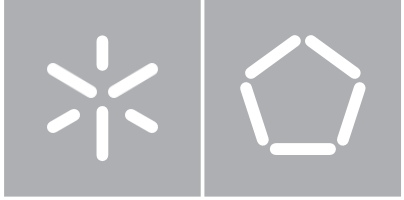
Universidade do Minho
Escola de Engenharia

Fábio Diogo Peixoto Dias

Um Sistema de Recomendação para o Produto RAID

Novembro de 2018



Departamento de Informática

Universidade do Minho
Escola de Engenharia

Fábio Diogo Peixoto Dias

Um Sistema de Recomendação para o Produto RAID

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho realizado sob orientação de:

Professor Orlando Belo

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Professor Orlando Belo por todo trabalho, orientação e disponibilidade que prestou ao longo desta dissertação.
De seguida, queria deixar o meu obrigado à WeDo Technologies pela excelente oportunidade que me foi dada e a todos os seus colaboradores que me acolheram, especialmente ao Engenheiro Carlos Martins pelo acompanhamento que forneceu ao logo deste projecto.
Além disso, gostaria de agradecer a toda a minha família, especialmente aos meus pais e irmão que me apoiaram e ajudaram ao longo do meu percurso académico.
Não posso, também, de deixar o meu obrigado à minha namorada e a todos os meus colegas de curso que tiveram a paciência e disponibilidade necessária para ultrapassar os desafios que foram aparecendo.
A todos, o meu sincero obrigado!

Resumo

Sistema de Recomendação para o Produto RAID

O ciclo de vida dos produtos informáticos é um processo contínuo, durante o qual vão surgindo de uma forma regular novas actualizações, dia após dia, que frequentemente visam a melhoria dos processos de interação do utilizador com o produto. Contudo, este processo é complicado, uma vez que responder de uma forma apropriada às necessidades de cada cliente não é uma tarefa fácil. De forma a combater esta tarefa, a *WeDo Technologies* decidiu implementar no seu principal produto de *software*, o *RAID*, um sistema capaz de monitorizar as diversas interações dos clientes com o produto e com base na informação recolhida apresentar-lhes cenários de interação avançados que de alguma forma reflitam as suas necessidades, por exemplo, em termos de introdução de dados ou configurações de serviços. Desta forma, é possível retirar algum trabalho aos utilizadores do software uma vez que o sistema adianta-se no seu fornecimento. Com este sistema, a *WeDo* pretende alargar um conjunto diverso de funcionalidades dos seus produtos de forma a facilitar os processos de interação com o utilizador, promovendo o desenvolvimento de um protótipo para a geração de recomendações adequadas a cada utilizador dos seus produtos de software, a partir de um conjunto base das suas preferências de utilização do sistema. Basicamente, com este trabalho de dissertação estudou-se, projetou-se e desenvolveu-se esse sistema de recomendação, materializando, de certa maneira algo que M. O'Brien referiu: *"The Web is leaving the era of search and entering one of discovery. What's the difference? Search is what you do when you're looking for something. Discovery is when something wonderful that you didn't know existed, or didn't kown how to ask for, finds you." (M. O'Brien, 2006).*

Abstract

Recommendation System for the RAID Software

The life cicle of software products it's a continuous process. In the middle of this process, new updates will appear in a regular basis, with the single goal of a continuous improvement between the interaction of the user and the software. However, this process can be complex, because it is not an easy task to meet all customer needs. With this goal in mind, WeDo Technologies took the first steps to build, in the main software product, *RAID*, a system capable of monitoring every customer interaction with the software. With this data, we can present to the user a set of scenarios that reflect the client needs, like, fill in the data inputs or advance service configurations that could be done accordingly to what the customers want. With this system, WeDo wants to expand a set of current features that already exists in the core product to ease the integrate the recommendation system in *RAID*. To accomplish this, we intend to develop a prototype that should be able of generating a set of recommendations base on each user preferences. Basically, with this dissertation we have studied, projected and developed this recommendation system, that in a certain way correspond to what M. O'Brien said: *"The Web is leaving the era of search and entering one of discovery. What's the difference? Search is what you do when you're looking for something." (M. O'Brien, 2006).*

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

Nos dias de hoje, apesar dos avanços tecnológicos na área de desenvolvimento de *software*, existe uma necessidade crescente de evoluir e melhorar, não só os produtos existentes, mas também a experiência dos clientes com o produto desenvolvido. Nesse sentido, as empresas que desenvolvem as suas atividades nesta área têm feito um grande esforço para fornecer aos utilizadores novas experiências, formas de navegação e de apresentação do conteúdo, com o objectivo de melhorar a sua interação com o produto de *software* produzido. Porém, tal como foi dito por *Feng Qiu* e *Junghoo Cho* da Universidade da California, cada utilizador tem a sua própria opinião e as suas próprias necessidades (*"one hundred users one hundred needs" (Qiu & Cho, 2006)*. Isto faz com que a criação de um sistema que seja capaz de fornecer respostas às questões de cada utilizador seja uma tarefa bastante árdua, sem se obter usualmente uma solução que se possa considerar óptima. Ao longo de muitos anos de análise de como os utilizadores do *RAID* usavam a ferramenta, a *WeDo Technologies* concluiu que utilizadores distintos procuravam respostas distintas ao longo do dia e nem sempre padronizáveis entre si.

Com vista no objectivo de melhorar a experiência do utilizador com os seus produtos, uma empresa especializada na área das telecomunicações decidiu dar início a um projecto cujos principais propósitos assentavam na construção de um sistema de recomendação adequado a cada utilizador a partir do seu padrão de utilização dos produtos. Além disso, esse projecto deveria permitir também ao utilizador fornecer *feedback* sobre a recomendação gerada pelo sistema, de forma a que fosse possível recalcular a recomendação caso o utilizador fornecesse um *feedback* negativo. Contudo, de

forma a obter respostas individualizadas de cada utilizador, era necessária fazer a recolha de uma quantidade enorme de dados. Para que isso fosse possível, realizámos um estudo detalhado sobre os dados angariar que nos permitisse fazer a construção de um *log* completo – contendo os eventos dos processos de interação do utilizador com a aplicação -, de forma a ser possível posteriormente realizar *queries* relacionadas com o comportamento do utilizador, como pode ser observado no capítulo 4.1.

Em suma, este documento retrata a implementação que foi realizada de um sistema de recomendação que tinha a finalidade de fornecer aos utilizadores uma experiência adaptativa às suas necessidades com base na sua interação com a aplicação.

1.1 Sistemas de Recomendação

Em meados dos anos 90, verificou-se que os produtos ditos informáticos estavam a produzir uma quantidade enorme de dados e que, a sua grande maioria, não era utilizada *(Adomavicius & Tuzhilin, 2005)*. Os sistemas de recomendação podem seu um dos instrumentos de trabalho para a resolução desse problema, dada a sua capacidade de filtrar informação essencial de acordo com os interesses, preferências e comportamentos dos utilizadores (por exemplo: dados do utilizador, de navegação, de onde passa mais tempo, de onde clica, que informação visita mais vezes, a que horas visita essa informação, ect...). Os sistemas de recomendação conseguem beneficiar quer os utilizadores finais quer os fornecedores do serviço, uma vez que permitem aos primeiros uma melhor experiência de navegação bem como a descoberta de novos itens baseados no seu perfil, e aos segundos a possibilidade de compreender a forma como os utilizadores usufruem dos serviços que disponibilizam. Tais padrões permitem sugerir novas funcionalidades aos utilizadores, novas formas de navegação, e novas formas de "prender" os utilizadores ao serviço que providenciam, para cada utilizador individual. Em 2014, Xavier Amatriain (director de investigação da *Netfilix*) fez um estudo sobre o "valor" que é obtido através das recomendações efectuadas *(Amatriain, 2014)* concluindo que 2/3 dos filmes visualizados na *Netflix* foram indicados através de recomendações sugeridas aos utilizadores ou que 35% das compras realizadas são feitas a partir de recomendações. Conclusões

como estas fornecem-nos uma noção bem mais clara da evolução deste tipo de *software* e do que realmente este pode oferecer aos utilizadores.

Figura 1 – As Fases do desenvolvimento de um sistema de recomendação

Essencialmente, os sistemas de recomendação são desenvolvidos em três fases (Figura 1), nomeadamente: 1) a captação dos dados (filtragem dos dados essenciais referido anteriormente), 2) o tratamento e aprendizagem dos dados e, por fim, 3) a recomendação final. A primeira fase visa a captação dos dados que são retirados dos processos de interação do utilizador com a aplicação. Os dados captados passam por guardar a informação específica do utilizador, os conteúdos que este visualizou, a etiqueta temporal (*timestamp*) relativa ao momento em que o utilizador utilizou o recurso ou o serviço, os comportamentos de navegação demonstrados, entre outras coisas. Usualmente, esta informação pode ser obtida de diferentes maneiras, nomeadamente *(Isinkaye, Folajimi, & Ojokoh, 2015)*:

- **Explícita** – consiste no *feedback* fornecido pelo utilizador, isto é, o sistema apresenta uma série de questões ao utilizador de forma a obter a sua opinião sobre diversos temas (as formas utilizadas podem ser de vários tipos, quer seja por formulário, quer seja em forma de *rating*). A principal desvantagem deste método passa por depender do *input* fornecido pelo utilizador, os quais nem sempre estão dispostos a providenciar a quantidade de informação necessária. Contudo, o facto da informação ser dada pelo utilizador faz com que os dados sejam mais confiáveis, uma vez que foram fornecidos pelo próprio *user*.
- ***I*mplícita** – informação obtida através do comportamento do utilizador na aplicação, ou seja, o sistema automaticamente infere as preferências dos utilizadores depois de captar os dados a partir da sua interação com a aplicação. A principal vantagem deste sistema, comparativamente à opção anterior, é que esta abordagem não necessita de qualquer *input* do utilizador, removendo a desvantagem do *feedback* explícito. Porém, a subjectividade é

difícil de avaliar, o que faz com que as recomendações inferidas pelo sistema sejam completamente erradas ao que o utilizador necessitava.

- ***H*ibrido** – consiste numa combinação dos dois anteriores, é como aproveitar o melhor dos dois mundos, permitindo assim que o sistema de recomendação seja muito mais robusto e intolerante a falhas. Este sistema pode ser obtido inferindo o que utilizador necessita, mas possibilitando que este forneça *feedback* ao sistema se tal recomendação foi útil ou não.

O feedback híbrido foi o que acabou por ser utilizado neste projecto uma vez que havia a necessidade de obter *feedback* (positivo ou negativo) do utilizador sobre a recomendação.

A segunda fase do processo consiste na obtenção de conhecimento (aprendizagem) sobre os dados recolhidos. Basicamente, através da aplicação de algoritmos, pretende-se filtrar os dados captados anteriormente, de forma a construir o perfil de utilização de cada utilizador. Quanta mais informação for captada mais completo será o perfil desenhado pelo sistema. É nesta fase que se acaba por dar significado/sentido à informação recolhida. Inicialmente, os dados são recebidos pelo sistema em formato bruto, sendo necessário processá-los para que passem a conter algum significado, isto para que se consiga, de forma eficiente, interpretar a recomendação que irá ser gerada. Por último, a terceira fase tem como intuito a geração da recomendação. Aqui o software solicita ao sistema a indicação de que necessita de uma recomendação para um determinado utilizador e este responde com uma recomendação que ache ser mais precisa com base no perfil do utilizador construído anteriormente. Através da Figura 1podemos ver que o processo volta ao início. Isto acontece quando o utilizador fornece ao sistema *feedback* da recomendação que este lhe forneceu, permitindo ao sistema que reajuste do seu *profile,* para que numa próxima sugestão inclua o *feedback* fornecido. Além disso, podemos ainda concluir que todas as fases são dependentes umas das outras, pois se a captação dos dados for escassa o perfil do utilizador ficará incompleto acabando pela sugestão não ir de encontro ao que este esperaria/necessitaria.

A principal desvantagem de qualquer sistema de recomendação passa pelo tempo de processamento da informação. É uma situação provocada pela ocorrência de *queries* em tempo real que visam fornecer ao utilizador recomendações ajustadas ao seu padrão de utilização, aquando da sua utilização – o tempo de execução destas *queries* pode ser demasiado elevado para as necessidades do cliente. Além disso, é necessário que o sistema de recomendação seja construído com o intuito de fornecer resposta a qualquer necessidade particular do utilizador. Isto faz com que a sua implementação seja um grande desafio, dadas as grandes dificuldades que se enfrenta quer na

escolha quer na interpretação dos dados que o sistema terá ao seu dispor. A melhor forma de o fazer é ter sempre em conta o objectivo principal do utilizador.

O ser humano tem tendência para criar padrões nas suas tarefas diárias, aquilo que designamos de rotina, apesar de ser inconsciente acaba por ser captada pelo sistema de recomendação. Porém, para obter essa informação o sistema de recomendação precisas de aliar à pesquisa realizada pelo utilizador a sua *timestamp*, isto é, toda a informação temporal, de ano, mês, dia, hora e dia da semana, com esta informação é mais simples para o sistema conseguir compreender os padrões de uso (e de hora) do utilizador. Por exemplo, um utilizador no primeiro dia de cada semana (segunda-feira) consulta o seu histórico de tarefas de forma a construir um relatório com as suas tarefas realizadas, com um sistema de recomendação poderíamos compreender que no primeiro dia de cada semana este utilizador faz sempre a mesma tarefa podendo enviar-lhe um alerta de que necessita de preencher o relatório, ou ainda melhor, compreendendo o conteúdo desse relatório preenchendo-o automaticamente de forma a possibilitar ao utilizador debruçar-se sobre outras tarefas.

