UNIVERSIDADE DO ALGARVE

PLATAFORMA PARA GESTÃO E MONITORIZAÇÃO DE EVENTOS

JOSÉ ISABEL DOS SANTOS

Projeto

Mestrado em Engenharia Elétrica e Eletrónica

Trabalho efetuado sob a orientação de:

Professor Doutor Jânio Monteiro & Professor Doutor Pedro Cardoso

2014

PLATAFORMA PARA GESTÃO E MONITORIZAÇÃO DE EVENTOS

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências bibliográfica incluída.

_____

À minha família, em especial ao meu filho,

Péricles dos Santos

AGRADECIMENTOS

Começo por agradecer primeiramente a Deus que me deu mais uma oportunidade de viver e de continuar com os meus objetivos depois de um trágico acidente que me atirou para uma cadeira de roda durante quase um ano e que me fez ficar dois anos afastado dos estudos. Agradeço a minha família que me apoiou incondicionalmente nos momentos mais difíceis e que me fez acreditar que conseguia vencer. Agradeço também a todos os colegas e professores do Departamento de Engenharia Eletrotécnica da Universidade do Algarve, em especial aos meus orientadores, os Professores Doutores Jânio Monteiro e Pedro Cardoso, por todo o apoio e atenção que me concederam durante o desenrolar deste projeto. E por fim, mas não menos importante, a todos aqueles que direto ou indiretamente me apoiaram durante estes últimos anos e que contribuíram de certo modo, para concretizar este trabalho.

José Santos, Faro, 22 de janeiro de 2014

RESUMO

A popularidade da Internet e a crescente utilização das Tecnologias de Informação e Comunicação permitem atualmente utilizar uma gama variada de terminais para obter os mais diversos tipos de informação, como por exemplo informação sobre eventos que acontecem no dia-a-dia. Mas se por um lado esse tipo de informação se encontra na sua grande maioria disponível online, também acontece que a mesma está tipicamente dispersa por inúmeros locais e plataformas, com indicação pouco clara em relação às datas e não correlacionadas com a localização do utilizador que as pesquisa. Refira-se a este nível e a título de exemplo, a informação que os municípios todos os meses disponibilizam sobre atividades de índole cultural. Talvez por isso, nos deparemos inúmeras vezes com informação sobre eventos aos quais gostaríamos de ter assistido, mas que já se realizaram, ou sobre eventos que verificamos estar a realizar-se, sem que saibamos de que se tratam. De facto, em termos de localização geográfica ou temporal, se por um lado temos uma gama de soluções que nos permitem encontrar um restaurante mais próximo, poucas soluções permitem identificar os eventos próximos. É neste âmbito, que surge o projeto *LifeSpeeder* cujo objetivo é desenvolver uma plataforma que permita localizar com facilidade, determinados tipos de eventos no espaço e no tempo, mas que também seja capaz de apresentar o resultado da pesquisa, de acordo com as preferências de cada utilizador e com informações suficientes sobre o evento, para lhe permitir chegar até o local onde o mesmo está a ser realizado.

PALAVRAS-CHAVE: Base de Dados, Localização Geográfica e Temporal, Eventos, Aplicação Móvel, NoSQL, Sistema de Informação, MongoDB

.

ABSTRACT

The popularity of the Internet and the increasing use of Information and Communication Technologies currently enables the usage of a range of terminals to get several kinds of information, as for example information about events that happen on a day-to-day basis. However, if in one hand this kind of information is mostly available online, it also happens that it is typically dispersed by numerous sites and platforms, commonly unclear regarding the associated dates and not correlated with the location of the user that performs the search. As an example, we can mention the information that the majority of the city halls offer every month on their cultural activities. On the other hand, we find many times information about events that we would like to have participated in, but that have already happened, or events that we verify that are happening without knowing what they are about. In fact, if in terms of geographical location, we have a range of solutions that allow us to find a restaurant near us, very few solutions can identify an event that is happening or will occur near us, geographically or time speaking. In this context, the *LifeSpeeder* project emerged with the goal to develop a platform to easily locate certain types of events in the space and time, being able to show the results of the search, according the user preferences and with enough information to allow him to get to the event site.

KEYWORDS: **Database, Geolocation, Events, Nosql, Information Systems, Mobile Application, Mongodb**

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ACRÓNIMOS

ADSL	*Asymmetric Digital Subscriber Line*
API	*Application Programming Interface,* (Interface de Desenvolvimento de Aplicativos)
App	Termo utilizado para designar aplicações desenvolvidas para dispositivos móveis (telemóveis, smartphones, tablets, etc.)
CC	*Cloud Computing* (Computação na nuvem)
DNS	*Domain Name System*
GPRS	*General Packet Radio Services*
GPS	*Global Positioning System,* Sistema de Posicionamento Global
GSM	*Global System for Mobile Communications*
IP	*Internet Protocol*
MR	Modelo Relacional
NoSQL	*Not Only SQL* (Base de Dados não Relacionais)
LTE	*Long Term Evolution*
RDBMS	*Relational Database Management System*
RPC	*Remote Procedure Call*
SMS	*Short Message Service* (Serviço de Mensagens Curtas)
SMTP	*Simple Mail Transfer Protocol*
SOAP	*Simple Object Access Protocol*
TI	Tecnologias de Informação
UDDI	*Universal Description Discovery and Integration*
W3C	*Word Wide Web Consortium*
Wi-Fi	*Wireless Fidelity*
WiMAX	*Wordwide Interperability for Microwave Acess*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*

1. INTRODUÇÃO

Muitas vezes deparamo-nos com a vontade de localizar certo tipo de evento (e.g., cultural, desportivo, social, académico) quer no tempo quer no espaço. Se é verdade que esse tipo de pesquisa assume especial relevo quando nos deslocamos a uma cidade ou país fora da nossa área de residência, também é verdade que tal interesse não se resume a esse tipo de situações.

Neste âmbito, assume especial relevo a crescente utilização de terminais móveis, que integram num mesmo dispositivo diferentes formas de comunicação, facilitando o acesso aos conteúdos disponibilizados diariamente na Web. Muitos desses terminais móveis incorporam funcionalidades que permitem obter informações correlacionadas com a nossa localização geográfica.

No âmbito do turismo, não só existe a clara necessidade de identificar eventos que estejam relacionados com os interesses pessoais de cada utilizador, mas também existe uma clara necessidade de uma ferramenta que permita dinamizar comercialmente uma série de atividades que de outra forma encontram dificuldade em atingir os grupos de interesse respetivos.

Para desenvolver um sistema de informação capaz de responder a essas necessidades, a sua implementação deve incluir facilidade de utilização, rapidez, mas também permitir obter as informações detalhadas relativas aos eventos. Para além disso, deve apresentar-se essas informações de uma forma estruturada de acordo com as preferências de quem está a realizar a pesquisa.

1.1 CONTEXTO DO TRABALHO

O projeto que recebeu o nome de *LifeSpeeder*, consiste no desenvolvimento de uma plataforma para gerir e monitorar eventos de uma forma simples e rápida. A motivação por detrás deste projeto, deve-se por um lado ao facto de estarmos situados numa das regiões mais turística de Portugal, onde principalmente na época de verão, acontecem inúmeros eventos. Apesar disso, por norma os anúncios dos eventos encontram-se espalhados por diversos locais

e com informações pobres em termos de detalhe, não existindo uma plataforma dedicada ou quando existe é relativamente limitada. Tirando partido das várias tecnologias disponíveis atualmente, pretende-se com este projeto, colocar à disponibilidade do utilizador, uma ferramenta capaz de auxiliá-lo na pesquisa de eventos.

A plataforma *LifeSpeeder* é constituída por:

1. Uma aplicação Web, onde para além de pesquisar os mais diversos tipos de eventos, qualquer utilizador registado no sistema poderá publicar gratuitamente os seus próprios eventos.
2. Uma aplicação para dispositivos móveis, destinada a terminais com sistema operativo Android.

Pretende-se com o desenvolvimento desta plataforma deixar o utilizador a par do que está a acontecer à sua volta, quer em termos espaciais, quer em termos temporais. Ou seja, o objetivo é reunir num único sítio, informações de eventos ou atividades realizados no quotidiano, com todos os detalhes necessários, facilitando assim a localização de eventos com interesse para o utilizador, que ao utilizar o *LifeSpeeder* terá centralizado numa única aplicação as informações mais relevantes, evitando pesquisas em vários locais na Web. Deste modo o utilizador terá acesso a informações de vários tipos de eventos (desportivos, culturais, sociais, académicos, entre outros), que estão ou vão ser realizados num determinado sítio. Para além de eventos, a plataforma poderá também integrar outras informações consideradas úteis como por exemplo, informações de locais de interesse que o utilizador possa visitar, locais de alojamentos (pensões, hotéis), restaurantes, entre outros.

A plataforma deverá contar com um suporte multilingue e oferecer opções de pesquisa avançadas. Dito por outras palavras, as pesquisas na plataforma, deverá ir para além das clássicas, no sentido em que deverá permitir fazer uso das preferências do utilizador para filtrar e personalizar os resultados das referidas pesquisas. As preferências do utilizador podem ser baseadas num determinado tipo de evento (e.g., desporto) ou numa determinada subcategoria de evento (e.g., futebol). Poderá incluir parâmetros relativos à distância (km ou milhas), coordenadas geográficas da localização do utilizador, data, hora, local e idioma do evento, entre outras.

1.2 ESTRUTURA DA TESE

A tese está estruturada da seguinte forma. No Capítulo 1 faz-se a introdução e apresentação do projeto. No Capítulo 2 faz-se o apanhado geral de algumas tecnologias de suporte utilizadas no desenvolvimento da plataforma *LifeSpeeder*, tais como as tecnologias de armazenamento de dados, tecnologias de acesso à Internet, documentos estruturados, *Web Service* e das *Application Programming Interfaces* (API). No Capítulo 3, analisam-se as diferentes soluções de bases de dados que existem atualmente. No Capítulo 4 será apresentado o modelo de dados que foi adotado para a plataforma *LifeSpeeder*. No Capítulo 5 apresentam-se as aplicações Móvel e Web, descrevendo o processo de comunicação e as funcionalidades de cada uma delas. No sexto e último Capítulo são apresentadas as conclusões e trabalhos futuros.

2. ENQUADRAMENTO DAS TECNOLOGIAS DE SUPORTE

2.1 INTRODUÇÃO

Atualmente a produção e o acesso à informação é conseguido por diversos meios, principalmente através dos meios eletrónicos. No entanto, para que tenhamos chegado a este ponto, as tecnologias de suporte, têm-se baseado numa forte evolução tecnológica, tais como nas redes de comunicação, nas ferramentas e nas linguagens de programação, bem como os dispositivos ou terminais utilizados na produção e visualização da informação. Essa evolução fez aumentar a quantidade de dispositivos a aceder simultaneamente a um mesmo recurso assim como o volume de dados gerados e processados por algumas aplicações, a ponto de em certos casos as tecnologias de armazenamento de dados e processamento tradicionais deixarem de ser suficientes ou adequados para lidar com a grande demanda, o que levou alguns profissionais da área das tecnologias de informação (TI) a pensar em novas soluções para contornar essas limitações [1].

Devido à grande diversidade de dispositivos e *software* provenientes de diferentes fabricantes que nem sempre seguem os mesmos padrões, verificou-se inicialmente uma certa dificuldade em relação a interoperabilidade entre sistemas desenvolvidos em plataformas e/ou linguagens de programação diferentes. Para ultrapassar essa barreira de comunicação entre sistemas diferentes, foram propostos novos formatos de dados (e.g., XML) [2] e implementadas novas soluções (e.g., *Web Service*) [3] que permitem atualmente que dois sistemas comuniquem de uma forma normalizada.

Para acompanhar e adaptar-se a essas evoluções tecnológicas, a Web teve que ser repensada, aplicando novos conceitos e metodologias que permitem aproveitar da melhor forma os recursos tecnológicos existentes. O crescimento e sucesso da Web têm contado com o apoio de alguns programadores de *software* e empresas do ramo das TI que têm disponibilizado ferramentas importantes tais como, *frameworks* e APIs que auxiliam os programadores no desenvolvimento de novas aplicações, permitindo deste modo, gastar menos tempo de programação, para acrescentar mais funcionalidades às suas aplicações.

2.2 TECNOLOGIAS DE ACESSO A INTERNET

Ao longo do tempo tem-se verificado uma constante evolução nas tecnologias de acesso à Internet, o que tem tornado cada vez mais fácil, melhor e mais económico o modo de aceder às informações que são diariamente disponibilizadas *online.* Dessa evolução podemos destacar o surgimento das redes de comunicação sem fios, que proporcionaram ao utilizador uma maior mobilidade com o seu terminal de acesso à Internet [4][5]. Deste modo o utilizador para além de deixar de estar dependente de um cabo para aceder à Internet, passou a contar com mais terminais de acesso, principalmente terminais pequenos, como por exemplo o telemóvel ou as *tablets*.

No que toca as redes cabladas, as soluções de banda larga (*broadband*) [6] e desenvolvimento de novas tecnologias na transmissão de dados tais como, *Asymmetric Digital Subscriber Line* (ADSL) [7] vieram permitir transmissões digitais com altas taxas de transferência. Ainda neste domínio a crescente introdução de fibra ótica [8] que em relação ao cobre, permite transmitir mais informação, com maior grau de fidelidade e para distâncias maiores, tem aumentado consideravelmente a velocidade nas conexões, oferecendo ao utilizador uma maior agilidade de navegação na Web.

No âmbito das redes de comunicação sem fios também se tem assistido ao longo do tempo evoluções significativas, com a aplicação de novas normas para transmissão sem fios. Refira-se por exemplo a norma IEEE 802.16 *Worldwide Interoperability for Microwave Access* (WiMAX) [9] que proporciona um melhor desempenho de comunicação e permite maiores distâncias e velocidades de transmissão quando comparada com a norma IEEE 802.11 (também conhecida como Wi-Fi) [10]. Todas estas redes têm contribuído para aumentar não somente o número de utilizadores, mas também a quantidade de dispositivos conectados simultaneamente à Internet.

2.2.1 Evolução da Web

Há alguns anos atrás os conteúdos disponibilizados na Web era praticamente feito por profissionais da área e o utilizador era apenas um consumidor que acedia às páginas estáticas para consultar a informação lá contida. Dito por outras palavras, os conteúdos

disponibilizados *online* não podiam ser alterados pelos utilizadores finais. Nessa época, tal como referido, a Web era caracterizada por conter conteúdos praticamente estáticos e ficou conhecida historicamente por Web 1.0 [11].

Atualmente o conceito e a forma de fazer a Web mudou completamente. Para tirar proveito de um grande número de dispositivos que passaram a estar conectados à Internet, a Web também teve que evoluir no sentido de se adaptar às caraterísticas desses dispositivos. As simples páginas estáticas contendo basicamente textos e imagem, evoluíram e deram lugar a sofisticadas aplicações dinâmicas que interagem com o utilizador e que oferecem uma vasta gama de conteúdo. A Web atual, também conhecida por Web 2.0 [12], é uma Web participativa, onde os internautas desempenham uma grande função na produção dos seus conteúdos. O termo Web 2.0 não se refere a uma atualização de especificações técnicas, mas sim a uma nova forma de pensar e fazer as coisas. Neste novo contexto da Web, a maioria das aplicações permitem ao utilizador criar e compartilhar conteúdos na comunidade virtual. Dito por outras palavras, o utilizador deixou de ser um simples consumidor e passou a ser o maior produtor de conteúdo disponibilizado na Web. Um exemplo claro dessa realidade pode ser verificado nas atuais redes sociais, onde diariamente são disponibilizados milhões de conteúdos (e.g., fotos, vídeos, músicas) pelos internautas [11][12].

Contudo a evolução da Web tal como a conhecemos hoje deve-se também a outros factores. O utilizador é cada vez mais um consumidor exigente, que espera sempre utilizar os melhores dispositivos e aplicações. Por outro lado temos um conjunto de tecnologias de suporte que de certo modo tem contribuído para essa evolução.

2.2.2 Terminais de acesso à Internet

Ao longo do tempo os terminais de acesso à Internet têm vindo a diversificar-se e a melhorar as suas características. Essa melhoria é bastante evidente no caso dos terminais móveis (e.g., *smartphones*, *tablets*), devendo-se em parte essa evolução aos desenvolvimentos verificados noutras áreas, como por exemplo na eletrónica e nas telecomunicações.

Em relação à eletrónica, com a redução das dimensões dos circuitos, atualmente é possível incorporar num pequeno chip, um número maior de portas lógicas e de funcionalidades. Tal permite ter dispositivos cada vez mais reduzidos, mas que simultaneamente incorporam um número maior de funcionalidades e capacidades. Graças à redução do tamanho dos circuitos,

hoje em dia é possível ter num *tablet*, quase todas as funcionalidades que estão disponíveis num computador normal [13].

No que se toca às telecomunicações, tem-se também verificado uma grande evolução, principalmente na parte das comunicações móveis. O telemóvel que dantes era um terminal de funcionalidades reduzidas, cuja principal função era a comunicação de voz, agora é um terminal com as mais diversas funcionalidades e capacidades, capaz de correr diversos tipos de aplicações. Também a velocidade de acesso à Internet por meio destes terminais tem melhorado consideravelmente com o passar do tempo. O suporte ao serviço de transmissão de dados digitais nestes tipos de terminais, iniciado com a segunda geração (2G) das comunicações móveis, proporcionou grandes avanços nesta área, sendo o *Global System for Mobile Communications* (GSM) [9], uma das tecnologias mais populares dessa geração. O débito binário conseguido com a tecnologia 2G é bastante limitado e portanto com o intuito de aumentar a qualidade de serviço e o débito binário, algumas melhorias foram feitas ao 2G, a que passou a ser chamada de 2.5G. No 2.5G foi por exemplo introduzido o serviço de mensagens curtas (*Short Message Service* - SMS) [14], bastante popular atualmente. Entretanto, para utilizar estes dispositivos para aceder à ampla gama de serviços e aplicações disponíveis atualmente na Internet (e.g., comércio eletrónico, vídeos, áudio, videochamadas), foi necessário aumentar ainda mais o débito binário, introduzindo novas tecnologias tais como *Universal Mobile Telecommunications System* (UMTS) [15] e *Wideband Code Division Multiple Access* (WCDMA) [15] que caraterizam a terceira geração (3G) das comunicações móveis. Atualmente a quarta geração (4G) já está a ser adotada em alguns países, cujas principais tecnologias implementadas são o *Long Term Evolution* (LTE) [16], o WiMAX e o *Evolved High-Speed Packet Access* (HSPA+) [9][10]. A tecnologia 4G visa essencialmente melhorar alguns aspetos da tecnologia 3G como por exemplo, reduzir a latência e aumentar a qualidade de serviço nas videochamadas e videoconferências, oferecer serviços de dados com taxas mais elevadas de largura de banda, permitir aos utilizadores enviar e receber ficheiros de grandes dimensões, aceder a conteúdos multimédia e assistir transmissões televisivas em tempo real, entre outros. Ou seja, o foco central da tecnologia 4G é a banda larga móvel [10][16].

2.3 TECNOLOGIAS DE ARMAZENAMENTO DE DADOS

No que se refere às tecnologias de armazenamento de dados, mais especificamente as bases de dados, por muito tempo as soluções baseadas no Modelo Relacional [17] foram as que dominaram o mercado, tanto em aplicações desenvolvidas para *desktop* como em aplicações Web [18]. Quando o Modelo Relacional foi projetado, a realidade dos sistemas de informação em geral e da Internet em particular era completamente diferente do atual, não se prevendo um crescimento e mudança na Web tão repentino. Nessa altura, quando se implementava uma aplicação, quase que se podia prever a quantidade de utilizadores que iria utilizar o sistema, bem como o espaço de armazenamento que seria necessário.

