os dados necessários para a gestão do rendimento através do *interface*.

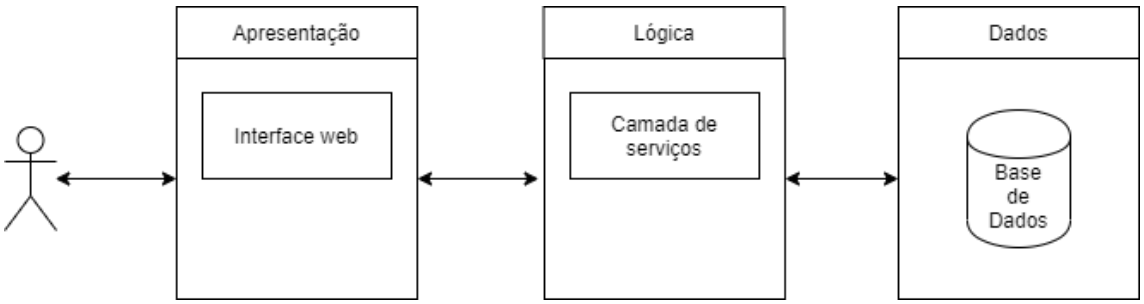

Figura 2 - Arquitetura da aplicação Web

Nesta aplicação, a camada de apresentação contém a maior parte da lógica. Após o utilizador inserir as alterações através do *interface*, a camada de aplicação recebe o *input* e aplica as alterações nos objetos que contêm os dados, ou seja, toda a manipulação com o *input* recebido é feito no *front-end*. Foi escolhida esta abordagem por causa da boa integração de *front-end framework* Material-UI (escolhida para a implementação da aplicação) com a biblioteca ReactJS, que facilita a manipulação dos dados diretamente nesta camada. A camada de serviços é responsável por receber os objetos já alterados e fazer a conexão à base de dados para guardar as alterações efetuadas.

5.2 Breve descrição da implementação da aplicação com React Hooks

Nesta secção é feita uma breve descrição da ligação entre o *back-end* e o *front-end* e a abordagem usada na implementação do *front-end* da aplicação *Web* – abordagem ReactHooks.

ReactHooks é uma abordagem recente e inovadora, utilizada na construção de *interfaces* de utilizador. A maior parte dos projetos encontram-se implementados com uma linguagem de programação simples como, por exemplo, JavaScript, ou então, recorrem a bibliotecas dessa linguagem, no caso de JavaScript as bibliotecas poderiam ser: React[19], Angular [20] ou Bootstrap[21]. No entanto, React adicionou uma nova proposta de implementação que é recorrer aos Hooks, que serão explicados mais adiante. Do ponto de vista de *front-end* é uma abordagem interessante de analisar e perceber que atualmente existem alternativas simples e rápidas que ao serem usadas apenas facilitam e reduzem o trabalho.

Dado isso, abaixo é feita a análise e descrição dos Hooks, sendo também referidos alguns dos Hooks mais significativos usados no projeto e explicados no contexto da aplicação *Web* implementada.

5.2.1 Ligação entre *front-end* e o *back-end*

O projeto encontra-se separado em duas partes, *back-end* e *front-end*; a ligação das duas foi realizada através do protocolo HTTP.

O *back-end* tem guardados todos os objetos com o conteúdo dos ficheiros JSON (que contêm os dados das tabelas) e envia os mesmos através dos pedidos POST e GET para um *Route* URL definido por nós; é criado um URL diferente para cada objeto para estes dados poderem ser recebidos separadamente.

O *front-end* recebe estes dados, guardando-os em objetos, que são posteriormente lidos e processados.

5.2.2 Análise da abordagem usada para implementação do *front-end (*ReactHooks*)*

Os ReactHooks apareceram pela primeira vez na versão v16.8.0 do React, em fevereiro de 2019. Os criadores descrevem-nos como uma nova maneira de usar o estado e outros recursos do React sem

[19] https://reactjs.org/

[20] https://angular.io/

[21] https://getbootstrap.com/

escrever uma classe. ReactHooks foram criados para introduzir a gestão de estado e efeitos colaterais nos componentes das funções.

A construção do *front-end* foi realizada com a abordagem de ReactHooks. Foi uma maneira de tornar mais fácil o uso isolado de componentes da função React, sem a necessidade de modificar o código para alterar a sua estrutura para um componente da classe, para ser possível usar métodos do ciclo de vida ou estado local.

Hooks proporcionam uma API mais direta para os conceitos React já conhecidos; além disso, oferecem uma nova forma útil e poderosa de os combinar. Resumidamente, Hook é uma função especial que permite '*Hook into*' (conectar-se a/ir buscar) recursos do React.

5.2.3 Aplicação dos ReactHooks para implementação do *front-end*

No *front-end* da aplicação, para os dados serem lidos e apresentados no *interface*, os dados são recebidos do *back-end* e guardados através dos métodos 'useFetch' e 'useState' (um Hook que permite adicionar o estado React aos componentes da função).

- **useFetch** – Hook 'useFetch' declara uma variável que serve para guardar o valor que é recebido através do único argumento que o Hook leva, que é o URL através do qual são recebidos os dados; os URL's são os mesmos que foram definidos no *back-end*. Assim, os dados são recebidos e ficam guardados na variável.
  Ex: 'const [variable] = useFetchOnce('http://URLExample');'

- **useState** – este Hook declara uma variável *state*, que serve para "preservar" alguns valores entre as chamadas de funções e retorna um par de valores: o *state* atual e uma função que atualiza o *state*. O único argumento que o Hook leva é o estado – estado inicial que se quer atribuir à variável. Assim, após receber os dados, através do método 'useState' é realizada a cópia do objeto, cópia com a qual se vai trabalhar; este método vai servir para atualizar o estado do objeto.
  Ex: 'const [state, setState] = useState(null);'

Esta abordagem permitiu uma fácil manipulação e atualização dos dados a apresentar no *interface*.

5.3 Funcionalidades implementadas e por implementar

Inicialmente existiam requisitos base gerais, tais como implementação de funcionalidades CRUD para manipulação de regras de preços, funcionalidade de ordenação de regras, criação de novas regras através da duplicação e gestão de grupos de regras.

Partindo destes requisitos base, foi então feito o estudo e levantamento de todas as funcionalidades necessárias para a manipulação do *interface*, que já foram referidas no capítulo anterior.

Todas as funcionalidades planeadas foram implementadas à exceção de uma – funcionalidade de gestão de grupos de regras. Esta funcionalidade não foi considerada indispensável para o bom funcionamento da aplicação, mas apenas do tipo *nice to have*. Sendo assim, foi deixada para uma próxima etapa de implementação.

5.3.1 Análise das funcionalidades implementadas

Nesta secção será feita uma breve descrição da aplicação e das funcionalidades que foram implementadas, tendo em conta que a descrição detalhada das ferramentas usadas e decisões tomadas foi descrita no capítulo anterior.

Abaixo segue um *print screen* da última versão do *BackOffice* implementado.

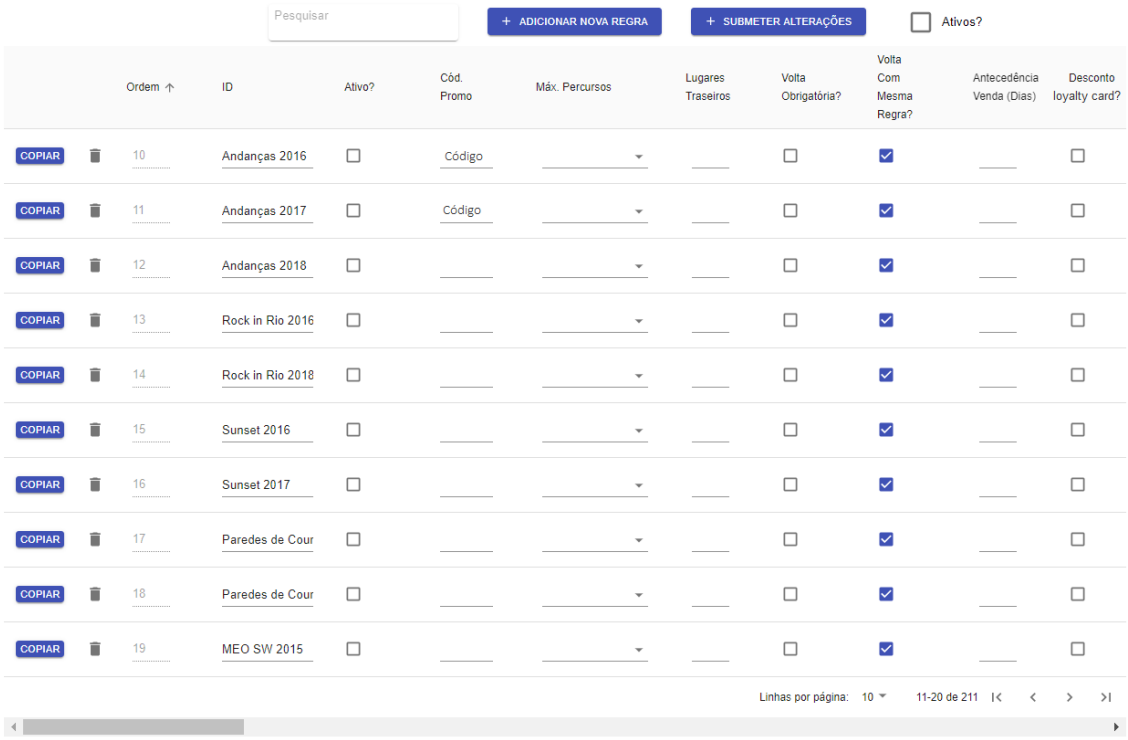

Figura 3 - Front-end do BackOffice – a)

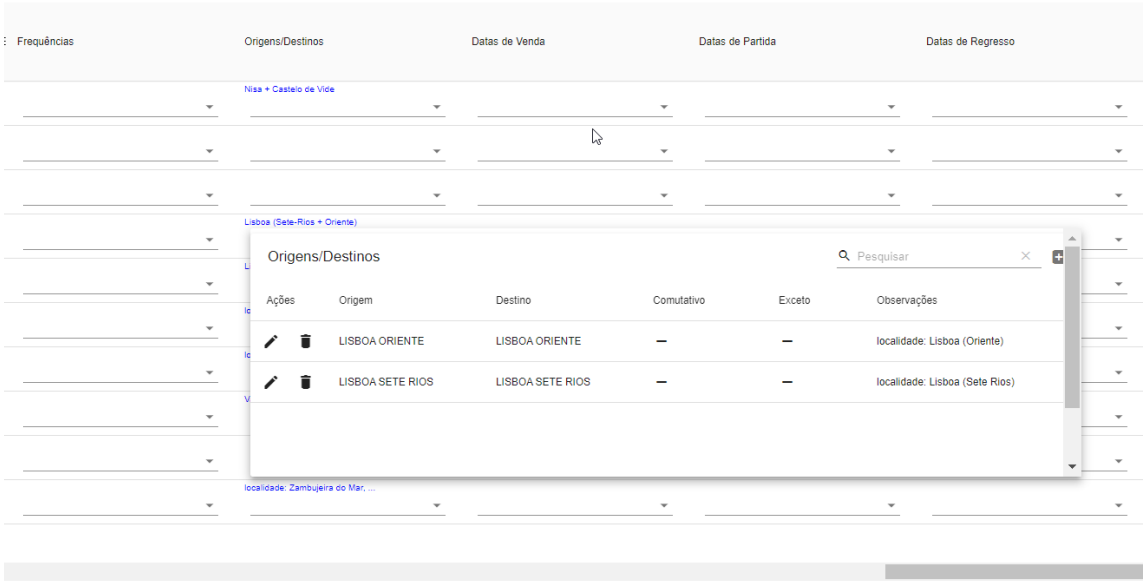

Figura 4 - Front-end do BackOffice – b)

❖ Visão geral da aplicação *Web*

A aplicação *Web* encontra-se estruturada em dois componentes principais: cabeçalho e corpo.
No cabeçalho é possível verificar a existência de quatro ações globais. As ações são denominadas como globais, visto que quando acionada uma das ações, esta atua diretamente sobre todo o corpo da aplicação, mais precisamente sobre todas as regras existentes na aplicação.
No que toca ao corpo da aplicação *Web*, este consiste numa tabela que contém todas as regras disponíveis na base de dados. Como teste foram consideradas cerca de 200 regras de descontos definidas, daí a tabela ter cerca de 200 elementos, pois cada elemento é representado por uma linha que por sua vez representa uma regra. A tabela da aplicação *Web* tem 34 colunas, ou seja, 34 campos que podem ser alterados para configurar a regra. Estes valores não são fixos visto que regras podem ser apagadas e criadas a qualquer momento e existe sempre a possibilidade de ser necessário acrescentar mais algum campo, coluna, para a definição da regra.
No corpo da tabela são realizadas ações individuais sobre cada regra. Além de editar, é possível também copiar regras existentes ou então apagar as que já existem e não são mais necessárias.