Além disso, é importante referir um tipo de arquitectura que é bastante comum nos sistemas de recomendação, EDA (*Event-Driven Architecture*), que tem como base os produtores e consumidores de eventos, Brenda Michelson designou estes eventos como sendo o que acontece à volta do *software*, isto é, são todas as acções que podem ser captadas de forma a serem interpretadas pelo sistema. O evento é composto por duas partes, o *header*, que contém todos os dados que conseguem descrever o evento, tais como: um indentificador específico gerado para o evento em questão, o tipo de evento, a sua *timestamp* para ser possível analisar a data e hora da sua ocorrência, entre outros. A outra parte, é referida como sendo o *body* do evento, este contém toda a informação que descreve o que foi feito, na secção 4.1 é possível visualizar toda a informação que foi captada e inserida no *body* do evento criado *(Michelson, 2011)*. De referir, ainda, que as EDA podem ser de duas tipologias diferetes, *Mediator* e *Broker*, a principal diferença ente elas é que a topologia de *Mediator* contém um mediador de eventos intermédio, tal como pode ser observado na Figura 2 - Exemplos das topologias *Mediator* e *Broker* . Este consiste numa camada de pré-processamento desta arquitectura, necessária para quando o tipo de informação que obtemos do evento é demasiada complexa ou quando simplesmente é necessário proceder a algumas operações de forma a tratar o evento. Por exemplo, numa transferência bancária realizada, antes de proceder à transação é necessário validar se a transferência é possível (se existe saldo suficiente na conta do utilizador, se o NIB para onde o dinheiro vai ser transferido é válido, ect...).

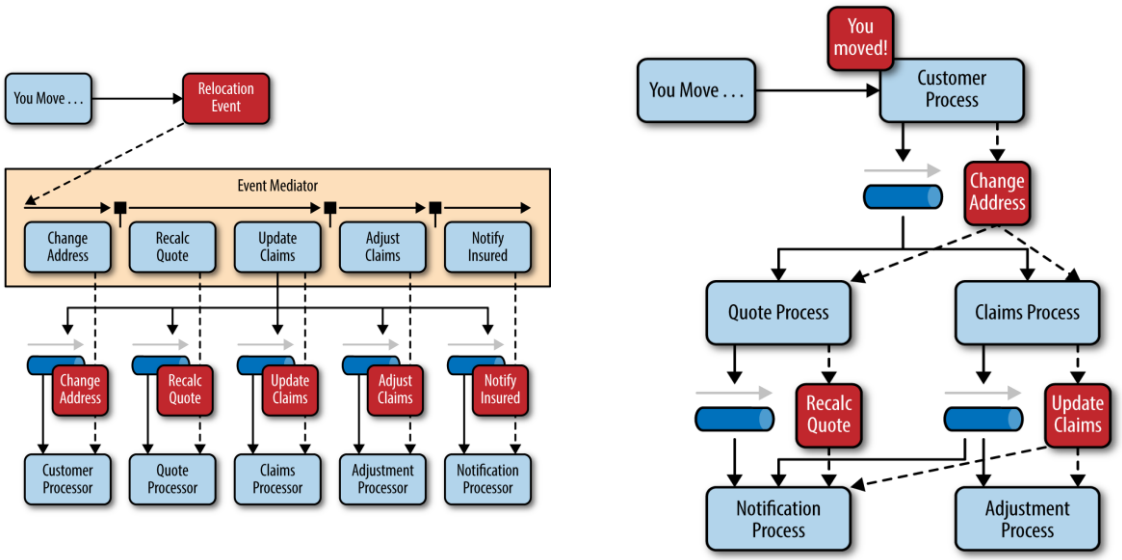


Figura 2 - Exemplos das topologias *Mediator* e *Broker* (Richards, 2015)

1.2 Domínio da Aplicação

Este trabalho de dissertação foi desenvolvido em ambiente industrial, na empresa WeDo *Technologies*. A WeDo é uma empresa de *software* que desenvolve maioritariamente os seus produtos para empresas na área de telecomunicações. A WeDo conta com mais de 600 profissionais espalhados por 10 escritórios no mundo inteiro, tendo os seus produtos distribuídos por mais de 170 clientes em 96 países. A *Gartner*, uma empresa de pesquisa e consultoria que fornece informações sobre o mundo tecnológico, em Junho de 2013, reconheceu a *WeDo Technologies* como líder global no que diz respeito ao fornecimento de soluções de *Revenue Assurance* e *Fraud Management* para o setor das telecomunicações, como pode ser visualizado no seu relatório *Market Share: Telecom Operations Management Systems (BSS, OSS and SDP) (Kurth, J. Scholz, & Bhatia, 2013)*. Além da Gartner, também a *Frost & Sullivan Stratecast*, uma empresa de consultoria e pesquisa, em 30 de Dezembro de 2014, reconheceu a WeDo como líder global após a sua análise da mudança do mercado de comunicações. O seu relatório "2014 Global CSP Financial Assurance: Market Share Analysis, Forecast, and Supplier Assessment" *(Frost & Sullivan, 2014)*) surgiu depois de terem sido

examinadas mais de 30 empresas responsáveis por soluções para problemas áreas de *Revenue Assurance* e *Fraud Management*.

O *software* produzido pela *WeDo* tem como objectivo a análise de grandes quantidades de dados *(Big Data)* das diversas empresas a si associadas, além de permitir combater as várias falhas operacionais dos sistemas ou de negócio. A *WeDo* contém no seu portfólio vários tipos de produtos, nomeadamente: *RAID, SHAPE, COLLECTIONSBROKER, ROAMBROKER, NETCLARUS* e uma oferta *Cloud (RAID Cloud).*

O *software* da WeDo alvo deste trabalho de dissertação foi o sistema *RAID* (Technologies, 2017). Este sistema é responsável pela maior parte das receitas obtidas pela WeDo. Com o sistema RAID (Figura 3) é possível monitorizar os dados de outras aplicações e plataformas de negócio, o que torna possível observar de forma detalhada as várias actividades de negócio, proporcionando bons elementos de gestão para a melhoria do desempenho empresarial.

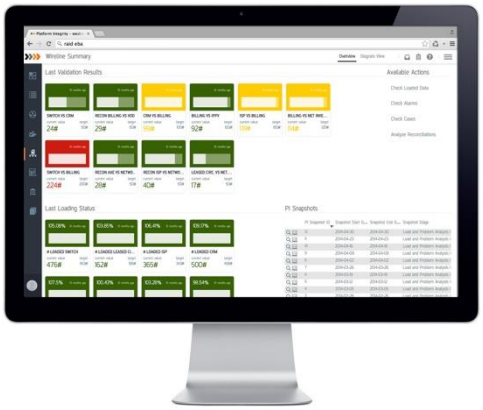


Figura 3 – Exemplo de um layout do produto de software RAID

O sistema *RAID* fornece aos seus clientes a possibilidade de reunir todos os seus dados, proporcionando uma plataforma de monotorização adequada para a identificação daquilo que é relevante para o cliente. Com esta informação os clientes podem descobrir a origem de muitos dos seus problemas e assim abordar a sua resolução de forma sustentada. O sistema *RAID* foi o principal contribuidor para os bons resultados dos relatórios referidos anteriormente, os quais consideravam a WeDo Technologies como líder global em *Revenue Assurance* e *Fraud Management*.

1.3 Motivação e Objectivos

A WeDo Technologies propôs como trabalho de dissertação a análise do seu produto *RAID* em todos os aspectos que implicassem algum tipo de interação com o utilizador. O objetivo era bastante claro: melhorar a interacção do cliente com o produto. Nesse sentido foi desenvolvido especificamente um *software* com o objectivo de satisfazer tal pretensão, o que originou um processo de desenvolvimento de um sistema capaz de gerar recomendações para os diversos utilizadores, de forma a que fosse possível obter recomendações individualizadas com base no perfil de cada utilizador. Contudo, como a versão corrente do sistema *RAID* não fazia a recolha de dados resultantes dos seus processos de interação com os utilizadores, foi necessário proceder a uma análise exaustiva do produto de forma a identificar as áreas em que tais processos aconteciam e aquelas que seria, de facto, fazer a recolha dessa informação. Com essa análise, iniciou-se o processo de construção da captação dos dados de interação, com o objetivo de que um dia todos os componentes existentes e a serem construídos possam incorporar este serviço de registo de interação. A captação dos dados foi, pois, o ponto mais importante deste trabalho de dissertação, uma vez que foi necessário projetar e implementar um conjunto de mecanismos muito diverso para que fosse possível registar de uma forma sistemática tudo o que acontece ao longo de um processo de interação do utilizador com o sistema, e, posteriormente, fornecer recomendações aos seus utilizadores com os conteúdos que estes visitam mais frequentemente.

O problema que a WeDo Technologies pretendeu resolver prendeu-se na melhoria da experiência que cada utilizador tem com o seu produto *RAID*, de forma a que fosse possível tornar a sua interação com o sistema mais fácil e agradável, tendo ao seu dispor um sistema capaz de gerar novas recomendações de utilização, como melhorias na navegação no sistema de serviços, na formatação de conteúdos, entre outras coisas. Em suma, o objectivo foi tornar o sistema *RAID* mais interativo, dar-lhe a capacidade de melhorar ao longo do tempo a sua interação com os seus utilizadores, tornando mais *user-frendly*.

1.4 Problemas e Desafios

Como referi primeiramente, este projecto incidiu no produto *RAID*, *software* este que gera uma quantidade enorme de dados, quer estes sejam referentes aos diversos componentes do produto, quer sejam obtidos através da interacção do cliente com o produto. Sendo que, foi principalmente neste último ponto, que a WeDo Technologies decidiu alterar o seu pensamento e dar início a este projecto inovando assim o seu produto e a sua forma de funcionamento, com o intuito de melhorar a interacção do cliente com o produto. Para isso foi desenvolvido um *software* com o objectivo de combater este problema dando início a um sistema capaz de gerar recomendações para os diversos utilizadores, baseando-se na interacção destes com a aplicação *RAID,* de forma a obter recomendações individualizadas com base no perfil de cada *user*. Contudo, como não havia qualquer tipo de dados guardados até ao momento sendo necessário proceder a uma análise exaustiva do produto de forma a identificar as áreas de mais fácil acesso. O primeiro passo passou pela captação dos dados para que um dia todos os componentes existentes e que possam ser criados no futuro contribuam para este sistema. A captação dos dados foi o ponto mais importante para este projecto, uma vez que era necessário obter dados da interacção do utilizador com o sistema para que estes sejam tratados e forneçam recomendações aos seus utilizadores com os conteúdos que estes visitam mais frequentemente.

Os clientes que já utilizam o produto *RAID* irão também beneficiar deste sistema, uma vez que este poderá oferecer um sistema de recomendações adequado a cada utilizador proporcionando uma navegação e interação com o *software* completamente diferente do que têm realizado até ao momento. O problema que a WeDo Technologies pretendia resolver passava por melhorar a experiência que cada utilizador tem com o seu produto de forma a tornar a sua interação mais fácil e agradável para cada um deles, ou seja, que os utilizadores tenham ao seu dispor um sistema capaz de gerar novas recomendações de visualização de dados. Este projecto sendo uma fase inicial desta grande alteração, tendo sempre em mente a elaboração de algo mais complexo, como melhorias na navegação, formato de conteúdo, entre outros. Com o pensamento de, mais tarde, obter um sistema interativo capaz de melhorar a interacção dos clientes com o produto, tornando este mais fácil e *user-frendly* para quem o usa, mas com as bases assentes no sistema de recomendação contruído ao longo deste projecto.

Como vimos, os sistemas de recomendação podem ter algumas desvantagens, que provocam problemas e restrições de implementação. No caso do sistema *RAID*, tais desvantagens foram reveladas em casos como:

- **A sobrecarga de serviços -** Um dos principais desafios deste trabalho de dissertação foi lidar com o tempo de processamento da recomendação a gerar. Para isso foi necessário retirar do *RAID* tudo o que fosse relacionado com o *software* de recomendação, colocando-o fora do produto para que os clientes que o utilizassem não notassem nenhuma quebra em termos de *performance*.

Figura 4 - *Overload* do *software* RAID

- **O armazenamento de dados em caso de falha na comunicação com o servidor -** Um dos problemas adicionais acontece quando ocorre uma falha do servidor e não é possível guardar os dados que estão a ser trabalhados. O servidor pode estar algumas horas inativo, e, obviamente, a informação envolvida não pode ser descartada. Seriam bastantes "horas de dados" desperdiçados pelos vários utilizadores.

- **O envio de informação previamente guardada -** Aquando da falha na comunicação do servidor é necessário voltar a enviar a informação guardada para o servidor. Essa tarefa terá de ser executada num período de tempo incremental (isto é, para não estar constantemente a enviar pedidos ao servidor que não está a responder, aumenta-se o tempo de tentativa de conexão ao servidor), de forma a realizar o pedido de inserção dos dados no servidor. Além disso, para não sobrecarregar o servidor com pedidos de inserção de larga escala, a melhor

estratégia será a de não enviar todos os registos guardados, mas sim um número de blocos (fixo) deles.

- **O tipo de dados -** A informação retirada dos processos de interação do sistema com o utilizador é muito diversa. Isto pode levantar alguns problemas, já que dificulta a construção dos objectos *JSON* envolvidos, isto é a informação em causa pode ser proveniente de um formulário ou de uma tabela. De facto, nunca se sabe o tipo de informação que se vai receber.

- **A obtenção da informação do utilizador que não consta no produto RAID -** De forma a construir um perfil completo da informação do utilizador é necessário obter dados sobre a sua localização (país, cidade, localidade, coordenadas geográficas, etc.). Porém, essa informação não é guardada no *core* do *RAID.*

- **O tratamento de toda a informação -** O principal desafio deste trabalho de dissertação foi o tratamento da informação envolvida no processo de interação, uma vez que se pretendeu construir um sistema de recomendação adequado ao padrão de utilização de cada utilizador, isto porque os dados eram diversos e o tipo de informação a retirar de cada um constituiu um problema.

Porém, o grande desafio deste trabalho foi dar ao sistema de recomendação a capacidade de se adaptar às necessidades de cada cliente do *software*. Basicamente, isto significa que as recomendações que o sistema fornecer o utilizador deveriam ser adequadas ao seu padrão de utilização e, caso este não ache a recomendação adequada, o sistema deveria adquirir esse *feedback* refletindo posteriormente essa informação em próximas recomendações.