No entanto, com a crescente evolução da Web, para certas aplicações deixou de ser possível fazer essas previsões, principalmente em aplicações cujo número de utilizadores cresce quase que exponencialmente com o passar do tempo e que dá ao utilizador a permissão para fazer *upload* de conteúdos simultaneamente, para aplicações deste género, poderá haver milhares ou mesmo milhões de utilizadores a utilizá-las em simultâneo, tornando a disponibilidade um factor a ter em conta. Dito por outras palavras, a tecnologia de armazenamento de dados e de processamento utilizada, para além de poder suportar grande número de utilizadores concorrentes, precisa ser rápida nas operações de leitura e escrita, como por exemplo, se for uma aplicação onde os utilizadores comunicam em tempo real uns com os outros através de troca de mensagens instantâneas [1][19].

O Modelo Relacional tem sido a solução de armazenamento de dados mais utilizada, graças às suas várias características entre as quais se destaca, uma estrutura rígida que proporciona robustez e segurança aos sistemas. No entanto, com o advento da Web 2.0, surgiram novos tipos de aplicações que geram grandes fluxos de dados e o Modelo Relacional começou a demostrar certas limitações em lidar com as novas necessidades de leitura/escrita destes[19] [20].

Na tentativa de solucionar essa lacuna, surgiu uma nova geração de tecnologias de armazenados de dados, que mais tarde passou a ser conhecida por NoSQL [19]. Esse novo modelo de dados, também designado por bases de dados não-relacionais, para além de prometer facilidade de escalabilidade das bases de dados, alta disponibilidade e desempenho, comparado ao Modelo Relacional, oferece por norma uma maior flexibilidade na estruturação dos dados [1][20]. Este tema será abordado na secção 3.5 com mais detalhes.

2.4 DOCUMENTOS ESTRUTURADOS

Muitas das aplicações atuais, principalmente as desenvolvidas para dispositivos móveis, necessitem de comunicar via Internet com outras aplicações de modo a trocar informações. A solução encontrada para facilitar essa troca de informação entre aplicações provenientes de plataformas e/ou linguagens diferentes, foi utilizar uma linguagem que seja independente de plataforma. Neste contexto várias soluções têm sido desenvolvidas, tendo algumas delas ganho popularidade a ponto de serem largamente utilizadas em diversas áreas, como veremos nas próximas secções.

2.4.1 XML

O XML (*Extensible Markup Language*), tem sido a solução que mais se tem destacado pela maturidade e larga margem de usabilidade [2]. A sua capacidade para serialização de estruturas computacionais e de transporte de dados, independentes de plataforma, tem feito dele uma linguagem de marcação por excelência. O XML adquiriu uma grande popularidade e tem sido escolhido como o novo formato de documentos na maior parte das aplicações de computadores e adotado em diferentes áreas, onde estão presentes as Tecnologias de Informação [2][21].

Na elaboração de um documento XML, algumas regras de sintaxe têm que ser respeitadas, para que o documento possa ser considerado um documento-bem-formado, principalmente se não for utilizado algum validador. Basicamente existem dois validadores de documentos XML [21]: o *Document Type Definition* (DTD) e o XML-*Schema Definition* (XSD), sendo que o DTD, foi o primeiro validador de documentos XML a ser desenvolvido. Com o passar do tempo o DTD começou a apresentar algumas limitações, de entre as quais as mais apontadas são a limitação de tipos de dados que podem ser declarados e a sintaxe utilizada que é totalmente diferente do XML. Para minimizar as limitações do DTD, foi pensada uma nova forma de validar documentos XML, tendo surgido desde modo o XSD, que para além de desempenhar a função principal do DTD (validar documentos XML), trouxe várias vantagens em relação ao DTD e que de certo modo contribuíram para o sucesso do XML. Das principais vantagens dos XSD em relação ao DTD pode-se destacar, a possibilidade de reutilização do *schema* para validar outros documentos XML; a utilização de uma sintaxe igual à do XML e a possibilidade de definir vários tipos de dados e novos tipos de documentos.

Apesar do XML ter revolucionado a forma de representar e transferir dados ou documentos entre aplicações e de ter sido adotado com facilidade em diversas áreas, a ponto de ser considerado pela W3C como um *standard* definido e recomendado para anotação de documentos, o XML apresenta algumas imperfeições. A maior limitação do XML é a sintaxe que é redundante e como consequência pode afetar a eficiência das aplicações e exigir um maior espaço de armazenamento, o que pode limitar a sua aplicação em casos onde o limite da largura de banda é um factor a ter em conta [2][22].

Na tentativa de solucionar algumas limitações do XML, outras alternativas têm sido propostas. Dessas algumas ganharam destaque e têm tido uma boa aceitação, como é o caso do YAML [23] e o *JavaScript Object Notation* (JSON) [24], que em vários casos têm sido utilizados como um substituto ou como um complemento do XML.

2.4.2 YAML

O YAML é um formato de serialização de dados que oferece quase todas as características do XML ou de outras linguagens semelhantes, mas em contrapartida com uma notação simples e de fácil leitura pelos humanos. Foi proposto pela primeira vez em 2001 por Clark Evans [23], tendo sido inspirado em conceitos de linguagens tais como Python, C, Perl, XML, entre outras.

Os documentos no formato YAML podem ser produzidos, modificados, editados com facilidade em qualquer editor de texto. A sua sintaxe é relativamente simples, tendo sido projetada tendo em mente não somente a legibilidade pelos humanos, mas também a facilidade de transformar os dados em outras estruturas de dados mais comuns utilizadas em várias linguagens de alto nível. Uma das principais características do YAML é a serialização de dados, que permite com facilidade representar tipos de dados mais comuns na maioria das linguagens de programação. Deste modo, é bastante utilizado principalmente para escrever ficheiros de configurações e representar estruturas de dados de um forma limpa e compacta da maior parte das linguagens de programação da atualidade, como por exemplo o Ruby, Python, Perl, entre outras. A representação de dados suportados pelo YAML, normalmente corresponde a tipos de dados nativos das linguagens, gastando deste modo menos tempo a transformar os dados numa estrutura que será processada pela aplicação, o que torna mais rápida a importação e exportação de dados [22] [23].

2.4.3 JSON

Proposto pelo engenheiro de *software* Douglas Crockfor, outro formato que tem vindo a ganhar muita popularidade é o JSON [24]. O JSON é um formato de dados leve que, tal como os formatos anteriores, permite fazer troca de dados entre aplicações. Foi baseado no subconjunto da anotação de objeto *JavaScript*, mas o seu uso não requer exclusivamente *JavaScript*. O JSON rapidamente teve uma grande aceitação por parte da comunidade Web graças às suas características, entre as quais se destaca a facilidade de criar um analisador *(parser*) JSON para o processar [24].

Um documento JSON é uma coleção de pares chave/valor, onde é possível guardar quatro tipos de dados (*strings*, inteiros, booleanos e *arrays*). Além disso é um formato fácil de ler e escrever pelos seres humanos e processado facilmente por máquinas.

O JSON proporciona várias vantagens ao desenvolvedor de aplicações Web, principalmente em projetos sofisticados onde poderá ser necessário uma política de transferência de dados simplificada e que seja eficiente e leve. É usado para vários propósitos, como por exemplo, para trocar dados entre linguagens, para criar *Web Services* que permitem aplicações de plataformas diferentes comunicar, entre outros.

O JSON em relação ao XML traz algumas vantagens, entre as quais se destaca uma sintaxe muito mais simplificada, de fácil implementação e aprendizagem. Outra das vantagens é a diminuição do tamanho da resposta que é devolvida quando duas aplicações troquem informações, razão pela qual é bastante usado em ambientes onde a quantidade do fluxo de dados trocado entre o cliente e servidor é de suma importância [22][24].

Na Tabela 2.1 podemos notar a diferença de sintaxe e legibilidade entre os três formatos de dados aqui apresentados, onde podemos notar a redundância na sintaxe do XML.

XML	<?xml version="1.0" encoding="UTF-8" ?> <user> <firstname>Manuel</firstname> <surname> Miranda</surname> <birthday>10/10/1980</birthday> <email>mmiranda@ualg.pt</email> <friends>Carla Jesus</friends> <friends>Moisés Soares</friends> <friends>Arielle Fernandes</friends> </user>
YAML	--- user: firstname: "Manuel" surname: " Miranda" birthday: "10/10/1980" email: "mmiranda@ualg.pt" friends: - "Carla Jesus" - "Moisés Soares" - "Arielle Fernandes" ---

JSON	{
	"user":{
	"firstname": "Manuel",
	"surname": " Miranda",
	"birthday": "10/10/1980",
	"email":"mmiranda@ualg.pt",
	"friends": ["Carla Jesus", "Moisés Soares", "Arielle Fernandes"]
	}
	}

Tabela 2.1: Exemplo de diferenças de sintaxe entre XML, YAML e JSON

2.5 SERVIÇOS WEB

Os avanços tecnológicos, bem como a popularização da Internet e a evolução das redes de computadores, deu origem a um novo tipo de aplicações conhecidas por aplicações distribuídas [25]. Com essa nova categoria de aplicações, surge também a necessidade de partilhar informações, ou seja, a necessidade de interoperabilidade entre os diversos sistemas. É nesta vertente que surgem os *Web Services* [3], que funcionam digamos, como um elo de ligação, ou como um "intérprete", entre sistemas que "falam" línguas diferentes e que precisam trocar informações.

O surgimento dos *Web Services* a partir de novos conceitos de interoperabilidade permitiu ultrapassar as barreiras de comunicação entre aplicações que residem em plataformas diferentes ou que foram escritas em linguagem de programação diferentes. A partir de então a comunicação entre novas aplicações e as já existentes, independentemente da plataforma ou linguagem de programação utilizada no seu desenvolvimento, tornou-se mais fácil e mais prática.

O princípio de funcionamento de um *Web Service*, consiste em aceitar/interpretar solicitações de outros sistemas através da Internet e expor funcionalidades/serviços para os utilizadores, através de protocolos normalizados. Por outras palavras, através de um *Web Service*, uma aplicação pode invocar outra para executar determinadas tarefas simples ou complexas, ainda que as duas aplicações estejam em plataformas diferentes ou mesmo desenvolvidas em linguagens de programação diferente [3][26].

Nas implementações de Web *Services* costuma-se utilizar basicamente tecnologias normalizadas, das quais podemos destacar: o XML, o *Simple Object Access Protocol* (SOAP), a *Web Service Definition Language* (WSDL) e o *Universal Description Discovery and Integration* (UDDI) [3] .

O SOAP é o protocolo de comunicação utilizado entre as aplicações clientes e servidores, cujo principal papel é fazer o encapsulamento das mensagens XML trocadas entre as aplicações e que pode ser utilizado em combinação com outros protocolos, como HTTP, HTTPS, SMTP, FTP e RPC [26][27].

O WSDL é a linguagem utilizada para descrever a interface do *Web Service*. É uma especificação desenvolvida pelo W3C baseada no XML e é através dela que se faz a descrição

dos serviços disponibilizados à rede, ou seja, serviços que uma determinada aplicação coloca à disposição para as demais acederem [3].

O UDDI é considerado como um repositório, onde os *Web Services* anunciam os serviços que dispõem e é onde também as aplicações clientes devem localizar serviços ou os *Uniform Resorce Locator* (URL) dos serviços disponíveis nos *Web Services*. Existe um registo global público, denominado de UDDI *Business Registry* que disponibiliza toda a informação dos *Web Services*, mas para além desse serviço, é possível fazer um registo privado, principalmente quando se deseja garantir um maior controlo de segurança e restrição de acesso aos dados [3].

Atualmente existe uma alternativa ao protocolo SOAP para implementação de *Web Services*, designado por *Representational State Transfer (*REST) [27], que consiste num conjunto de princípios que exploram a arquitetura Web de forma a tirar proveito dela. Um dos serviços do REST que é bastante utilizado é o RESTful, usado para implementação de *Web Services*, muitas vezes como uma alternativa à tecnologia baseado no protocolo SOAP, pelo facto de ser mais leve e de utilizar apenas o protocolo HTTP na transmissão de dados. O formato dos dados transmitidos na solução RESTful, pode ser o XML ou o JSON, sendo que o uso do JSON tem algumas vantagens, principalmente se o *Web Service* for utilizado por aplicações móveis, onde o tamanho dos dados a serem transmitidos podem influenciar o desempenho da aplicação [26][27] (secção 2.4).

2.6 INTERFACES DE PROGRAMAÇÃO DE APLICAÇÕES (APIS)

No desenvolvimento de *software* ou aplicações Web, na maioria das vezes há a necessidade de integrar certas funcionalidades para executar determinadas tarefas. Algumas dessas funcionalidades porém, podem apresentar elevada complexidade de implementação que para além de aumentar o tempo de desenvolvimento de um projeto, podem ser uma tarefa árdua e custosa, o que tornaria certos projetos inviáveis. Para minimizar essas limitações, algumas empresas e serviços ligados às Tecnologias de Informação, desenvolveram as chamadas APIs que para além de facilitar significativamente o trabalho dos programadores, disponibilizam publicamente soluções tecnológicas úteis, que permitem transformar e dinamizar serviços e negócios *online*.

Uma *Application Programming Interface* (API) [28] representa um conjunto de funções padrão, previamente programadas para executar determinadas funcionalidades e que podem ser utilizadas por aplicações que delas necessitam. No que toca a aplicações Web, o funcionamento consiste basicamente num conjunto de mensagens de requisição e respostas HTTP, cujos formatos de dados mais utilizados são o XML e o JSON.

A utilização de códigos de terceiros no desenvolvimento de sistemas tornou-se numa prática comum, porque há situações em que não há necessidade de andar a "reinventar a roda", implementando de novo funcionalidades que já existem. Com o auxílio das APIs no desenvolvimento de *software* e aplicações Web, para além reduzir o tempo que seria gasto num projeto sem o uso delas, ajudam também agregar várias funcionalidade que contribuem no enriquecimento dos conteúdos disponibilizado aos utilizadores finais.

Existem diversas empresas e serviços que disponibilizam os seus códigos para serem utilizados por terceiros. Ao disponibilizar uma API para uso de terceiros, o proprietário pode impor algumas restrições no uso da mesma, como por exemplo, limitar a frequência de utilização da API por uma aplicação ou limitar o período de tempo que a mesma pode estar disponível para uso.

O uso de uma API pode ser gratuito ou mediante uma licença. Em termos de formalidade de uso, as APIs podem ser classificadas em dois tipos [28]:

- **Pública –** significa que a API está disponível para uso de qualquer utilizador. Muitas delas para utilizá-las basta concordar com os termos de uso, outras poderão ou não exigir um contrato de uso.
- **Privada –** são APIs utilizadas para diversos propósitos, algumas são de uso apenas interno, para agilizar processos de negócios ou para aproximar equipas/colaboradores de projetos dentro de um empresa/organização. Outras, para além do uso interno podem ser facultadas para o uso externo (parceiros e grandes clientes), mediante um contrato estabelecido com o proprietário da API.

No contexto da Web, um exemplo comum do uso de uma API verifica-se quando entramos em determinados sítios, fazemos uma pesquisa e verificamos que o resultado devolvido, muitas vezes é apresentado de acordo com o local (e.g., Cidade) de onde estamos. Uma API que por exemplo faz isso é o Google Maps Geolocation API [29], disponibilizada pela empresa Google e que oferece métodos de geolocalização que permitem saber com uma precisão relativamente elevada, em que local do globo se encontra o utilizador.

Existem vários tipos de APIs que realizam diversas funções e podem ser encontradas em diversos tipos de aplicações/*software*, nomeadamente nos navegadores de Internet, nos sistemas operativos, em linguagens de programação e sobretudo em aplicações Web.

Veremos no capítulo seguinte a análise de algumas tecnologias de gestão e armazenamento de dados.

3. ANÁLISE DE ALGUNS MODELOS DE BASE DE DADOS

3.1 INTRODUÇÃO

A maior parte das aplicações Web desenvolvidas atualmente, são aplicações que processam grandes quantidades de dados. Incontornavelmente, à medida que o tempo vai passando e o número de utilizadores dessas aplicações aumenta, o volume de dados a serem armazenadas e processados, também aumentará.

As bases de dados que seguem o Modelo Relacional, têm sido utilizadas em vários ambientes como a solução que permite lidar com o armazenamento e gestão de dados gerados e processados por diversos tipos de aplicações. No entanto, na altura em que o Modelo Relacional foi projetado, as aplicações desenvolvidas até então não geravam por norma grandes fluxos de dados comparáveis aos atuais. Com o passar do tempo e com a evolução de algumas áreas tais como, a eletrónica, as telecomunicações e as tecnologias de informações de um modo geral, começaram a surgir novos tipos dispositivos portáteis com capacidades para aceder à Internet. Entres outros motivos, a facilidade de aceder à Internet através dos vários dispositivos atualmente existentes, aumentou o número de utilizadores simultaneamente *online* e como consequência deste aumento, o fluxo de dados a serem processados e armazenados também aumentou.

As bases de dados do Modelo Relacional, que durante muitos anos tinham sido a principal solução encontrada para lidar com esses dados, começaram a mostrar em alguns casos certas limitações, principalmente em aplicações Web de grande porte, como por exemplo as atuais redes sociais, cujo número de utilizadores aumenta exponencialmente e que permitem aos seus utilizadores trocar grandes quantidades de dados [30] .

Algumas das empresas proprietárias dessas aplicações de grande porte, começaram a enfrentar o desafio de conseguir garantir os seus serviços aos utilizadores espalhados pelo mundo, principalmente quando o número de utilizadores é na ordem de milhares ou mesmo milhões. Essas empresas aperceberam-se que para aplicações deste tipo, o Modelo Relacional estava a atingir o seu limite e que precisavam de encontrar outras soluções para lidar com o armazenamento e a manipulação destes grandes volumes de dados gerados por estas novas

aplicações. Motivados principalmente pela necessidade de escalabilidade e da disponibilidade da informação, nasceu um novo conceito de bases de dados, denominadas por bases de dados não relacionais e que mais tarde passou a ser conhecido por NoSQL [19][31]. Tendo em conta que para o sistema de monitorização de eventos que foi projetado no âmbito do projeto *LifeSpeeder*, como é óbvio, seria necessário fazer uso de uma base de dados para armazenar não somente as informações dos eventos, mas também outras informações que se julgar importantes, como por exemplo, informações dos utilizadores que farão uso da plataforma, para saber que tipo de base de dados que melhor se enquadrava no projeto, foi necessário fazer uma análise prévia dos modelos de bases de dados existentes atualmente. Neste âmbito, este capítulo tem por objetivo, apresentar uma análise comparativa entre Modelo Relacional e a nova geração de bases de dados não relacionais, denominada por NoSQL, a fim de chegar a uma conclusão de qual dos dois modelos de dados será a solução mais adequada para a plataforma do *LifeSpeeder*.

3.2 EVOLUÇÃO DOS MODELOS DE DADOS

Não se pode falar dos modelos de dados atuais sem antes fazer uma referência aos antigos modelos de dados que de certa forma foram úteis e impulsionaram o surgimento de outros modelos. Desde os primórdios da informática, o homem tem procurado encontrar a melhor forma de armazenar e manipular informações do seu mundo real, em sistemas informatizados. Antes dos modernos sistemas de gestão de bases de dados, outras formas de armazenamento e modelos de dados foram utilizadas, cujas mais populares serão apresentadas a seguir.

3.2.1 Sistemas de gestão de ficheiros (SGF)

A primeira forma encontrada para lidar com o armazenamento de dados, baseou-se nos sistemas de gestão de ficheiros (SGF), que representam uma forma de organizar e armazenar dados em meios de armazenamento em massa. Com os SGF foi possível automatizar algumas tarefas, que até então eram realizadas manualmente, tornando a execução dessas tarefas mais rápida. Para cada nova aplicação era necessário definir novos ficheiros de dados e desenvolver os programas necessários para os processamentos específicos. Nos SGF cada sistema era considerado como um sistema isolado, sem relação com outros sistemas já existentes [32] [33].