❖ Análise das funcionalidades do cabeçalho da aplicação *Web*

- **Filtrar** – na Figura 3 podemos observar o campo de pesquisa, que filtra as regras pela palavra chave ou caráter introduzido;
- **Adicionar nova regra** – existe um botão "adicionar nova regra" que adiciona uma nova linha no final da tabela com os campos por preencher;
- **Submeter alterações** – o botão "submeter alterações" serve para submeter todas as alterações efetuadas à base de dados, caso contrário, qualquer alteração inserida apenas será guardada nos objetos JSON no *back-end*. Foi tomada esta decisão uma vez que guardar instantaneamente qualquer alteração que se introduza no *interface*, como alterar um campo *check box* ou fazer uma alteração *inCell*, acabava por ser muito custoso, tornando o processo muito mais lento; para configurar uma regra é necessário preencher bastantes campos e com esta solução é possível submeter as alterações após efetuar todas as manipulações necessárias;
- **Filtrar por regras ativas e não ativas** – existem muitas regras que não se encontram ativas no momento, porque foram configuradas para serem testadas ou porque se encontram preparadas para posterior ativação. O filtro do campo *check box* "Ativos?" ajuda a visualizar as regras que estão de momento a ser utilizadas.

❖ Análise das funcionalidades mais relevantes do corpo da aplicação *Web*

- **Copiar/Apagar** – à frente de cada regra existem botões que permitem apagar e copiar a regra (copiar facilita na configuração, pois caso exista uma regra semelhante à que queremos criar basta duplicá-la e simplesmente inserir as alterações necessárias);
- **Campos simples** – os restantes campos são simples, como *check box*, para alterar os campos com valor binário, *inCell*, que permitem alterar textos e valores de uma forma simples e rápida, e *dropdown*, em que temos já as opções predefinidas para a escolha;
- **Campos complexos** – na Figura 4 está representado o campo mais complexo de todo o interface, o qual, embora pareça à primeira vista um simples *dropdown*, permite-nos visualizar uma tabela completa, com paginação, pesquisa, possibilidade de apagar o item (linha) ou editá-lo, e de criar itens novos. A tabela do *doropdown* já não foi montada componente a componente; como tem poucas colunas para configurar, foi

possível usar um componente de tabela de dados já criado, “material-table”; a implementação deste componente é baseada nos componentes do *Material-UI*;

- ***Drag and drop*** – com esta funcionalidade é possível arrastar as linhas da tabela, alterando a ordem das regras e por conseguinte a sua relevância, visto que as regras se encontram ordenadas, por *default*, pela sua relevância.

5.4 Testes realizados à aplicação

Os testes de *software* têm como objetivo monitorizar a qualidade funcional do produto que se encontra em desenvolvimento e garantir que este cumpra todas as especificações pré-definidas. Existem diversos tipos de testes com diferentes finalidades que têm de ser aplicados em diferentes fases do ciclo de vida de projeto.

Segundo o trabalho de (Pressman) os testes de software podem ser divididos em quatro tipos principais: *(i)* testes de unidade; *(ii)* testes de integração; *(iii)* testes de validação; *(iv)* testes de sistema (funcionais).

- **Teste de unidade** – “concentra-se no esforço de verificação de menor unidade de projeto de software – o módulo”;
- **Teste de integração** – após testar cada módulo individualmente, ou seja, funcionalidades do *software*, é testada a integração entre diferentes módulos que formam o sistema, a fim de verificar se estes funcionam corretamente quando interligados. Existem duas abordagens de fazer testes de integração: *top-down e bottom-up*. A primeira consiste em começar pelo módulo principal (de mais alto nível), que sucessivamente é decomposto, a fim de testar os módulos de nível inferior. A segunda abordagem é o oposto, primeiro são testados os módulos localizados nos níveis hierárquicos mais baixos, que posteriormente são agrupados em *clusters*, que também são testados e agrupados entre si [(Rodrigues), (Pressman)];
- **Teste de sistema (testes funcionais)** – consistem numa série de diferentes testes, que visam testar o comportamento de todo o sistema, ou seja, verificar se todos os elementos do sistema foram integrados de forma correta e realizam as funções atribuídas. De uma forma geral os testes concentram-se em verificar (Carvalho):
    a) Os valores dos *inputs* válidos e inválidos, com intuito de perceber que valores aceitar e que valores rejeitar;
    b) Funções que deverão ser executadas em determinadas chamadas do sistema;
    c) *Output* do sistema;
    d) Sistemas ou procedimentos que são chamados.

    Os testes a realizar ao sistema, podem também ser de seguinte natureza:
    e) Testes de recuperação;
    f) Testes de stresse;
    g) Testes de segurança;
    h) Testes de desempenho.
- **Teste de validação** – teste criado a fim de simular a utilização da aplicação pelo utilizador final. A validação é considerada como bem-sucedida quando o software funciona de uma maneira esperada pelo utilizador [(Carvalho), (Pressman)];

No processo de implementação do *BackOffice*, foram realizados os respetivos testes em todas as fazes do ciclo de vida da aplicação, a fim de garantir o correto funcionamento da aplicação. A aplicação foi apenas testada num único *browser* – Google Chrome (versão 86.0.4240.111).

Os testes iniciais foram os testes de unidade, ou seja, a implementação de cada componente era acompanhada por testes individuais, a fim de verificar se cada módulo funcionava de acordo com a especificação. A fase seguinte de verificação, consistiu em realizar testes de integração, com o objetivo de verificar a correta comunicação entre os módulos (por exemplo, se os dados eram guardados corretamente). Após a correta integração dos módulos foram realizados testes funcionais (do tipo a, b, c e d), a fim de fazer verificações em termos do conteúdo que era guardado, verificar que valores fazia sentido serem aceites e rejeitados, ou verificar que campos era obrigatório serem preenchidos. Para fazer esta última verificação, foi necessário realizar o estudo acerca de regras existentes, com o intuito de perceber que valores eram aceites, que limitações havia, resumidamente qual a construção e funcionamento correto de regras.

Assim sendo, para evitar que o utilizador inserisse regras inválidas como, por exemplo, criar ou guardar regras vazias, recorreu-se ao uso de objeto *window* do JavaScript que tem diversos métodos associados, e entre estes, o "alert ()". Quando o método "alert ()" é chamado, exibe uma caixa de alerta com uma mensagem, por nós previamente escrita, e um botão "OK". Além deste método foi também utilizado o método "confirm ()" que é semelhante ao anterior, sendo a diferença em que, para além do botão "OK", tem o botão de "Cancelar". A diferença entre os dois métodos é que o primeiro exibe apenas uma mensagem de alerta e no segundo é possível escolher entre confirmar (continuar com a alteração inserida) ou cancelar (voltar atrás e não aplicar as alterações). Esta foi a técnica essencial utilizada para controlar o correto preenchimento e inserção de conteúdo.

Por fim, foram realizados testes de validação, simulando possíveis ações do utilizador final, com o objetivo de verificar se o programa funcionava de acordo com o esperado pelo utilizador.

Atualmente a ferramenta implementada encontra-se funcional e em testes.

Capítulo 6

Yield Management Dinâmico

Gestão de preços dinâmicos é uma área muito complexa que requer o uso de diferentes tecnologias e conhecimento de diferentes áreas científicas, entre as quais:

- ***Marketing*** – fundamental para fazer o estudo do mercado em que se quer aplicar a gestão de rendimentos dinâmica, a fim de aplicar as melhores estratégias de preços e serviços;
- **Gestão** – é necessário fazer a gestão financeira e a de recursos para ajudar na tomada de diversas decisões relacionadas;
- **Matemática** – aplicada praticamente em cada área, para criar preços dinâmicos é necessário o seu conhecimento profundo, a fim de criar os melhores e mais eficientes algoritmos de cálculo;
- **Programação** – é necessária a fim de integrar e implementar no sistema o processo de *Yield Management* Dinâmico.

Neste capítulo pretende-se mostrar, através de exemplos e aplicações simples e específicas, aplicadas no contexto de uma empresa exemplo (designada daqui em diante como "empresa exemplo"), as várias facetas de *Yield Management* dinâmico, sem a complexidade desejável num sistema de uso profissional, dado que para tal seria necessária intervenção de especialistas de diferentes áreas científicas. Mais especificamente, neste capítulo serão descritas várias etapas necessárias para a construção do sistema de *Yield Management* dinâmico. Serão abordados tópicos como:

(i) fatores/variáveis necessárias de introduzir no protótipo do novo sistema de *Yield Management*, a fim de diversificar e dinamizar os preços;
(ii) ferramentas, *bots* e algoritmos necessários para recolher a nova informação para o cálculo de preços;
(iii) esboço da nova arquitetura, introduzindo as novas ferramentas;
(iv) sugestão de algumas regras e algoritmos para o cálculo de preços recorrendo à nova arquitetura sugerida e aos novos fatores.

6.1 Novos fatores para tornar *Yield Management* dinâmico

Nesta secção é feita a análise de fatores/variáveis empregues, no módulo exemplo (referido no Capítulo 3), no cálculo de preços dos bilhetes de autocarros. São analisados tipos de fatores que poderão influenciar os preços e descritas as variáveis necessárias a serem introduzidas, a fim de tornar os preços mais diversificados e dinâmicos.

6.1.1 Fatores atualmente utilizados

No sistema do módulo exemplo, já analisado, encontram-se empregues diversas variáveis no cálculo de preços dos bilhetes. Entre elas, tal como já foi referido num dos capítulos anteriores, são usadas variáveis como:

(i) perfil do cliente;
(ii) informação sobre origem e destino da viagem pretendida;
(iii) datas de partida ou chegada;

(iv) *loyalty card* gratuito que oferece ofertas exclusivas aos membros;
(v) diferentes pontos de venda de bilhetes que podem ser físicos e *online*;
(vi) frequência com que os descontos/promoções são ativados;
(vii) número do serviço / expresso para o qual será aplicado um determinado desconto;
(viii) número de lugares a que é aplicado um desconto, entre outras.

Essencialmente são estas as variáveis/fatores utilizados para definir os preços dos bilhetes ou para introduzir promoções e descontos pontuais. No entanto, para acompanhar a dinâmica do mercado atual estes fatores só por si não são suficientes. Os valores no mercado mudam a cada instante, as ofertas estão constantemente a subir e a descer, promoções e descontos aparecem à última hora. Para empresas que não têm os sistemas autónomos e mais atuais de gestão de rendimento implementadas torna-se impossível acompanhar as mudanças empregues pelos seus concorrentes, uma vez que monitorizar o mercado e os concorrentes, manualmente, é um processo exaustivo e exige bastantes recursos. Além disso, provavelmente no momento em que for decidido aplicar alguma alteração nos preços e até o processo estar concretizado, os preços do mercado já podem ter sido alterados várias vezes. Desta forma as empresas que têm os sistemas manuais passam a estar em grande desvantagem.

Torna-se então necessário a inserção de novos fatores no cálculo. Dado que existem bastantes fatores internos já empregues no cálculo de preços, o foco seria adicionar fatores externos do mercado e do ambiente em que a indústria de autocarros de longo curso se encontra inserida. Além disso, criar um sistema que torne dinâmico todo o processo de cálculo, de modo a que os preços estejam sempre atualizados de acordo com os dados mais atuais.

6.1.2 Tipos de fatores possíveis de usar na formação de *Dynamic Pricing*

Existem diversos trabalhos de pesquisa acerca dos fatores que influenciam a precificação dinâmica, contudo, não existe uma classificação sólida geral aceite por todos os investigadores. Sendo assim, é possível verificar que existe uma ampla variedade de fatores disponíveis, agrupados e classificados de forma distinta. Compete ao investigador/cientista/autor decidir que fatores usar e com que classificação, tendo sempre em contra o trabalho que se encontra em desenvolvimento (Deksnyte and Lydeka).

Uma ideia que é necessário ter presente quando se quer implementar preços dinâmicos, é que um ou dois fatores não são suficientes para resolver o problema. Por essa razão, cada vez mais, é possível verificar a existência de mais fatores diferentes, tanto fatores novos que fazem realmente diferença, como fatores que já existiam, mais gerais, que foram divididos em fatores mais específicos.

Nesta secção serão abordados e descritos os fatores mais interessantes e com maior impacto na formação da precificação dinâmica.

- **Cliente** – um dos fatores mais importantes que modela os preços dinâmicos, uma vez que o sucesso da empresa depende da satisfação dos clientes. A empresa deve certificar-se de que está a prestar um serviço de qualidade a um preço justo, pois se o cliente ficar satisfeito, a empresa irá conseguir construir um bom relacionamento com o mesmo a longo prazo. Sendo assim, é necessário conseguir perceber a elasticidade na procura dos consumidores e o seu nível de conhecimento. À primeira vista parece um procedimento bastante simples, no entanto, é difícil ter a perceção da satisfação dos clientes num mercado dinâmico e competitivo.
  Em alguns trabalhos realizados (Deksnyte and Lydeka), para a melhor compreensão do comportamento dos clientes, estes são classificados como: clientes míopes ou clientes estratégicos. Um cliente míope é aquele que faz uma compra quando o preço sugerido é inferior

ao que desejava pagar inicialmente, ou seja, não recorre a estratégias complexas no momento de decisão da compra. Um cliente estratégico é aquele que otimiza o seu comportamento de compra em resposta às estratégias de preços. Recorrendo a esta última classificação, a modelagem de preço dinâmico torna-se mais realista.