1.5 Organização da Dissertação

Para além do presente capítulo, esta dissertação inclui mais 6 capítulos, nomeadamente:

- **Capítulo 2** – *Trabalho Relacionado* -, capítulo em que são apresentados alguns casos reais que se aproximam daquilo que foi realizado ao longo deste trabalho de dissertação. Nesta secção é possível encontrar dois exemplos práticos de sistemas de recomendação que utilizamos todos os dias, o *Spotify* (aplicação de música) e a *Netflix* (serviço de *streaming* que permite ver séries e filmes *online*).

- **Capítulo 3** – *Evolução do Processo e Decisões* -, no qual são retratadas todas as decisões que permitiram resolver todos os problemas mencionados acima (p.e.: sobrecarga de serviços, o armazenamento e o tipo de dados, ect.). Para a solução da maioria dos casos recorremos a serviços externos que permitiam uma solução fácil e eficiente para os problemas em mão (recorremos a serviços como *Web Workers*, *IndexedDB*, *Apache Flink*, *Apache Kafka*, ect.).

- **Capítulo 4** – O Sistema de Recomendação -, aqui apresentamos e descrevemos as duas principais etapas da construção do sistema, a captação dos dados que sustentasse o sistema de recomendação e a arquitectura da aplicação desenhada. A primeira fase, retrata as dificuldades sentidas na captação dos dados e como foram ultrapassadas, terminando na explicação da extração de informação - dos componentes do *RAID* e da interação do utilizador com este – bem como, toda arquitectura de captação em que está inserida. Na segunda fase, é apresentada toda a arquitectura do sistema de recomendação, os seus componentes e serviços (como *Apache Kafka*, *Apache Flink*, *Weka*, ect.), que têm como principal objectivo a recomendação final ao utilizador o mais precisa possível.

- **Capítulo 5** – Mineração de Dados e seus Resultados -, no qual apresentamos toda a parte mais técnica deste trabalho, desde do processo de mineração dos dados, que retrata todo o processo de implementação dos modelos de *machine learning*, quais os modelos implementados e uma descrição de cada um deles. De referir, quem é nesta secção que é apresentada a análise dos resultados obtidos, análise esta com base não só nos modelos implementados – prós e contras de cada –, mas também nos dados que foram captados.

- **Capítulo 6** – Aplicação dos Resultados Obtidos -, neste capítulo descrevemos toda a interação que o utilizador tem com a aplicação, isto é, a informação que é apresentada ao utilizador – que é enviada pelo sistema de recomendação –, forma como os dados são representados, em tabela, gráfico ou cartões. Além disso, é ainda, descrito a forma como o utilizador pode fornecer *feedback* ao sistema para que este possa aprender com o *input* fornecido.

- **Capítulo 7** – Conclusões e Trabalho Futuro -, aqui apresentamos para além das conclusões obtidas, as dificuldades que foram surgindo e como foram ultrapassadas ao longo deste trabalho de dissertação. Também, é apresentada, nesta secção, uma visão geral dos resultados e dos dados obtidos, bem como, as ilações retiradas pela WeDo Technologies e o trabalho futuro que desejam implementar no seu produto de *software*.

Capítulo 2

Trabalho Relacionado

Os sistemas de recomendação são algo que hoje está inerente às mais diversas aplicações, podendo ser encontrados em vários domínios como o retalho (*amazon, ebay, gearbest,* etc.) ou o mundo musical (*spotify, youtube,* etc.). Até no próprio motor de busca mais utilizado (*Google Chrome*) podemos encontrar também elementos de recomendação, atuando sobre as pesquisas realizadas pelos vários utilizadores nas diversas aplicações que utilizam. Aquilo que estes sistemas revelam, o seu *output,* varia de pessoa para pessoa, uma vez que o motor de pesquisa avalia os pedidos que lhe vão fazendo de forma a poder fornecer ao utilizador aquilo que determinou como sendo aquilo que acha ser mais indicado para ele, de acordo com o padrão de utilização que o utilizador tem revelado ao longo dos seus processos de interação. Os resultados deste projecto de dissertação não evidenciarão funcionalidades tão complexas quanto aquelas que as ferramentas atuais de recomendação disponibilizam, mas o conceito e a base no qual este projecto assenta visa isso mesmo: a construcção de um sistema de recomendação que forneça recomendações com base em padrões de utilização de utilizadores.

Como já foi referido anteriormente, Xavier Amatriain, director de investigação da *Netflix* realizou um estudo no qual concluiu que 2/3 dos filmes visualizados na *Netflix* surgem de recomendações sugeridas aos utilizadores e que 35% das compras realizadas são feitas a partir de recomendações

realizadas. Uma das aplicações mais bem-sucedidas no domínio dos sistemas de recomendação tem sido o *Spotify (Pasick, 2015)*. é possível observar uma *playlist* semanal, gerada de forma automática, baseada nos gostos e preferências de cada utilizador. Se pensarmos com quantos utilizadores este sistema interage diariamente, de imediato surge-nos a questão de como é que é possível gerar tantas *playlists* individualizadas para tantos utilizadores. O *Spotify* encontrou uma maneira bastante interessante de remover trabalho aos seus clientes no processo de escolha de músicas e na criação das suas próprias *playlists,* oferecendo-lhes semanalmente uma lista gerada por ele. Essa lista é produzida por inferência, com base naquilo que os utilizadores ouviram anteriormente. Basicamente, o que o motor de inferência do Spotify faz é bastante simples. Na verdade, este observa todas as *playlists* criadas por todos os utilizados e cruza a informação nelas contida com os gostos que determina serem mais próximos dos gostos de cada utilizador. Na Figura 5 é possível ver como isso funciona. Através da utilização de um *"taste profile",* calculado pelo próprio motor do *Spotify*, combinado com as milhões de *playlists* que tem ao seu dispor, o motor gera a *playlist "Weekly Discover" (Pasick, 2015)*. Temos, assim, dois grandes elementos (*taste profile e playlist*) que fazem com que as recomendações semanais sejam tão úteis, ou seja:

1. A captação e compreensão dos gostos dos utilizadores, que é feito com base no(s) estilo(s) de música, ano, banda, vocalistas, entre outros, que o utilizador costuma ouvir – daí que as categorias das músicas são tão importantes para este motor;
2. As *playlists* de outros utilizadores, o peso das cotações atribuídas para cada *playlist* não é igual, as *playlists* de utilizadores designados como profissionais tem mais peso que a de que utilizadores normais; isto pode causar alguma controvérsia, porém se pensarmos bem acaba por fazer sentido, uma vez que qualquer pessoa pode ter criado várias *playlists* com gostos musicais misturados o que para o sistema é mais complicado de compreender, enquanto que os "profissionais" acabam por criar *playlists* individuais, podendo combinar estilos musicais, mas sempre sendo estilos semelhantes.

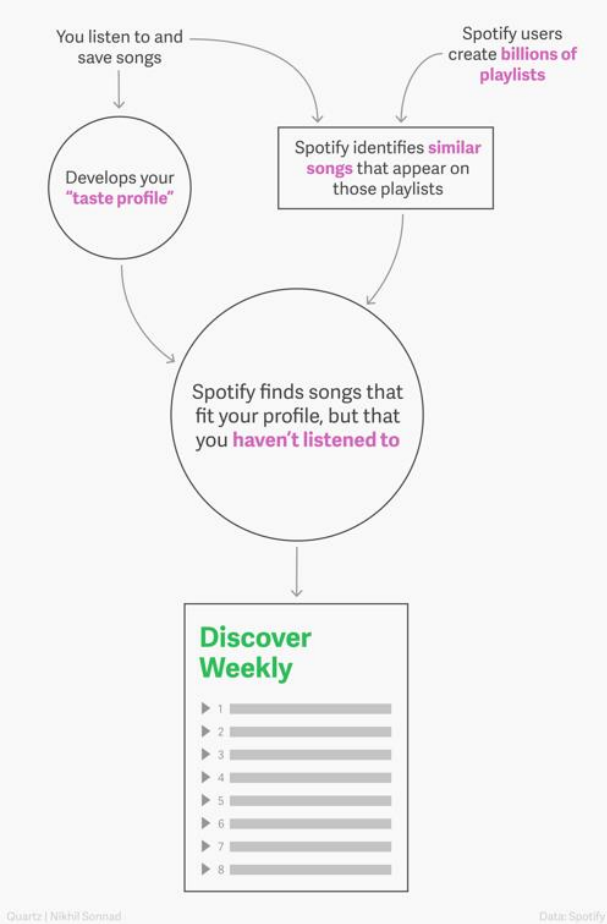


Figura 5 – Ilustração do método de *Weekly Discover* da *Spotify (Pasick, 2015)*

Este processo fez com que o *Spotify* conseguisse combater o problema da ambiguidade de gostos entre utilizadores. Quem, também, fez algo para combater este problema foi a *Netflix* que ao longo dos anos foi enfrentando o problema dos gostos serem relativos. No seu caso, uma série ou um filme pode ter uma cotação bastante elevada – a *Netflix* utiliza um sistema de estrelas a serem atribuídas – o problema neste cenário é que facto de um determinado filme ter uma cotação elevada

não significa que um determinado utilizador vá gostar, isto pode acontecer ou porque pessoalmente não gosta daquele tipo de filmes ou simplesmente lhe apetecia ver um filme diferente naquela altura. Com o surgimento da televisão via Internet foi possível melhorar as recomendações que seriam feitas aos utilizadores, uma vez que desta forma é possível obter muitos mais dados do que era possível anteriormente. A apreciação dos gostos dos utilizadores agora pode ser obtida de forma mais credível, não só pela quantidade de informação agora recolhida, substancialmente bem maior, mas também pelo significado que essa informação agora permite produzir. Coisas como guardar todos os momentos de *play*, pausa e procura, de todos os utilizadores, permite que qualquer sistema de recomendação ganhe robustez e consiga optimizar o *output* fornecido ao utilizador, indo mais ao encontro daquilo que necessita.

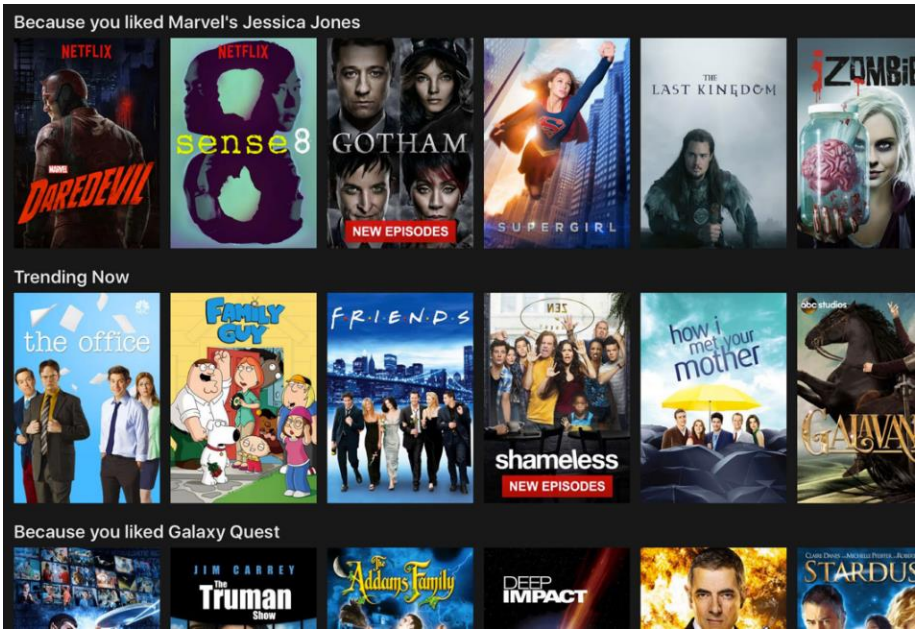


Figura 6 - Recomendação sugerida na aplicação da Netflix (Lusbin, 2016)

Na Figura 6 podemos observar a *interface* da *Netflix* após a entrada na televisão por Internet. Nessa imagem, na parte superior, vemos que o motor de recomendações da *Netflix* está a sugerir uma série de *TV Shows* apenas porque o utilizador gostou de *Marvel's Jessica Jones*. Porém, quanto maior for a interação do utilizador com a aplicação melhores os resultados fornecidos.

Gomez-Uribe, vice-presidente para a inovação do produto e personalização dos algoritmos da *Netflix*, afirmou que *"If we show you something that leads you to watch six minutes and then abandon it, that's a failure from our point of view. It's even worse than you not watching it, because we just made you waste your time on something you didn't enjoy." (Lusbin, 2016).* Basicamente, isto significa que, mesmo com algumas contrariedades, um motor de recomendações é capaz de aprender mesmo com o *feedback* silencioso fornecido pelo utilizador.

Capítulo 3

A Evolução do Processo e Decisões

3.1 Web Worker

Uma das dificuldades identificadas neste projeto estava relacionada com o reflexo da carga que os trabalhos a desenvolver deste projecto poderia ter no desempenho da aplicação *RAID*. Tal reflexo não poderia ser muito acentuado para que os clientes da aplicação não notassem um decréscimo de *performance* nas ações realizadas diariamente. Assim, na solução encontrada utilizámos *Web Workers*, uma vez que estes permitem a execução de múltiplas tarefas sem bloquearem a *interface* do utilizador - "*Workers utilize thread-like message passing to achieve parallelism. They're perfect for keeping your UI refresh, performant, and responsive for users*." (Bidelman, 2010)). Estas execuções sem bloqueio só são possíveis porque os *Web Workers* funcionam como uma *thread* que corre paralelamente com a *main thread* do *browser*, isto é, enquanto a aplicação corre naturalmente na *main*, o *worker* desempenha todas as suas tarefas numa *thread* diferente.