Mas os sistemas de gestão ficheiros apresentavam no entanto algumas limitações, como por exemplo a dificuldade em fazer o controlo de acesso, quando o sistema ou aplicação era utilizado por vários utilizadores ou por vários processos ao mesmo tempo. Para além dessas limitações, existiam outros inconvenientes nos SGF, nomeadamente [33][34]:

- Redundância de informação - originada pela forma desarmonizada de criar os ficheiros e da forma independente de efetuar atualizações dos dados, que dava origem a duplicação de dados.
- Inconsistência dos dados - muitas vezes uma consequência direta da redundância.
- Dificuldade em extrair a informação - devido à estrutura dos ficheiros que dificultava o acesso aos campos onde estava a informação.
- Informação dispersa - estando os dados em diferentes ficheiros, uma mesma informação poderia estar em formatos diferentes, em cada um dos ficheiros.
- Integridade dos dados - tarefa difícil de conseguir e que só poderia ser assegurada, mediante a criação de regras em cada aplicação que fizesse uso dos ficheiros.

As técnicas de processamento utilizadas nos SGF evoluíram e dessa evolução surgiram novos modelos de gestão de dados, dos quais podemos destacar o Modelo Hierárquico e o Modelo em Rede (secções 3.2.2 e 3.2.3).

3.2.2 Modelo de Dados Hierárquico

O Modelo de Dados Hierárquico é reconhecido como o primeiro modelo de dados efetivo e foi bastante utilizado principalmente nos primeiros sistemas de gestão de bases de dados *mainframe*. Neste modelo os dados são estruturados em hierarquias ou árvores, em que os nós que constituem a hierarquia contêm ocorrências de registos. Cada registo é uma coleção de atributos e cada atributo contém apenas uma informação. Um registo que precede outros registos é denominado por registo-pai e os outros que estão abaixo do registo-pai, são denominados por registos-filhos. O relacionamento de um registo-pai com registos-filhos é de 1:N (um para muitos) ou seja, um registo-pai pode ter vários registos-filhos, mas um registo filho só está relacionado com um registo pai. O acesso aos dados é feito segundo uma sequência hierárquica, do topo para a base e da esquerda para a direita. Para que seja possível associar um registo a vários registos diferentes ao mesmo tempo, é necessário fazer a replicação deste, mas a replicação de registo traz duas desvantagens: a primeira é a possível inconsistência dos dados quando um dos nós fizer alguma atualização no registo replicado; a segunda é o desperdício de espaço de armazenamento. Na Figura 3.1 podemos ver um

exemplo de um modelo de dados hierárquico, onde o registo-pai (Nível 0), está relacionado com dois registos-filhos (Nível 1). O acesso a qualquer um dos registos-filhos é feito a partir da raiz, ou seja a partir do registro-pai (Nível 0). Neste modelo se um registo-filho tiver dois ou mais registos-pai, então o registo-filho deve ser duplicado para cada um dos registos-pai. Isso faz aumentar a redundância dos dados, uma caraterística inerente ao próprio modelo. Um outro problema deste modelo é que ao eliminar um registro-pai, eliminam-se todos os registos-filhos que poderão conter informações importantes para o funcionamento do sistema. O sistema comercial mais divulgado deste modelo, foi o IMS (*Information Management System*) da IBM *Corporation* (1968) desenvolvido para *mainframe*s[33][34] .

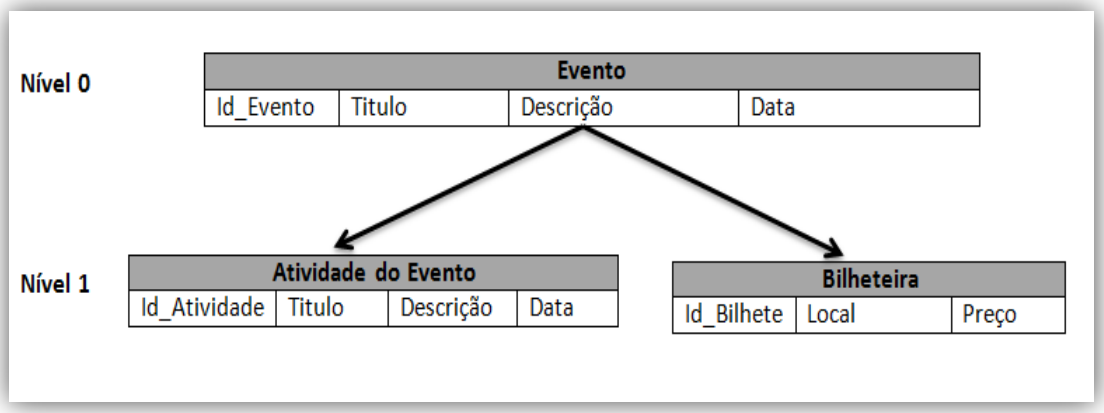


Figura 3.1: Modelo de dados hierárquico

3.2.3 Modelo de Dados em Rede

O Modelo de Dados em Rede surgiu como uma forma de melhorar o Modelo de Dados Hierárquico. Este modelo não utiliza o conceito de hierarquia, sendo a organização dos dados feita em forma de grafos. Neste modelo existe apenas um tipo de associação, chamado de *set*, que estabelece uma relação de 1:N entre dois tipos de registos: registro proprietário (*owner*) e registo membro (*member*). Ao contrário do Modelo de dados Hierárquico onde um registo-filho só podia estar relacionado com um registo-pai (isso sem ter em conta a replicação do registo-filho), neste modelo um registro *member* pode estar relacionado com mais de um registo *owner*, sem a necessidade de replicar o registro *member*. Esta estrutura torna as pesquisas mais rápidas e mais flexíveis, porque não depende de um único nó raiz, como ponto de partida das pesquisas. Por outras palavras, a navegação numa base de dados deste modelo, pode ser feita de várias formas, conforme ilustrado na Figura 3.2: do *owner* para o primeiro ou último *member*, de um *member* para o *owner*, de um *member* para um *member* anterior ou próximo. Este modelo, para além de possuir as propriedades básicas de registos existentes no

Modelo Hierárquico, utiliza também uma linguagem DDL (*Data Definition Language*) para definir os dados e uma linguagem DML (*Data Manipulation Language*) para manipulação dos dados. Uma das vantagens deste modelo em relação ao outro modelo, reside na possibilidade em criar relacionamentos de M:N(muitos para muitos) e a flexibilidade de acesso aos dados. Para este modelo, o sistema comercial mais divulgado foi IDMS (*Integrated Database Management System*) da Computer Associates [32] [34].

A necessidade de um modelo de dados com mais recursos, capaz de superar as limitações apresentadas pelos modelos anteriores, deu origem a um novo modelo de bases de dados, o Modelo Relacional, que substitui os modelos anteriormente mencionados, trazendo várias vantagens, nomeadamente: maior facilidade de acesso aos dados, maior velocidade nas respostas, acesso múltiplo através de *software* de gestão, integridade dos dados armazenados e uma maior facilidade de gestão da informação, conforme veremos na próxima secção [33].

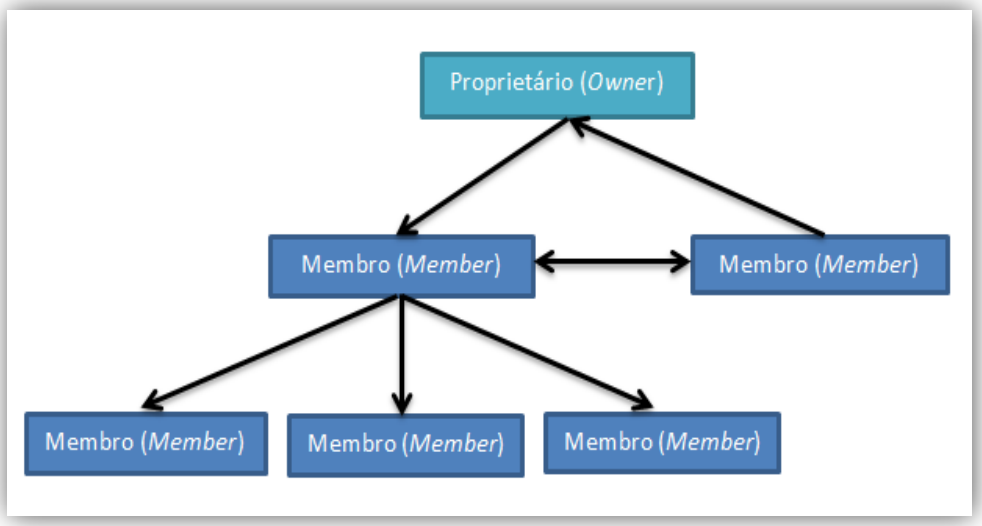


Figura 3.2: Modelo de dados em rede

3.3 MODELO RELACIONAL

O Modelo Relacional revolucionou a indústria das bases de dados e tornou-se num modelo padrão para a maioria dos SGBD (Sistemas de Gestão de Bando de Dados) existentes. Este modelo é resultado de um estudo realizado por um investigador da IBM (Edgar Frank Codd) em 1970 [17] [32], onde ele utilizou pela primeira vez o termo relacional para as bases de dados. Apesar de esse estudo ter sido publicado em 1970, uma primeira implementação do

Modelo Relacional só foi feita na década de 1980. No estudo apresentado, Codd propunha uma base de dados onde existisse:

- Independência física - a forma de armazenar os dados não pode influenciar na forma de recuperá-los.
- Independência lógica - uma aplicação que utiliza uma base de dados, não pode ser afetada devido a alterações efetuadas na base de dados.
- Flexibilidade - uma base de dados deve oferecer formas distintas de visualização, em função dos utilizadores e das aplicações que fazem uso dela.
- Uniformidade - as estruturas lógicas devem ter uma única forma conceptual, que são as tabelas.

Quando este modelo foi implementado, revelou-se na altura ser o modelo mais adequado para contornar as limitações que até então se tinham verificado com os seus antecessores. As fortes caraterísticas de segurança, integridade dos dados, entre outras, oferecidas pelo Modelo Relacional, fizeram com que ele fosse rapidamente eleito e adotado como o principal modelo de bases dados por pequenas, médias e grandes empresas que necessitavam de bases de dados para otimizar os seus processos de negócio [32][35].

3.3.1 Estrutura de uma Base de Dados Relacional

Uma base de dados relacional é formada por entidades, também designadas por tabelas, e relacionamentos. Uma tabela é uma estrutura formada por linhas (tuplos) e colunas (atributos), onde são armazenados os dados. Cada linha formada por um conjunto de atributos, representa um registo dentro da base de dados. Não é obrigatório um registo ter dados em todos os atributos (colunas), isto é, podem existir atributos com valor nulo. O número de tabelas numa base de dados é limitado exclusivamente pelo SGBD. A linguagem mais popular na definição e manipulação dos dados das bases de dados relacionais é a linguagem de consulta estruturada, SQL (*Structured Query Language*) [32][36].

A estrutura lógica de uma base de dados pode ser expressa graficamente, num nível de abstração, utilizando o Modelo de Entidades e Relacionamentos (MER) [35] que é um modelo conceitual de alto nível, empregue em projetos de sistemas de informações. A principal ferramenta utilizada neste modelo é o Diagrama Entidade-Relacionamento (DER), em que os componentes são retângulos, elipses, e losangos. A Figura 3.3 mostra exemplo de uma DER, onde os retângulos representam as entidades (tabelas), as elipses representam os atributos e os losangos representam um relacionamento entre entidades.

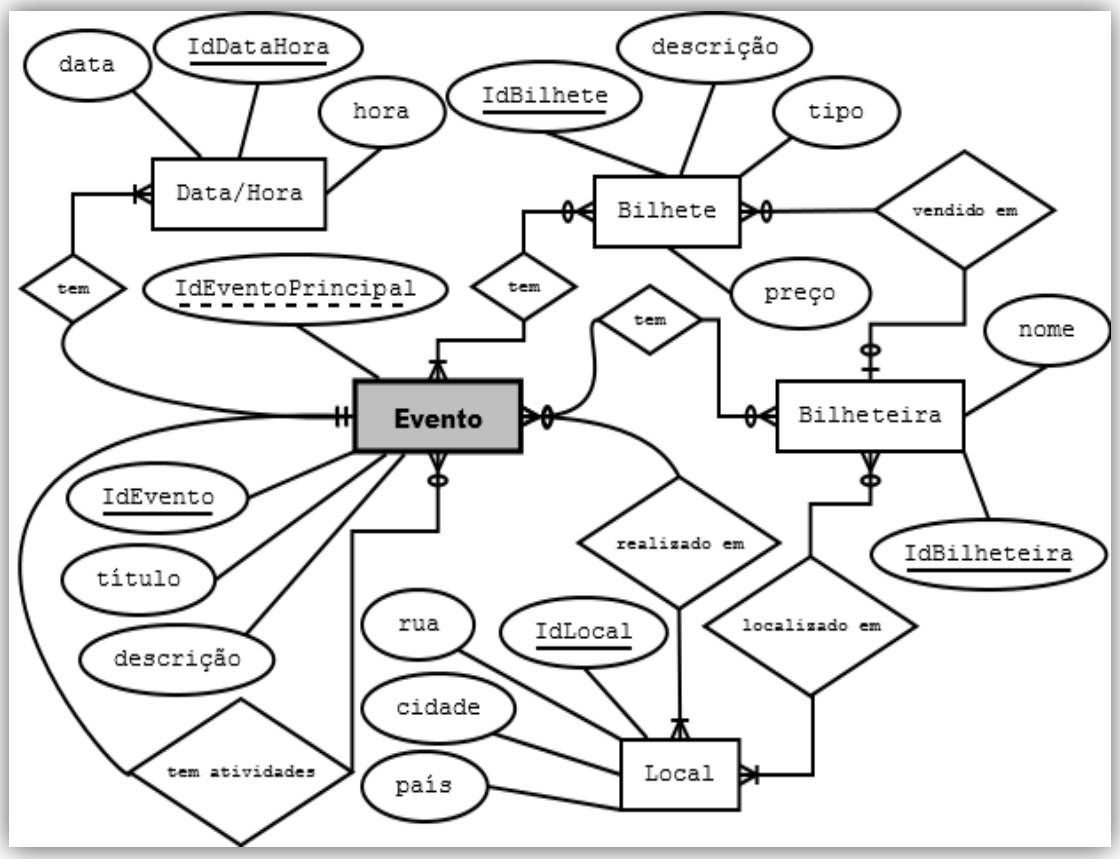


Figura 3.3: Exemplo de uma DER com as entidades que armazenam dados de um evento

O relacionamento entre entidades de uma base de dados do Modelo Relacional é feito através de chaves que podem ser formadas por um ou mais atributos. Um atributo pode ser considerado como um elemento de dados que descreve uma entidade e pode assumir um único valor (atributo simples) ou múltiplos valores (atributo composto). Para cada atributo existe um conjunto de valores permitidos. Ao conjunto de valores aceitáveis por um atributo designa-se por domínio. Numa tabela poderá existir um conjunto de atributos que permitem identificar univocamente um registo e que são designados por chaves candidatas. Do conjunto de chaves candidatas mínimas, uma é eleita como chave primária (PK - *Primary Key*). Para além da chave primária, existe outro tipo de chave no Modelo Relacional, chamada de chave estrangeira (FK - *Foreign Key*), cuja principal finalidade é garantir a integridade referencial. Quando um atributo é considerado como uma chame estrangeira, significa que o mesmo está a referir uma chave primária de outra ou da própria tabela.

Um relacionamento no MER, é uma associação entre entidades ou tabelas distintas e pode ser do tipo [32][37]:

- Um para um (1:1) - quando a relação entre as tabelas é uma relação unívoca. Neste caso a escolha da tabela que receberá a chave estrangeira, fica ao critério do projetista.
- Um para muitos (1:N) - quando um registo de uma tabela do "lado 1", se relaciona com um ou mais registos da outra tabela do "lado N". Por norma a chave primária da tabela do "lado 1", deverá aparecer na tabela do "lado N" como chave estrangeira.
- Muitos para muitos (M:N) - quando um registo de uma das tabelas relaciona com vários registos da outra tabela e vice-versa. Neste tipo de relacionamento será necessário criar uma nova tabela que conterá as chaves primárias das tabelas envolvidas no relacionamento, formando por norma uma chave composta.

Na implementação de uma base de dados relacional, um dos passos importantes consiste em criar um modelo conceptual dos dados. Um modelo conceptual é um modelo lógico, abstrato e independente de qualquer implementação. Para implementar o modelo lógico resultante do modelo conceptual, é necessário transformá-lo num modelo suportado pelo SGBD onde vai ser implementado [33][35].

3.3.2 Sistema de Gestão de Base de Dados

O armazenamento e o acesso aos dados de uma base de dados relacional, normalmente é feito por meio de um Sistema de Gestão de Base de Dados (SGBD), que são programas ou conjunto de programas equipados com ferramentas que permitem ao utilizador definir, construir e manipular as bases de dados.

Os SGBDs facilitam o armazenamento e o acesso aos dados numa base de dados, através de uma interface proporcionada entre os dados armazenados a baixo nível e os utilizadores ou aplicações que precisam aceder ou processar esses dados. Muitos dos SGBDs trazem incorporados recursos que facilitam muito os programadores no desenvolvimento de novas aplicações, possibilitando que os programadores se concentrem em aspetos da aplicação. Os recursos ou ferramentas proporcionados pela maioria dos SGBDs, permitem fazer validação de dados, verificar e garantir a integridade, evitar redundâncias, fazer controlo de concorrência e controlo de transações. Uma outra característica interessante de alguns SGBDs é a possibilidade do sistema recuperar de forma adequada de possíveis falhas, ou seja, existe a possibilidade de voltar a um ponto anterior antes de ter ocorrido uma falha, garantindo assim a consistência da base de dados. Para além dos recursos mencionados, muitos dos SGBDs

possibilitam que vários utilizadores acedam e manipulam simultaneamente e de forma eficiente, uma mesma base de dados [33][35].

Existem algumas regras que diferenciam um SGBD de um tradicional sistema de gestão de ficheiro (SGF), algumas dessas regras são:

- Autocontenção - esta regra também conhecida por Meta-Base de Dados, diz que para além dos dados em si, um SGBD deve armazenar completamente toda a descrição dos dados, os relacionamentos e as formas de aceder a esses dados.
- Independência dos dados - uma aplicação não precisa de sofrer alterações na sua estrutura de armazenamento ou de acesso aos dados sempre que precisar criar algo novo na base de dados, isto é, nenhuma definição dos dados deverá estar contida a nível de aplicação.
- Abstração dos dados - somente se deve apresentar ao utilizador, uma representação conceitual dos dados, sem detalhes sobre a forma de armazenamento.
- Forma de visualizar os dados - deve permitir que cada utilizador visualize os dados de forma diferente daquela previamente utilizada para armazenar os dados e cada visualização pode ser constituída por um subconjunto de dados derivados dos dados existentes na base de dados.
- Transações - um SGBD deve fazer gestão completa da integridade referencial definida no seu esquema, sem depender do auxílio da aplicação. Deve existir pelo menos uma instrução que permita gravar uma série de modificações simultâneas e uma instrução capaz de cancelar uma série de instruções, caso alguma coisa corra mal durante uma transação.
- Acesso automático - a responsabilidade de evitar o *dead-lock* (fenómeno de bloqueio que acontece muito nos sistemas de gestão de ficheiros), não pode ser da aplicação, mas sim do SGBD.

Uma das principais características da maioria dos SGBD é implementação do ACID que é um acrónimo derivado das primeiras letras das seguintes propriedades [33]:

- Atomicidade - propriedade que garante que uma transação seja efetuada com sucesso na base de dados. Se alguma falha ocorrer durante a execução, então nenhuma parte da operação será efetuada, ou seja, a transação será desfeita.