Outra forma de classificar os clientes (Deksnyte and Lydeka) é identificar se a população de potenciais clientes é finita ou infinita. A população é infinita quando a informação sobre as procuras anteriores não influencia o número de potenciais clientes e a sua disposição de pagar por um determinado produto ou serviço (por exemplo, se comprar um cacho de bananas hoje, não deixo de ser um potencial cliente para comprar outro amanhã). A população é finita quando um cliente deixa de ser um potencial cliente por algum tempo (tempo depende da durabilidade do produto) pelo facto de comprar o produto (por exemplo, se um cliente comprar um carro hoje, por alguns anos deixa de ser um potencial cliente, visto que a duração do produto é bastante longa).

De acordo com a literatura existente, nesta temática, a convicção geral é que a reação dos clientes a alterações dos preços é mudar para o serviço com o preço inferior; desta forma, caso um dos concorrentes baixe o preço é necessário criar resposta para não perder os seus clientes para ele (Almeida);

- **Procura do produto** – previsões de procura de um produto são essenciais para as empresas. Com uma boa previsão é possível proporcionar um melhor atendimento dos clientes e fazer uma melhor gestão do *stock*. A procura do produto está intimamente ligada ao preço do mesmo – se o preço mudar, a procura também muda em resposta; por outro lado, o preço pode ser ajustado em termos de procura (por exemplo, se a procura estiver baixa, é possível diminuir um pouco o preço para aumentá-la).

  (Bi and Liu) no seu artigo, demonstram que o cálculo mental que os consumidores realizam tem impacto significativo na procura do produto, logo, é conveniente ter este aspeto em consideração para obter uma previsão da procura mais precisa.

  Segundo Dickson e Urbany, a elasticidade na procura é determinante no comportamento dos preços. A procura de cada produto varia ao longo do ano e, em resposta às variações, as empresas tentam acompanhar através de promoções ou outro tipo de campanhas;

  A procura está também relacionada com os consumidores e o valor que estes lhe atribuem, estando este valor dependente do grau de satisfação que os clientes retiram do consumo do produto ou serviço, ou seja, a utilidade que o bem proporcionou (Almeida);

- **Estrutura do mercado** – a estrutura do mercado tem uma grande influência na definição de preços. É necessário fazer um estudo do mercado de modo a identificar o mercado-alvo, ou então, perceber que estratégia aplicar em cada segmento. Segundo (Kanagal), cada segmento do mercado pode ter perceções diferentes e, como tal, ofertas de preços diferentes. Assim sendo, é necessário customizar ofertas e adaptar os preços para grupos de consumidores diferentes. Kanagal propõe a divisão do mercado nos seguintes grupos de consumidores [(Almeida) , (Kanagal)]:

  - Clientes de grande, médio e baixo nível de compra;
  - Clientes leais, leais e lucrativos, leais mas indecisos, leias que alteram entre os concorrente e os que vêm e vão;
  - Clientes bem informados e clientes com experiência;
  - Clientes potenciais, novos, regulares, da concorrência;

- Clientes insatisfeitos e satisfeitos;
- Clientes inovadores, primeiros a adotar novos produtos, clientes do mercado principal, maioria tardia, e clientes retardatários;
- Clientes conscientes do valor e conscientes do luxo;
- Clientes experientes, novatos, de conhecimento limitado, sensíveis às informações e em processamento de informações;
- Clientes com conhecimento, líderes de opinião, seguidores e de mercado de massas;
- Cliente que cria produtos e que revende;
- Clientes de uma única compra/visita, de tempo limitado de transação e de relacionamento;
- Clientes sensíveis ao preço e menos sensíveis ao preço;
- Clientes que procuram qualidade, marcas e produtos resistentes;
- Clientes que procuram satisfação e prazer, e que procuram serviços associados;
- Clientes avessos ao risco e propensos ao risco;
- Clientes que procuram funcionalidade, e clientes que procuram estilo e prazer.

❖ **Perceção do valor do produto** – a perceção do valor justo por parte do cliente depende principalmente da quantidade de informações que as empresas revelam. Na dissertação de (Dai) o conceito de justiça do valor percebido é definido como: as avaliações dos consumidores sobre se o preço de um vendedor pode ser razoavelmente justificado.
Como é sabido, os clientes têm o habito de adiar as compras com o objetivo de no futuro obter uma oferta melhor, no entanto, podem existir outros motivos para adiar as compras como, por exemplo, não saberem como avaliar o produto/serviço, o que os leva a esperar até obter mais informações necessárias para a tomada de decisão segura (Deksnyte and Lydeka). As perceções dos clientes sobre o produto/serviço são também definidas como as perceções do cliente sobre qualidade, satisfação e valor, ou então, como resultado de uma série de observações. As perceções dos clientes formam-se com base na sua experiência, sendo sempre ligeiramente diferentes de cliente para cliente (Bertsimas and Vayanos).
(Deksnyte and Lydeka), no seu trabalho referem autores como Dada, Petruzzi (2002), Yu, Kapuscinski e Ahn (2005), visto que, segundo os trabalhos deles, os clientes usam o preço como um indicador de custo e de qualidade percetível do produto que irão incidir durante a compra de um produto/serviço.
Segundo (Deksnyte and Lydeka), é possível concluir que "a perceção do valor do produto está diretamente relacionada com a preferência e escolha dos clientes, ou seja, quanto maior a perceção de valor, maior é o desejo de compra ou preferência do produto".

❖ **Sazonalidade** – de uma forma curta e geral, a sazonalidade consiste na variação da procura de um produto ou serviço de acordo com a época do ano. No entanto, as coisas podem complicar, visto que a sazonalidade pode ser semanal, mensal ou outra completamente diferente, tudo dependendo do produto ou serviço que se está a oferecer. Na indústria da moda é possível verificar que no fim de cada estação as coleções de roupa antigas baixam o preço e esgotam, e novas coleções vêm substituí-las. Outro exemplo, completamente diferente é o de bebidas alcoólicas, em que a sazonalidade corresponde a cada sexta-feira, sendo que pelos estudos feitos é o dia de maior aquisição de álcool, assim sendo, às sextas-feiras é possível observar muitos descontos nestas secções.
No estudo realizado por Chevalier, é mostrado que durante os períodos sazonais de maior procura os preços baixam e o lucro é menor devido às promoções realizadas para diminuir os

preços e propaganda, no entanto, os custos marginais permanecem inalterados [(Chevalier, Kashyap and Rossi), (Deksnyte and Lydeka)].

- **O inventário** – inventário de *stock* é uma prática que recorre à identificação, classificação e contagem de todos os produtos a fim de conferir se o controlo de *stock* ocorre de forma correta. Esta prática tem diversos benefícios, entre eles:
  - Redução de perdas;
  - Redução de desperdícios;
  - Melhor gestão do *stock*;
  - Melhor atendimento ao cliente;
  - Melhor organização do *stock*;

  O preço e a prática de inventário podem ser combinados e usados, por exemplo, da seguinte forma: tendo um *stock* de 60 lugares disponíveis no autocarro, quando 50 forem vendidos, os últimos serão vendidos a 0,80 vezes o preço (Dodeja, Suresh and Mehta);

- **Preços dos concorrentes** – com a evolução da tecnologia e o aumento de competição em todas as áreas industriais, torna-se difícil "sobreviver" calculando os preços apenas com base nos fatores internos, surge então a necessidade de recorrer a fatores externos, como o preço dos concorrentes. No entanto, acompanhar os preços dos concorrentes é uma tarefa complicada, uma vez que, os preços dos seus produtos podem alterar-se várias vezes ao dia, ou, se considerarmos o caso da Amazon, a cada dez minutos. Assim sendo, é necessário adotar novas tecnologias, *softwares*, ou lógicas de trabalho, visto que analisar a todo o tempo os *sites* ou catálogos dos concorrentes é uma tarefa tediosa e que não traz os melhores resultados.
  No fim, ao ter os dados dos concorrentes é possível comparar os seus preços com os nossos e perceber qual a nossa posição no mercado, quais as melhores decisões a tomar, e ofertas a fazer aos clientes, para fazer frente aos concorrentes;

- **Períodos** – as vendas e a procura não são constantes 24/7, existem períodos em que a procura aumenta, outros em que diminui. A dimensão dos períodos varia de empresa para empresa, por exemplo, para o negócio de gelados, estes vendem-se durante todo o ano, no entanto, no caso de Portugal, as maiores vendas ocorrem a partir de meados de março até meados de outubro.
  Analisando o caso da indústria de autocarros de longo curso, a tarefa complica-se um pouco, dado que os períodos de vendas oscilam demasiado e com maior frequência. Algumas mudanças que têm impacto na variação de procura são imprevisíveis como, por exemplo, ocorrência de eventos ou alterações climáticas.
  Posto isto, é de grande importância analisar e detetar todos os períodos de maior e menor venda ou movimento, de modo a que seja possível tirar o maior proveito de cada época.
  Toda esta análise poderá ser realizada manualmente, numa fase inicial, no entanto, para ter os melhores resultados, mais rápidos e precisos é necessário investir em algoritmos que detetem diferentes padrões de procura, a fim de construir uma melhor estratégia de preço final. Um exemplo de lógica a aplicar na estratégia poderá ser: caso seja detetado que sexta-feira é o dia mais ativo, é possível aumentar ligeiramente os preços, visto que sabemos que esse dia é sempre de grande procura, no entanto, se detetarmos dias de pouca procura, como por exemplo, quarta-feira, é possível descer um pouco o preço até a procura normalizar e depois voltar a subir até encontrar o melhor preço para esse período.

- **Eventos locais** – eventos locais não é um fator que seja relevante para a maioria das indústrias, contudo, é importante para os autocarros de longo curso. O objetivo dos autocarros é levar os passageiros de uma origem para um determinado destino. Os preços costumam ser constantes, pelo menos na maioria das empresas que não têm os preços dinâmicos implementados, tirando alguma promoção que possa ser feita localmente. Deste modo, para criar alguma diversidade é possível introduzir algumas alterações nos preços, tendo em conta os eventos que ocorrem em determinados destinos. Estas novas alterações serão válidas, visto que, quando um destino tem algum evento, este torna-se por defeito um local de maior procura e os bilhetes esgotam com maior facilidade. Desta forma é possível tirar o proveito da situação de várias formas, por exemplo, subir ligeiramente os preços, para obter maior lucro, tendo em atenção para que essa mudança não seja muito relevante para os clientes, ou então, pelo contrário, aumentar o número de carreiras para o destino, para poder transportar maior número de clientes e descer um pouco o preço em relação à competição para obter maior número de vendas de bilhetes, que no final irá compensar. Existem outras formas de conjugar as coisas e com outros fatores, mas para tomar alguma decisão é necessário fazer um bom estudo do mercado e dos clientes.

- **Estado do tempo** – este é um outro fator que não tem muita popularidade em nenhuma indústria em particular, e não é mencionado como fator para cálculo de preços dinâmicos em nenhum dos trabalhos consultados para a escrita desta dissertação. Contudo, é interessante analisá-lo e aplicá-lo para estudar os seus efeitos, pois possivelmente é um fator que será reconsiderado e valorizado por muitos mercados.
  Para o caso específico de autocarros de longo curso, é um fator interessante de aplicar em determinados casos, como por exemplo, nas rotas de verão que têm como destino praias. Analisando a meteorologia é possível verificar os dias de maior calor e dias de possível chuva, sendo possível subir um pouco os preços no primeiro caso, conjugando, por exemplo, com os dias de maior procura e descer nos dias com previsão de chuva, que são os dias em que os clientes ficam com mais dúvida na compra dos bilhetes.
  Também seria um fator interessante a introduzir no mercado de gelados, subindo os preços nos dias de maior calor, que são os dias em que as pessoas procuram formas de se refrescar.

Todos estes fatores são interessantes só por si, porém, têm mais efeito se conjugados entre si. É necessário analisar sempre o mercado, os concorrentes, e os clientes, a fim de perceber o que realmente é relevante à empresa para crescer, e que estratégias de preços é necessário aplicar, pois, mesmo recorrendo a iguais fatores, as estratégias ou lógicas no cálculo diferem para cada caso, dependendo das necessidades de cada empresa.

6.1.3 Fatores mais relevantes para *Long-Distance Bus*

Nesta secção serão abordados e descritos de uma forma mais resumida, os fatores que se consideram mais relevantes, dos descritos na secção anterior, para a primeira etapa e primeiros passos na direção de cálculo de preços dinâmicos, no mercado de autocarros de longo curso.

Para melhor perceção da informação, as variáveis irão ser divididas em fatores externos, que correspondem a toda a informação que vem de fora da empresa exemplo, e fatores internos, que correspondem a todos os dados que são calculados ou simplesmente se encontram à disposição para

serem usados, nos ficheiros ou base de dados da empresa exemplo. Todas as variáveis serão referidas com base nos dados dos clientes da empresa exemplo, a fim de se ter um exemplo mais concreto e percetível.

Todas os fatores mencionados abaixo serão usados numa secção mais a frente, que corresponde às regras para o cálculo de preços dinâmicos.

- ❖ Fatores Internos
  Os fatores mais óbvios a serem utilizados são os dados que são inseridos pelos clientes, no *site online* de venda de bilhetes, no ato de compra do bilhete. Estes dados correspondem à maior parte dos dados internos que serão utilizados.