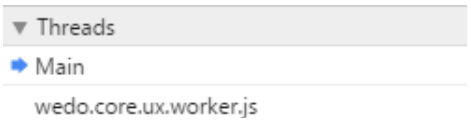
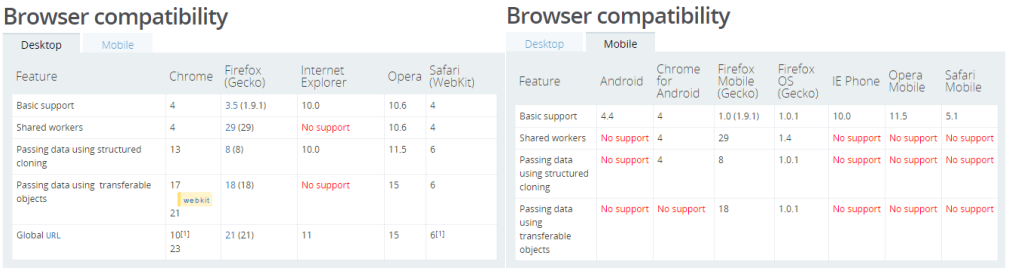


Figura 7 - *Thread* criada pelo *Web Worker*

Na Figura 7 podemos ver a *thread* que é criada, com o uso dos *Web Workers,* a correr paralelamente a *thread Main* do *browser*. Porém esta *API* (*application programming interface*) tem várias limitações *(Mozilla, Web Worker API, 2017)*, sendo a principal restrição o facto de não ter acesso ao *DOM* da página. De forma a ultrapassar este problema os *workers* têm uma técnica baseada em troca de mensagens que permite a comunicação entre o *manager*, um *script* que cria e executa o *worker*, e o próprio *worker* que realizará as funções desejadas.



Browser compatibility

Desktop | Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support	4	3.5 (1.9.1)	10.0	10.6	4
Shared workers	4	29 (29)	No support	10.6	4
Passing data using structured cloning	13	8 (8)	10.0	11.5	6
Passing data using transferable objects	17 webkit 21	18 (18)	No support	15	6
Global URL	10[1] 23	21 (21)	11	15	6[1]

Browser compatibility

Desktop | Mobile

Feature	Android	Chrome for Android	Firefox Mobile (Gecko)	Firefox OS (Gecko)	IE Phone	Opera Mobile	Safari Mobile
Basic support	4.4	4	1.0 (1.9.1)	1.0.1	10.0	11.5	5.1
Shared workers	No support	4	29	1.4	No support	No support	No support
Passing data using structured cloning	No support	4	8	1.0.1	No support	No support	No support
Passing data using transferable objects	No support	No support	18	1.0.1	No support	No support	No support

Figura 8 - Compatibilidade da API Web Worker nos browsers para as versões Desktop e Mobile (Mozilla, Web Worker API, 2017)

A única forma desta *API* poder ser implementada e utilizada seria se esta fosse compatível com todos os *browsers* que os utilizadores do produto *RAID* pudessem utilizar, tal como podemos observar na *Figura 8*. Todos os *browsers* permitem realizar as funções básicas necessárias para implementar e trabalhar com *Web Workers*, mesmo em versões *mobile*.

3.2 A API IndexedDB

Após a escolha da utilização de *Web Workers* foi necessário verificar se a tecnologia a utilizar para guardar os dados em caso de falha do servidor era compatível com a escolha anterior.
As *IndexedDB (W3C, 2015)* são um tipo de *API* para base de dados que existe nas novas versões dos *Web browsers,* baseada em *Javascript,* que permite guardar objetos de dados e realizar *queries*. Na Figura 9 - Compatibilidade da API *IndexedDB* nos browsers para as versões *Desktop* e *Mobile* é possível visualizar a compatibilidade destas nos diversos *browsers (Mozilla, IndexedDB API, 2017)*. A compatibilidade nos diferentes *browsers* pode ser uma desvantagem enorme para este tipo de ferramenta, principalmente porque os produtos a utilizar têm que ser versáteis e compatíveis com vários *browsers* e as suas diversas versões. Contudo as diferentes versões desta *API* coincidem com as versões que o software *RAID* suporta.

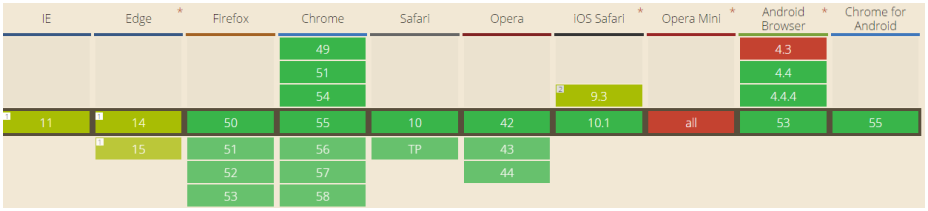


Figura 9 - Compatibilidade da API *IndexedDB* nos browsers para as versões *Desktop* e *Mobile (Mozilla, IndexedDB API, 2017)*

Além da *API IndexedDB* existem outras duas *API* (*WebSQL* e *LocalStorage*) que têm como objectivo servir de base de dados local ao *browser*. Porém nenhuma delas permitem a comunicação com *Web Workers (W3C, 2015)*. Além disso, a *API IndexedDB* permite guardar os dados mesmo quando é encerrada a sessão do utilizador. Isto possibilita que os dados sejam guardados mesmo de um dia para o outro em caso de falha do servidor – este pode estar desligado durante dias devido a falhas técnicas.

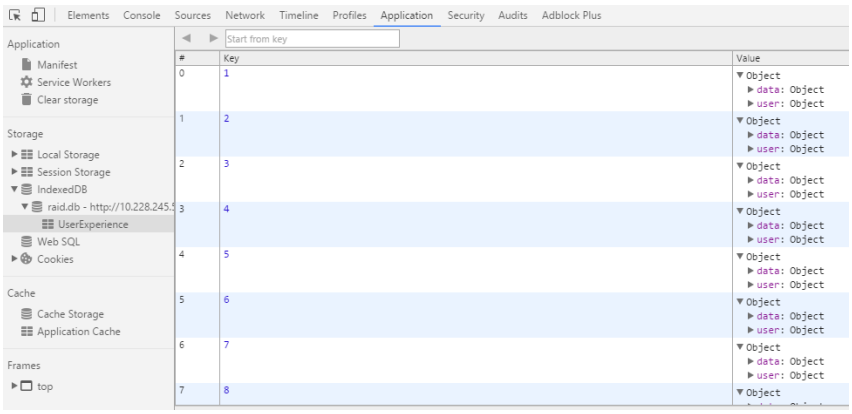


Figura 10 - Exemplo prático de uma *IndexedDB*

A Figura 10 mostra como os dados são guardados pela API IndexedDB e como consultá-los no browser - neste caso, em particular, foi utilizado o Web browser *Chrome*. A informação guardada pela API *IndexedDB* está organizada em registos do tipo par chave valor, no qual a *chave* é o *ID* do registo a que pode aceder e o *valor* o objecto de dados, em *JSON,* que contém a informação do utilizador mais a informação que foi retirada da interação do cliente como o produto de *software*.

3.3 O Servidor NodeJS

Para sustentar a execução do projeto de monitorização da utilização da aplicação era necessário escolher um servidor que realizasse o encaminhamento dos pedidos, quer de envio de informação (envio do objecto *JSON* construído), quer do resultado da recomendação. O servidor que escolhemos foi implementado em *NodeJS* pela facilidade com que permite fazer o tratamento de formatos *JSON* e pela larga comunidade de utilizadores existente. Tudo isto faz com que este tipo de servidor seja, actualmente, dos mais implementados (Figura 11). Além disso, o produto *RAID* foi construído sobre *javascript* (linguagem utilizada nesta *framework*) o que permite que o seu suporte, manutenção e possíveis melhorias sejam mais fáceis de atingir.

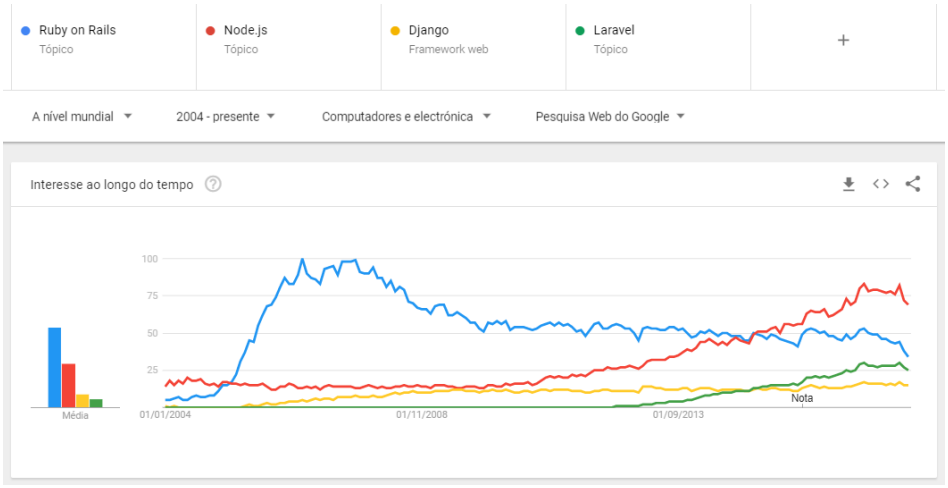


Figura 11 - Tendência do mercado relativa às frameworks *Ruby on Rails, NodeJS, Django* e *Laravel (GoogleTrends, 2018)*

Na Figura 11 podemos ver a tendência da utilização do *NodeJS* comparativamente a outras *frameworks* para implementação de servidores, como o *Ruby on Rails, o Django* ou o *Laravel*. Assim, podemos concluir que atualmente a preferência da maioria dos *developers* recai sobre o NodeJS, dando por dar uma maior garantia à escolha desta *framework*, uma vez que estão a aparecer, cada vez mais, novos contribuidores para o desenvolvimento e melhoria desta ferramenta.

3.4 O Apache Kafka

Numa fase seguinte, foi necessário escolher uma ferramenta que fosse capaz de consumir as mensagens entregues pelo servidor e que, posteriormente, as entregasse a um ou mais *endpoints* com capacidade para tratar os dados recolhidos. Basicamente, pretendia-se um serviço que fosse

responsável por servir de ponte entre os dados em formato bruto e a ferramenta responsável pelo seu tratamento e respectiva recomendação a ser gerada – neste caso específico o serviço utilizado foi o *Apache Flink*. Esta troca de mensagens deveria ser feita quase em tempo real, de forma a diminuir a latência do tempo de espera do utilizador pela entrega dos dados. Para isto, foram estudadas algumas *frameworks* com serviços para tratar o consumo e a entrega de mensagens, nomeadamente: o *Apache Kafka*, o *Apache ActiveMQ*, o *RabbitMQ* e o *zeroMQ*.

Figura 12 - Tendência do mercado nas frameworks *Apache Kafka, Apache ActiveMQ, RabbitMQ* e *ZeroMQ (GoogleTrends, 2018)*

Na Figura 12 podemos ver que a ferramenta que mais é utilizada atualmente é o *RabbitMQ*. Porém, é importante referir que o *Apache Kafka* é uma *framework* relativamente recente e que ao longo do tempo foi sendo reformulada e melhorada até aos dias de hoje - é notável observar que desde 2014, esta *framework* teve 8 versões diferentes, sem contabilizar pequenos *upgrades* de *bug-fixing*. Comparativamente com as outras *frameworks* mais antigas (*ActiveMQ* e *RabbitMQ, zeroMQ)*, o Apache *Kafka* apresenta um maior rendimento, uma vez que trás consigo um sistema de particionamento, de replicação de dados e de tolerância de falhas. Isto torna esta ferramenta mais adequada para aplicações com envolvam um grande número de troca de mensagens, tal como

acontece no sistema de recomendação que quisemos implementar. Além disso, esta ferramenta é bastante escalável, visto que permite aos seus utilizadores fazer a criação de múltiplos tópicos com múltiplas partições, cujas mensagens podem ser entregues e consumidas por diferentes produtores e consumidores (Figura 13). Os dados que são enviados através das mensagens permanecem em disco, o que permite uma durabilidade elevada dos dados.

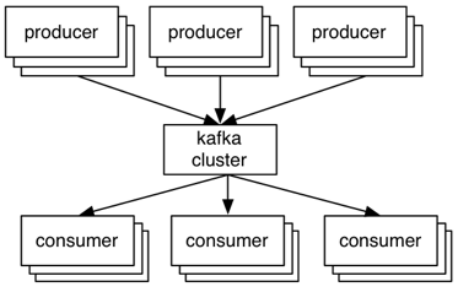


Figura 13 - Funcionamento e arquitectura da *framework Apache Kafka (Kafka, 2014)*

Um dos *use-cases* referenciados na página do *Apache Kafka* pode ser incorporado no desenvolvimento do núcleo deste projeto, em particular na implementação dos serviços de *tracking* das actividades dos clientes na aplicação, ou seja, na criação e tratamento de um *log* de atividades bastante completo, gerado através da interação dos diversos utilizadores com a aplicação. Devido a características como estas, optámos por escolher o Apache *Kafka* para fazer a gestão dos ficheiros de *log* de atividades.