- Consistência - propriedade que garante que as transações isoladas (sem qualquer outra transação executando simultaneamente) preservam a consistência da base de dados. As transações devem respeitar as restrições que foram impostas ao SGBD.
- Isolamento - garante que caso duas transações estejam a ser executadas simultaneamente, o sistema de controlo de concorrência do SGBD deve garantir que cada transação seja executada sem estar ciente das outras transações existentes no sistema, ou seja, as transações devem ocorrer sem interferências de outras transações, como se estivesse isolado das demais.
- Durabilidade - após uma transação ser concluída com sucesso, as mudanças que ela efetuou na base de dados persistem, mesmo que haja qualquer falha no sistema.

Existem no mercado várias soluções de SGBD, muitas delas são *open source*, enquanto outras requerem uma licença para usá-las. Nas soluções *open source*, os mais populares são: o MySQL, o PostgreSQL e o Apache Derby. Por outro lado, os mais populares nas soluções pagas são: o Oracle Database e o SQL Server [37].

3.3.3 Normalização de dados no Modelo Relacional

Para conseguir um armazenamento consistente e um eficiente acesso aos dados em bases de dados relacionais, é necessário saber organiza-los. A técnica utilizada para este fim é designado por normalização. A normalização é um processo que permite examinar os atributos de uma entidade de forma a prevenir de eventuais anomalias que possam surgir principalmente nas operações CRUD (*Create, Read, Update, Delete*) e tem como principais objetivos, estruturar relacionalmente uma base de dados de forma a suportar dados de um determinado universo, reduzir a redundância dos dados, garantir a consistência dos dados, reduzir o espaço de armazenamento e proporcionar facilidade de manipulação e manutenção dos dados [32][35][37].

Quando se aplica as regras de normalização de dados, em alguns casos certos atributos dão origem a novas tabelas, o que acaba por gerar no final um número maior de tabelas do que as que existiam originalmente. Mas por outro lado, esse aumento do número de tabelas pode ser compensado pelos benefícios que advém da normalização.

Uma série de regras são aplicadas durante o processo de normalização em cinco estágios diferentes, que vão desde a 1ª forma normal (1FN), até a 5ª forma normal (5FN). Para além dessas cinco formas, existe uma forma que se pode considerar intermédia (entre a 3FN e a 4FN), conhecida como Forma Normal de Boyce-Codd. Existem pré-requisitos para alcançar

cada uma dessas formas e esses pré-requisitos são cumulativos, isto é, para alcançar uma forma normal, como por exemplo a 3FN, tem que se atender aos pré-requisitos das formas normais anteriores, neste caso, a 1FN e a 2FN. As condições que têm que ser cumpridas para chegar a cada uma das formas normais, são descritas a seguir:

Primeira forma normal (1NF) – uma relação estará na 1NF quando não contiver atributos compostos, ou seja, quando todos os atributos que o compõem são atómicos. Essa forma elimina valores repetidos, transferindo-os para tabelas separadas [36][37]. No exemplo da Figura 3.4, em relação à tabela T_0 se considerarmos que um evento pode ocorrer em mais do que um local (por exemplo, a Semana Académica da Universidade do Algarve tem eventos em vários locais da cidade de Faro), então a tabela T_0 não se encontrava na 1NF.

Alcançar a primeira forma normal, poderá passar por criar uma nova tabela e decompor o atributo "Locais". O resultado da 1NF seria as duas tabelas T_1 e T_2, que foram decompostas a partir de T_0 **(**Figura 3.4**)**.

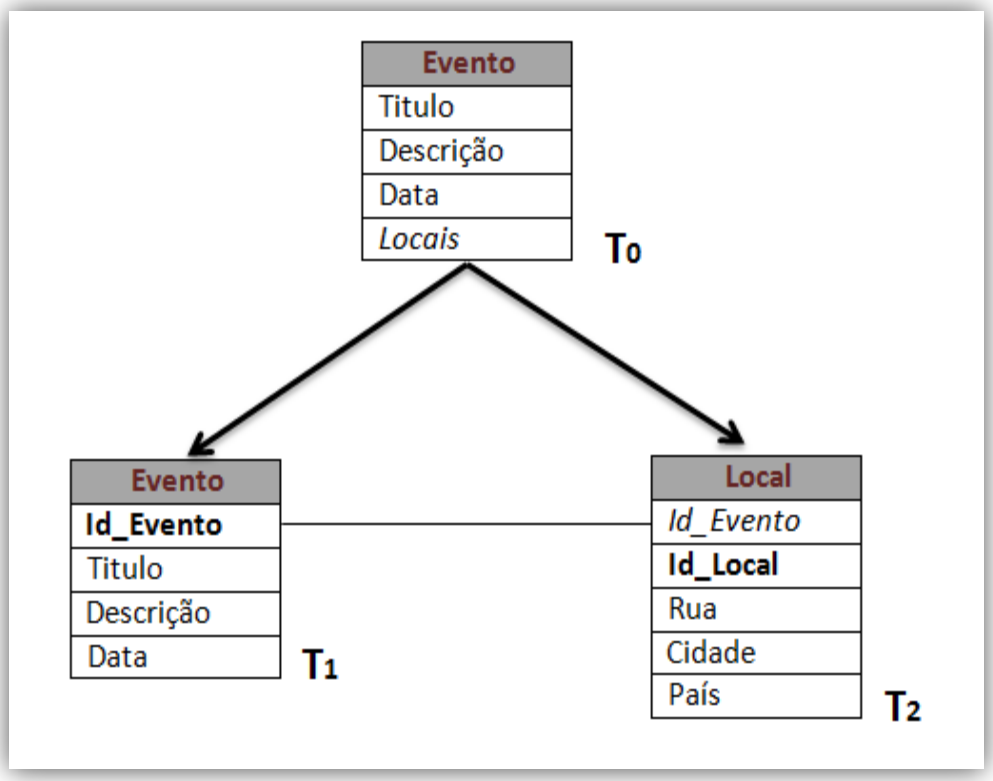


Figura 3.4: Exemplo da 1NF do Modelo Relacional

Segunda forma normal (2NF) – para alcançar a 2NF, duas condições deve ser satisfeita. A primeira é que as relações devem estar na primeira forma normal e a segunda condição exige

que todos os atributos não chaves devem depender totalmente da chave primária e não de apenas de uma parte dela [35][37] .

Terceira forma normal (3NF) - uma tabela está na 3NF se ela estiver na 2FN e não tiver nenhum atributo não-chave a depender de atributos também não-chaves [35][37].

Forma Normal de Boyce-Codd (BCNF) – pode ser considerada uma forma intermédia entre a 3NF e a 4NF. Uma tabela está na forma BCNF, se ela se encontrar obrigatoriamente na 3FN e os atributos não-chaves dependerem funcionalmente da chave primária [35][37].

Quarta forma normal (4FN) – considera-se que uma tabela está na 4NF, se estiver na 3FN e não existem atributos que não sejam participantes da chave primária, com multi-valores [32][35].

Quinta forma normal (5FN) - é aplicada a tabelas que estejam na 4FN e está vinculado a relacionamentos múltiplos, ou a tabelas cuja chave primária está relacionada com três ou mais atributos [32][35].

Aplicando corretamente as principais regras na implementação do Modelo Relacional, obtêm-se como resultado, bases de dados com uma alta consistência e com um nível de segurança elevada. No processo de normalização convém referir que as soluções raramente implementam a 5NF, uma vez que a 5NF traz benefícios em relação à redução da redundância, mas em contrapartida aumenta o número de tabelas e consequentes relacionamentos o que pode afetar significativamente a performance. Deste modo é na generalidade aceite que uma base de dados seja normalizada até a 3FN, como sendo um ponto de equilíbrio entre a performance e a redundância.

3.4 ESCALABILIDADE EM BASES DE DADOS

A escalabilidade é uma característica fundamental e desejável nos sistemas atuais, porque é cada vez mais difícil prever a quantidade de utilizadores ou o volume de dados que os sistemas terão de suportar com o passar do tempo, de modo que apostar em tecnologias escaláveis é quase uma necessidade atual.

A escalabilidade é a capacidade ou habilidade de um sistema em suportar um aumento substancial de carga sem perder o desempenho ou de aumentar o seu desempenho quando são adicionados mais recursos ao sistema [19][38].

A escalabilidade de um sistema pode ser aplicada de duas formas: verticalmente ou horizontalmente.

A escalabilidade vertical, consiste em aumentar o "poder" da máquina (nó) através de adição de mais recursos de *hardware*. Os recursos adicionais podem ser memória RAM e/ou disco rígido com mais capacidade e mais velozes, capazes de atender uma demanda crescente de requisições e armazenamento ou a troca do processador por um mais rápido. Dito por outras palavras, o que se faz na escalabilidade vertical é um *upgrade* de *hardware* do servidor, para aumentar a capacidade de processamento e armazenamento [38][39].

A escalabilidade horizontal consiste em adicionar mais máquinas (nós) na arquitetura do sistema. É uma solução considerada por muitos como mais viável para sistemas atuais cuja demanda e volume de dados crescem rapidamente [38][39].

Nas bases de dados tradicionais (Modelo Relacional), a escalabilidade mais simples de implementar é a vertical (*upgrade* de *hardware*), sendo que a implementação da escalabilidade horizontal, é um processo mais complexo requerendo em alguns casos grandes investimentos.

Os novos conceitos que têm vindo a surgir, como por exemplo o *Cloud Computing* (computação na nuvem), têm aumentado a necessidade de fornecer serviços cada vez mais escaláveis. Deste modo a escalabilidade vertical deixou de ser suficiente e a escalabilidade horizontal passou a ser quase que uma necessidade nas bases de dados.

Este conceito de escalabilidade horizontal está eminente à grande maioria dos sistemas de gestão de bases de dados que serão apresentados nas próximas secções. Como veremos, teremos de sacrificar algumas das características bases dos SGBD que implementam o Modelo Relacional.

3.5 NoSQL

Com o aumento significativo do número de dispositivos conectados simultaneamente à Internet, algumas grandes empresas do ramo das TI proprietárias de aplicações de grande porte, começaram a enfrentar dificuldades em fazer gestão do crescente volume de dados gerados e processados por essas aplicações. Além da dificuldade de escalar as bases de dados relacionais, em certas situações essas empresas precisavam de alguma liberdade na forma de

estruturar, armazenar e recuperar esses dados, algo difícil de se conseguir com o Modelo Relacional devido à sua estrutura pouco flexível [19].

Para fazer face às limitações enfrentadas com o Modelo Relacional, os projetistas começaram a pensar num modo alternativo para modelar as bases de dados, cuja solução proposta tinha como base eliminar ou minimizar a sua estrutura. Nesse sentido, algumas das empresas de TI, que estavam a enfrentar a dificuldade em lidar com grandes volumes de dados, passaram a desenvolver novas estratégias de armazenamento. Muitas das soluções apresentadas pareciam estar a recuar no tempo, para a época em que os sistemas utilizados para armazenar e manipular dados eram simples sistemas de gestão de ficheiros. Nessas novas soluções propostas, perdia-se um pouco a consistência que é um ponto forte no Modelo Relacional, mas por outro lado, esta perda é fortemente compensada com ganho de performance e flexibilidade na forma de estruturar as bases de dados. Assim nascia uma nova geração de bases de dados que foi designada de NoSQL, que promete resolver o problema de escalabilidade e proporcionar uma maior flexibilidade na estruturação das mesmas, para além de poder usufruir de um melhor desempenho e disponibilidade dos dados [40].

O termo NoSQL (*Not Only SQL*), foi utilizado pela primeira vez por Carlo Strozzi em 1998, numa solução de base de dados apresentada, que não oferecia uma interface SQL. A partir de 2009 este termo foi largamente difundido, depois da realização de um evento que tinha por objetivo falar sobre base de dados *open source* distribuídas. Neste evento, o nome NoSQL foi utilizado para descrever o surgimento de um número crescente de bases de dados não relacionais e desde então o termo passou a representar as soluções de base de dados utilizadas em casos onde o Modelo Relacional não apresentava uma performance desejada. Por outras palavras, NoSQL é um termo utilizado para definir qualquer base de dados que não adota os paradigmas do Modelo Relacional [19].

As soluções de bases de dados NoSQL não pretendem substituir por completo as bases de dados tradicionais, mas sim ser uma solução em cenários onde o Modelo Relacional não seja a melhor solução.

Em resumo, de entre as características fundamentais que diferenciam as bases de dados NoSQL das bases de dados relacionais, podemos destacar a ausência de bloqueios, que facilita a escalabilidade horizontal, contornando deste modo o problema de gestão de grandes volumes de dados que crescem rapidamente; estruturas flexíveis e suporte a replicação, uma característica nativa deste modelo.

3.5.1 Tipos de bases de dados NoSQL

Este movimento da nova geração de base de dados deu origem a quatro grandes categorias de base de dados não relacionais (NoSQL) [19][40], que poderemos de uma forma geral caraterizar do seguinte modo:

- Bases de dados de armazenamento chave-valor – de todas as categorias é a mais simples de implementar e quando bem implementada, permite obter ganhos elevados de performance. Os dados são armazenados em forma de objetos indexados por chaves e a cada chave está associado um único valor. A sua facilidade de implementação permite um acesso rápido aos dados através das chaves. Dynamo, GenieDB, Redis, Riak, Table Storage, Couchbase, são alguns exemplos de bases de dados nessa categoria.
- Bases de dados orientadas a colunas - como o próprio nome diz, são baseadas em colunas e foram projetadas para armazenar e processar grandes quantidades de dados distribuídas em diversas máquinas. É um pouco mais complexo que o modelo chave-valor. Neste modelo os dados são indexados por linha, coluna e *timestamp*, em que as linhas são identificadas por chaves e o *timestamp* serve para diferenciar versões diferentes de um mesmo dado. Exemplos de bases de dados dessa categoria: BigTable, Cassandra, Amazone SimpleDB, Hbase e Hypertable.
- Bases de Dados orientadas a documentos – os documentos das bases de dados dessa categoria são coleções de atributos (chaves) e valores. Esta categoria de base de dados permite ter documentos embebidos, em que cada documento pode ter qualquer número de campos e oferecem suporte a consultas mais eficientes. Apesar de permitir ter documentos embebidos, este modelo não exige ter uma estrutura fixa dos documentos, o que permite atualizar a estrutura do documento a qualquer momento, adicionando ou removendo campos. Exemplos de bases de dados dessa categoria são: o CouchDB, o MongoDB e o RavenDB.
- Bases de Dados orientadas a grafos – essa categoria de bases de dados é baseada na teoria de grafos. As pesquisas nas bases de dados que enquadram nesta categoria, podem ser feitas tanto por algum dos atributos dos vértices, quanto pelos relacionamentos entre os mesmos. Exemplos de bases de dados dessa categoria são: Neo4j, AllegroGraph, Virtuoso e Infinite Graph.

A tecnologia NoSQL está a ser largamente difundida e adotada em diversos domínios de aplicação, principalmente pelos gigantes da TI que contam com grande fluxo de informações, como por exemplo, Google, Amazon, Facebook, Twitter, Yahoo e eBay.

Desde que surgiu o movimento NoSQL, vários projetos têm sido desenvolvidos, promovendo diversas soluções inovadoras, no âmbito de processamento e armazenamento de grandes volumes de dados. Alguns destes projetos têm sido largamente utilizado, tais como:

- Cassandra - inicialmente desenvolvida pelo Facebook é uma base de dados distribuída e altamente escalável, implementada na plataforma Java. Reúne a arquitetura de outras bases de dados NoSQL, nomeadamente o Dynamo, Amazon e o modelo de dados do BigTable do Google. Até o ano de 2008 teve o seu código-fonte aberto à comunidade, mas atualmente é mantido por programadores da Fundação Apache e colaboradores de outras empresas [41].
- BigTable - desenvolvida pelo Google para fazer a distribuição de dados por vários servidores com o objetivo de promover uma maior disponibilidade e escalabilidade. Várias aplicações do Google usam o BigTable, tais como o Google Earth, Maps, Blogger, YouTube e Gmail. Apesar do BigTable ser proprietário, o seu modelo de dados é aplicado em algumas implementações em código aberto [1].
- Dynamo - foi um projeto desenvolvido pela Amazon em 2007 para oferecer suporte a armazenamento de dados de alta disponibilidade [42].
- CouchDB - base de dados orientado a documentos escrita na linguagem Erlang e de código-fonte aberto. A sua arquitetura proporciona replicação bidirecional e escalabilidade horizontal. Usa JSON para armazenar os dados e JavaScript como uma linguagem de consulta [43].
- MongoDB - Base de dados não relacional, também orientada a documentos e de código-fonte aberto, livre de esquema e que oferece alto desempenho. Foi desenvolvida na linguagem C++ [44].

Para além das bases de dados aqui apresentadas, muitas outras têm vindo a ser utlizadas e o número dessas soluções não para de crescer. Na secção 3.8 analisaremos com mais detalhe a solução MongoDB, justificando a sua escolha como SGBD para o projeto *LifeSpeeder*.

3.6 NOSQL VERSUS MODELO RELACIONAL

Conforme já se tinha referido anteriormente, o NoSQL não veio para substituir o Modelo Relacional. Cada um destes modelos de dados tem a sua aplicação no ambiente certo. O NoSQL por exemplo, é empregue principalmente em situações onde é necessário ter uma maior flexibilidade na estruturação das bases de dados e onde se espera um alto desempenho e disponibilidade de dados, assim como facilidade de escalabilidade. O Modelo Relacional por sua vez, continua a ter uma grande aplicabilidade, principalmente em aplicações tradicionais onde é possível estimar o número de utilizadores que farão uso do sistema e em ambientes onde a forte consistência dos dados é o principal requisito. Em termos de usabilidade, podemos assim dizer que as bases de dados NoSQL estão mais voltadas para ambientes Web, onde se espera um bom nível de desempenho, disponibilidade e sobretudo facilidade de escalabilidade à medida que aumenta o número de utilizadores e o volume de dados. Enquanto as bases de dados relacionais estão mais orientadas para ambientes corporativos, como por exemplo instituições financeiras, onde as propriedades ACID precisam ser respeitadas. Mas o Modelo Relacional e o NoSQL podem coexistir ao mesmo tempo num mesmo sistema, cada um desempenhando uma função diferente ou um como um complemento do outro [19][39][44].

3.7 TEOREMA DE CAP

Existem uma grande motivação na utilização das bases de dados NoSQL, principalmente em consequência da flexibilidade nos esquemas, o desempenho, a facilidade de escalabilidade e replicação. Mas para alcançar certos benefícios do NoSQL, alguma coisa se perde quando comparada com as funcionalidades do Modelo Relacional. Por outras palavras, para conseguir certos características do NoSQL, é necessário sacrificar algumas funcionalidades das bases de dados tradicionais [40][45].

O teorema de CAP (*Consistency, Availability, and Partition Tolerance*) foi apresentado em 2000 durante um simpósio sobre princípios de computação [45]. Este teorema afirma que num sistema é desejável obter essencialmente três requisitos: consistência, disponibilidade e tolerância a particionamento; mas que infelizmente não é possível garantir os três requisitos em simultâneo, isto é, apenas dois destes requisitos podem ser garantidos ao mesmo tempo.

A consistência (*Consistency*) é uma característica que descreve o estado de um sistema após uma operação. Para um sistema ser considerado consistente, deve garantir que após efetuar uma transação, a integridade dos dados mantem-se. Um sistema pode ser considerado fortemente consistente ou de fraca consistência. As bases de dados do Modelo Relacional que implementam as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) podem ser consideradas sistemas de consistência forte. De fraca consistência têm-se as bases de dados baseadas no conceito BASE (*Basic Availability, Soft-state, Eventual consistency*) em que o sistema pode atingir um estado de consistência, mas é uma consistência eventual, como é em geral o caso por exemplo das bases de dados não relacionais (NoSQL). Num sistema com fraca consistência, como por exemplo uma base de dados onde os dados são replicados, num determinado instante só se pode garantir que num nó existe a versão mais recente de um registo, podendo noutro nó, existir uma versão antiga do mesmo registo ou não existir sequer [40][43][45].