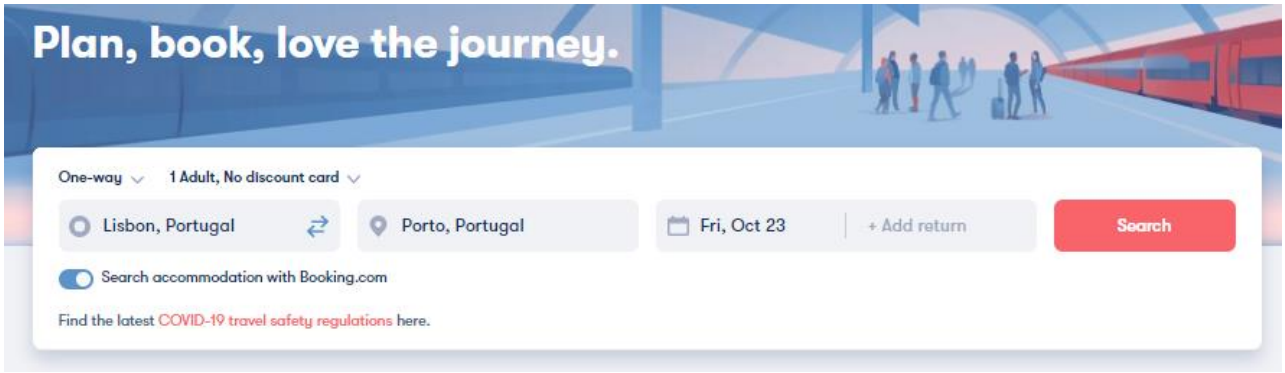

Figura 5 - Inputs inseridos pelos utilizadores na compra de bilhetes

- Origem;
- Destino;
- Data de ida;
- Data de volta;
- Número de passageiros e o seu tipo;
- Tipo de passageiros;
- Código de *loyalty card* (gratuito) conjugado com o tipo de passageiro.

Fatores que se encontram no sistema da empresa exemplo e podem ser utilizados no cálculo.

- Valor fixo tabelado para cada origem e destino;
- Lotação de cada carreira.

- ❖ Fatores Externos
  A informação sobre estes fatores já não se encontra nas "mãos" da empresa exemplo, havendo por isso a necessidade de os obter com ajuda de algoritmos, *softwares* ou, então, recorrer ao serviço de outras empresas que tratam de fornecer os dados, como por exemplo, dados relativos à concorrência.
  - Preços dos concorrentes;
  - Eventos locais;
  - Estado do tempo;

- Períodos de maior procura.

Em termos de fatores externos, as razões de terem sido escolhidos os quatro mencionados acima para se aplicarem nesta primeira fase, foram as seguintes:

(i) em termos de lógica, estratégia e análise de dados são fáceis de se perceberem e aplicarem;
(ii) com ajuda de algum *software* ou *bot* gratuito disponível online, será relativamente rápida e fácil a obtenção dos dados necessários;
(iii) após a sua aplicação, rápidos resultados se verificarão na mudança de preços;
(iv) interessantes e inovadores no setor de transportes.

6.2 Ferramentas existentes para construir *software de Web scraping*

A técnica *Web scraping* consiste em extrair grandes quantidades de dados de *websites*. Este processo pode ser feito manualmente acedendo ao *site* do qual se pretende extrair a informação e usando copiar e colar dos dados que forem necessários, para um determinado ficheiro ou base de dados, por exemplo, para posteriormente analisá-los.

Contudo, é possível evitar o processo manual – tedioso e dispendioso em termos de tempo – e automatizar o processo, construindo um *software* de *Web scraping*, que pode ser implementado para extrair dados de um *site* específico ou configurado para funcionar com qualquer *site*, dependendo a decisão de vários fatores, entre eles:

(i) de quantos *sites* se pretende extrair informação;
(ii) em quantos projetos será necessário utilizar esta técnica;
(iii) ter em conta o futuro, ou seja, se será necessário acrescentar novos *sites*.

Existem duas opções para *Web scraping* automático: *(i)* usar um *software* comercializado, *(ii)* criar um novo.

- *(i)* existe *software* para utilizadores técnicos, que necessitam de conhecimento de programação, no entanto a maior parte das soluções disponíveis são à base de *Web scraping* visual que não requer conhecimentos de programação. Nas soluções de *Web scraping* visual é apenas necessário fornecer o URL do *website* do qual se pretende extrair a informação, apontar para os dados que se querem extrair, clicar em cima deles e por fim extrair. É possível também configurar a hora, o dia ou a frequência de extração de dados. Contudo, a maior parte das soluções disponíveis são pagas, o que se torna dispendioso a longo termo.

- *(ii)* é uma solução mais dispendiosa nas primeiras etapas, no entanto, a longo prazo compensa todo o investimento e trabalho dedicado. Esta opção requer conhecimentos de programação para desenvolver um *software* de extração de dados personalizado e adaptado às necessidades específicas da empresa. Neste processo é usual recorrer-se ao uso de ferramentas *open-source* (existem várias, com diferentes características) que ajudam na extração da informação dos *websites*.

Alguns exemplos de ferramentas possíveis de utilizar no *Web scraping*:

- **Scrapingbot[22] e ScrapingBee[23]**
  São ferramentas que proporcionam acesso a diferentes *Web scraping* APIs especializadas em diferentes setores. São capazes de localizar a informação necessária a ser retirada do HTML da página *Web*.

- **Scrapy[24]**
  Scrapy é uma *open-source* Python *framework*, usada para rastrear *sites* e extrair dados estruturados das suas páginas, podendo a extração também ser via APIs. Além de extrair e armazenar dados, Scrapy fornece muitos outros recursos poderosos como: suporte integrado na seleção e extração de dados, suporte para gerar exportações em vários formatos, suporte de codificação e deteção automática, entre outros.

- **Jaunt[25]**
  Jaunt é uma biblioteca Java para *Web scraping*, automatização da *Web* e consulta JSON, que permite que os programas Java sejam capazes de:
  (i) extrair dados JSON e fazer *Web scraping*;
  (ii) trabalhar com formulários e tabelas;
  (iii) pedidos/respostas;
  (iv) trabalhar com *interfaces* com APIs REST ou aplicações *Web* (JSON/ HTML/ XHTML ou XML).

- **Selenium[26]**
  Selenium é um projeto muito abrangente, mas de uma forma sucinta é um conjunto de ferramentas e bibliotecas que permitem dar suporte à automatização do navegador *Web*.
  Esta ferramenta é usada muitas vezes para fazer testes de *front-end*, tendo em conta que permite aos utilizadores simular atividades comuns que são realizadas pelos utilizadores finais, tais como: inserir textos em campos, selecionar valores e caixas de seleção e clicar em links em documentos. Além disso, fornece também outros tipos de funcionalidades como o controlo do movimento do rato, a execução arbitrária de JavaScript, entre outros.
  Sucintamente, além de permitir fazer *Web scraping*, a ferramenta também permite testar *sites* e automatizar qualquer ação demorada na *Web*.
  Selenium usa a sua *WebDriver* API em conjunto com um *driver* de navegador (no caso de Google Chrome seria o ChromeDriver), deste modo funcionando da mesma forma que o utilizador final a abrir manualmente o navegador *Web* para realizar determinadas ações.
  A ferramenta é muito adaptável e diversificada para desenvolvedores com diferentes tipos de conhecimento, visto que a *WebDriver* pode ser usada em Java, Python, C#, Haskell, Ruby, entre outros.

22 https://www.scraping-bot.io/

23 https://www.scrapingbee.com/

24 https://scrapy.org/

25 https://jaunt-api.com/

26 https://www.selenium.dev/

- **Nightmare.js**[27]
  Nightmare é uma biblioteca de automatização de navegador de alto nível. Originalmente foi projetada para automatizar as tarefas em *websites* que não têm API, no entanto atualmente é usada com maior frequência para fazer testes de UI e *Web scraping*. O objetivo da ferramenta é expor métodos simples, que simulam as ações do utilizador final, que compõem uma API simples, mas poderosa.

Além das ferramentas mencionadas, existem muitas mais, com características completamente diferentes. Foram aqui escolhidas as mais interessantes do ponto de vista de *Web scraping* e mais diversificadas, a fim de se poder ter uma visão geral da temática e das possibilidades existentes.

Para obter especificamente o fator de preços dos concorrentes, foi implementado um *software* de *Web scraping*, recorrendo à ferramenta Selenium.

6.3 Implementação do *Web scraping* com Selenium para fator preços dos concorrentes

Nesta secção serão descritos os passos da implementação do *software* de *Web scraping* para obter a informação relacionada com o fator correspondente à preços dos concorrentes.

Para implementar o *Web scraping* optou-se por utilizar a ferramenta *open-source* Selenium. A escolha foi baseada na documentação, simplicidade, tipo de linguagem e quantidade de informação disponível online. Selenium tem uma documentação simples de entender, com vários exemplos em código, e opções em diferentes tipos de linguagem, entre os quais C# que é a linguagem preferencial neste projeto. Além disso, é simples de aplicar e existe bastante informação disponível *online*, com exemplos de implementação e tutoriais, o que para um programador é sempre uma grande vantagem.

O objetivo do *software* de *Web scraping* implementado é extrair as rotas e os preços dos clientes da empresa exemplo e dos seus concorrentes, a fim de poder compará-los e criar regras de preços de acordo com os dados extraídos, tornando possível acompanhar o mercado e criar preços dinâmicos.

Nesta implementação foi realizado *Web scraping* a dois *sites*, o *site* dos clientes da empresa exemplo e o *site* dos seus concorrentes, a empresa de transporte de passageiros de longo curso FlixBus.

O *software* implementado tem as seguintes funcionalidades:

- Acede ao *site* dos clientes da empresa exemplo;
- Acede ao *site* dos concorrentes;
- Simula a pesquisa para um determinado par origem e destino, e para um tipo de clientes específico, em cada *site*;
- Localiza os dados necessários para a extração – hora de partida, hora de chegada e preço – de todas as rotas, em cada *site*;
- Extrai os dados de cada *site* para ficheiros diferentes do tipo ".txt".

A seguir segue descrição e análise mais detalhada dos passos realizados na criação do *software*.

[27] http://www.nightmarejs.org/

A solução foi implementada na linguagem C#, recorrendo ao uso da ferramenta Visual Studio 2019. Foi necessário instalar a *framework* Selenium e fazer o *download* do WebDriver *(*mais exatamente o Google Chrome Driver*)* que vai permitir simular o comportamento do utilizador final, a abrir manualmente o navegador *Web*, e posteriormente simular outras ações.

Após esta parte preparatória terminada, segue a parte que corresponde à programação do *Web scraper*.

- Para aceder aos *sites* do cliente da empresa exemplo e do concorrente FlixBus, foi necessário criar uma instância do ChromeDriver e indicar os endereços dos *sites* pretendidos.

  Ex:
  ```
  private static IWebDriver driver =
  new ChromeDriver(Directory.GetCurrentDirectory());
  driver.Navigate().GoToUrl("https://www.flixbus.pt/");
  ```

- No *site* da FlixBus é necessário aceitar o serviço de *cookies* que aparece no formato de *Pop-up*, clicando num botão de confirmação. Então, o *Web scraper* tem de realizar os seguintes passos: localizar o botão e de seguida simular o *click* nesse botão.
  Para localizar o elemento é necessário encontrar o seu XPath (usado para encontrar a localização de qualquer elemento numa página *Web).*

  Ex:
  ```
  var cookie = driver.FindElement(By.XPath("/html/body/…/div[2]/button"));
  cookie.Click();
  ```

- Próximo passo é simular a entrada de dados que se pretendem inserir, a fim de simular a pesquisa de uma viagem. Os primeiros dados que se pretendem inserir são a "Origem" e o "Destino".
  O processo é semelhante ao descrito no ponto acima, mas com algumas diferenças:
  - *(i)* localizar as caixas de *input* de texto;
  - *(ii)* simular o *click* em cima delas;
  - *(iii)* inserir a informação pretendida, tendo em conta as sugestões que normalmente aparecem para a palavra introduzida, e a necessidade de localizar e selecionar a sugestão correta.

  Ex:
  ```
  var origemFLIX = driver.FindElement(By.ClassName("flix-input__field"));
  origemFLIX.Click();
  origemFLIX.SendKeys(origem_destino[0]);
  origemFLIX.Click();
  driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(2);
  var clickTextoOrigemFLIX =
  driver.FindElement(By.ClassName("Option__cityName___1c6ZH"));
  clickTextoOrigemFLIX.Click();
  ```

- Outra informação que é necessário introduzir é o tipo de cliente. Na maioria dos *sites*, incluindo o da FlixBus, alguns campos podem estar preenchidos com informação pré-definida, para acelerar o processo de compra/consulta.

No caso da FlixBus isto acontece no campo que corresponde ao tipo de passageiro, em que são apenas disponibilizados três tipos de bilhetes, dispostos sempre pela mesma ordem, e tendo sempre o mesmo tipo pré-selecionado, o tipo "Adulto", com quantidade de bilhete "um". Se for necessário simular a pesquisa para o tipo de bilhete "Criança", por exemplo, é necessário desmarcar a opção pré-selecionada, decrementando o número de bilhete do tipo "Adulto", de forma a deixar o campo "limpo", e de seguida incrementar a quantidade de bilhetes no campo que se refere ao tipo de cliente "Criança", ou outro tipo pretendido.