3.5 O Apache Flink

Depois de se obter os dados sobre as atividades desenvolvidas na aplicação, passa-se ao seu tratamento. Para isso foi necessário recorrer a uma outra ferramenta, especialmente orientada para esse fim e que permitisse a sua realização quase em "tempo real". Tal como anteriormente,

realizámos uma breve análise a duas ferramentas: o *Apache Flink* e o *Apache Spark*. Com base nessa análise foi possível concluir que ambas as ferramentas têm bastantes semelhanças entre si, tais como: o processamento dos dados, a realização de *queries* (com o intuito de filtrar os dados de forma mais eficiente), a aplicação de modelos de *machine learning* e o processamento de dados em *streamming*. Contudo, deve-se referir que, apesar das suas semelhanças, o *Apache Spark* é mais estável. O *Apache Flink* é uma ferramenta mais recente, apesar de já ter uma forte comunidade de utilizadores à sua volta, tendo já os seus próprios eventos de demostração. Na nossa opinião, este último ponto é bastante importante, uma vez que a arquitetura do sistema de recomendação implementado foi desenhada tendo este ponto em consideração, de forma a que tivesse a possibilidade de realizar a troca de informação em tempo real. É, pois, neste caso que se encontra a principal diferença entre estas duas ferramentas. O *Apache Spark* foi desenhado e optimizado com o objectivo de permitir grandes cálculos em *batch*, ou seja, apesar de também permitir o processamento em *streamming*, o seu ponto forte é o processamento de dados em *batch.*

3.6 O Hadoop (HDFS)

Um dos principais objectivos da construção do sistema de recomendação era guardar os dados em formato bruto (sem qualquer tipo de processamento efectuado) de forma a ter os dados captados todos guardados num único sítio. Desta forma, seria possível, um dia, realizar análises mais minuciosas dos dados angariados, caso a empresa necessitasse. Para isso, foi necessário recorrer a alguma(s) ferramenta(s) que permitissem guardar os dados sem afectar a *performance* da aplicação e que disponibilizassem um fácil acesso aos dados. Além disso, era imperativo que também fosse possível guardar um volume enorme de dados, uma vez que estamos a lidar com um sistema que é utilizado diariamente por centenas de pessoas, que irão enviar vários dados várias vezes ao dia. Uma base de dados dita convencional não seria, assim, muito recomendável para suportar tais situações e manusear um tão grande volume de dados. Tendo todos esses aspetos em consideração, a nossa escolha caiu sobre o *Hadoop (HDFS - Hadoop Distributed File System),* uma vez que esta

ferramenta disponibiliza uma *store* de dados (em formato de ficheiro) gerido de forma eficiente. Com esta ferramenta os dados são guardados em *file-system* o que permite uma mais fácil ingestão de dados, e de forma individualizada, uma vez que os utilizadores podem definir como querem ver os seus dados guardados – e.g. por ano, por mês, por dia, por hora, etc. -, diminuindo assim o tratamento o a aplicação de *queries* adicionais. Além disso, o *HDFS* permite, ainda, guardar uma quantidade enorme de dados sem afectar a performance de outras tarefas no sistema, algo que sempre foi um dos requisitos básicos do sistema de recomendação a implementar.

3.7 O Elasticsearch

O último serviço a ser implementado no âmbito deste projeto foi o *ElasticSearch*. O *ElasticSearch* "*is a search index database*", isto é, uma base de dados que permite realizar processos de procura sobre um grande número de registos de forma extremamente eficiente. Porém, o principal objectivo desta ferramenta não é guardar os dados, mas sim permitir aos seus utilizadores fazer uma procura eficiente e elaborada com base nos seus diversos *use-cases*, permitindo-lhes fazer a realização de *queries* de forma bastante simples e obter os dados pretendidos praticamente em *real-time*. O *ElasticSearch* permite a análise e o estudo de dados não-estuturados e semi-estruturados, com o intuito de fornecer aos seus utilizadores uma forma completamente diferente de visualizar os dados consultados. Isto só é possível graças a uma segunda ferramenta que vem incorporada com o *ElasticSearch*: o *Kibana*. Além disso, o *ElasticSearch* permite realizar um conjunto enorme de análises de forma a enquadrar-se com aquilo que o utilizador necessita, como por exemplo recomendações de *marketing*, detecção de fraude (uma das principais áreas em que a *WeDo Technologies* se debruça), agregações de registos, e outras coisas mais. As potencialidades desta ferramenta revelam-se em processos de pesquisa, exploração e visualização de dados. A utilização ferramenta neste projecto justifica-se pela sua eficácia na entrega dos dados ao *endpoint* encarregue de se comunicar com o *RAID,* mas principalmente, pela possibilidade de realizar processos de análise em formato visual sobre as recomendações sugeridas pelo sistema construído para os seus utilizadores.

Capítulo 4

O Sistema de Recomendação

4.1 A Captação dos Dados

Para que fosse possível fornecer ao utilizador recomendações acerca dos seus processos de interação com o produto foi necessário proceder à captura da informação relacionada com as acções desenvolvidas com a aplicação. O *RAID* é um produto de software de grande dimensão o que dificultou e demorou a proposta e desenvolvimento da solução de monitorização das acções realizadas pelos utilizadores. Como não havia qualquer tipo de *log* implementado, tivemos que definir que tipo de conteúdo iríamos guardar e quais os elementos de dados a manter. Porém, com a enorme quantidade de dados que o sistema *RAID* gera e com a quantidade de pequenas funcionalidades que este permite foi necessário reduzir as secções de gráficos, listas de dados e *infographics*, de forma a que fosse possível simplificar a captação de dados e obter um resultado final. O sistema *RAID* não foi idealizado e contruído de forma a incorporar este tipo de ferramenta de monitorização. Os seus processos de acesso a dados não foram feitos de forma genérica, o que dificultou a tarefa de captação dos dados. Assim foi necessário encontrar o equilíbrio entre a redução de secções do sistema *RAID*, a quantidade e a qualidade do *log* de ações desenvolvidas pelos

utilizadores do sistema*, para que pudéssemos* ter um conjunto de dados de monitorização de atividades capaz de ser tratado e que gerasse um resultado final de acordo com as nossas expectativas. Nesse sentido, procedemos à captação de dados de monitorização de atividades desenvolvidas em três das secções (*gadgets*) mais utilizadas do sistema *RAID*, nomeadamente:

- *Data List* – que é responsável por apresentar os dados dos clientes em formato de tabela.
- *Chart*: *gadget* – que é responsável por apresentar os dados dos clientes em formato de gráfico.
- *Infographics* – o qual é responsável por apresentar os dados dos clientes em formato de cartões (com diversos formatos de apresentação), permitindo a visualização dos dados de uma forma mais simples e apelativa.

Todos estes elementos permitem realizar operações de filtragem de dados, consoante os valores de *output*, ou formatações específicas sobre os dados obtidos, ou ações de ordenação pelo um dado campo de dados. A estrutura dos dados que decidimos angariar tem o seguinte formato:

- **Informação do Evento**:
  - ***ID*** – Identificador único.
  - ***Username*** - Nome do Utilizador.
  - ***Timezone*** - Fuso horário.
  - ***Server Timezone*** - Fuso horário do servidor - útil para realizar comparações entre comparação entre o fuso horário do servidor e do cliente, de forma a realizar as recomendações com base na *timezone* do utilizador e não do servidor.
  - ***ClientIP*** - IP do cliente.
  - ***DeviceType*** - Tipo do dispositivo utilizado pelo Utilizador – com o intuito de saber com que tipo de dispositivo os utilizadores visitam o produto.

- ***Data***:
  - ***Timestamp*** - Horas (com ano, mês, dia, horas, minutos e segundos) e ***DayOfTheWeek*** (dia da semana).
  - ***LogicalFields*** - Campos da base de dados (de qualquer tipo: *string, integer, boolean, double, ect...).*
  - ***ConditionalFormattings*** - Formatações especiais aplicadas aos dados através de condições.
  - ***Breadcrumb*** - Dados da navegação do utilizador.
  - ***Filters*** - Filtros aplicados aos diversos gadgets (estes filtros consistem maioritariamente em condições aplicadas aos campos dos diversos componentes).
  - ***Gadget*** - Informação do *gadget* do produto.
  - ***ClickedData*** - Informação dos dados clicados.

A informação acima representa a informação do evento mencionado anteriormente (secção 1.1), em que temos a informação do evento que é referente ao *header* e o campo *data* que representa o *body* do evento.

A arquitectura da captação dos dados (Figura 14 - Arquitectura da aplicação responsável pela captação dos dados) permite mostrar como foi realizada a entrega dos dados. Como já referido, era importante que este software de monitorização de atividades não provocasse qualquer tipo de *delay* no desenvolvimento do sistema *RAID*, de forma a que o utilizador não sentisse o efeito de qualquer sobrecarga computacional que pudesse ser gerada por a nova aplicação de monitorização de atividades. Para que isso fosse garantido utilizámos a *API Web Worker* que permite trabalhar os dados paralelamente ao funcionamento da *thread main* que o produto *RAID está a* correr. Quanto à *IndexedDB* esta surge para atuar em *fallback,* em caso de ocorrer uma falha no envio dos dados para o servidor, o que nos permite evitar a perda de informação importante que eventualmente possa acontecer durante um processo de interação do cliente com o sistema *RAID*.

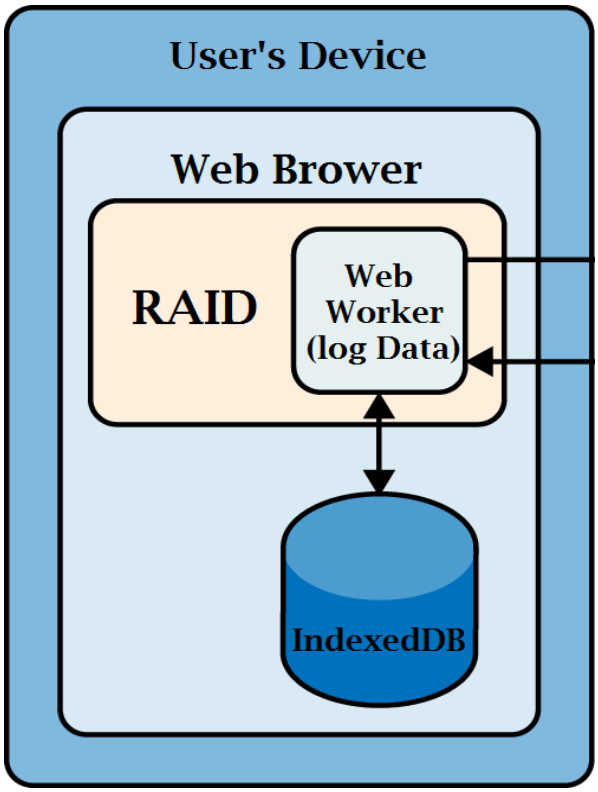


Figura 14 - Arquitectura da aplicação responsável pela captação dos dados

Ao longo do desenvolvimento da aplicação de monitorização, as principais dificuldades sentidas na captação dos dados passaram, principalmente pela forma como estes estavam definidos no sistema *RAID*. Apesar dos *gadgets*, mencionados anteriormente, terem bastantes semelhanças os objectos que os acolhem são sempre diferentes, desde o nome das variáveis e configurações até à sua estruturação. Isso dificultou e atrasou bastante o processo de concepção de um objecto uniforme para o acolhimento de qualquer um dos *gadgets* referidos. Para isso foi necessário analisar de forma bastante exaustiva o código fonte de cada uma das suas instanciações. Isso, obviamente, acabou por condicionar o resultado final.

4.2 Arquitectura da Aplicação (*NodeJS, ElasticSearch, Flink, Kafka, HADOOP HDFS*)

Após termos terminado o processo de captação dos dados, bem como a construção de um objecto uniformizado para representação e acolhimento dos vários *gadgets* para monitorização, passámos à implementação do sistema responsável pela geração de recomendações, com base nos dados angariados. Como já referenciado, uma das nossas principais preocupações associadas com este sistema era o funcionamento em paralelo deste sistema com o sistema *RAID*, que não deveria ser reflectido de alguma forma no desempenho do sistema *RAID*. Tendo isso em consideração desenhámos uma arquitectura particular para o sistema de recomendação (Figura 15 - Arquitectura do sistema de recomendação e seus).

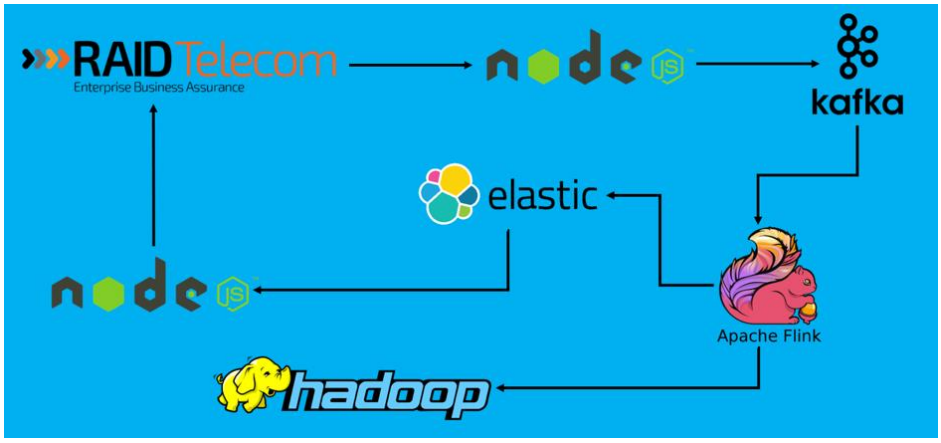


Figura 15 - Arquitectura do sistema de recomendação e seus intervenientes

Nessa arquitetura podemos ver que o sistema apenas realiza duas conexões ao sistema *RAID*, nomeadamente, uma para entregar os dados e outra para consumir os dados, com as respectivas recomendações geradas. Além disso, é importante referir que toda a arquitectura da aplicação foi desenvolvida sobre o princípio das EDA (*Event-Driven Architecture*) e todas os serviços utilizados têm como objectivo tratar os eventos gerados a partir da interação que o utilizador tem com o

produto *RAID*. Como podemos verificar pela Figura 15, as duas conexões ao sistema *RAID* foram realizadas recorrendo a dois servidores montados em *NodeJS*. Estes servidores foram desenvolvidos e montados de forma autónoma e instalados em máquinas diferentes para que não existisse qualquer sobrecarga. Tudo isto para assegurar de forma eficiente as várias comunicações entre a *thread* do *worker* e o servidor responsável pela ligação existente entre o *RAID* e a *framework Apache Kafka*. O *Kafka* permite-nos realizar o *streaming* dos dados (em formato de *string*) entre o servidor e o consumidor dos dados, que é responsável pelo seu tratamento e pela geração da recomendação final. Os dados são enviados em formato *string* (texto) para o *Kafka*, já que este apenas permite trabalhar com este tipo de dados. Isto acontece, porque o Kafka trabalha sobre uma arquitetura de mensagens e, para que estas sejam entregues de forma o mais eficiente possível o sistema requer esse tipo de formato de dados.