O conceito da disponibilidade (*Availability*) diz que um sistema deve ser implementado de forma que mesmo que haja alguma falha de *hardware* ou *software*, os serviços ou recursos disponibilizados pelo sistema, devem permanecer ativos, isto é, o acesso aos serviços e recursos fornecidos, não deve ficar indisponível [40][43][45].

A tolerância ao particionamento (*Partition Tolerance*) está relacionada com a capacidade de um sistema continuar a operar, mesmo na presença de perdas de mensagens ou da falha de parte do sistema. Isto garante que um sistema continua a realizar operações, como leitura e escrita, mesmo quando ocorre alguma falha de rede [40][43][45].

Segundo Orend [31], um sistema que é tolerante a particionamento, para fornecer uma forte consistência, tem que admitir alguns cortes na disponibilidade, isto porque para ter uma forte consistência é necessário garantir que cada operação de escrita, só termina quando os dados foram replicados em todos os nós do sistema. De modo que, tentar implementar sistemas capazes de garantir simultaneamente as três propriedades que são dados como incompatíveis no teorema de CAP, implicaria um aumento significativo dos custos computacional e da complexidade do sistema e ainda assim não havia a garantia de que o objetivo seria alcançado.

A Figura 3.5 ilustra a aplicação do teorema de CAP e mostra alguns exemplos de bases de dados onde se aplicam as propriedades. Dependendo do sistema que se pretende implementar,

pode-se optar por CA (*Consistency*, *Availability*), por CP (*Consistancy*, *Partition Tolerance*), ou por AP (*Availability*, *Partition Tolerance*).

As características CA normalmente são encontradas nas bases de dados do Modelo Relacional, enquanto AP e CP aplicam-se nas bases de dados NoSQL, que são sistemas com grande nível de tolerância a particionamento.

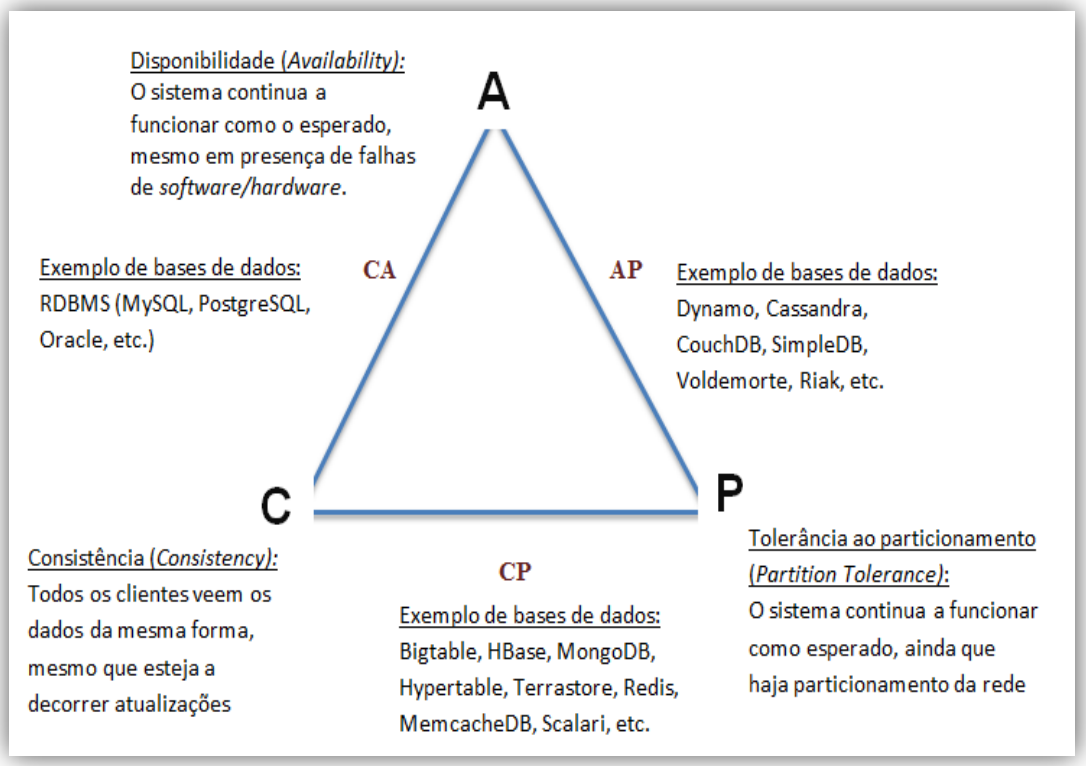


Figura 3.5: Ilustração do Teorema de CAP

3.8 A BASE DE DADOS MONGODB

No projeto da plataforma *LifeSpeeder*, em consequência da complexidade dos dados que se pretendiam armazenar, optou-se pelo modelo de dados NoSQL. A razão na escolha deste modelo, deve-se às várias vantagens proporcionadas, tanto no modo de armazenamento, como na recuperação dos dados. Das quatros categorias de base de dados não relacionais deste modelo, foi escolhida a categoria de base de dados orientadas a documentos, por se tratar de uma categoria que melhor se enquadrava nos objetivos do projeto. Dentro da categoria das bases de dados orientadas a documentos foi escolhida o MongoDB. O MongoDB conta com uma excelente documentação e é mantida por uma empresa que pode dar suporte ou fazer

melhorias no *software*. Para além disso oferece excelentes características que fazem da mesma, uma das melhores dentro da sua categoria [46][47].

MongoDB é uma base de dados de código fonte aberto, desenvolvido pela empresa 10gen [44][46]. Uma base de dados em MongoDB é formada por uma ou mais coleções. Por sua vez cada coleção poderá ter um ou mais documentos e um documento pode ter outros documentos embebidos. Fazendo uma analogia, uma coleção em MongoDB representa aquilo que no Modelo Relacional é chamado de tabela e um documento representa um registo ou linha. Ao contrário do modelo relacional, não é necessário definir uma estrutura dos dados à priori, e a qualquer momento pode-se modificar um documento, adicionando ou removendo campos. A Tabela 3.1 mostra uma comparação dos termos utilizados nas bases de dados relacionais e no MongoDB:

BD Relacional	**MongoDB**
Tabela	Coleção
Linha (Registo)	Documento
Coluna (Atributo)	Campo

Tabela 3.1: Termos utilizado numa BD Relacional e no MongoDB

O MongoDB tem uma forma de armazenamento dos dados totalmente diferentes das bases de dados relacionais. Enquanto nas bases de dados relacionais os dados são armazenados num esquema rígido de tabelas constituídas por linhas e colunas, unidas por relacionamentos, nas bases de dados orientadas a documentos, os dados são armazenados em formas de documentos. Para cada documento de uma coleção, é gerado automaticamente uma chave (_id), que funciona como um identificador único do documento, havendo a opção de criar índices manualmente. A API utilizada no MongoDB é uma mistura de objetos JSON (*JavaScript Object Notation*) e funções de *JavaScript*. Os documentos são criados utilizando o JSON, mas para o armazenamento o MongoDB usa outro formato chamado de BSON [44], que é uma forma binária de JSON e que oferece uma maior eficiência no processo de armazenamento. A interação com o MongoDB para armazenar ou manipular os dados, pode ser feito através da consola, por meio de alguma ferramenta GUI (*Graphical User Interface*) ou também através de alguma interface desenvolvida pelo próprio programador que estiver a fazer uso dela. Uma caraterística importante do MongoDB é a velocidade, principalmente nas

gravações, em que primeiro os dados são armazenados na memória e posteriormente em segundo plano, são gravados em disco.

Um exemplo simples de um documento em MongoDB é mostrado na Figura 3.6 onde podemos constatar que apenas num documento pode ser guardada, a informação completa referente a um utilizador. Tal deve-se a uma das características do MongoDB, que permite ter documentos embebido em outros documentos. No exemplo apresentado o "endereço" e "contato" são documentos embebidos.

```
{
"id": ObjectId("50b18f6a643453786"),
"name":"Sandra Fernandes",
"password":"fernandes",
"username":"fernandes",
"dateofbirth":ISODate("1988-06-10"),
"address":{
"country":"Cabo Verde",
"city":"Mindelo"
},
"contact":{
" email":[
"sfernandes@hotmail.com",
"fernandes@gmail.com"
],
"phone":[
"+2389875402",
"+351963445633"
]
}
}
```

Figura 3.6: Exemplo de um documento em MongoDB

Como já foi referido, o MongoDB apresenta caraterísticas interessantes, que o tornam numa base de dados adequada para situações onde se espera uma grande disponibilidade dos dados, performance e facilidade de escalabilidade [46][48]. Uma dessas caraterísticas é a simplicidade das consultas, sem transações e sem joins, o que reduz o tempo de resposta. Por exemplo no documento apresentado na Figura 3.6, para saber a informação completa do utilizador "Sandra Fernandes", bastava efetuar uma consulta simples, como por exemplo: db.user.find({'id':'50b18f6a643453786'}). O mesmo esquema numa base de dados relacional normalizada, seria normalmente traduzido por pelo menos três tabelas de forma que os documentos embebidos ("endereço" e "contato") estariam em tabelas separadas uma vez que

são atributos multivalores. Deste modo uma consulta para obter toda a informação do utilizador "Ana Bela Silva", envolveria neste caso três tabelas, o que tornaria a consulta mais complexa e mais demorada.

Existem no entanto outras características que tornam o MongoDB interessante e que iremos expor de seguida.

3.8.1 *Sharding* no MongoDB

O *Sharding* é uma técnica muito usada atualmente para escalar sistemas. Trata-se de um conceito simples que permite dividir os dados em diversos nós (máquinas), aumentando deste modo a performance e a capacidade de armazenamento. Esta técnica é bastante interessante principalmente para sistemas que processam conteúdos multimídia de alta definição (*HD*), sistemas de mensagens em tempo real, sistemas que trabalham com geolocalização, entre outros [44][46].

No MongoDB, existe um módulo de *Shadring* que permite a construção de um *cluster* de base de dados escaláveis horizontalmente. Os principais constituintes de um *cluster* em MongoDB são:

- *Shards* – são blocos de dois ou mais servidores. Em cada servidor corre um processo *mongod* que é responsável pelo sistema de armazenamento estruturado e pela gestão das requisições de dados.
- Servidores de configuração – servem para armazenar as configurações do *cluster*. Nestas configurações estará o mapeamento dos *shards* que constituem o *cluster*.
- *Mongos* – processo que faz o roteamento e a coordenação das configurações dos *Shards* e faz interface com a camada de aplicação. Do ponto de vista da aplicação, *mongos* comporta de forma idêntica a qualquer instância do MongoDB.

À medida que a demanda aumentar, basta acrescentar mais *shards* ao *cluster* para continuar a garantir o desempenho do sistema. Na Figura 3.7 podemos ver um exemplo de *cluster*, bem como a arquitetura de *sharding* do MongoDB [44][46][47].

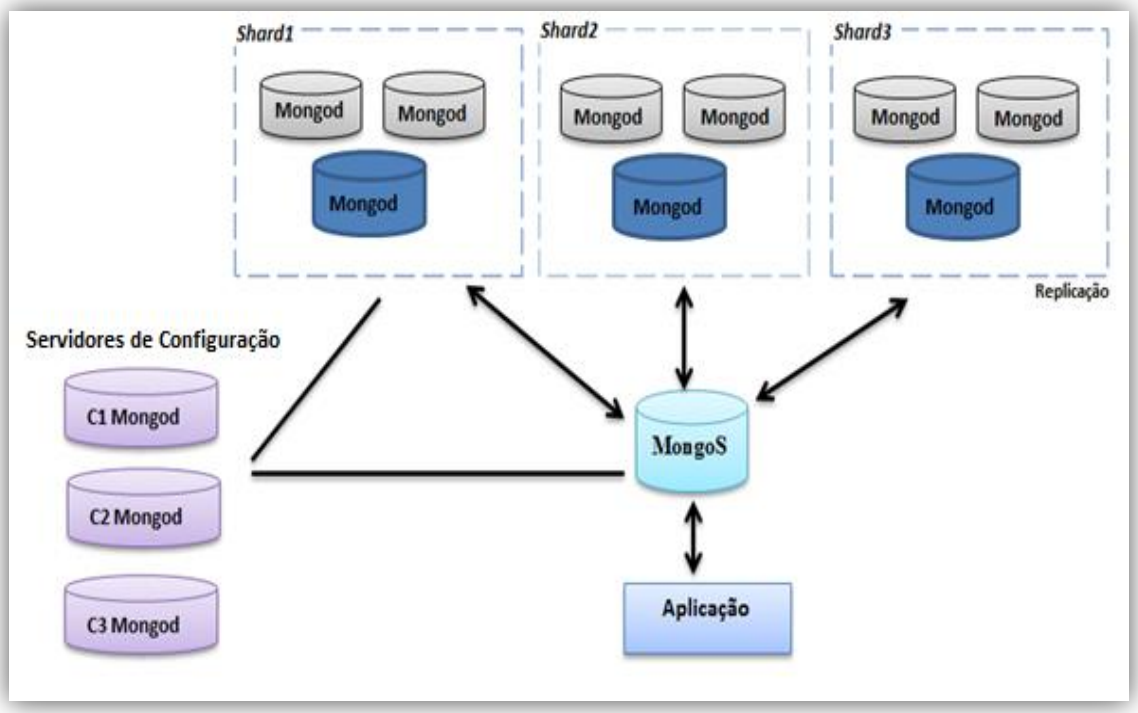


Figura 3.7: Esquema de implementação de *Sharding* no MongoDB

3.8.2 GridFS

O *GridFS* é um outro recurso útil do MongoDB e serve para armazenar ficheiros na base de dados. O seu uso é recomendado principalmente quando o tamanho dos ficheiros a serem armazenados ultrapassa os 4Mbyte. Este recurso é bastante interessante principalmente se a aplicação que o utiliza, permita aos seus utilizadores fazer partilha de ficheiros (e.g., fotos, vídeos, áudios). Utilizando o *GridFS* o ficheiro é diretamente armazenado dentro da base de dados MongoDB, o que pode diminuir muito o tempo de acesso ao ficheiro [44][46].

3.8.3 Geolocalização

O termo geolocalização (*geolocation*) refere-se à ação de obter a localização geográfica de um lugar de interesse ou a posição de um equipamento. Essa localização é obtida na maioria das vezes através do cálculo da longitude e latitude, utilizando algum dos métodos disponíveis nos dispositivos (GPS, IP, Rede Móvel, etc).

No que se refere à metodologia utilizada para conseguir obter a localização, atualmente é possível obter a posição geográfica de um determinado dispositivo de diversas formas, cujos métodos mais populares são [49]:

- **Geolocalização IP** – é um dos métodos mais utilizado pelos navegadores, que através de consultas *whois* (protocolo TCP que permite consultar informações de contacto e DNS sobre entidades na Internet) e serviços de localização de IP, permite determinar em que região ou cidade um utilizador se encontra.
- **Triangulação GPRS** – este método permite determinar a posição através de triangulação de antenas GPRS (*General Packet Radio service*) próximas de dispositivos que estão ligados a uma operadora móvel, mas que não tem GPS ou que tenham o GPS desligado, apresenta uma maior precisão em relação ao método baseado em IP.
- ***Global Positioning System* (GPS)** – comparado com os outros métodos, a utilização do GPS oferece uma maior precisão no cálculo da posição geográfica, com pequena margem de erro em condições ideais.

Em alguns casos, a localização que se obtém é uma localização aproximada, principalmente se o método utilizado for o geolocalização IP, porque para obter uma posição com mais precisão é necessário que o utilizador concorde em partilhar a sua localização, o que nem sempre acontece.

O MongoDB tem o suporto nativo para trabalhar com sistemas de coordenadas geográficas. As coordenadas geografias podem ser armazenadas dentro de um *array* (e.g., *loc* : [-7.930440, 37.019355]) ou como um documento embebido (e.g., *loc* : {*lng* : -7.930440, *lat* : 37.019355}). Antes de poder utilizar as funções de geolocalização nos *queries*, tem que criar os índices correspondentes (*Geospatial Indexes*), que permitem o MongoDB reconhecer aquele campo (*loc*) como um campo de geolocalização [44][46].

3.8.4 MapReduce

O MapReduce é um modelo de programação que permite processar em paralelo, grandes volumes de dados e cujo princípio consiste em dividir uma tarefa maior em subtarefas independentes. Isto é, cada subtarefa é executada de forma independente e depois o resultado de cada subtarefa são combinados para produzir o resultado final. Conforme esquematizado na Figura 3.8, o modelo de programação MapReduce, é constituído por duas fases principais:

- Fase de mapeamento – consiste em decompor a tarefa maior em subtarefas menores e executá-las para produzir resultados intermédios.

- Fase de redução - nesta fase será feita a combinação dos resultados intermédios e a seguir será produzido o resultado final

A ideia do *MapReduce* foi inicialmente proposta pelo Google em 2004 num artigo publicado que demostrava como o modelo *MapReduce* pode ser usado para processar grandes volumes de dados em várias máquinas em simultâneo. Uma máquina no *cluster* assume o papel de um nó-mestre e é a responsável por dividir uma entrada/tarefa em subtarefas e distribuí-las pelos nós-trabalhadores. As subtarefas são executadas em paralelos pelos nós-trabalhadores que depois devolvem os resultados para o nó-mestre que encarrega de combiná-los e produzir o resultado final [44][48].

Este modelo de programação proposto pelo Google foi rapidamente adotado por outras empresas, sendo que algumas lançaram os seus próprios *frameworks* que implementam o *MapReduce*, como é o caso do MongoDB.

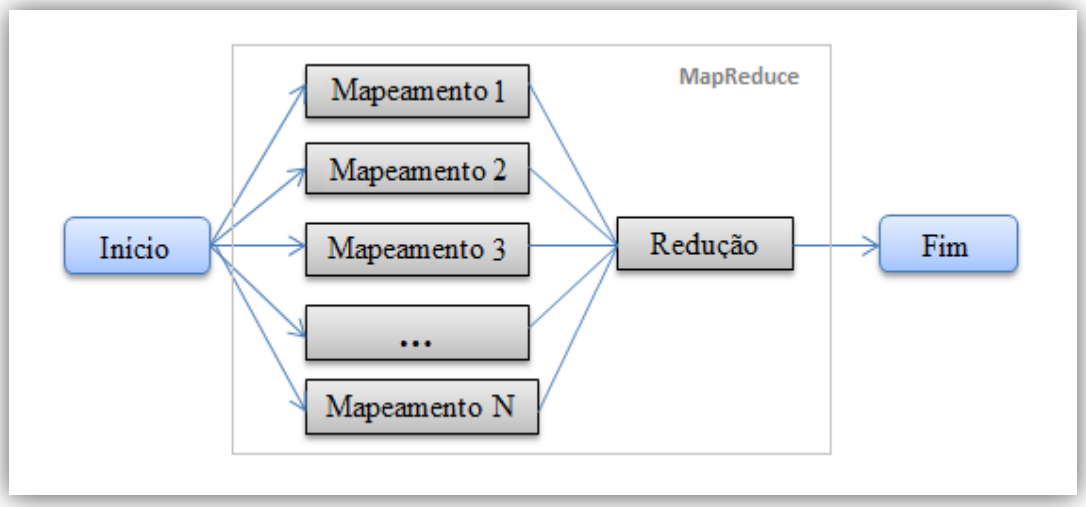


Figura 3.8: Exemplo de *MapReduce*

3.8.5 Replicação

A replicação é outra característica interessante e importante do MongoDB. A sua aplicação permite garantir a durabilidade dos dados através da replicação dos mesmos por diversos nós. Numa configuração de replicação, a base de dados está hospedada em diversos nós ao mesmo tempo, com um dos nós a funcionar como mestre (*master*) e os restantes nós como escravos (*slaves*). Todas as operações de escritas ocorrem no nó *master*, sendo os restantes nós (*slaves*) para leitura, conforme esquematizado na Figura 3.9. As alterações nos dados são copiadas de forma assíncrona do *master* para os *slaves*. Se ocorrer alguma falha no nó *master*, continua a

ser possível ler os dados dos nós *slaves*, havendo a possibilidade de configurar um dos nós *slave* para que atue como *master* [44][46] [47].