Ex:

```
var tipoPassageiro = driver.FindElement(By.XPath("//*[@id='search-mask-
component']/div/div[4]/div/div[1]/div/div/input"));
tipoPassageiro.Click();
var descelecionar = driver.FindElement(By.XPath("//*[@id='search-mask-
component']/div/div[4]/div/div[2]/div/div[2]/div/div[2]/…"));
descelecionar.Click();
if (origem_destino[2].Equals("Crianças"))
{
       var selectCriancas = driver.FindElement(By.XPath("//*[@id='search-mask-
       component']/div/div[4]/div/div[2]/div/div[2]/div/div[2]/…"));
       selectCriancas.Click();
}
```

- Após preencher os campos com informação necessária e localizar o botão "Procurar" e efetuar a pesquisa, resta identificar, nos resultados obtidos, a informação relevante que se pretende guardar.
  Neste caso, relevante para a comparação dos preços é saber informação relativamente ao número de rotas existentes para o dia, qual a hora de partida, a hora de chegada do autocarro e o preço do bilhete.
  Para obter os dados, o processo é semelhante, com a diferença de que é necessário obter a informação de todos os elementos (rotas) da lista que resultou da procura e não apenas de um único elemento específico. Outra diferença é que a informação no *site* relativamente às rotas disponíveis é dinâmica, visto que os percursos desaparecem da lista quando atingem a lotação máxima.
  Para identificar todas as rotas e os seus elementos (hora de partida, hora de chegada e preço) para extração, é necessário realizar o seguinte processo para cada elemento (exemplo para extrair o preço de todos as rotas):
  - *(i)* localizar e extrair XPath do preço da primeira e da segunda rota da lista, de entre todos os resultados apresentados;
  - *(ii)* comparar os XPath extraídos, localizar a diferença entre os dois caminhos e retirá-la, obtendo deste modo um caminho generalizado para extrair o preço de todas as rotas. O mesmo processo aplica-se para os outros elementos.

  Ex:
  Tendo o seguinte XPath para o preço da primeira rota da lista:
  //*[@id="search-results-component"]/div/div[8]/div/div/div/div[4]/**div[1]**/div/div/div[2]/div[1]/div

E tendo o seguinte XPath para o preço da rota a seguir:
//*[@id="search-results-component"]/div/div[8]/div/div/div/div[4]/**div[2]**/div/div/div[2]/div[1]/div

A única diferença que se pode observar entre ambos é o preço da primeira rota localizar-se no "div" de cima ("/div[1]/") e o segundo no de baixo ("/div[2]/"). A forma de generalizar o XPath de forma a obter a informação de todas as linhas é tão simples quanto retirar o valor que indica o número da linha do elemento "[1]" deixando apenas "/div/", deste modo obteremos os preços de todos os elementos. Após ter o XPath generalizado e usar o método que procura múltiplos elementos, este irá devolver os preços de todas as rotas:

Ex:
```
var precosFLIX = FindElements(By.Xpath("//*[@id='search-results-component']/div/div[8]/div/div/div/div[4]/div/div/div/div[2]/div[1]/div"));
```

- A última etapa consiste em guardar a informação obtida num ficheiro ".txt" para posteriormente ser possível visualizar e analisar todos os dados. Para isso, é necessário apenas criar um ficheiro local com uma determinada localização e escrever nesse ficheiro o conteúdo da nossa variável que contém os múltiplos elementos extraídos da página *Web*.

  Ex:
```
using (System.IO.StreamWriter file =
new System.IO.StreamWriter(@"C:\Users\...\PrecosFLIX.txt"))
{
   file.WriteLine("FlixBus" + "\n");
   foreach (var preco in precosFLIX)
   {
       file.WriteLine(Preco: " + preco.Text);
   }
}
```

  De seguida, é necessário percorrer as variáveis que contêm o resto da informação e juntar ao ficheiro.

Abaixo segue a representação da informação real recolhida dos dois *sites* (para o mesmo dia). Os dados inseridos foram:

- Origem: Lisboa;
- Destino: Porto;
- Tipo de passageiro: Adulto.

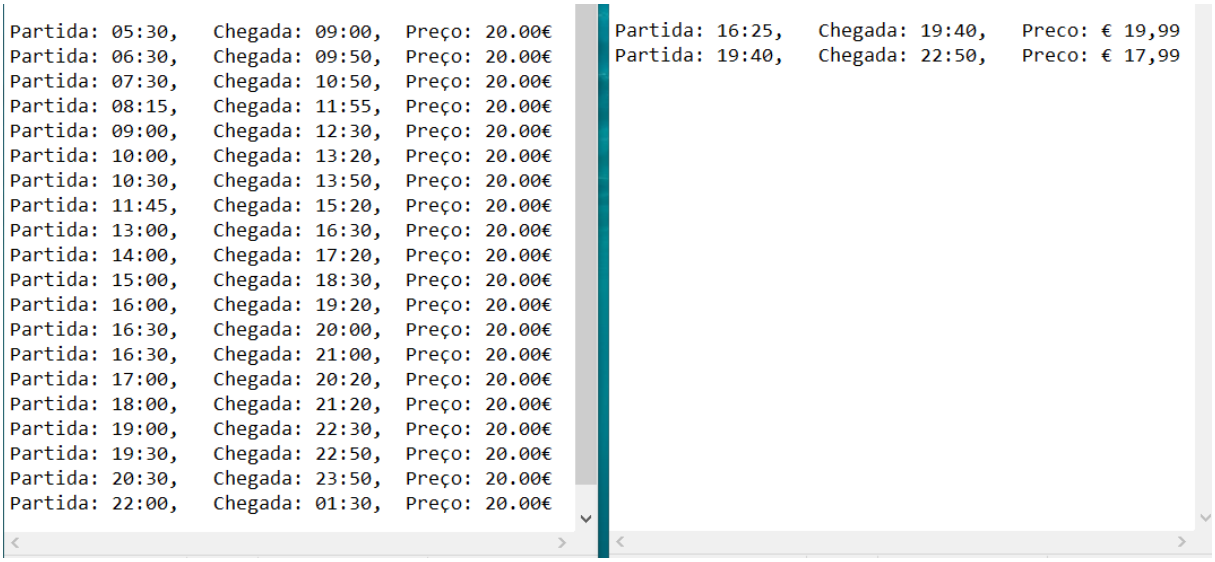

Figura 6 - Dados extraídos com Web scraper construído – dados da empresa exemplo e de empresa concorrente

Embora construído para efeitos de exemplificação, este *Web scrape*r, muito simples, já permite poupar bastante tempo na pesquisa e monitorização de preços dos concorrentes – com um simples *click*, em segundos é possível obter ficheiro com a informação sobre as rotas dos concorrentes.

6.3.1 Automatização, dificuldades e melhorias

Nesta secção serão descritas alterações inseridas no *software* para automatizar o processo de simulação de procura nos *sites*, dificuldades que surgiram ao longo da implementação e melhorias a introduzir nas próximas etapas.

- **Automatização**

  Para automatizar o *software* nesta primeira etapa de implementação, e para que a informação não seja *hard-coded*, foi criado um ficheiro com extensão ".txt". O papel do ficheiro é servir de lugar de configuração dos dados de entrada, usados para simular a pesquisa nos *sites*.

  Os dados de entrada usados, são a origem, o destino e o tipo de cliente, desta forma ficou determinado que a primeira linha do ficheiro corresponderia à origem, a segunda linha ao destino e a terceira ao tipo de passageiro.

  Esta abordagem facilita e torna mais rápido o processo de extração de dados, tendo em conta que não é necessário andar à procura no código do lugar onde substituir a informação, mas sim apenas aceder ao ficheiro e inserir os dados pretendidos.

  Este é apenas um exemplo de critérios que se podem usar para realizar a pesquisa, no entanto, tudo é adaptável às necessidades de cada empresa, podem ser adicionados novos critérios ou usados outros completamente diferentes.

- **Dificuldades**

  Embora à primeira vista o código pareça ser simples e sem complicações, no entanto surgiram algumas dificuldades no processo de implementação.

  O primeiro *site* ao qual foi feito o *Web scraping* foi o dos clientes da empresa exemplo. O processo foi relativamente fácil e rápido visto que os campos estavam todos bem identificados no HTML e todos os componentes tinham nomes diferentes, o que facilitou a sua localização.

  O *site* dos concorrentes (FlixBus) tem o sistema de preços muito avançado e arrojado, o que levou a que fosse escolhido como o concorrente a monitorizar, no entanto tem também algumas outras características que vieram dificultar o processo de recolha automática de informação.

  Por exemplo, *(i)* tem o serviço de *cookies*, no formato de *Pop-up*, que obriga o cliente a aceitá-lo – houve que conseguir identificá-lo e realizar a confirmação, *(ii)* tem muitos campos com os mesmos nomes, o que obrigou a conseguir a identificar a melhor forma de os identificar, *(iii)* pelo facto de ser muito rápido, o processo de *Web scraping* pode perder alguma informação em situações como, por exemplo, no caso da FlixBus, que o *software* já queria saltar para a introdução dos dados no campo "Origem" enquanto o campo da confirmação dos *cookies* era procurado e identificado – para não haver concorrência entre as ações, foi necessário recorrer ao método "Timeouts()" definido para o uso em conjunto com o *driver*. Esta técnica não foi apenas usada para o caso referido, mas também noutras situações como, por exemplo, entre identificação de dois campos diferentes, para estes não entrarem em concorrência.

- **Melhorias a inserir numa próxima etapa**

  O *software* que se encontra implementado no momento da escrita desta dissertação é apenas um primeiro passo para um *Web scraping* complexo e sofisticado. Este ainda não tem a autonomia desejada nem a complexidade da informação extraída, havendo ainda muito por onde evoluir. Uma das melhorias a introduzir numa próxima etapa é a possibilidade de escolher o dia para o qual se pretende fazer a simulação da pesquisa – atualmente a pesquisa é simulada sempre para o dia corrente.

  Outra melhoria a acrescentar é tornar o programa autónomo ao ponto de ser possível configurar a frequência com que as pesquisas são simuladas. Por fim, mais uma alteração que irá facilitar o processo de análise, após extração dos dados, é a do formato dos dados extraídos (por exemplo ficheiros CSV que podem ser abertos em Excel e logo editados).

Um grande desafio na construção de um *Web scraper* é lidar com a diversidade das páginas *Web*. Apesar de haver páginas com estruturas ou padrões semelhantes, todas acabam por ter uma estrutura única. Desta forma, torna-se necessária a criação de um *Web scraper* personalizado para cada página que se quer consultar. Pode haver blocos de código, como os de análise e tratamento de dados, que são reutilizáveis, mas os blocos de consulta e extração de dados têm de ser adaptados de acordo com a estrutura de cada *site*.

Outro desafio é acompanhar a atualização das páginas *Web*, pois estas encontram-se em atualizações periódicas, com objetivo de melhorar a experiência do utilizador, podendo levar a alterações na estrutura da página e, consequentemente, a falhas no momento de procura de uma determinada informação. Porém, estes problemas costumam ser fáceis de resolver, necessitando de ajustes mínimos no código.

Em termos de automatização do processo, é possível implementar um simples *scheduler*, a fim de executar o *Web scraper* em segundo plano, agendando uma hora/dia para a sua execução.

6.4 Implementação do *software* de *Web scraping* com Selenium para fator estado do tempo

Para obter o fator correspondente ao estado do tempo, recorreu-se ao mesmo processo do fator preços dos concorrentes, explicado na secção anterior. A informação relevante a ser extraída do *site* de meteorologia é a previsão do tempo, para uma determinada localidade, para os próximos dez dias, contendo informação relativamente às temperaturas mínima e máxima de cada dia.

A lógica, o processo de extração e as ferramentas utilizadas são exatamente as mesmas da secção anterior, sendo por isso nesta secção apenas apresentado o resultado da extração realizada do *site* do Instituto Português do Mar e da Atmosfera (IPMA).

Os dados extraídos são apresentados no formato ilustrado na Figura 7. O formato é simples e permite com facilidade localizar a informação necessária quando esta for utilizada para construir regras de preços dinâmicos.

Em termos de dificuldades, não surgiu nenhuma complicação ou imprevisto na escrita do código, foi bastante simples e rápido, ocupando o programa completo apenas umas quantas linhas.

Em termos de melhorias, no trabalho futuro, poderia ser acrescentada a automaticidade ao programa, a fim de este fazer as tarefas por conta própria numa hora pré-definida.