A ferramenta utilizada e responsável por consumir os dados provenientes do *Kafka* foi o *Apache Flink*. Esta ferramenta assegura as tarefas *core* do sistema de recomendação desenvolvido. Para que isto acontecesse, primeiramente, realizámos a conexão ao sistema *Kafka,* de forma a consumir os dados enviados, recorrendo aos conectores fornecidos pelo *Flink,* que permitem realizar conexões a variadíssimas ferramentas sem grande esforço. Além desta conexão, foram estabelecidas mais duas, em particular:

- ao *HDFS*, para permitir que os dados fossem guardados em formato bruto sem qualquer tipo de tratamento, com o intuito de no futuro se conseguir executar *queries* mais complexas - a própria empresa poderá ter interesse em ter num lugar todos os dados da interação dos clientes com a sua aplicação, por exemplo recomendações gerais para clientes que entrem no produto pela primeira vez, agregações e *queries* que sejam computacionalmente mais exigentes ou tarefas onde seja necessário um tratamento especial dos dados.
- ao *elasticSearch*, com o objectivo de obter os dados o mais rapidamente possível, assim que o servidor faça o pedido com a *querie* desejada.

Após estarem completas todas as tarefas de consumo dos dados obtidos pelo *Kafka,* é realizado o envio dos dados para o *HDFS*, aqui é, também, executado o processamento e *parse* dos dados. Esta

tarefa foi realizada para reduzir a informação enviada, construindo, assim, um "objecto tipo" mais reduzido comparativamente com o inicial, com o intuito de simplicar a recomendação a gerar. Depois, realizámos a implementação do sistema de recomendação. Aqui sentimos algumas novas dificuldades, uma vez que os modelos do *Apache Flink* estão maioritariamente disponíveis para a linguagem *Scala*. Por isso chegou-se à conclusão que o melhor seria utilizar uma ferramenta externa. Esta decisão acabou por trazer novas dificuldades relativamente à forma como incorporar os modelos de *machine learning* na arquitectura já desenvolvida. Por isso e com o intuito de simplificar o desenvolvimento em curso, já atrasado face ao planeado inicialmente, acabámos recorrer a uma ferramenta de fácil implementação, que não requeria grande tratamento de dados adicional, o *Weka* – no próximo capítulo poderão ser encontradas as decisões tomadas sobre a sua escolha e o tratamento adicional que foi necessário aplicar aos modelos de *machine learning*. Por fim construímos uma página no sistema *RAID*, de forma a permitir aos utilizadores visualizar as respectivas recomendações. Mais tarde, desenvolvemos ainda o serviço responsável por permitir aos vários utilizadores fornecerem *feedback* acerca das recomendações que lhes foram fornecidas. Além disso, de forma a visualizar os dados no mesmo formato com que foram visitados, recriámos as diferentes visualizações dos *gadgets* já citados (*charts*, *datalist* e *infographics*) de forma muito semelhante ao dos conteúdos visualizados.

Capítulo 5

Mineração de Dados e seus Resultados

5.1 O Processo de Mineração

Após a conclusão da captura dos dados e seu respectivo tratamento, passámos à implementação do sistema de recomendações. Porém, logo no início desta nova fase do trabalho deparámo-nos com algumas dificuldades na utilização *FlinkML*, o componente do Apache *Flink* responsável pela aplicação dos algoritmos de *machine learning* sobre os dados de monitorização que foram angariados. Contudo o *FlinkML* está pouco desenvolvido para ambientes *Java*, sendo mesmo quase um exclusivo para para ambientes da linguagem *Scala*. Como grande parte do desenvolvimento da aplicação *(Kafka, Flink, Hadoop)* já estava realizado em *Java*, optámos por recorrer a outras *frameworks* externas para construir as recomendações para cada um dos utilizadores monitorizados. Refira-se que, nesta altura do projeto, o tempo disponível para o desenvolvimento do sistema já era bastante escasso devido a atrasos ocorridos no desenvolvimento e implementação dos processos de captação dos dados.

As desvantagens de recorrer a ferramentas externas são principalmente provocadas pela necessidade de fazer um tratamento extra aos dados para que estes estejam de acordo com os requisitos das próprias ferramentas. Tendo isto em consideração, decidimos utilizar uma ferramenta *open-source*, de fácil acesso, que permitisse fazer a aplicação de modelos de *machine learning* sobre os dados de monitorização que angariámos sem ser necessário recorrer a um esforço computacional exagerado ou a um tratamento demasiado exigente dos próprios dados. Após uma breve análise das ferramentas que tínhamos à nossa disposição, e de forma a avançar o mais rapidamente possível com o desenvolvimento do projeto, optámos pela ferramenta *Weka*, que consideramos uma ferramenta de *Machine Learning* de simples utilização e de fácil acesso. Além disso, esta ferramenta tem uma *API* externa para *Java* que permite conectar-se ao *Hadoop*, o que evitou a procura de uma outra ferramenta que servisse como elo de ligação entre o *Hadoop* e a própria ferramenta de *Machine Learning*.

A primeira fase da implementação do processo de mineração dos dados envolveu o tratamento dos dados de forma a estes serem interpretados pelos mecanismos de mineração da ferramenta *Weka*. Para isso, foi necessário estruturar os dados, com um formato semelhante ao *CSV*, no qual os diversos elementos de dados estão separados por vírgula. Além disso, foi necessário também colocar no cabeçalho do ficheiro os respectivos valores encontrados em cada um dos campos, para facilitar a conversão do texto dos diversos campos para inteiros (sistema utilizado pelo *Weka* para optimizar a respectiva recomendação). Todo este tratamento foi realizado recorrendo a métodos disponibilizados pela ferramenta, de forma a perder o mínimo de *performance* possível. Após terminada esta primeira fase passámos para a fase de análise de dados e implementação dos modelos de *machine learning* definidos. É importante referir que neste processo foram implementados dois modelos de análise diferentes, de forma a que fosse possível realizar uma análise comparativa. Isso permitiria verificar a correção da implementação do modelo e se os resultados obtidos iam de encontro ao esperado, quer estes fossem os resultados do processo de recomendação quer fossem os resultados acerca do desempenho do sistema.

O primeiro modelo que implementámos foi o de árvores de decisão - *decision tree*. Iniciarmos o nosso trabalho de análise com este modelo não foi resultado de uma escolha aleatória. Os dados

que obtivemos e o tipo de recomendação que pretendíamos obter determinou essa escolha. Além disso a fácil implementação dos seus resultados no sistema de recomendação foi também algo que contribui para a utilização do modelo baseado em árvores de decisão em primeiro lugar. Este modelo permite, também, uma maior compreensão dos resultados obtidos, quando comparados com outros mais complexos. Este modelo permite fazer a identificação de *outputs*, ou seja, com base nos dados treinados anteriormente conseguimos identificar os cenários com maior probabilidade de ocorrerem através das suas árvores de decisão. Normalmente, este tipo de modelo permite uma abordagem típica de decisão binária, isto é, assenta num conjunto de questões, com uma simples resposta de de sim ou não, ou simplesmente permitindo uma opção entre vários cenários definidos A, B ou C. No nosso caso, optámos por esta última modalidade, em que se pergunta no contexto do modelo de decisão qual o cenário com maior probabilidade com base nos dados treinados *(Sunlin, 2015)*. Após a conclusão da implementação do primeiro modelo de mineração escolhido, passámos à implementação de um segundo modelo, com o objectivo de alcançar um conjunto de elementos que nos permitisse analisar os dados de forma comparada. O segundo modelo implementado foi definido para *Logistic Regression*, que, na nossa opinião, é mais robusto que o modelo implementado anteriormente, uma vez que permite um tratamento estatístico mais poderoso que o anterior, permitindo através de dados binomiais calcular as diversas probabilidades de *output* com base num ou em vários cenários *(Isinkaye, Folajimi, & Ojokoh, 2015)*.

O modelo baseado em *Logistic Regression*, pelo seu mecanismo interno, é mais preciso que o modelo baseado em *Decision Trees.* Todavia, impõe uma carga computacional maior, devido ao nível da sua precisão no processo interno de cálculo que desenvolve. Com tal, acaba por ser mais lento que o primeiro modelo implementado. Na implementação de cada um destes modelos os dados foram processados sabendo que o utilizador pode fornecer a indicação que a recomendação gerada não foi de acordo com o aquilo que estava à espera, é então realizado um esforço extra para que com base no seu *feedback* as próximas recomendações sejam alteradas (quer seja problema com os dados, quer seja de visualização dos mesmos) baseando-se sempre no padrão de uso do utilizador.

5.2 Análise de Resultados

Após termos completado a implementação dos dois modelos de mineração de dados foi preciso analisar os resultados obtidos em cada um deles. De referir que neste processo de análise foram considerados 1404 registos de atividades de 6 utilizadores diferentes, contendo informação muito variada sobre a utilização do produto e das suas funcionalidades. Tal, permitiu-nos retirar um conjunto muito diverso (e precioso) de informação sobre a utilização do sistema *RAID*, rentabilizando-se assim a implementação do sistema de recomendação implementado. Na Tabela 1 podemos ver uma recomendação que foi gerada para dois utilizadores – 'Ricardo Faria' e 'Pedro Pires' – que têm o mesmo tipo de output, quer para modelo de *Decision Tree* quer para o modelo de *Logistic Regression*. Para estes casos, é importante que se refira que não foram encontradas quaisquer discrepâncias entre os resultados de ambos os modelos, para os diversos utilizadores analisados. Contudo, dado o grau de simplificação dos dados utilizados, não podemos garantir que não existam alguns falsos resultados no leque apresentado.

Tabela 1 - Representação dos dados obtidos

Utilizador	***Varíaveis***	***Decision Tree***	***Logistic Regression***
Ricardo Faria	LogicalFieldValue	PLAYER Player Name PLAYER_ATTRIBUTES Potential	• PLAYER Player Name PLAYER_ATTRIBUTES Potential
	LogicalFieldAgrregation	None Avg (Average)	None Avg (Average)
	LogicalFieldScopeId	151cc530-0f87-4e0e-a845-1153ee4ed79c	151cc530-0f87-4e0e-a845-1153ee4ed79c
	LogicalFieldsOrder	{Player Name} asc	{Player Name} asc
	BaseResAdvanced	None	None
	BaseResField	None	None

	BaseResFieldType	None	None
	BaseResOperator	None	None
	BaseResValue	None	None
	GadgetClass	Infographics	Infographics
	GadgetType	indicator-renderer-table-card	indicator-renderer-table-card
	LogicalFieldValue	PLAYER Player Name PLAYER_ATTRIBUTES Dribbling	PLAYER Player Name PLAYER_ATTRIBUTES Dribbling
	LogicalFieldAgrregation	None Avg (Average)	None Avg (Average)
	LogicalFieldScopeId	151cc530-0f87-4e0e-a845-1153ee4ed79c	151cc530-0f87-4e0e-a845-1153ee4ed79c
	LogicalFieldsOrder	{Player Dribbling } asc	{ Player Dribbling } asc
	BaseResAdvanced	None	None
	BaseResField	None	None
	BaseResFieldType	None	None
	BaseResOperator	None	None
	BaseResValue	None	None
	GadgetClass	Infographics	Infographics
	GadgetType	indicator-renderer-arrow	indicator-renderer-arrow

Tal como já foi mencionado anteriormente, uma das principais optimizações realizada no tratamento dos dados passou por dividir estes mesmo por hora e na Tabela 2 podemos visualizar a performance obtida para esses casos. Além disso, estão, ainda, presentes as diferentes horas (mais o total de registos, ou seja, para todas as horas juntas) e a respectiva diferença de tempos obtida para ambos os modelos.

Tabela 2 - Tabela com os dados retirados para medir a performance dos dois modelos (DT e LR)

Hora	***Nº Records***	***Tempo (DT) (ms)***	***Tempo (LR) (ms)***	***% Ganho do modelo DT***
8	195	86,837674	1152,730581	≈ 92%
9	113	75,327377	904,605871	≈ 92%
10	238	90,307782	1232,457267	≈ 93%
11	21	67,022692	765,295044	≈ 91%
12	144	77,958903	1028,817886	≈ 92%
13	65	73,400837	863,388441	≈ 91%
14	220	88,958132	1189,407162	≈ 93%
15	232	89,262271	1200,324761	≈ 93%
16	176	82,047015	1048,047293	≈ 92%
Total	1404	146,78933	11465,61363	≈ 98%

Analisando os dados apresentados na Tabela 2 e nas Figura 16 e Figura 17 é possível verificar que o modelo de *Logistic Regression* acaba por ser mais demorado que o modelo em comparação, isto porque, e tal como foi referido no capítulo anterior, este modelo por ser mais preciso nos cálculos que efectua acaba por trazer consigo uma carga computacional mais elevada, acabando, assim, por afectar a sua *performance* comparativamente com o modelo de *Decision Tree,* de notar que todos os dados da tabela indicada acima foram medidos em milissegundos.