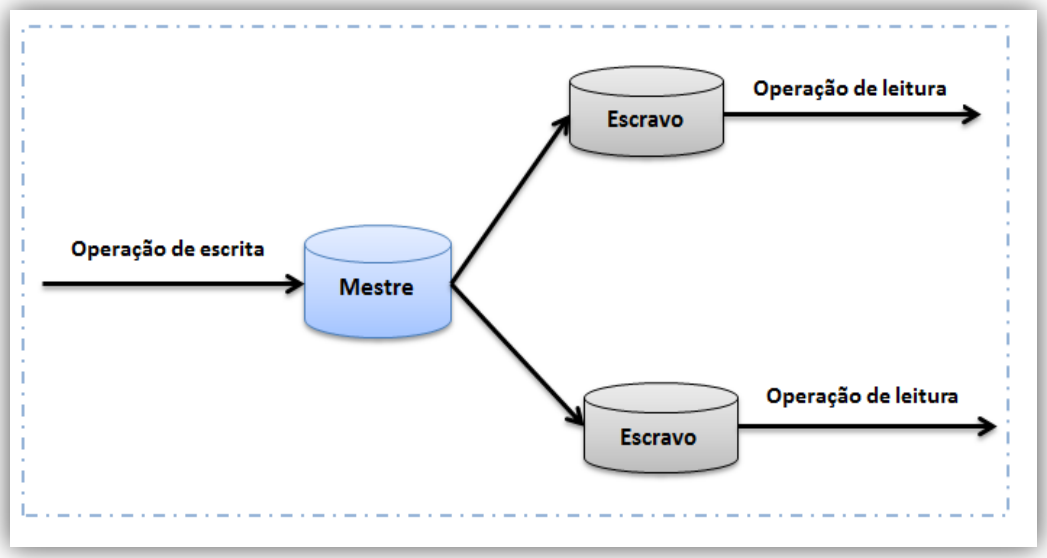


Figura 3.9: Replicação de dados no MongoDB

No capítulo seguinte será apresentada uma análise breve comparativa dos dois modelos de dados apresentados aqui.

4. MODELO DE DADOS DO LIFESPEEDER

Neste capítulo serão apresentados os critérios utilizados na escolha da base de dados que servirá de suporte da plataforma *LifeSpeeder*. Começaremos por apresentar uma visão geral das principais entidades que farão parte da base de dados, seguido de um breve estudo comparativo entre o Modelo Relacional e o NoSQL em relação a estruturação dos dados que se pretende armazenar.

4.1 VISÃO GERAL DO PROJETO

Comecemos por dar uma visão muito geral do *LifeSpeeder*. As aplicações que constituem a plataforma *LifeSpeeder* serão suportadas por uma mesma base de dados. Algumas das principais entidades desta base de dados serão apresentadas a seguir. Dependendo do modelo de dados em análise, estas entidades consideradas como principais poderão dar origem a novas entidades ou a documentos embebidos. Por exemplo se consideramos a entidade "países", no Modelo Relacional esta entidade pode originar pelo menos mais duas entidades ("distritos" e "cidades") e no MongoDB a mais dois documentos embebidos ou subdocumentos. Deste modo, as entidades consideradas como as principais são as seguintes:

- "**Utilizadores**" – entidade que armazenará as informações dos utilizadores, como por exemplo, o nome, o email, o *username* e *password*, o tipo de utilizador, entre outras.
- "**Tipos de eventos**" – entidade que armazenará os tipos e subtipos ("subtipos" será uma nova entidade ou documento embebido) de eventos em diferentes idiomas.
- "**Países**" – esta entidade armazenará dados dos países, tais como, o código e nome do país, distritos e cidades (que serão novas entidades) que fazem parte do país.
- "**Idiomas** " – nesta entidade será armazenado um conjunto de idiomas.
- "**Eventos**" – será a entidade que armazenará as informações dos eventos inseridos pelos utilizadores. Dependendo das informações à armazenar em relação a um evento, essa entidade também pode dar origem a novas entidades, como por exemplo, a entidade "tradução" que armazenará as diferentes traduções que um evento pode ter.

Para inserir um evento na base de dados, o utilizador precisa estar registado no sistema. Os eventos deverão conter informações suficientes, tais como, o tipo de evento, o local, a data/hora da realização do mesmo, entre outros. Para facilitar a inserção dum evento pelo utilizador, certas informações, como por exemplo, o tipo (e.g., "Universitário") e subtipo (e.g., "Palestra") de evento, o país (e.g., "Portugal"), a cidade (e.g., "Faro"), o idioma (e.g., "Português"), poderão ser selecionados diretamente de uma lista (*drop-down*). Portanto, essas informações precisam ser previamente armazenadas na base de dados.

No caso da aplicação móvel, a comunicação com a base de dados dá-se por meio de um *Web Service*, que interpretará as requisições e fica encarregada de solicitar as informações requeridas à base de dados. A resposta que o *Web Service* devolva a aplicação móvel deverá ser um ficheiro com o resultado da consulta, num formato de dados facilmente interpretada pela aplicação (e.g., JSON).

A Figura 4.1 apresenta uma visão macro da plataforma *LifeSpeeder*. Qualquer utilizador em qualquer local do globo, que dispõe de um dispostito com Android (e.g., *Smartphone*, *tablet*) ou um computador (e.g., PC, Mac, *Notebook*) ou mesmo um outro dispositivo qualquer (e.g., iPad) com um *browser* e com acesso a internet, poderá tirar proveito desta plataforma.

Figura 4.1: Visão da plataforma LifeSpeeder

Os restantes detalhes das aplicações que constituem a plataforma *LifeSpeeder* serão dadas no capítulo 5. Nas próximas secções que se seguem são apresentados os critérios utilizados na escolha da base de dados que serve de suporte da referida plataforma.

4.2 A ESCOLHA DA BASE DE DADOS PARA O *LIFESPEEDER*

No projeto da plataforma *LifeSpeeder*, foram analisadas as duas possibilidades de implementação para a base de dados: o Modelo Relacional e o NoSQL. O primeiro a ser analisado foi o Modelo Relacional. Depois de fazer um levantamento dos dados que seriam armazenados na base de dados, iniciou-se a modelação dos mesmos. Durante o dimensionamento da base de dados utilizando o Modelo Relacional, vários obstáculos foram aparecendo aos poucos à medida que se se avançava. Algumas das razões destes obstáculos, têm haver por exemplo, ao facto da plataforma requerer suporte multilingue e dos eventos poderem conter diversas atividades, realizadas em locais e em datas diferentes. Tratar das traduções e estruturar os eventos no Modelo Relacional com uma normalização adequada, exigiam um número elevado de tabelas e respetivos relacionamentos entre elas, o que provavelmente tornaria o processo de obtenção de dados mais pesado, dado a necessidade de efetuar muitos *joins*. Além disso, sendo difícil prever todos os tipos de características dos eventos a armazenar, possíveis atualizações ao modelo de dados podia implicar alterações significativas no código. Na Figura A-1 do Anexo A, está representado o esquema a nível lógico, com tabelas e relacionamentos, daquilo que seria a base de dados do *LifeSpeeder* no Modelo Relacional.

Utilizando o NoSQL, mais propriamente a base de dados MongoDB, conseguiu-se ganhar uma certa flexibilidade na estrutura da base de dados, nomeadamente o facto já referido de permitir ter documentos embebidos dentro de outros documentos. Apesar de esta base de dados ser *schema-less*, convém criar uma estrutura (padrão) dos documentos de modo a facilitar o processo de recuperação os dados. Por outras palavras, *schema-les*s não quer dizer ausência de estrutura, mas sim, ter a liberdade no modo de estruturar a base de dados, adicionar ou remover campos, a qualquer momento sem afetar o funcionamento do sistema. Deste modo foi necessário planificar a base de dados, de forma a saber que coleções seriam necessárias e que informações cada coleção devia conter. Na Figura 4.2 está esboçado o modo como um evento pode ser estruturado na base de dados. Todos os retângulos dentro dos

esquemas (e.g., “Tradução”, “Comentários”, “Bilheteira”, etc.) são documentos embebidos. Caso se tratasse do Modelo Relacional, cada um desses documentos embebidos, seriam um forte candidato a nova tabela. O esquema **A** da Figura 4.2 representa um evento que podemos considerar um dos mais simples. Por outras palavras, um evento único, realizado num único local e numa única data. Por outro lado, no esquema **B** da mesma figura, temos um evento composto, isto é, um evento que tem várias atividades e cujas atividades são realizados em locais e datas diferentes.

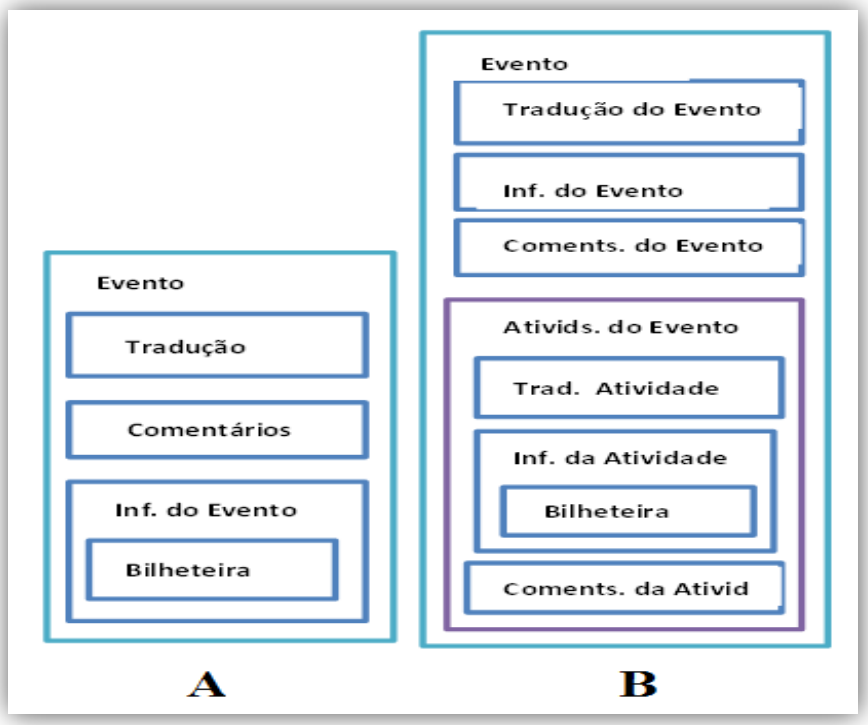
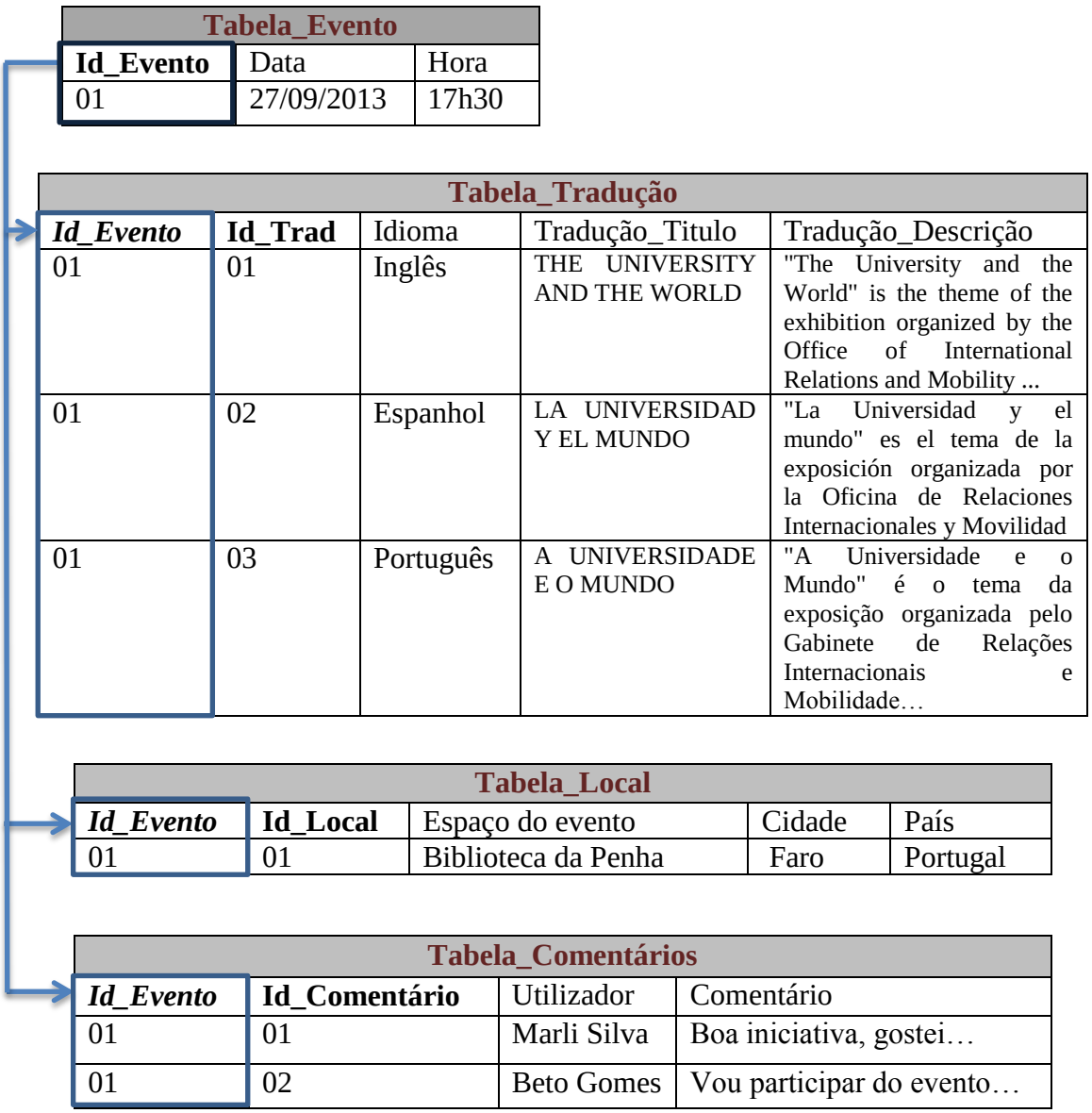


Figura 4.2: Esquema de um documento simples (A) e complexo (B) no MongoDB

Na Figura B-2 do Anexo B é apresentado um exemplo de um documento que será um evento na base de dados MongoDB. O documento representa um evento que tem mais que uma atividade, realizadas em locais diferentes, isto é, um evento com uma estrutura aproximada daquela apresentada no esquema da Figura 4.2 (**B**).

Para exemplificar a flexibilidade na forma de estruturar os dados no MongoDB em relação ao Modelo Relacional, é também apresentado um exemplo que mostra como um mesmo evento podia ser estruturado e armazenado nestes dois tipos de base de dados. No exemplo apresentado são utilizadas quatro tabelas no Modelo Relacional (“Tabela_Evento”, “Tabela_Tradução”, “Tabela_Local” e “Tabela_Comentários”) que contêm informações de um determinado evento, conforme mostra a Figura 4.3. Nas tabelas apresentadas, há alguns

atributos que a princípio deviam estar em tabelas separadas (e.g., "Idioma", "Cidade", "País", "Utilizador"), mas para simplificar o exemplo, as informações destes campos foram inseridas diretamente. O mesmo se aplica ao MongoDB. Ainda que não seja obrigatório criar uma estrutura de armazenamento, para uma boa organização e localização dos dados, o campo "Cidade" e "País" convinham estar numa coleção separada, assim como o campo "Utilizador", por uma questão de uma melhor organização dos dados.



Tabela_Evento

Id_Evento	Data	Hora
01	27/09/2013	17h30

Tabela_Tradução

Id_Evento	**Id_Trad**	Idioma	Tradução_Titulo	Tradução_Descrição
01	01	Inglês	THE UNIVERSITY AND THE WORLD	"The University and the World" is the theme of the exhibition organized by the Office of International Relations and Mobility ...
01	02	Espanhol	LA UNIVERSIDAD Y EL MUNDO	"La Universidad y el mundo" es el tema de la exposición organizada por la Oficina de Relaciones Internacionales y Movilidad
01	03	Português	A UNIVERSIDADE E O MUNDO	"A Universidade e o Mundo" é o tema da exposição organizada pelo Gabinete de Relações Internacionais e Mobilidade…

Tabela_Local

Id_Evento	**Id_Local**	Espaço do evento	Cidade	País
01	01	Biblioteca da Penha	Faro	Portugal

Tabela_Comentários

Id_Evento	**Id_Comentário**	Utilizador	Comentário
01	01	Marli Silva	Boa iniciativa, gostei…
01	02	Beto Gomes	Vou participar do evento…

Figura 4.3: Tabelas com dados do evento exemplificado

Uma das principais características da base de dados MongoDB conforme já tínhamos visto anteriormente, é a possibilidade de ter vários documentos embebidos num mesmo documento.

Na Figura 4.4 podemos notar como é que as quatro tabelas que representam o evento ("A UNIVERSIDADE E O MUNDO") no Modelo Relacional, podem ser transformadas num único documento na base de dados MongoDB. Para isso, na estrutura do documento criado, três das quatro tabelas ("Tabela_Tradução", "Tabela_Local" e "Tabela_Comentários") da Figura 4.3, passaram a ser documentos embebidos. Essa possibilidade de ter documentos embebidos num mesmo documento é uma das razões da performance da base de dados MongoDB. Enquanto no Modelo Relacional uma consulta que devolve toda a informação do evento, seria necessário fazer alguns *joins*, uma vez que estaria envolvida na consulta no mínimo quatro tabelas, que no MongoDB como as informações estão num único documento e não existe *joins*, o tempo de resposta numa consulta que retorna o documento, será menor.

Representação do documento do evento do exemplo no MongoDB

```
{
  "Id_Evento": "1",              // ID do evento, podia ter utilizado o _id gerado pelo MongoDB
  "Tradução_Evento": [      // array com  o evento nos 3 idiomas (português, inglês, espanhol)
          { "Idioma": "Português",          // evento em português
            "Título": " A UNIVERSIDADE E O MUNDO",
            "Descrição": " "A Universidade e o Mundo" é o tema da exposição organizada pelo
                          Gabinete de Relações Internacionais e Mobilidade…" },
          { "Idioma": "Inglês",             // evento em inglês
            "Título": " THE UNIVERSITY AND THE WORLD",
             "Descrição": " "The University and the World" is the theme of the exhibition
                           organized by the Office of International Relations and Mobility ..." },
          { "Idioma": "Espanhol",           // evento em espanhol
            "Título": " LA UNIVERSIDAD Y EL MUNDO",
             "Descrição": " "La Universidad y el mundo" es el tema de la exposición organizada
                           por la Oficina de Relaciones Internacionales y Movilidad ..."},
            ],
"Data_Hora":{  "Data": "27/09/2013":
                 "Hora": "17h30"      },
"Local": [   { "Espaço_do_evento": "Biblioteca de Gambelas ",
               "Cidade": "Faro",
               "País": "Portugal" }],
"Comentários": [{ "Utilizador": " Marli Silva ",
                  "Comentário": " Boa iniciativa, gostei…" },
                { "Utilizador": "Beto Gomes",
                  "Comentário": " Vou participar do evento…"}
            ]
 }
```

Figura 4.4: Comparação entre o Modelo Relacional e o MongoDB em termos de estruturação dos dados

Ainda na análise dos dois modelos de dados em estudo, com o objetivo de determinar qual o mais adequado para suportar o armazenamento no projeto *LifeSpeeder*, foi realizado um teste comparativo em relação a performance entre uma base de dados do Modelo Relacional e a base de dados MongoDB. A base de dados Modelo Relacional utilizada no teste foi o MySQL. O computador utilizado no teste tinha as seguintes caraterísticas:

- Sistema Operativo: Windows 7 Profissional (64 bits)
- CPU : Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz 3.10 GHz
- Memória RAM : 8,00 GB

Os *software* utilizados no teste, foram:

- MongoDB versão 2.2.2
- Servidor Apache versão 2.2.22
- MySQL versão 5.5.24
- PHP versão 5.4.3

O teste consistiu na operação de escrever e ler os dados do evento que foi apresentado no exemplo anterior (Figura 4.3 e Figura 4.4). O primeiro teste iniciou com 10 registos e foi aumentando este valor até 1 milhão de registos. Cada teste foi repetido cinco vezes, sendo que os resultados apresentados nas tabelas seguintes, referem ao tempo médio em segundos. Na Tabela 4.1 estão resumidos os resultados obtidos, onde podemos constatar que o tempo médio de inserção para o MongoDB é inferior ao tempo médio do MySQL em todos os casos. Para o número mais elevado de registos (1 milhão), o MongoDB foi pelo menos quatro vezes mais rápido que o MySQL, o que é uma mais-valia para sistemas que interagem em tempo real com utilizadores e precisam estar constantemente a gravar e atualizar os conteúdos.