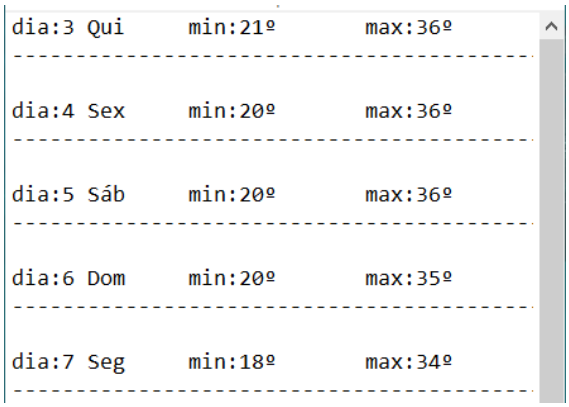

Figura 7 - Dados extraídos com Web scraper construído – dados do estado de tempo

6.5 Ferramenta para obter fator eventos locais

Atualmente existem bastantes *sites* que reúnem informação de todos os eventos a decorrer nos próximos tempos no país. Devido a isso, a tarefa de obter a informação torna-se bastante simples. É possível criar um *Web scraper* simples tal como foi feito nas secções anteriores ou recorrer a uma *framework* comercializada.

Para diversificar um pouco o processo de extração de dados, considerou-se interessante analisar e aplicar outra técnica de extração. Optou-se então por extrair os dados com uma *framework* comercializada – WebScraper.io.

Os dados relevantes para a extração acerca dos eventos locais, são: o local onde o evento irá decorrer, o seu nome e a data.

❖ **WebScraper.io** [28]

WebScraper é uma ferramenta de extensão do Chrome, que permite recolher dados de qualquer *site*, diretamente dentro do Chrome, sem escrever código adicional. O conceito da aplicação é muito simples: criar um seletor para cada elemento (que irá conter configurações de extração para cada elemento), dar-lhe um nome, selecionar o tipo de elemento que se quer extrair e por fim apontar para o elemento no *site* e selecioná-lo.

Tendo como exemplo um *site* que disponibiliza uma lista de eventos, e como objetivo a extração dos nomes de todos os eventos, basta criar um seletor com designação "nome do evento", selecionar o tipo (neste caso "texto"), e selecionar o nome de um evento na página.

Para a ferramenta identificar os nomes de todos os eventos da página é necessário ativar a funcionalidade "Multiple" (que permite selecionar mais que um elemento) e após isso selecionar o título de um evento que esteja ao lado. Desta forma a ferramenta irá automaticamente assumir que os nomes dos eventos a seguir também são para a extração e identificá-los.

A seguir, basta repetir o processo para o resto de informação. No final, após selecionarmos a opção "*scrape*" que extrai a informação, esta fica arrumada numa espécie de tabela, em formato CSV, que permitirá a análise dos dados.

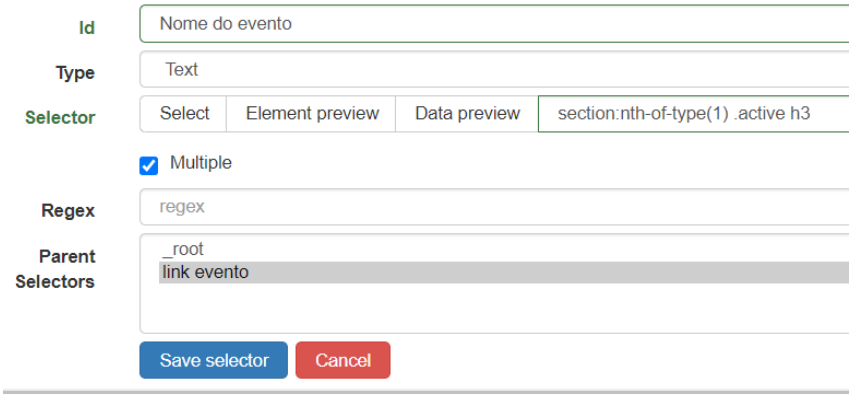

Figura 8 - Campos de configuração para a extração de dados com WebScraper.io

Formato final dos dados extraídos:

Nome	Data	Local
Concerto de Homenagem a Bernardo Sassetti	05 Set 2020 - 18:30	Coimbra, Quinta das Lágrimas
MEAJAZZ 2020	04 Set 2020 » 05 Set 2020	Mealhada - Mealhada
O Drive in volta a estar na moda com música e cinema em Lagos	01 Ago 2020 » 05 Set 2020	Lagos
"Sonho Europeu: Obras da Coleção Norlinda e José Lima"	14 Dez 2019 » 05 Set 2020	Centro de Cultura Contemporânea de Castelo Branco
DESCONCERTO	01 Set 2020 » 05 Set 2020	Coliseu de Lisboa - Lisboa

Figura 9 - Formato de dados de eventos extraídos com WebScraper.io

[28] https://webscraper.io/

Comparando esta técnica de extração com a de escrita de um *Web scraper* (simples) específico, esta acabou por ser mais demorada e mais maçadora. Tentou-se usar esta ferramenta em vários *sites* de eventos diferentes, no entanto, apenas à terceira tentativa foi possível extrair todos os dados desejados. A dificuldade consistiu em obter a data dos eventos, pois várias vezes, dependendo da data do evento, o tipo de caixa mudava, de evento para evento (caso o evento ocorra no dia de hoje, a data é apresentada num tipo de caixa, que difere do tipo de caixa usado quando o evento ainda está por ocorrer). Procurar *sites* em que a ferramenta funcione bem é completamente inconveniente, visto que poderá ser preciso aceder a um *site* específico, que por azar não sincronize bem com a ferramenta.

6.6 Como obter fator períodos de maior procura

Para obter o fator que corresponde aos períodos de maior procura, é necessário estudar e observar bem o mercado e posteriormente desenvolver algoritmos complexos que sejam capazes, de forma autónoma, de prever os períodos de maior e menor procura para determinadas origens e destinos. Além destas abordagens, existe uma outra, que é realizar o levantamento de dados estatísticos do mercado pois normalmente todas as empresas têm a informação armazenada em atualizações regulares. Esta última alternativa foi aqui adotada para uma primeira abordagem ao cálculo de preços dinâmicos.

Os dados a obter podem ser os seguintes: origens/destinos de maior procura, dias de maior procura, horários em que existe maior movimento, destinos que têm mais procura em determinadas estações do ano.

Observando as regras e os preços que são aplicados atualmente pelos clientes da empresa exemplo, é possível extrair as seguintes informações:

- Quarta-feira e terça-feira – dias de menor procura, com menos vendas para qualquer par origem e destino;
- Sexta-feira – dia de maior procura;
- Domingo à tarde – tem uma procura mais elevada;
- Rotas de maior procura: Porto↔Lisboa, Lisboa↔Fátima, Lisboa↔Braga, Lisboa↔Bragança**;**
- Horas de maior procura: de segunda-feira a sexta-feira: 17h-19h e das 6h-9h**;**

6.7 Esboço de arquitetura para *Yield Management* Dinâmico e análise dos seus módulos

Nesta secção será definida e proposta uma nova arquitetura de *Yield Management*, com integração das novas ferramentas escolhidas e analisadas nas secções anteriores. Além disso, será também realizada a análise dos novos módulos que irão compor a nova arquitetura.

A maior parte dos módulos já se encontra parcialmente implementada, outros apenas criados para efeitos de exemplificação. No entanto, todos de forma isolada, havendo necessidade de, no futuro próximo, interligar todos os módulos de forma a funcionarem como um único mecanismo.

Abaixo está representada aquela que se sugere ser a nova arquitetura funcional, mostrando a estrutura do sistema de *software* que se visa construir numa próxima etapa.

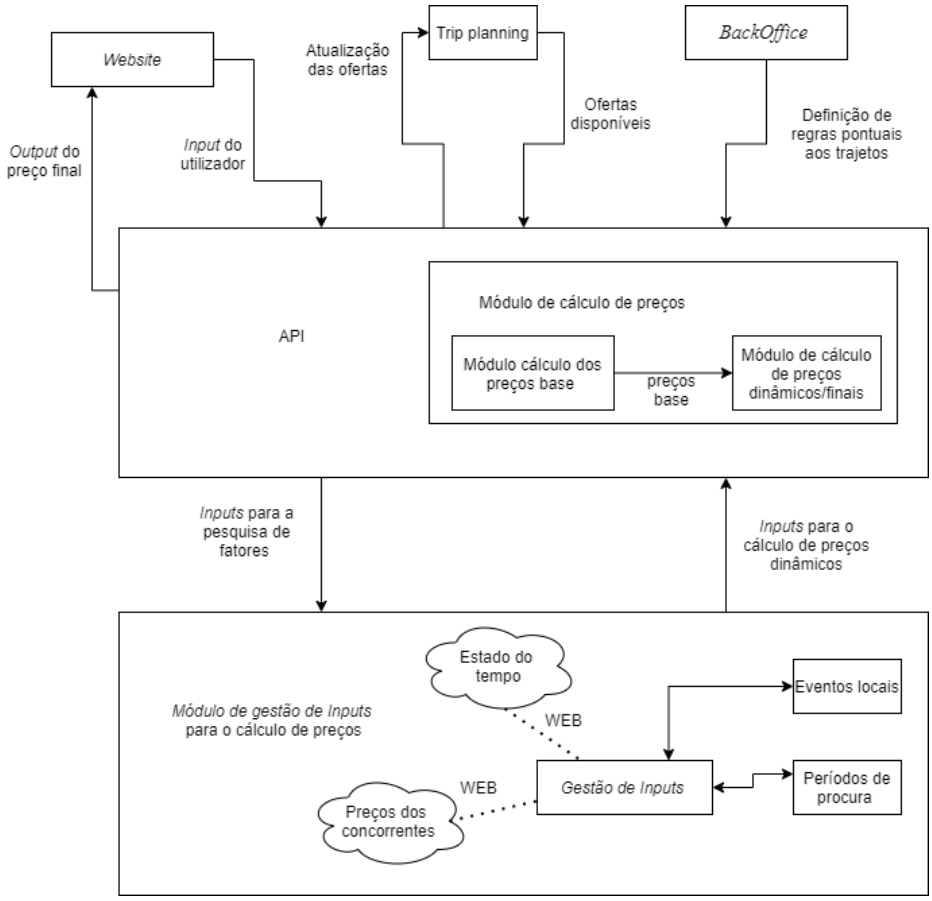

Figura 10 - Arquitetura sugerida para Yield Management dinâmico

6.7.1 Comunicação entre módulos

Para perceber a interação e comunicação entre todos os módulos, primeiro será descrito o processo anterior ao da arquitetura sugerida e de seguida o processo que se pretende implementar a fim de otimizar os preços.

- Processo de cálculo de preços anterior ao da arquitetura implementada
  De acordo com o *input* inserido pelo utilizador no *website*, primeiro é feita uma pesquisa no módulo das ofertas para verificar quantas e que tipo de ofertas existem disponíveis – estas vêm sempre com preços base fixos que, posteriormente, de acordo com determinadas regras já definidas, são ajustados. Todas as regras de preços, promoções e regras de trajetos encontram-se definidas na base de dados e, caso exista alguma regra definida para a pesquisa realizada, esta é aplicada sobre o preço. Por fim, os preços calculados com todas as alterações são devolvidos ao utilizador final.

- Processo de cálculo de preços com a nova arquitetura

  Na nova arquitetura, as primeiras etapas que correspondem à inserção do *input* pelo utilizador e à pesquisa de ofertas disponíveis de acordo com o mesmo, continuam a existir da mesma forma. Após se obterem as ofertas disponíveis do módulo *Trip planning* é que começam a ser inseridas as alterações. O próximo passo consiste em obter fatores para o cálculo do novo preço de acordo com a pesquisa do utilizador – esta etapa acontece no módulo de gestão de *inputs*, visto que é aí que são obtidos novos fatores (*inputs*) que irão influenciar o preço final apresentado. Após ser recolhida toda a informação, esta é recebida pelo módulo de cálculo de preço final/dinâmico. Neste último, aplica-se o sistema baseado em regras para criar a lógica de cálculo, são analisados todos os fatores recebidos e regras criadas no *BackOffice* e tomadas as decisões relativamente ao preço. Por fim o preço calculado é apresentado ao utilizador final num *website* ou quiosque por exemplo.

6.8 Formato de fatores a serem usados no cálculo de preços dinâmicos

Antes de os dados serem recebidos pelo módulo de cálculo de preços e usados como input na lógica de cálculo de preços dinâmicos, é necessário perceber em que formato será recebida e tratada toda a informação:

- **Fator preço dos concorrentes**

  O formato de representação de dados numa das secções anteriores foi em ficheiro de texto com extensão ".txt", visto que para análise visual é mais do que suficiente e, além disso, é rápido e fácil de interpretar.

  No entanto, para posterior utilização e leitura destes ficheiros no cálculo de preços, o mais adequado seria colocar o conteúdo dos ficheiros estruturado em formato JSON, visto que este formato é interpretável por máquinas e é de fácil processamento e manuseamento, enquanto que para tratar o ficheiro em formato de texto simples seria necessário criar um sistema de separação de variáveis (como por exemplo, separar a informação com virgulas) para poder filtrar e extrai a informação necessária.

  Tendo os dados estruturados em formato JSON, após a sua leitura os dados poderiam ser guardados no programa numa única variável, como por exemplo *Array* Multidimensional (que é um *Array* que contém mais de uma linha para guardar os dados, em C#).

  Normalmente, após o utilizador fazer a pesquisa no *website* para ver a disponibilidade de bilhetes para um determinado dia e um determinado par origem e destino, tem como *output* vários resultados correspondentes às viagens que ocorrem a diferentes horas e talvez até a diferentes preços; neste caso, um Array Multidimensional (mais especificamente com duas dimensões) seria a melhor opção para guardar a informação. É a melhor opção, dado que cada elemento de *Array* Multidimensional é uma linha de dados.