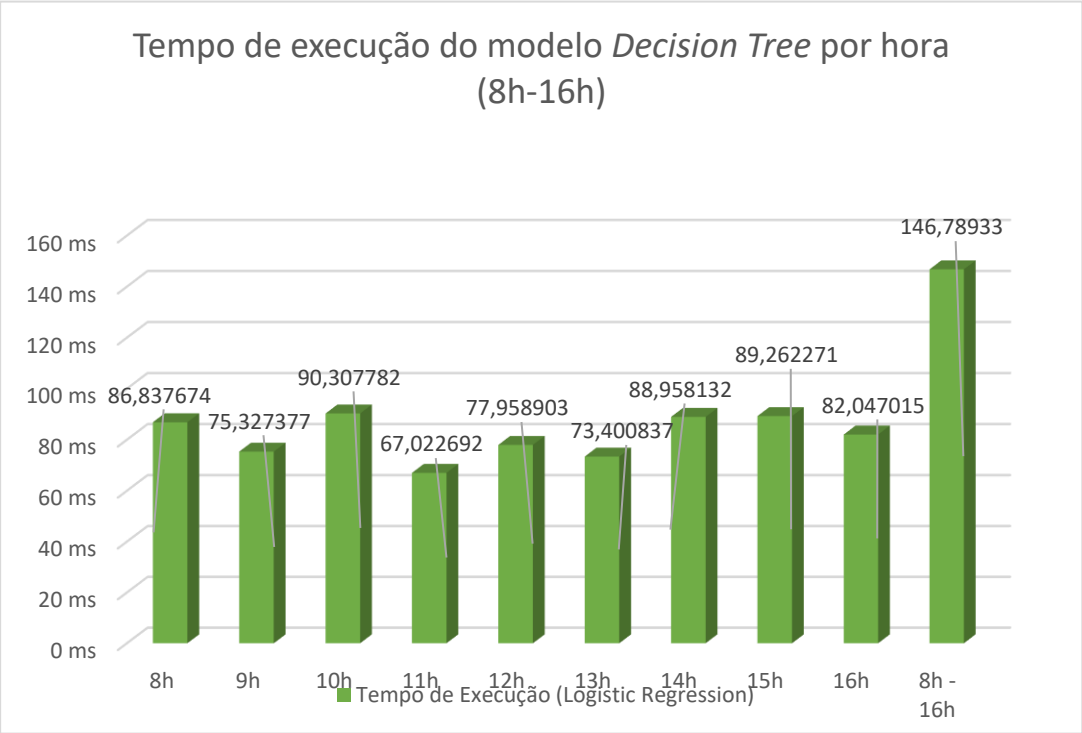


Figura 16 – Tempos de execução obtidos com o modelo de *Decision Tree*

Além disso, é, ainda, possível observar a percentagem de ganho que o modelo de *Decision Tree* tem sobre o de *Logistic Regression* que em média ronda os 92%, isto acontece, uma vez que a recomendação que queremos obter é muito simples, ou seja, a precisão de cálculos que nos é

fornecida pelo modelo *Logistic Regression* acaba por não compensar num *use-case* tão simples como é aquele que temos em mão.

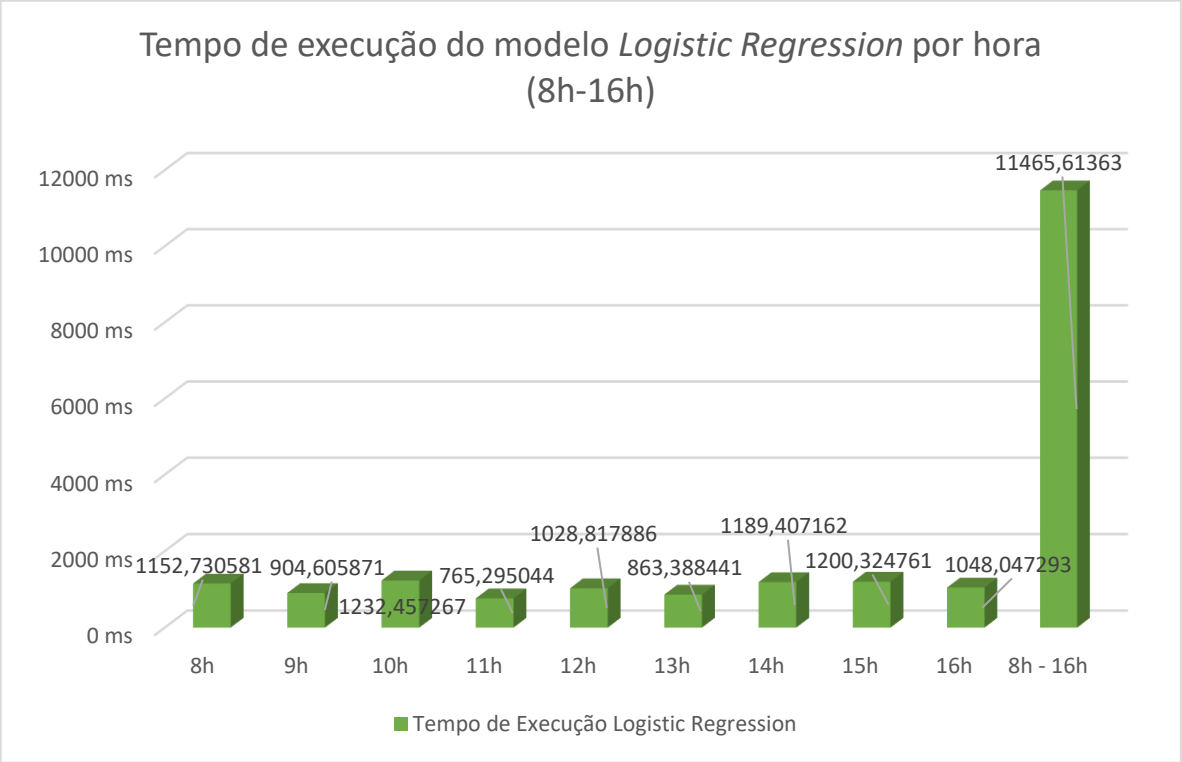


Figura 17 - Tempos de execução obtidos com o modelo de *Logistic Regression*

Além do cálculo do desempenho individual, para cada hora, foi ainda realizada uma análise comparativa do ganho obtido com a divisão dos registos colectados, por hora. Basicamente, realizámos uma média relativa a quanto tempo se perde ao se analisar os diferentes registos de cada hora. O cálculo desse valor médio foi realizado multiplicando o número de registos de uma determinada hora (p.e. 238) pelo valor que foi obtido por modelo para o número total de registo (p.e. 90,307782 *(DT)*), a dividir pelos 1404 *logs* obtidos ((238 * 90,307782)/1404). Este cálculo dá-nos a média de quanto é gasto pelos 238 registos quando é aplicado a todos os registos. De salientar que este valor apenas serve como um valor de referência, uma vez que existem vários factores que

podem contribuir para a quebra de desempenho, como por exemplo, a informação contida nos dados ou até mesmo a diversidade de configurações que foram encontradas para aquele período de tempo.

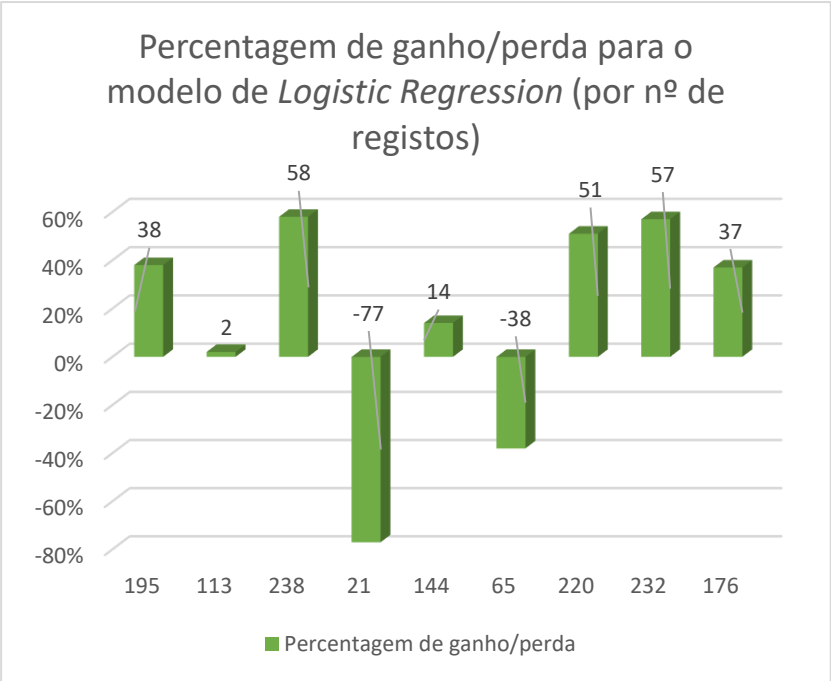


Figura 18 - Percentagem de ganho e de perdas obtidos com o modelo de Logistic Regression

Na Figura 18 podemos ver o ganho obtido com o modelo baseado em *Logistic Regression*. De notar que, neste caso, apenas não é obtido o ganho para os registos que contêm o menor número, 21 e 65 respetivamente. Isto acontece porque o tempo que o modelo demora na fase de preparação dos dados, e respectiva recomendação, para um número tão pequeno de dados, acaba por não compensar relativamente ao cálculo efectuado anteriormente. Além disso, verificamos que, mesmo para o valor de 113 registos, o ganho que é obtido é quase nulo. Porém, para um número de registos mais elevado conseguimos obter um ganho considerável, uma vez que o tempo que o modelo demora para calcular as diferentes probabilidades compensa, já que o cálculo requerido para o número de registos inicial (1404) acaba por afectar o seu desempenho.

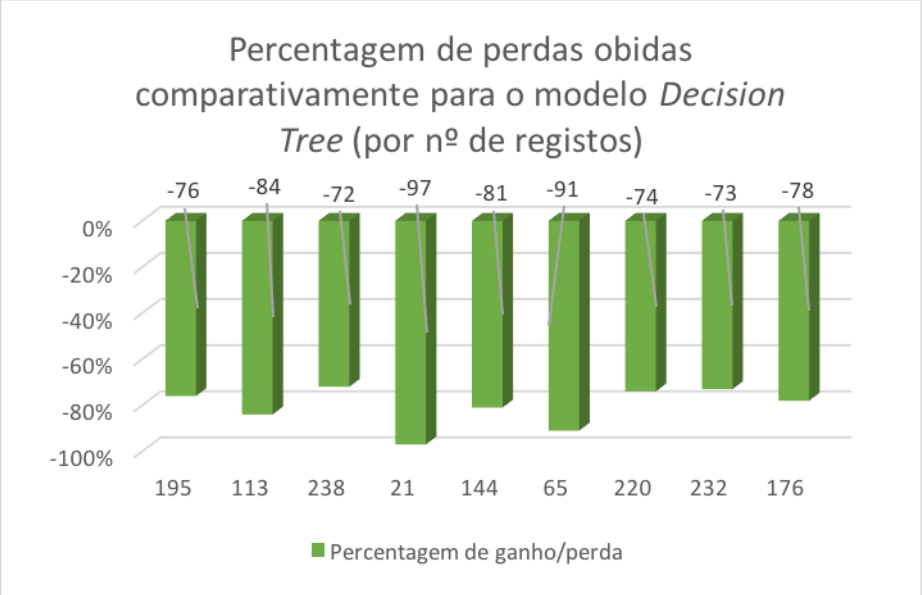


Figura 19 - Percentagem de perdas obtidas com o modelo de *Decision Tree*

Na Figura 19 podemos observar o ganho que foi obtido comparativamente com o valor calculado anteriormente e observarmos não só que não temos nenhum ganho, mas também que as percentagens relativamente às de perdas obtidas são elevadas. Este resultado foi obtido graças à forma como o modelo foi construído. Uma vez que este modelo tem um formato em árvore, a maior parte do tempo que a execução deste modelo consome não é no cálculo das probabilidades, tal como acontece no modelo baseado em *Logistic Regression*, mas sim na atribuição de cada valor a cada nodo. Por isso é que para valores tão pequenos, como 21, obtemos uma percentagem tão elevada. Assim, com base na análise nos dados apresentados nas Figura 18 e Figura 19, podemos concluir que o modelo baseado em *Decision Trees* acaba por apresentar melhores resultados que o modelo baseado em *Logistic Regression,* em situações nas quais o número de registos é mais elevado. Enquanto que o modelo baseado em *Logistic Regression* foi aquele que obteve melhores

percentagens de ganho, com a divisão dos registos por hora, reduzindo assim o seu volume e permitindo uma redução nos cálculos efectuados, o que conduziu um melhor desempenho.

Apesar dos benefícios obtidos com divisão dos registos por hora, isso acabou por trazer também uma desvantagem no processo de recomendação gerada a longo prazo, uma vez que nestes casos serão sempre considerados registos antigos, que o utilizador simplesmente deixou de os considerar. De forma a melhorar o processo de recomendação final seria necessário aplicar um modelo de análise diferente, com o objectivo de atribuir uma maior cotação a registos mais recentes, tal como é realizado noutros sistemas de, como por exemplo no caso da *Weekly Playlist* que é calculada no *Spotify*, em que as músicas mais recentes ouvidas "recebem" uma maior cotação daquelas que foram ouvidas anteriormente.

Tabela 3 - Medição do desempenho para ambos os modelos implementados

Hora	*Nº Records*	*Tempo (DT)*	*Tempo (LR)*	*Tempo Comparação (DT)*	*Tempo Comparação (LR)*	*% Ganho (DT)*	*% Ganho (LR)*
8	195	86,837674	1152,730581	20,38740694	1592,446338	≈ -76%	≈ 38%
9	113	75,327377	904,605871	11,81424095	922,8022369	≈ -84%	≈ 2%
10	238	90,307782	1232,457267	24,88309155	1943,601172	≈ -72%	≈ 58%
11	21	67,022692	765,295044	2,195566902	171,494221	≈ -97%	≈ -77%
12	144	77,958903	1028,817886	15,0553159	1175,960373	≈ -80%	≈ 14%
13	65	73,400837	863,388441	6,795802315	530,815446	≈ -91%	≈ -38%
14	220	88,958132	1189,407162	23,00117707	1796,606125	≈ -74%	≈ 51%
15	232	89,262271	1200,324761	24,25578672	1894,602823	≈ -73%	≈ 57%
16	176	82,047015	1048,047293	18,40094165	1437,2849	≈ -78%	≈ 37%
Total	1404	146,78933	11465,61363	146,78933	11465,61363	0	0

5.3 Aplicação dos Resultados Obtidos

Os resultados obtidos foram inseridos no sistema *RAID* para que os possíveis clientes pudessem consultar as recomendações sugeridas pelo sistema. Assim, no menu inicial do sistema *RAID* os utilizadores podem agora encontrar e escolher a página relativa à sugestão de recomendações (Figura 20).