Nº de Registos	MongoDB	MySQL
10	0.000556325912	0.032444462013
100	0.006245326995	0.108494615554
1000	0.029833507537	0.934835767745
10000	0.302766561508	8.892370939254
100000	3.063234043121	89.82641859054
1000000	36.04251680374	896.5549920558

Tabela 4.1: Tempo médio de escrita dos registos (segundos)

De modo semelhante, na Tabela 4.2 estão representados os tempos médios de leitura dos registos das duas bases de dados. Apesar das diferenças do tempo médio de leitura não ser tão acentuada, o MongoDB continuou a ser mais rápido, em particular para grandes números de leituras.

Nº de Registos	MongoDB	MySQL
10	0.000771665573	0.010589170455
100	0.006936359405	0.100223588943
1000	0.062518167495	0.962285995483
10000	0.617778301239	10.02487039566
100000	6.155860567092	101.4591174602
1000000	62.12916541099	1038.494795608

Tabela 4.2: Tempo médio de leitura dos registos (segundos)

4.3 A FERRAMENTA DE BASE DE DADOS ADOTADA NO *LIFESPEEDER*

Depois de ter analisado as duas possibilidades de implementação da base de dados (Modelo Relacional e NoSQL), chegou-se à conclusão que o MongoDB seria uma base de dados adequada para a plataforma que estava a ser desenvolvida. Em resumo, várias foram as razões que conduziram à escolha do MongoDB como a base de dados para a plataforma do LifeSpeeder, começando pela flexibilidade proporcionada pelo MongoDB na forma de estruturar a base de dados. Conforme já tínhamos vistos anteriormente, as bases de dados relacionais são escaláveis, sendo que a escalabilidade no Modelo Relacional é mais fácil de ser aplicada verticalmente. No entanto nem sempre a escalabilidade vertical é suficiente, sendo o problema muitas vezes resolvido recorrendo à escalabilidade horizontal, um ponto fraco das bases de dados relacionais. Tratando-se o LifeSpeeder de uma aplicação com características da Web atual, à medida que se aumenta o número de utilizadores do sistema e aumentando a quantidade de eventos, é necessário que o sistema esteja preparado para suportar a demanda de requisições provenientes dos utilizadores. Tendo em conta que a escalabilidade é uma das características natas das bases de dados NoSQL, utilizando o MongoDB é possível aplicar a escalabilidade horizontal com rapidez e facilidade, garantindo deste modo a disponibilidade do sistema. O desempenho é outro factor importante para aplicações atuais. As bases de dados NoSQL normalmente oferecem um melhor desempenho

do que as bases de dados relacionais e isso está diretamente relacionado com a estrutura das bases de dados. No Modelo Relacional temos tabelas e relações, em que qualquer operação deve respeitar as propriedades ACID, o que de certo modo tira algum desempenho ao sistema, principalmente em ambientes onde há grande número de utilizadores conectados ao mesmo tempo e é necessário fazer o controlo de concorrência. Dado que as bases de dados NoSQL baseiam-se no conceito BASE (Basic Availability, Soft-state, Eventual consistency), o desempenho tende a ser maior.

Convém referir que ao optar por solução NoSQL, neste caso em concreto o MongoDB, perder-se de certo modo algumas das vantagens que o Modelo Relacional nos traria, nomeadamente a capacidade de fazer questionários mais complexos envolvendo joins. De salientar ainda que esta situação é até certo ponto mitigada com a utilização de documentos embebidos. Quando tal não é possível, os "joins" serão feitos do lado da aplicação, o que acarreta um "overhead" mais elevado.

No capítulo seguinte será apresentado as aplicações que constituem a plataforma *LifeSpeeder*.

5. PLATAFORMA LIFESPEEDER

A plataforma *LifeSpeeder* é constituída por uma aplicação móvel e uma aplicação Web, suportadas por uma mesma base de dados (MongoDB), onde são armazenados e extraídos os dados que são manipulados por ambas. O projeto *LifeSpeeder* foi desenvolvido por uma equipa de vários elementos, sendo que, no âmbito deste documento está descrita a implementação da componente Web e da base de dados que serve de suporte para ambas as aplicações (Web e móvel) e dos serviços web que fornecem dados à aplicação móvel. Nas secções seguintes serão apresentadas as aplicações, descrevendo as respetivas funcionalidades, bem como a comunicação entre elas.

5.1 APLICAÇÃO MÓVEL

A aplicação móvel foi desenvolvida em paralelo com a aplicação Web, no âmbito desta tese está descrita a parte correspondente à comunicação desta aplicação com o *Web Service* que fornece os dados requeridos pela aplicação móvel, depois de consultar a base de dados.

Atualmente o sistema operativo para dispostitos móveis, mais utilizado em Portugal e no mundo é o Android (Figura C-3 do Anexo C). Por essa razão, para primeira versão da aplicação móvel foi escolhida a plataforma de desenvolvimento Android.

Na Figura 5.1 é apresentada interface da aplicação Móvel.

Figura 5.1: Interface de entrada na aplicação móvel

Nesta interface, o primeiro ecrã apresentado dá-nos um menu com as seguintes opções:

- Próximos Eventos – ao escolher está opção, pede-se ao utilizador para selecionar o modo que ele pretende utilizar para detetar a sua localização geográfica, que pode ser através da rede ou utilizando o GPS do seu dispositivo. Para obter uma localização com mais precisão, o GPS será a melhor escolha, mas em contrapartida, pode demorar mais tempo a obter a localização. Depois de escolher o modo para obter a localização, será calculada a localização do terminal móvel. O terminal móvel envia depois essa informação para a aplicação intermediaria (*Web Service*) que serve de interface de comunicação entre a aplicação móvel e a base de dados.

- Pesquisa Avançada – esta opção permite ao utilizador aplicar filtros nas suas pesquisas. Nessa categoria de pesquisas, o utilizador pode escolher o tipo e/ou subtipo de evento e filtrar eventos no espaço e no tempo.

- Configurações – opção que permite ao utilizador reajustar algumas funções da aplicação, de acordo com as suas preferências. Pode alterar o idioma da aplicação, alterar os parâmetros de distância máxima em relação ao espaço e ao tempo ou escolher o modo que os eventos devem ser ordenados (por distância, por data ou de acordo com os favoritos). Ainda nas "Configurações" existe a opção "Favoritos" que permite o utilizador escolher entre uma listagem de tipos e subtipos de eventos disponíveis na base de dados, quais os que ele mais prefere.

- Login – permite ao utilizador autenticar-se no sistema, caso o acesso a determinados recursos da plataforma exijam um controlo de acesso (*username* e *password).* A autenticação permitirá armazenar as opções do utilizador na base de dados de modo que estes fiquem disponíveis em todos os seus equipamentos.

Como já foi referido, todas as transações entre a aplicação Móvel e a base de dados passam primeiro pela aplicação Web, onde existe uma página específica, com a função de um *Web Service* e que se encarrega de:

1. Interpretar as requisições recebidas da aplicação móvel.
2. Montar as *queries* com os parâmetros recebidos.
3. Solicitar as informações à base de dados.
4. Devolver uma resposta à aplicação que fez a requisição.

Pelo facto de ser um formato de dados leve e proporcionar um melhor desempenho nas transações, optou-se pelo JSON para efetuar a troca de informações entre a aplicação móvel e o *Web Service*.

Depois de instalar a aplicação no terminal móvel, a primeira vez que o dispositivo fizer alguma requisição, é-lhe atribuído um *UID*, o qual passa a ser o seu identificador no sistema e em todas as requisições posteriores, deverá passar a constar o *UID* do dispositivo. Para além disso, e como também já foi referido, o *UID* servirá para registar as ações do utilizador, mantendo-as mesmo que mude de dispositivo móvel ou passe para a aplicação Web.

As quatros etapas no processo de comunicação entre a aplicação Móvel, o *Web Service* e a base de dados MongoDB, estão ilustradas na Figura 5.2.

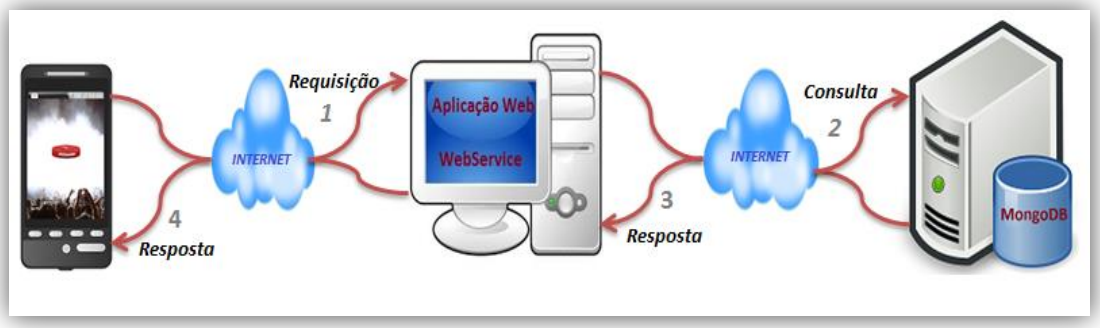


Figura 5.2: Etapas de comunicação entre a aplicação móvel, o *Web Service* e MongoDB

De um modo mais detalhado, as etapas de comunicação, desde o primeiro momento que a aplicação faz uma requisição, até quando recebe uma resposta são as seguintes:

1. A aplicação móvel, através do protocolo HTTP, inicia o processo de comunicação, enviando uma requisição através de uma URL. O *Web Service*, por sua vez interpreta essa requisição e verifica se o dispositivo enviou o *UID* de identificação. Se sim, passa diretamente para a etapa 2, se não, então o dispositivo será registado no sistema e um novo *UID* será gerado, que depois será enviado na etapa 4, junto com a resposta da requisição.
2. Depois de o *Web Service* interpretar a requisição, serão construídas as *queries* que satisfazem os parâmetros de pesquisa recebidos da aplicação móvel. Em seguida, o *Web Service* estabelece uma ligação com a base de dados e solicita as informações, de acordo com os parâmetros utilizados na consulta.

3. Em função da consulta efetuada, a base de dados devolve o resultado ao *Web Service*, que por sua vez, utilizando esse resultado, constrói um documento JSON com estes dados.
4. Com o resultado da consulta no formato JSON, o *Web Service* envia-o para a aplicação Móvel, fechando assim o ciclo de comunicação.

A Figura 5.3 mostra a primeira comunicação entre a aplicação móvel e o *Web Service*.

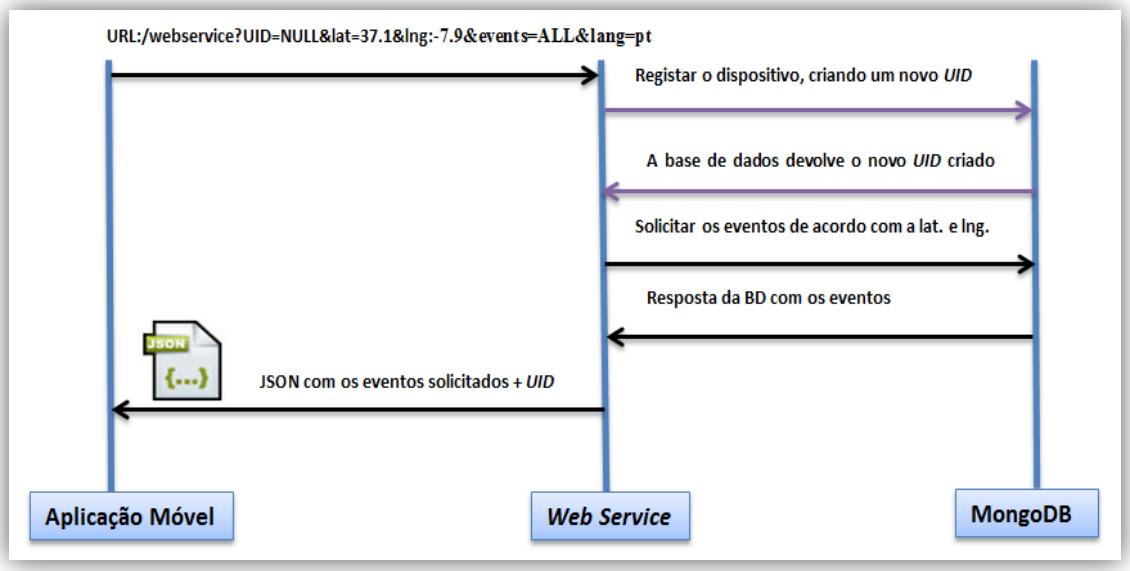


Figura 5.3: Esquema de mensagens na primeira requisição da aplicação móvel ao *Web Service*

No exemplo da Figura 5.3, através do método GET do protocolo HTTP temos o pedido traduzido pela URL: */webservice?UID=NULL&lat=37.1&lng:-7.9&events=ALL&lang=pt*. Nesta requisição o utilizador pretende ver qualquer tipo de evento ("*events=ALL"*) que esteja próximo da sua localização (*lat*=37.1 e *lng*=-7.9) e de preferência que estejam em português ("*lang:* pt"). No entanto, como é a sua primeira requisição ao *Web Service*, ainda não existe um *UID* associado ao dispositivo ("*UID=NULL*"). Logo, a primeira tarefa do *Web Service* depois de receber este pedido, consiste em registar o dispositivo no sistema. Depois da base de dados devolver o *UID* criado para este novo dispositivo, será realizada uma consulta, solicitando os eventos que satisfazem os parâmetros da requisição. Como já foi referido, no final a resposta que é devolvida à aplicação móvel, consiste num documento JSON com os eventos encontrados nas proximidades do utilizador, mais o *UID* criado para identificar o dispositivo.

Nas requisições posteriores serão omitidos os passos de atribuição de *UID*, que já virá identificado na URL.

A informação de resposta colocada num documento JSON que é devolvida à aplicação móvel, depende da requisição que foi feita ao *Web Service*. Dito por outras palavras, o JSON que é enviado à aplicação, pode ser uma lista de:

- Todos os países e cidades (no modo de pesquisa avançada e apenas uma única vez).
- Tipos e subtipos de eventos que estão cadastrados no sistema (no modo de pesquisa avançada e apenas uma única vez).
- Eventos que estejam mais próximos do utilizador. Neste caso os únicos parâmetros de pesquisa requeridos são a latitude e a longitude, que serão enviados ao *Web Service* automaticamente, ao escolher a opção "Eventos Próximos".
- Eventos que satisfazem determinados critérios, ou seja, quando o utilizador escolhe o que ele quer ver de acordo com as suas preferências (e.g., tipo/subtipo de evento, local, distância máxima).

Quando a opção de pesquisa padrão ("Próximos Eventos") é escolhida, a lista de eventos que é devolvida, resulta da localização do terminal móvel. A lista de eventos é filtrada de modo a conter apenas eventos que ainda estão por acontecer, evitando deste modo a listagem de eventos que já tenham sido realizados.

Se a opção de pesquisa escolhida for "Pesquisa Avançada", automaticamente é enviada uma requisição ao *Web Service*, solicitado a lista dos países e cidades que já estão registados na base de dados, bem como a lista de tipos e subtipos de eventos existentes. Em função disso é dada ao utilizador a opção de escolher qual o tipo ou subtipo de evento que ele está interessado, assim como o local (cidade ou país) onde ele pretende efetuar a pesquisa. Resolve deste modo situações em que não queremos saber sobre eventos próximos de nós, mas sim numa localização definida. Para além de o utilizador ter a opção de escolher o tipo e/ou subtipo de evento, país e/ou cidade, pode também especificar um intervalo de tempo (data de início e data de fim) e a distância máxima em relação a localização escolhida. Pode também selecionar o local diretamente no mapa.

A título de exemplo, na Figura 5.4 é apresentado um documento JSON em resposta à uma requisição feita por um utilizador localizado em Faro. Nesta requisição o utilizador solicita eventos do tipo "Desporto" com uma distância máxima de 50 km e que estejam em português ("lang = pt"). Conforme podemos constatar na referida figura, a resposta devolvida à

aplicação móvel consiste numa lista de eventos que satisfazem os critérios indicados pelo utilizador em questão.

Requisição - *URL:/webservice?lang=pt&lat=37.0193548&long=-7.9304397&Type=Desporto&MaxDist=50*

Resposta: JSON com um *array* de eventos


```
{
   "events": [
      {
         "id": "0",
         "type": "Desporto",
         "subtype": "Voleibol",
         "title": " Torneio de Apuramento – Voleibol masculino/feminino ",
         "description": " O I Torneio de Apuramento da Zona Nacional/Centro/Sul de Voleibol realiza-se em Faro, no Pavilhão do Sporting Clube Farense. Esta será a primeira competição que a Associação Académica da Universidade do Algarve (AAUAlg) em parceria com a Federação Académica de Desporto Universitário… ",
         "city": "Faro",
         "latitude": "   37.0227323 ",
         "longitude": "-7.9292563",
         "startdate": "12/11/2013",
         "enddate": "13/11/2013",
    },
  {
         "id": "1",
         "type": "Desporto",
         "subtype": "Regata",
         "title": "III TROFÉU - CENTRO NÁUTICO NA PRAIA DE FARO",
         "description": "O Centro Náutico da Praia de Faro irá acolher, no próximo dia 02 de novembro (sábado), o 3.º Troféu Centro Náutico, mais uma prova de aventura promovida pelo Município de Faro …",
         "country": "Portugal",
         "district": "Faro",
         "city": "Faro",
         "address": "Praia de Faro. Avenida Nascente 8005-520",
         "latitude": "37.0052078",
         "longitude": "-7.9897051",
         "startdate": "02/11/2013",
         "starttime": "11:00",
         "telephone": " +351 289870898, +351 28987005 ",
         "email": "centronauticofaro@cm-faro.pt",
         "url": "http://www.cm-faro.pt/",
        },
  {
         "id": "2",
         "type": "Desporto",
         "subtype": "Basquetebol",
         "title": "Supertaça Masculina 2013 SL BENFICA X VITÓRIA SC",
         "description": "O campeão nacional SL Benfica vai enfrentar o Vitória SC, equipa vencedora da Taça de Portugal. Vai ser de grandes emoções no Pavilhão Desportivo de Albufeira...",
         "country": "Portugal",
        "district": "Faro",
         "city": "Albufeira",
         "address": "Pavilhão Desportivo de Albufeira",
         "latitude": "37.086704",
         "longitude": "-8.2570957",
         "startdate": "19/10/2013",
         "starttime": "17:00",
        }
  ]
}
```


Figura 5.4: Documento JSON com um *array* de eventos devolvido à aplicação móvel

Para complementar a aplicação móvel, nas informações de localização dos eventos, se o utilizador pretender ver com mais detalhes onde é o local de um determinado evento, ele pode utilizar a opção "Ver no mapa", que abre o mapa do Google Maps exatamente nas coordenadas (latitude e longitude) que constam na informação do evento, podendo também utilizar o GPS do dispositivo Android para navegar até o local do evento.