  Neste caso, cada linha iria corresponder a cada viagem disponível, contendo horários de partida, horário de chegada e preço.

Ex:

```
string [ , ] viagens = new string[ , 3] {
{"05:30", "09:00", "20.00"},
{"06:30", "09:50", "20.00"},
{"07:30", "10:50", "20.00"},
…
};
```

- **Fator estado do tempo**

  Este fator será tratado da mesma forma que o anterior: os dados no ficheiro serão estruturados no formato JSON e posteriormente transformados no formato de *Array*.

  A pesquisa do utilizador na consulta dos bilhetes é normalmente realizada para uma determinada data; desta forma apenas é necessário obter o estado do tempo para essa data específica, contento a informação das temperaturas mínima e máxima para esse dia.

  O formato dos dados poderia ser representado da seguinte maneira.

  Ex:

```
int [] tempo = {21, 36};
```

- **Fator eventos locais**

  Para esta informação, o relevante é saber se existem, ou não, eventos a decorrer no destino selecionado pelo cliente, na data selecionada.

  Após análise dos eventos que foram extraídos de *sites* de eventos, chegou-se à conclusão de que a maioria não tem impacto ou influência praticamente nenhuns na compra de viagens para os destinos onde estes eventos se localizam. Isto acontece tendo em conta que normalmente são espetáculos de pequenas dimensões e muitos frequentados por grupos pequenos de público específico. Sendo assim, o que faria mais sentido seria a própria empresa criar um documento que contém os eventos mais importantes a decorrer em Portugal, os mais relevantes e com maior impacto para o seu negócio, como por exemplo, os festivais musicais: NOS Alive, Super Bock Super Rock, Sumol Summer Fest, entre outros. Sabendo que estes espetáculos têm as datas definidas com aproximadamente um ano de antecedência, seria fácil e rápido criar um documento todos os anos, ou criar uma vez e depois apenas atualizar, com a informação relevante e personalizada para a empresa.

  O formato poderia ser o mesmo utilizado no fator preços dos concorrentes, guardando em cada linha informação de cada evento, contendo o nome do evento, a data de início, a data de fim e o local.

  Ex:

```
string [ , ] eventos = new string[ , 4] {
{"Sumol Summer Fest", "03-07-2020", "04-07-2020", "Ericeira"},
{"MEO Sudoeste", "04-08-2020", "08-08-2020", "Zambujeira"},
{"Super Bock Super Rock", "18-07-2020", "20-07-2020", "Meco"},
{"NOS Alive", "09-07-2020", "11-07-2020", "Lisboa"},
            …
};
```

- **Fator períodos de maior procura**
  Após a análise deste fator e do tipo de informação que este traz (numa das secções anteriores), foi decidido utilizar a informação na construção das próprias regras do tipo "se-então". Como o fator períodos de maior procura implica, na maior parte das vezes, analisar dias, horas ou origens/destinos de maior procura, pode ser criado um ficheiro/método/excerto de código, em que todos os casos relativos a esse fator sejam tratados separadamente e, posteriormente, inserir este fragmento no código restante.

Todos os fatores mencionados serão utilizados na secção a seguir no cálculo de preço dinâmico.

6.9 Regras de cálculo de preços dinâmicos

Com base no estudo realizado nos capítulos anteriores, serão agora sugeridas algumas regras de preços que poderão ser inseridas no módulo de cálculo de preços dinâmicos. As regras serão apresentadas recorrendo a um algoritmo muito específico criado para efeitos de exemplificação. O objetivo da criação do algoritmo exemplo é perceber os passos gerais necessários na construção de regras, e formar uma ideia de como estas poderiam ser aplicadas e interligadas.

O processo de criação de regras de preços dinâmicas e inserção das mesmas no sistema, de forma profissional, é uma tarefa bastante complexa que requer conhecimento de diferentes áreas científicas e envolve no seu processo de criação a participação de diferentes indivíduos devido a essa complexidade. As regras que serão apresentadas a seguir são apenas meros exemplos, simples e com aplicação específica.

O sistema utilizado na criação de regras será o sistema baseado em regras no formato "se-então", ou seja, com base no conhecimento já adquirido, a fim de atribuir o melhor e o mais vantajoso preço aos bilhetes a serem disponibilizados ao utilizador final.

Como já foi referido, a aplicação *BackOffice* contém regras definidas para serem aplicadas nos preços. Na solução que será apresentada a seguir, foi decidido que esta aplicação irá apenas conter regras prioritárias no cálculo de preços. Como o processo não pode ser completamente autónomo e independente, tem de existir sempre a possibilidade de gerir e corrigir o sistema manualmente. Além disso, pode existir necessidade de inserir regras pontuais ou alterações nos preços independentes do cálculo que o sistema dinâmico efetua. Assim sendo, primeiro irá ser realizada a consulta ao *BackOffice*, com o objetivo de verificar se existe alguma regra definida para a pesquisa efetuada pelo utilizador. Caso exista, esta será aplicada sobre o preço; caso não exista, o preço irá ser calculado de forma dinâmica.

Antes de começar com a lógica e a construção de regras de cálculo dinâmicas, é necessário ter toda a informação pronta a ser utilizada e consultada. Será necessária a seguinte informação arrumada e disponível para a consulta, preferencialmente em formato JSON. A maior parte da informação (externa) obtém-se através dos dados de pesquisa introduzidos pelo utilizador, recorrendo aos módulos mencionados na secção anterior.

- Todas as tarifas base disponíveis (da empresa exemplo), para a pesquisa realizada pelo utilizador;
- Todas as tarifas disponíveis (dos concorrentes), para a pesquisa realizada pelo utilizador;
- Informação relativa a destinos de maior procura no verão e outra relativamente aos do inverno;
- Informações (precipitação e classificação do estado do tempo) do estado do tempo na data para a qual foi realizada a pesquisa do bilhete pelo utilizador;

- Classificações do estado de tempo existentes;
- Datas dos eventos importantes a decorrer;
- Horários de maior procura;
- Dias de maior procura.

Primeiro é preciso ter em conta que normalmente existem várias tarifas disponíveis a serem exibidas ao cliente, após este efetuar a pesquisa. Assim sendo, é necessário ajustar o valor de cada uma das tarifas antes de serem apresentadas.

O bloco de código principal será dividido em quatro partes principais. Na primeira parte, é verificado se existe uma regra definida no *BackOffice* para a tarifa a ser analisada. Caso exista, a regra é aplicada sobre o preço da tarifa, que é atualizada e guardada numa lista que contém todas as tarifas a serem apresentadas ao utilizador final. Caso contrário, na segunda parte, serão analisados os ficheiros que contêm a informação dos novos fatores introduzidos (referidos nas secções anteriores), que terão sido obtidos com base na pesquisa realizada pelo utilizador.

Após a análise de todos os fatores, a informação guardada nas respetivas variáveis será utilizada na terceira etapa, a fim de tomar decisões relativamente ao preço de acordo com o resultado da análise dos fatores.

Por fim, na última parte (quarta), será analisado o preço calculado com base nos fatores, a lotação do autocarro, a data de reserva e outras variáveis de decisão (que contêm a informação da quantidade de desconto a aplicar). Com base nesta informação serão aplicadas regras sobre os preços, a fim de calcular o preço final da tarifa. A tarifa com o preço atualizado será guardada tal como na primeira etapa, numa lista que contém todas as tarifas atualizadas.

Abaixo segue a descrição das etapas do bloco principal em pseudocódigo.

- **calculoDoPreco**

  Dados de *input*: Dados de pesquisa inseridos pelo utilizador (dadosPesquisa)
  Valor de *output*: Lista de tarifas com preços alterados (tarifasFinais)

```
List<tarifa> tarifasFinais = new List<tarifa>();
precoComFatores = 0;
evento = existeEvento(dadosPesquisa);
precipitacao = getPrecipitação();
estadoDotempo = getEstadoTempo();
procura = 0;
precoConcorrente = 0;

para cada tarifa da lista de tarifasBase{
       if (existeRegraBackOffice(tarifa, dadosPesquisa)) {
              tarifa.preco = aplicarRegraNoPreco(tarifa, dadosPesquisa);
       }
       else {
              procura = getProcura(tarifa);
              precoConcorrente = getPrecoConcorrente(tarifa);
              precoComFatores = calculoPrecoComFatores(evento, precipitacao,
              estadoDotempo, procura, precoConcorrente, tarifa);
```

```
                tarifa.preco = calculoPrecoFinal(precoComFatores, tarifa,
                dadosPesquisa);
            }
        tarifasFinais.Add(tarifa);
    }
```

Para perceber melhor os métodos da segunda parte e a informação que eles devolvem, estes serão explicados sucintamente recorrendo às variáveis definidas no bloco principal:

- **"evento"** – contém informação relativamente aos eventos – se existe, ou não, algum evento na data e no destino procurado pelo utilizador;
- **"precipitacao"** – guarda o valor da precipitação para a data que o utilizador efetuou a pesquisa do bilhete;
- **"estadoDotempo"** – guarda a descrição do estado do tempo para o dia em que é efetuada a pesquisa do bilhete (que posteriormente será analisada e classificada com valores a ponderar);
- **"procura"** – guarda um valor inteiro definido, recorrendo a varias verificações: se o dia para o qual foi realizada a pesquisa se encontra na lista de dias de maior procura, se o destino se encontra na lista de destinos de maior procura e se o horário da tarifa a ser analisada é um horário de maior procura, guardando no campo o valor entre zero e dois (em que zero corresponde a nenhuma característica de maior procura e dois a todas as características de maior procura);
- **"precoConcorrente"** – guarda o preço do concorrente – caso exista uma viagem com o horário semelhante, é guardado o valor dessa viagem, caso não exista nenhum horário semelhante, é guardado o valor da média de preços de todas as tarifas para aquela pesquisa.

Foi considerado interessante apresentar apenas dois métodos do bloco principal. Estes contêm a lógica principal a ser aplicada nos preços. As variáveis que não foram passadas aos métodos no bloco principal, são variáveis definidas fora do código, num ambiente configurável, como por exemplo, ficheiros, de modo a que seja possível controlar e alterar, por exemplo, valor de desconto ("descontoTarifa") ou o número de dias de antecedência a partir de quantos se quer aplicar o desconto, entre outras.

- **calculoPrecoComFatores**

  Dados de *input*:
  - informação sobre o evento (evento)
  - valor da precipitação (precipitacao)
  - estado do tempo (estadoDotempo)
  - valor da procura (procura)
  - preço do concorrente (precoConcorrente)
  - dados sobre a carreira (precoTarifa, destino)
  - precipitação mínima aceitável (precipitacaoMin)
  - desconto a aplicar sobre a tarifa (descontoTarifa)
  - valor mínimo para a variável "procura" (procuraMin)

  Valor de *output*: preço calculado com fatores (preçoComFatoresFinal)

```
pf = precoTarifa; // pf – preço com fatores aplicados
if (evento) { //Se existe evento
       pf = pf + precoTarifa*descontoTarifa;
} else {
  //Se não existe evento
       pf = pf - precoTarifa*descontoTarifa;
}
if (precipitacao < precipitacaoMin && !tempoMau(estadoDotempo) &&
destinoVerão(destino)) {//bom tempo e destino de verão
    pf = pf + precoTarifa*descontoTarifa;
}
if (precipitacao >= precipitacaoMin && tempoMau(estadoDotempo) &&
destinoVerão(destino)) {//mau tempo e destino de verão
    pf = pf - precoTarifa*descontoTarifa;
}
if (procura > procuraMin) {
       pf = pf + precoTarifa*descontoTarifa;
}
if (procura < procuraMin) {
    pf = pf - precoTarifa*descontoTarifa;
}
precoComFatoresFinal = pf – (ip*x); //preço com fatores, ajustado de acordo com
preço dos concorrentes

return precoComFatoresFinal;
```

A fim de aproximar o preço com fatores (pf) do preço do concorrente, foi utilizada a seguinte expressão:

$$y = pf - (ip * x)$$

Nesta expressão, o "y" corresponde ao valor final calculado, o "pf" é o preço com fatores aplicados e "x" corresponde à diferença entre o preço "pf" e o preço da tarifa dos concorrentes, e "ip" refere-se à importância percentual (em decimal), que se pretende atribuir à diferença.
Desta forma, a própria empresa consegue decidir o quanto se aproximar do preço do concorrente.