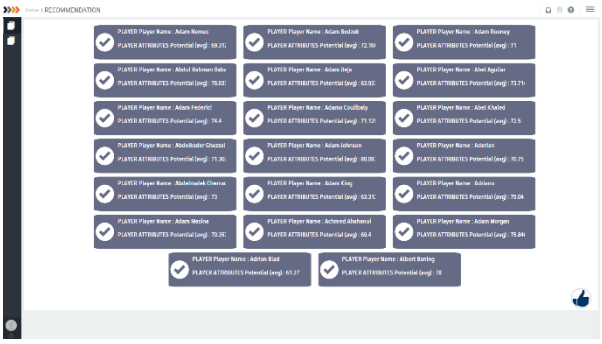


Figura 20 – Página do sistema da recomendação

Nesta página os utilizadores podem fornecer *feedback* acerca da recomendação apresentada. Para isso precisam apenas aceder a esse serviço através de um botão específico – *like*. Através desse botão ativa-se um pequeno *pop-up* que questiona o utilizador se a recomendação apresentada foi de encontro com as suas necessidades. Perante essa questão, o utilizador poderá dar feedback positivo, o que significa que a recomendação está enquadrada com aquilo que ele necessitava, ou negativo, que revela que o utilizador não concorda com a recomendação que lhe foi dada pelo sistema. Esta informação é integrada no sistema para que, em futuras recomendações, este a tenha em consideração Figura 21.

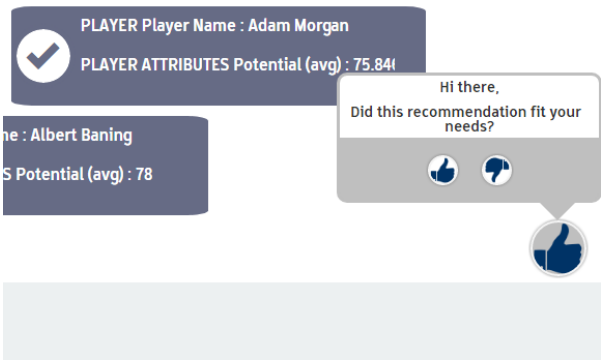


Figura 21 - *Pop-up* de *feedback* que o utilizador quer fornecer

Quando o utilizador seleciona o botão *dislike* o sistema questiona-o (Figura 22) se o problema da recomendação foi visual, a forma como os dados foram fornecidos, ou está relacionado com os dados, se o problema tem a ver com os dados apresentados que não vão de encontro aquilo que o utilizador estava à espera de encontrar, ou se envolve ambos os casos anteriores.

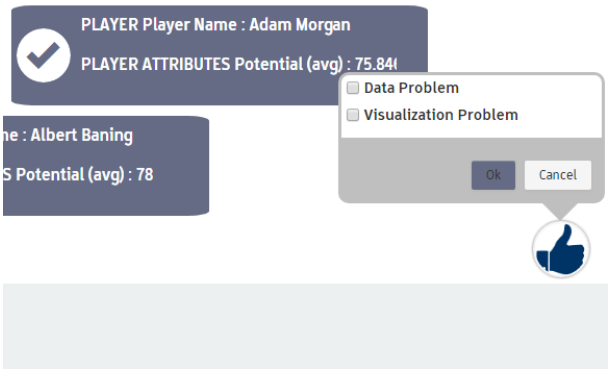


Figura 22 - *Pop-up* de problema de visualização ou conteúdo

Caso o utilizador tenha preenchido a *checkbox* de visualização o sistema questiona-o se este preferiria visualizar os dados em forma de *Chart* (gráficos) ou *Table* (tabela) – (Figura 23).

Figura 23 - *Pop-up* questionando o utilizador de como deseja visualizar os dados

Caso o utilizador diga que preferiria visualizar os dados em forma de gráfico o conteúdo da página é redesenhado, sendo os dados apresentados da forma como a Figura 24 apresenta.

Figura 24 - Apresentação dos dados em forma de gráfico

Todavia, se o utilizador preferisse visualizar os dados em forma de tabela, então o sistema reajusta a visualização dos dados apresentando-os como estão na Figura 25.

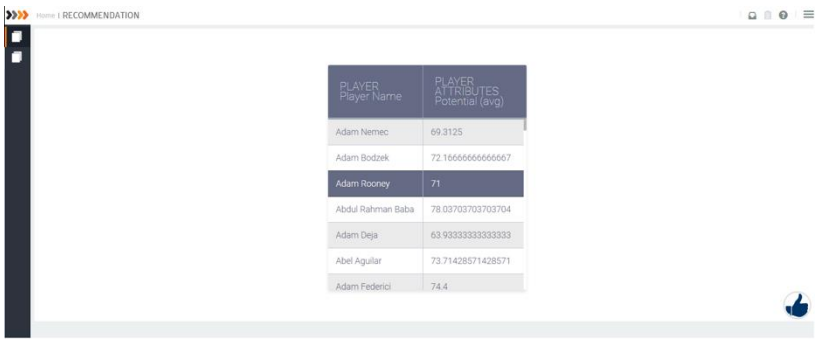


Figura 25 - Apresentação dos dados em forma de tabela

Por fim, o utilizador pode, ainda, indicar ao sistema que são os dados que não estão de acordo ao que ele esperaria (Figura 26):

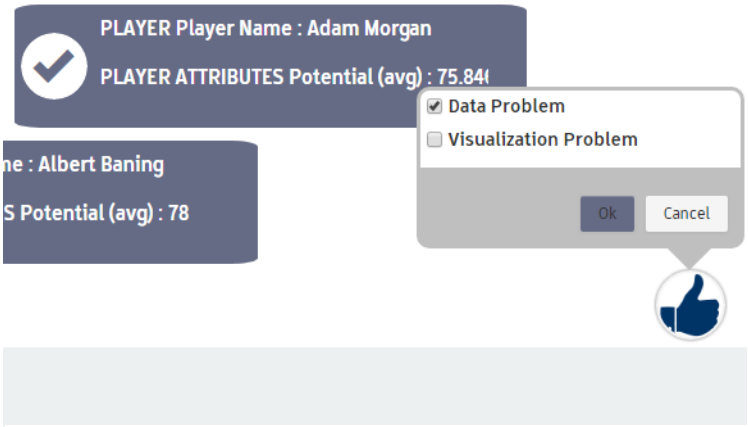


Figura 26 - *Pop-up* em que o utilizador fornece a indicação que o problema são os dados

Neste caso, o sistema irá calcular a segunda recomendação que obteve uma maior probabilidade aquando a aplicação do modelo, apresentando, assim, uma nova configuração com base nos dados calculados (Figura 27).

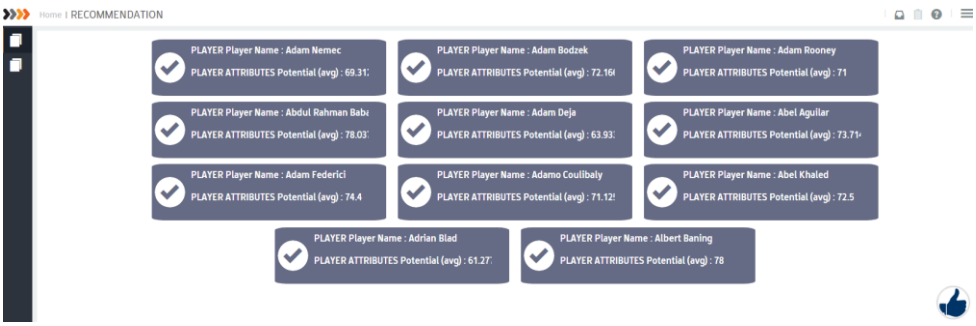


Figura 27 - Apresentação dos novos dados calculados

Capítulo 6

Conclusões e Trabalho Futuro

Após a finalização deste trabalho de dissertação podemos concluir que os resultados alcançados atingiram as expectativas dos responsáveis que lançaram este desafio, apesar dos vários percalços encontrados ao longo do seu processo de implementação. Basicamente, estes percalços foram provocados por problemas relacionados com os próprios desafios, em particular com o processo de implementação do *worker* e da *IndexedDB*. No primeiro caso, tais problemas foram devidas às dificuldades que esta *API* levantou por não permitir efetuar várias das funcionalidades requeridas para o sistema de recomendação e, no segundo caso, com a *IndexedDB* o processo de implementação dos métodos de inserção, remoção e pesquisa foi bastante difícil de forma a que se pudesse conseguir obter uma ligação com o *Web Worker* implementado. Além disso, a própria captação dos dados passou por alguns problemas principalmente devido à forma como os dados estavam estruturados nos diversos *gadgets* que foram alvo de operações de angariação de dados e pela informação que seria necessário guardar posteriormente, de forma a se poder obter uma recomendação que fosse adequada com aquilo que o utilizador estaria à espera. Contudo, o que tinha sido delineado no início dos trabalhos de dissertação acabou por ser ligeiramente modificado, uma vez que surgiram alguns contratempos originados por estarmos a trabalhar com um produto já construído, com alguma história considerável, e aquando da sua construção não houve a necessidade de providenciar o sistema RAID com mecanismos e estruturas adequadas à monitorização das tarefas realizadas pelos seus utilizadores. Isto, fez com que fosse necessário trabalhar um pouco o sistema RAID para o preparar para acolher o novo sistema de recomendação.

Este processo atrasou algumas das ações de desenvolvimento previstas fazendo com que alguns dos objetivos traçados tivessem que ser redefinidos para que no final do projeto fosse possível obter um sistema de recomendação perfeitamente integrado e útil para a análise e otimização da exploração do sistema RAID. Nesta vertente de desenvolvimento podemos afirmar que o resultado obtido foi bastante positivo.

Em termos gerais, os resultados obtidos deste trabalho de dissertação foram bastante positivos. Como prova disso, refira-se o facto dos utilizadores que usaram o sistema de recomendação implementado terem confirmado a utilidade das recomendações sugeridas pelo sistema que, na maioria dos casos, foram de acordo com a navegação que estes tinham realizado. Porém, nem tudo acabou por ser feito. Refira-se, em particular, o caso de alguns dados terem ficado por tratar. Por exemplo, os dados relativos aos processos de navegação pela aplicação não foram tidos em consideração. Estes dados poderiam dar uma ideia para uma restruturação mais precisa do sistema *RAID*, nos aspetos que têm a ver com as pesquisas realizadas pelo utilizador no seu ambiente, ou seja, as áreas que este mais visita ou naquelas em que permanece mais tempo. Todas estas questões foram debatidas com o responsável que supervisionou estes trabalhos no seio da empresa e só assim é que puderam ser tratados, dando origem a outros caminhos de desenvolvimento e soluções para atingir um resultado final que fosse de encontro às expectativas endereçadas pela empresa. O *feedback* que foi obtido após a apresentação do sistema de recomendação foi também bastante positivo, o que fez com que a *WeDo Technologies* afirmasse que pretende dar continuidade ao trabalho até agora realizado, melhorando os aspectos menos positivos que foram referidos ao longo desta dissertação. Para essa posição contribuiu o facto do sistema de recomendação implementado ter atuado como uma bela prova de conceito das ideias e pretensões endereçadas no início dos trabalhos e pelo facto da demonstração da utilidade do sistema quando incorporado no sistema *RAID*, atuando em paralelo, sem constranger a sua regular normal operacionalidade do sistema.

Bibliografia

Amatriain, X. (2014). *Recommender Systems : Collaborative Filtering and other approaches.* Netflix, Investigation. Netflix.

Amazon. (s.d.). *elasticsearch Service - What is Elasticsearch*. Obtido de Amazon (AWS): https://aws.amazon.com/elasticsearch-service/what-is-elasticsearch

Bidelman, E. (2010). *Workers Bascic*. Obtido em 12 de 2017, de html5rocks: https://www.html5rocks.com/en/tutorials/workers/basics/

Brasetvik, A., & (Elastic). (15 de Fevereiro de 2015). *Uses of Elasticsearch, and Things to Learn*. Obtido de Elastic: https://www.elastic.co/blog/found-uses-of-elasticsearch

Frost & Sullivan. (2014). *Global CSP Financial Assurance: Market Share Analysis, Forecast, and Supplier Assessment.* Frost & Sullivan.

GoogleTrends. (2018). *Google Trends*. Obtido de Google Trends: https://trends.google.com

Hortonworks. (25 de Julho de 2012). *THINKING ABOUT THE HDFS VS. OTHER STORAGE TECHNOLOGIES*. Obtido de Hortonworks: https://hortonworks.com/blog/thinking-about-the-hdfs-vs-other-storage-technologies

Hueske, F. (5 de Dezembro de 2015). *Building real-time dashboard applications with Apache Flink, Elasticsearch, and Kibana*. Obtido de elastic.co: https://www.elastic.co/blog/building-real-time-dashboard-applications-with-apache-flink-elasticsearch-and-kibana

Isinkaye, F., Folajimi, Y., & Ojokoh, B. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 261 - 273.

Kafka, A. (2014). *LinkedIn*. Obtido em 10 de 2018, de https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

Kurth, M., J. Scholz, N., & Bhatia, K. (2013). *Market Share: Telecom Operations Management Systems (BSS, OSS and SDP), Worldwide, 2011-2012.* Gartner.

Lusbin, G. (21 de Setembro de 2016). *How Netflix will someday know exactly what you want to watch as soon as you turn your TV on*. Obtido de businessinsider: http://www.businessinsider.com/how-netflix-recommendations-work-2016-9

M. O'Brien, J. (20 de November de 2006). The race to create a 'smart' Google.

Mozilla, D. (2017). *IndexedDB API*. Obtido em 12 de 2017, de https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

Mozilla, D. (2017). *Web Worker API*. Obtido em 12 de 2017, de Developer Mozilla: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

Pasick, A. (21 de December de 2015). *The magic that makes spotifys discover weekly playlists so damn good*. Obtido de QZ: https://qz.com/571007/the-magic-that-makes-spotifys-discover-weekly-playlists-so-damn-good

Qiu, F., & Cho, J. (2006). Automatic identification of user interest for personalized search. *In Proceedings of the 15th International Conference on World Wide Web* (pp. 727-736). New York: ACM.

Sunlin, R. (15 de Janeiro de 2015). *Decision Tree – Simplified!* Obtido de analyticsvidhya: https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/2

Technologies, W. (2017). *RAID - Risk Managment*. Obtido de wedotechnologies: http://ww1.wedotechnologies.com/en/

W3C. (2015). *IndexedDB*. Obtido em 11 de 2017, de https://www.w3.org/TR/IndexedDB