Exemplo de aplicação:

- pf = 22,99;
- precoDoConcorrente = 14,99;
- x = 22,99 – 14,99 = 8;
- ip = 0,55.

$$y = 22{,}99 - (0{,}55 * 8) = 18{,}59$$

❖ **calculoPrecoFinal**

Dados de *input*:

- preço calculado com fatores (pf)
- dados sobre a carreira (totalLugares, lugaresDisponiveis)
- dados da pesquisa do utilizador (dataViagem, ...)
- taxa estabelecida que indica uma alta ocupação (taxaOcupacaoAlta)
- taxa estabelecida que indica uma baixa ocupação (taxaOcupacaoBaixa),
- dias de antecedência com que se compra o bilhete (diasAntecedencia)
- preço mínimo possível do bilhete (precoMinimo)
- desconto a atribuir quando sobram pouco lugares disponíveis no autocarro (descontoPoucosLugares)
- desconto a atribuir quando sobram muitos lugares disponíveis no autocarro (descontoMuitosLugares)

Valor de *output*: preço final (precoFinal)

```
precoFinal = 0;
if(lugaresDisponiveis <= totalLugares*taxaOcupacaoBaixa && (|dataViagem –
dataAtual|) > diasAntecedencia){
       precoFinal = pf + pf*descontoPoucosLugares;
}
if(lugaresDisponiveis >= totalLugares*taxaOcupacaoAlta && (|dataViagem -
dataAtual|) <= diasAntecedencia){
       precoFinal = pf - pf*descontoMuitosLugares;
}
if(precoFinal < precoMinimo){
       precoFinal = precoMinimo;
}
return precoFinal;
```

No cálculo de preço final recorre-se ao preço calculado no método "calculoPrecoComFatores", a lotação do autocarro da tarifa em análise, e a diferença de tempo entre a data de pesquisa e a data de partida do bilhete. Deste modo é possível alterar o preço de acordo com a aproximação dos prazos, diminuindo o número de lugares não vendidos. Sendo assim, caso o número de lugares livres seja consideravelmente elevado e a data de partida seja próxima, é aplicado um determinado desconto ao preço do bilhete ("descontoMuitosLugares"). Caso existam poucos lugares e a data não estiver tão próxima, o preço é aumentado de acordo com o valor do "descontoPoucosLugares".

Estas regras de cálculo são apenas um exemplo específico de possíveis passos a seguir e forma de aplicar as regras, a fim de implementar uma possível solução de otimização do preço, que pode ser alterada e melhorada de diversas formas. Os algoritmos que foram apresentados são simples e específicos, e foram escritos para funcionar com determinados fatores exemplificativos (que foram considerados relevantes neste trabalho). No entanto, para se poderem utilizar na empresa profissionalmente, os algoritmos e expressões de cálculo teriam de ser melhorados e generalizados a fim de se poder usar outro tipo de informação no cálculo de preços.

Para esta primeira etapa de introdução aos preços dinâmicos, este processo de cálculo foi considerado uma grande mais valia, pois permitiu perceber como adaptar o preço de cada tarifa individualmente de acordo com as suas características e ajustável a fatores externos.

Capítulo 7

Considerações finais e trabalho futuro

Este capítulo é dividido em duas secções: considerações finais – onde é feita uma reflexão sucinta sobre o trabalho realizado, a fim de compreender a sua magnitude e importância; trabalho futuro – onde serão descritos planos futuros e melhorias a introduzir.

7.1 Considerações finais

Este trabalho surgiu com o intuito de modernizar e automatizar o sistema de gestão de rendimentos (*Yield Management*) que se encontra a ser utilizado nas empresas de autocarros de longo curso e, também, fazer o estudo e levantamento de requisitos a fim de criar um plano a introduzir no sistema, com o objetivo de tornar os preços dinâmicos.

Para a modernização, gestão mais rápida e eficiente do sistema de *Yield Management* foi criado o *BackOffice*. Foi realizado o estudo acerca das funcionalidades que seriam relevantes na organização e gestão de regras. Foram implementadas todas as funcionalidades consideradas essenciais (referidas num dos capítulos anteriores), que proporcionam uma fácil gestão de regras de preços já implementadas no sistema e também a criação de novas.

Quanto à parte de preços dinâmicos, foi realizado o estudo acerca de fatores que têm influência na alteração dos preços, selecionando os mais importantes nesta primeira abordagem à estratégia. Para que seja possível a futura inserção da estratégia no sistema, foi realizado o levantamento de ferramentas necessárias para obtenção dos fatores e construídos *softwares* simples, para efeitos de exemplificação, de obtenção de informação de *sites* externos. Foram também sugeridas regras de aplicação e inserção dos novos fatores de uma forma dinâmica e diversificada no sistema.

Por fim, com o objetivo de ligar todo o conhecimento, aplicações e ferramentas desenvolvidas, foi criada uma sugestão de arquitetura funcional que se poderá adaptar futuramente e inserir no sistema.

7.2 Trabalho futuro

No que toca ao *BackOffice*, a aplicação encontra-se integrada no sistema da empresa estando, no entanto, em fase de testes – tendo em conta que é uma aplicação que tem uma grande influência nos rendimentos e preços, é necessário continuar a realizar alguns testes exaustivos, com a finalidade de diminuir os riscos e assegurar que tudo se encontra a funcionar de acordo com o especificado.

Em termos de melhorias de preços dinâmicos é possível realizar o estudo mais profundo de regras de cálculo de preços, a fim de perceber que alternativas melhores podem ser sugeridas, como também realizar um estudo mais aprofundado acerca de fatores que influenciam os preços.

Outro ponto que seria interessante desenvolver melhor é o das *frameworks* (*Web scrapers*) que foram criados para efeitos de exemplificação e que, sendo funcionais, poderiam ser aplicadas no trabalho depois de evoluir para se tornarem mais sofisticadas e autónomas.

Por fim, com todo o conhecimento adquirido, criar e integrar todo o sistema de preços dinâmicos no sistema real da empresa. Dessa forma a empresa teria sempre os preços atualizados, tendo em conta o mercado e os concorrentes, e proporcionaria as melhores e mais vantajosas ofertas aos seus clientes.

Capítulo 8

Bibliografia

Almeida, Helder Manuel Goucha Gaspar Pais. *Estratégias de pricing no mercado retalhista: da concorrência a outras variáveis determinantes.* Master's thesis. FEUC, 2013.

Aviv, Yossi e Amit Pazgal. "A partially observed Markov decision process for dynamic pricing." *Management science 51.9* 2005: 1400-1416.

Balkoulis, Elaine M. ""Exploring revenue management in spas: Yield management concepts in the spa industry today." UNLV Theses, Dissertations, Professional Papers, and Capstones. 597, 2007.

Bandalouski, Andrei M., et al. "An overview of revenue management and dynamic pricing models in hotel business." *RAIRO-Operations Research 52.1* 2018: 119-141.

Bertsimas, Dimitris e Phebe Vayanos. "Data-driven learning in dynamic pricing using adaptive optimization." *Optimization Online* 2017.

Besbes, Omar e Assaf Zeevi. "Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms." *Operations Research 57.6* 2009: 1407-1420.

Bi, Wenjie e Mengqi Liu. "Product demand forecasting and dynamic pricing considering consumers' mental accounting and peak-end reference effects." *Journal of Applied Mathematics* 2014.

Board, American Software Testing Qualifications; , German Testing Board;. "Foundation Level Specialist Syllabus - Performance Testing." *International Software Testing Qualifications Board* 2018.

Boer, A.V. den. "Dynamic pricing and learning: historical origins, current research, and new directions." *Surveys in operations research and management science 20.1* 2015: 1-18.

Braden, David J. e Shmuel S. Oren. "Nonlinear Pricing to Produce Information." *Marketing Science 13.3* 1994: 310-326.

Carvalho, Márcio Filipe Alves. *Automatização de Testes de Software*. Relatório de Estágio. ISEC, 2010.

Chase, Richard B. e Sheryl E. Kimes. "The strategic levers of yield management." *Journal of service research 1.2* 1998: 156-166.

Chevalier, Judith A., Anil K Kashyap e Peter E Rossi. "Why don't prices rise during periods of peak demand? Evidence from scanner data." *American Economic Review 93.1* 2003: 15-37.

Dai, Bo. *The Impact of Perceived Price Fairness of Dynamic Pricing on Customer Satisfaction and Behavioral Intentions: The Moderating Role of Customer Loyalty*. Doctoral dissertation. Auburn University, 2010.

Deksnyte, Indre e Prof Zigmas Lydeka. "Dynamic pricing and its forming factors." *International Journal of Business and Social Science 3.23* 2012.

Dimicco, Joan Morris, Pattie Maes e Amy Greenwald. "Learning curve: A simulation-based approach to dynamic pricing." *Electronic Commerce Research 3.3-4* 2003: 245-276.

Dodeja, Chirag, Aparna Suresh e Dhruv Mehta . “Study on Dynamic Pricing in Indian Railways.” *International Journal of Innovative Science and Research Technology, 2.10* 2017.

Ferreira, Frederico Alexandre. “Inteligência Artificial na Verificação e Teste de Software para Desenvolvimento Ágil.” Doctoral dissertation, Instituto Superior de Engenharia de Lisboa. 2017.

Filho, Luiz Claudio de Freitas. *Dynamic Pricing in Transport: a study on social and personal implications to travellers*. Lunds University, 2018.

Fossati Figueiredo, Kleber , Alexandre Machado Silva e Paulo Cesar Pereira Pinto Junior. “Gestão da capacidade em serviços: a adoção do Yield Management por uma grande companhia área brasileira.” *ENCONTRO NACIONAL DA ANPAD 24* 2002.

Gaggero, Alberto A., Branko Bubalo e Lukas Ogrzewalla. “Pricing of the long-distance bus service in Europe: The case of Flixbus.” *Economics of Transportation 19* 2019: 100120.

“https://businessdocbox.com/69201832-Marketing/Yield-management-chapter-1.html.” s.d. *businessdocbox.com.* 23 de 11 de 2020.

Jallat, Frédéric e Fabio Ancarani. “Yield management, dynamic pricing and CRM in telecommunications.” *Journal of Services Marketing* 2008: 465-478.

Kanagal, Nagasimha Balakrishna. “Conceptualization of perceived value pricing in strategic marketing.” *Journal of Management and Marketing Research 12* 2013: 1.

Kimes, Sheryl E. e Richard B. Chase. “The Strategic Levers of Yield Management .” *Cornell University, School of Hotel Administration* 1998.

Kimes, Sheryl E. “The basics of yield management.” *Cornell Hotel and Restaurant Administration Quarterly 30.3* 1989: 14-19.

Krämer, Andreas e Regine Kalka. “How digital disruption changes pricing strategies and price models.” *Phantom Ex Machina. Springer, Cham,* 2017: 87-103.

Lieberman, Warren. “Revenue management in the travel industry.” *Wiley Encyclopedia of Operations Research and Management Science* 2010.

Malighetti, Paolo, Stefano Paleari e Renato Redondi. “Pricing strategies of low-cost airlines: The Ryanair case study.” *Journal of Air Transport Management 15.4* 2009: 195-203.

McAfee, R. Preston e Vera te Velde. “Dynamic pricing in the airline industry.” *forthcoming in Handbook on Economics and Information Systems, Ed: TJ Hendershott, Elsevier* 2006.

Monteiro, Alves Francisco José. *A importância do revenue management para o sucesso da companhia aérea no mundo competitivo de hoje*. Doctoral dissertation. Universidade Federal Fluminense, 2001.

Okumus, Fevzi. “Implementation of yield management practices in service organisations: empirical findings from a major hotel group.” *The Service Industries Journal 24.6* 2004: 65-89.

Pressman, Roger S. *Engenharia de Software. Terceira edição.* São Paulo: MAKRON Books do Brasil Editora Ltda., 1995.

Pupavac, Drago. “Dynamic Pricing: The Future of Retail Business.” *16th international scientific conference Business Logistics in Modern Management*. Osijek, 2016.

Qin, Jin, et al. “Time-Dependent Pricing for High-Speed Railway in China Based on Revenue Management.” *Sustainability 11.16* 2019: 4272.

Rodrigues, Raphael Julien. *Testes baseados em Modelos*. Doctoral dissertation, Master’s Thesis . Universidade do Minho, 2015.

Ryzin, Garrett van e Guillermo Gallego . “Optimal dynamic pricing of inventories with stochastic demand over finite horizons.” *Management science 40.8* 1994: 999-1020.

Santos, Flavio Andrew do Nascimento, Verônica Feder Mayer e Osiris Ricardo Bezerra Marques. “Precificação dinâmica e percepção de justiça em preços: um estudo sobre o uso do aplicativo Uber em viagens.” *Turismo: Visão e Ação 21.3* 2019: 239-264.

Schmidt Hahn de Lima, Patricia, et al. “Yield management em instituições de ensino superior: um estudo de caso na modalidade de ensino a distância de uma universidade comunitária.” *Revista Ibero Americana de Estratégia 15.3* (2016): 70-87.

Selmi, Noureddine e Raphael Dornier. “Yield management in the French hotel business: An assessment of the importance of the human factor.” *International Business Research 4.2* 2011: 58.

Souza, Melissa Mello e e Edson Nunes. “A Aviação Civil nos Estados Unidos: Um Estudo sobre o Papel do Estado na Regulação do Setor Aéreo.” *Documento de Trabalho n. 71–Observatório Universitário* 2007.

Treszl, Márta. *Yield Management*. Master’s Thesis. Budapeste Eötvös Loránd University, 2012.

Wang, Xuan Lorna e David Bowie. “Revenue management: the impact on business-to-business relationships.” *Journal of Services Marketing* (2009).

Weatherford, Larry R e Samuel E. Bodily. “A taxonomy and research overview of perishable-asset revenue management: Yield management, overbooking, and pricing.” *Operations Research 40.5* 1992: 831-844.

Withiam, Glenn. “A“4-C” Strategy for Yield Management. .” *Cornell Hospitality Report, 1* 2001.