5.2 APLICAÇÃO WEB

A aplicação Web pode ser considerada como a entrada principal da plataforma *LifeSpeeder*. É a partir dela que a base de dados é carregada com eventos, que depois são fornecidos para ambas as aplicações (a móvel que já foi abordado e a própria aplicação Web). As interfaces desta aplicação têm um suporte multilingue (atualmente português, espanhol e inglês), o que permite alargar o horizonte de usabilidade da plataforma. Para além das interfaces visíveis ao utilizador comum, existe uma área administrativa (*Back Office*) onde se faz a gestão dos eventos e dos utilizadores. É nesta área que, por exemplo, pode apagar um evento que foi assinalado como inadequado ou bloquear um utilizador que esteja a fazer uso abusivo do sistema.

Algumas das principais interfaces da aplicação são as seguintes páginas: "Início", "Registar", "Login", "Inserir Eventos" e página de perfil do utilizador. A página "Inicio" é a primeira interface que se vê quando entra na aplicação, onde o utilizador pode descarregar a aplicação móvel, conforme mostrado na Figura 5.5.

Alguns eventos são mostrados no rodapé da página "Início", ao clicar nalgum deles, a informação relacionado com esse evento será exibida. Nesta mesma página, clicando na aba "Pesquisar Eventos" conforme mostrado na Figura 5.6, ver-se-ão os eventos que estão mais próximos da localização do utilizador ("Eventos próximos de si") e também um formulário ("Pesquisa avançada") a partir do qual o utilizador pode iniciar as pesquisas avançadas.

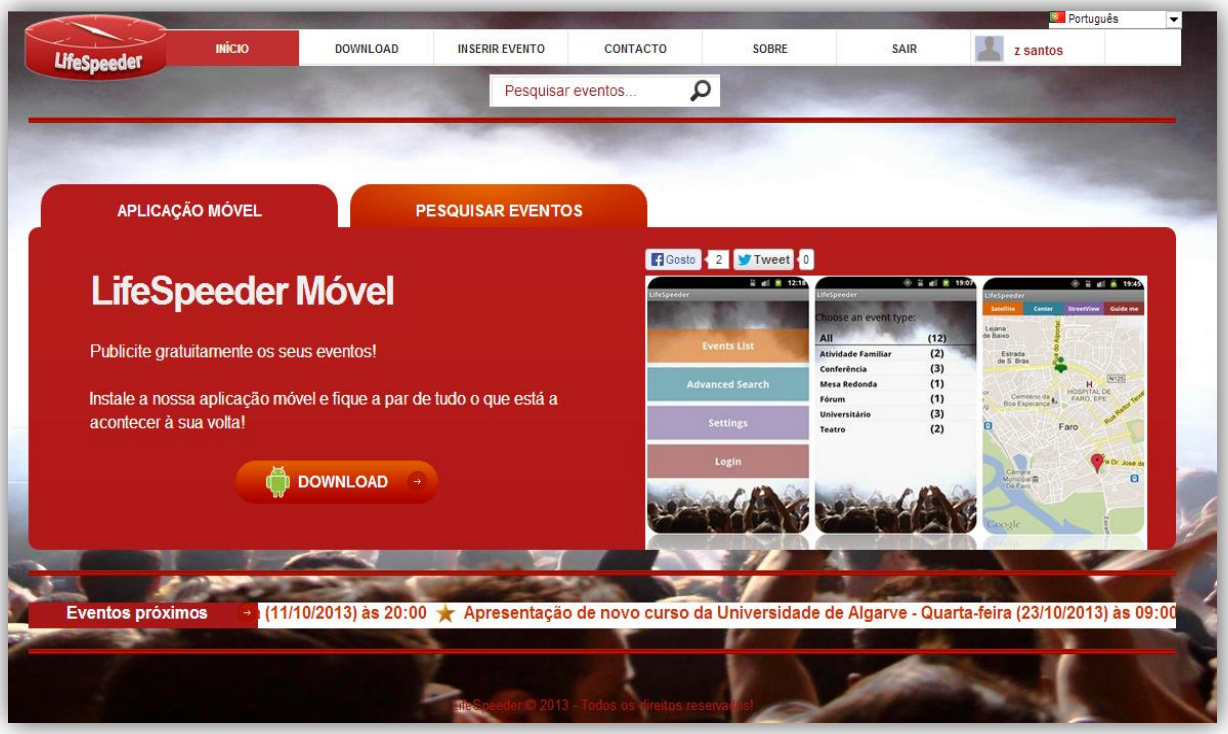


Figura 5.5: Interface de entrada na aplicação Web

Figura 5.6: Eventos próximos e pesquisas avançadas

A página "Registar", permite o utilizador criar uma conta na plataforma. A página "Login" é a interface onde o utilizador deve inserir as suas credenciais de acesso (Email e *password*), para poder ter acesso a áreas restritas, como por exemplo a página de inserção de eventos. Caso o utilizador se esqueça da sua palavra passe, existe a opção de reposição. Para isso basta escolher a opção "Esqueceu?" e indicar o seu email. Se o email digitado for encontrado no sistema, será enviado para caixa de correio eletrónico do utilizador um link que o leva a uma página onde ele pode criar uma nova palavra passe. A página "Inserir Eventos", como o próprio nome diz, é a interface onde qualquer utilizador previamente registado no sistema, pode inserir gratuitamente os seu eventos.

O utilizador registrado no sistema terá uma área pessoal (página de perfil), onde para além de poder ver e editar os seus dados pessoais, pode alterar a sua *password*, ver o histórico de eventos publicados e editar/eliminar eventos.

Tal como na aplicação móvel, na aplicação Web é possível fazer as seguintes pesquisas:

- Pesquisa normal (padrão) – com base no IP da máquina obtém-se, através da *IP Geolocation API*, a localização aproximada (para obter a localização com maior precisão, é necessário que o utilizador concorde em partilhar a sua localização) do local onde a pessoa se encontra e com base nessa informação é apresentado uma lista de eventos.

- Pesquisa avançada – do mesmo modo que na aplicação móvel, o utilizador tem a opção de escolher os tipos de eventos e locais que ele prefere. Se ele escolher o tipo e/ou subtipo de evento e não especificar o local, a lista de eventos devolvida reflete a localização estimada do terminal. Quando o utilizador escolhe o local (e.g., Mindelo, Cabo Verde), as coordenadas (latitude e longitude) do local selecionado serão obtidas por intermédio da *Google Maps API* e no caso de existirem eventos que satisfaçam as condições da pesquisa, estes serão devolvidos.

- Pesquisa livre – para além das duas formas de pesquisas atrás referidas, é possível efetuar pesquisa livre, que consiste em pesquisar por uma cadeia de carateres que o utilizador indique.

Tendo em conta que o factor tempo é importante para o utilizador, em ambas as aplicações os eventos estão organizados de forma a permitir que o utilizador obtenha diretamente o que lhe interessa. Neste sentido, quando algum item da lista dos tipos e subtipos de eventos registados

na base de dados não tiver nenhum evento associado, ou nenhum evento satisfaça os critérios da pesquisa em termos de localização e data, então os tipos e subtipos associados, não serão apresentados. Desta forma evita-se que o utilizador perca tempo a clicar num tipo de evento que esteja vazio.

No caso da aplicação Web, a página onde são listados os resultados das pesquisas de eventos está divida em três secções:

- Secção lateral esquerda – apresenta os eventos agrupados em diferentes intervalos de tempo. Cada intervalo de tempo mostra a quantidade de eventos encontrados (e.g., "Hoje (05) ", "Amanhã (07) ", "Um Mês (43) ", etc.). Caso o valor seja zero, o link do respetivo intervalo de tempo estará desativado. Ainda na referida secção é apresentada uma lista com a contagem dos tipos de eventos que aconteceram recentemente.
- Secção central – nesta secção é mostrada a quantidade de eventos, classificados por tipo. Caso a pesquisa tenha sido feita com base na localização do utilizador, os eventos são agrupados por distância: "Eventos mais próximos" e "Eventos mais distantes". Considera-se eventos próximos do utilizador, aqueles que se encontram a uma distância inferior a 50 km e distantes caso contrário.
- Secção lateral direta – um espaço onde podem ser apresentadas outras informações, como por exemplo eventos em destaque.

Na Figura 5.7 é apresentada a interface de listagem e visualização de eventos, onde podemos ver como os eventos se encontram organizados.

Para que seja possível fazer pesquisas de eventos em ambas as aplicações com base numa determina localização geográfica é necessário que esta informação esteja armazenada na base de dados. Assim, na fase de inserção de eventos utiliza-se a informação do local do evento escolhido pelo utilizador (rua, cidade, país), para através da API do Google de geolocalização obter as coordenadas (latitude e longitude). Essas coordenadas são depois armazenadas na base de dados MongoDB, num campo que será indexado, a fim de ser reconhecido como um campo de geolocalização nas pesquisas. Nos casos em que não é possível indicar corretamente o local da realização do evento, o utilizador pode optar por selecionar o local no mapa e de acordo com a sua indicação, são obtidas as coordenadas que serão armazenadas.

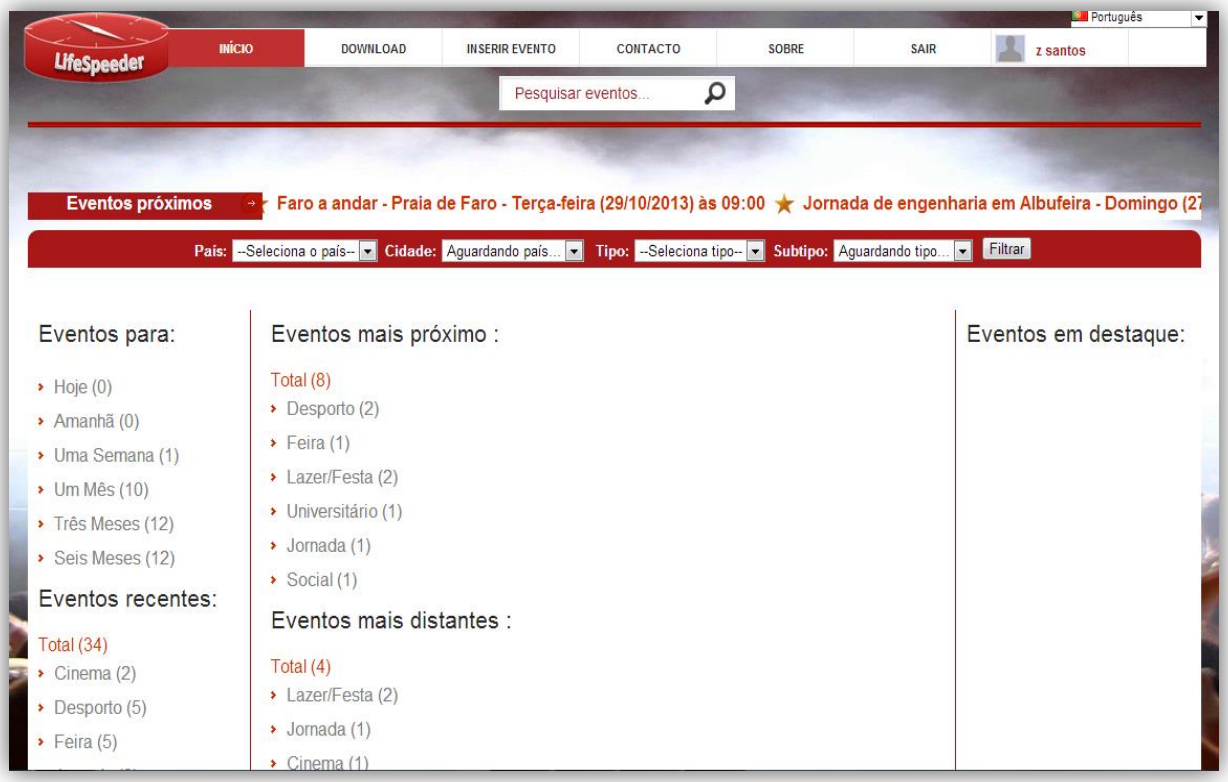


Figura 5.7: Interface de visualização e organização dos eventos

Nas pesquisas avançadas, quando, em ambas as aplicações (Web e móvel), o utilizador pretende fazer uma pesquisa de eventos num determinado país e cidade, essa informação é utilizada para obter a latitude e a longitude do local pretendido através da API de geolocalização do Google, conforme está ilustrado na Figura 5.8. Com base nas coordenadas geográficas devolvidas na resposta da referida API, serão realizadas as pesquisas à base de dados, podendo deste modo conseguir-se resultados mais precisos e que estejam num raio de *x* quilómetros (onde *x* é uma variável com o valor da distância máxima em km). A título de exemplo, suponhamos que um utilizador que esteja num sítio qualquer (Lisboa, Madrid, etc.) pretende passar um final de semana na cidade de Faro. Utilizando uma das aplicações da plataforma *LifeSpeeder* ele faz uma pesquisa por eventos que serão realizados na data em que ele se pretende deslocar à cidade de Faro. Se na consulta à base de dados fosse utilizado somente a informação da cidade (Faro) e do país (Portugal), o resultado devolvido a princípio teria apenas eventos onde havia a ocorrência da palavra "Faro" e "Portugal", ou seja, eventos que estavam dentro da cidade de Faro. No entanto poderiam existir eventos nas cidades vizinhas (Loulé, Tavira, etc.) que podiam interessar ao utilizador, mas que entretanto devido à filtragem utilizada na pesquisa, podiam não ser mostrados. Utilizando a latitude e longitude

do local em vez do nome, para além de obter os eventos que estejam mesmo dentro de Faro, obtêm-se também os eventos das localidades próximas. O resultado da pesquisa seria apresentado ao utilizador de acordo com a distância em relação ao ponto de interesse (neste caso, Faro) ou seja, começavam por ser mostrados os eventos encontrados dentro da cidade de Faro, seguido pelos restantes de acordo com a distância.

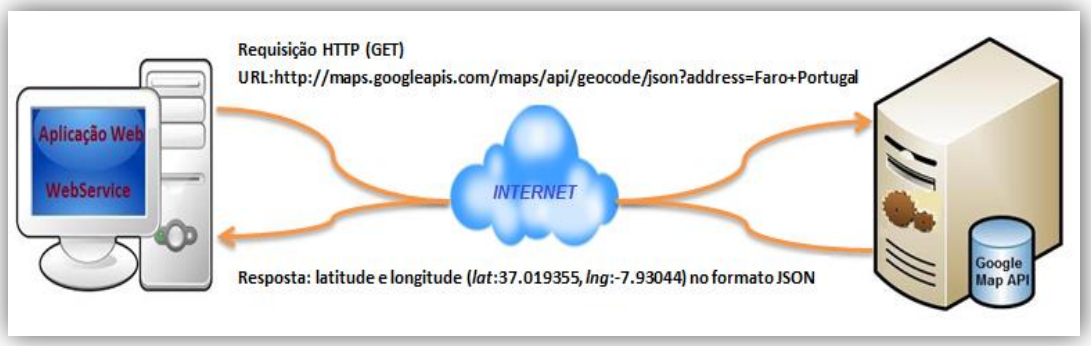


Figura 5.8: Comunicação da aplicação Web/Web Service com a Google Maps API

A Figura 5.9 ilustra como seria o resultado de uma pesquisa depois de transformar o endereço do local (Faro) em coordenadas geográficas (latitude e longitude). A título de exemplo, apenas algumas localidades vizinhas são mostradas na figura, mas na realidade todos os eventos que estejam a uma determinada distância (x km) do ponto de interesse, serão listados por ordem crescente de distância.

A seguir veremos como um evento é armazenado na base de dados do *LifeSpeeder*.

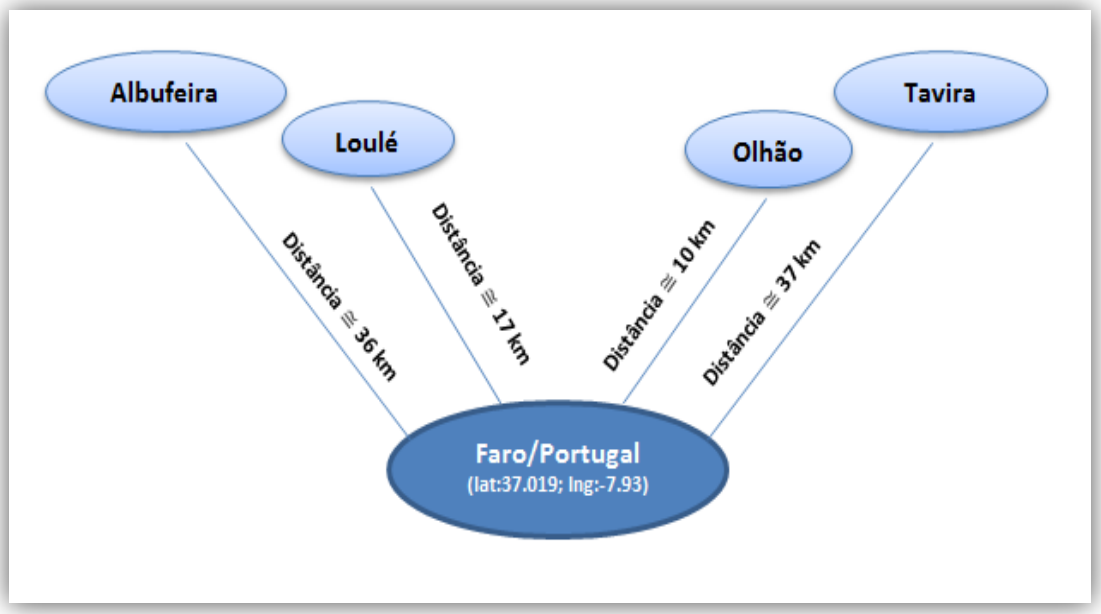


Figura 5.9: Pesquisa relacionada com a latitude e longitude

5.2.1 Armazenamento de um evento na base de dados do *LifeSpeeder*

O formulário da página de inserção de eventos na base de dados do *LifeSpeeder* dá ao utilizador a opção de inserir um determinado evento em vários idiomas. A vantagem disso é que ao inserir um evento em mais que um idioma, aumenta a probabilidade da informação do evento alcançar um público maior. Mesmo que um evento tenha sido inserido em vários idiomas, um utilizador que estiver a visualizar o evento, vê o evento apenas no idioma que ele escolheu. Suponhamos por exemplo que um evento foi inserido em português, inglês e espanhol. A um utilizador que na sua interface da aplicação, escolheu a língua inglesa, o evento é-lhe apresentado apenas em inglês. Os outros dois idiomas do evento não serão mostrados.

Algumas informações genéricas dos eventos, como por exemplo o tipo e subtipo do evento, o país, o distrito e a cidade, são armazenadas de forma que nas pesquisas essas informações sejam apresentadas ao utilizador, de acordo com o idioma que ele escolheu, independente do idioma utilizado no momento de inserir o evento à base de dados. Por outras palavras, supondo que um evento do tipo "Feira" foi inserido em português, ao visualizar o evento numa interface que esteja em inglês, o tipo de evento deve ser apresentado em inglês ("Fair").

Na base de dados que serve de suporte de armazenamento, existe uma coleção que armazena todos os tipos e subtipos de eventos e outra que armazena países, distritos e cidades, em diferentes idiomas. Conforme já tínhamos visto anteriormente, uma coleção em MongoDB é formado por um ou mais documentos e cada documento possui um _id único que o identifica na base de dados. Deste modo cada tipo de evento, assim como cada país é um documento com um _id único, que pode ser utilizado em outras instâncias para fazer referência ao objeto que ele representa. Um tipo de evento pode ter subtipos, do mesmo modo que um país pode ser formado por distritos/regiões, por cidades, etc. Isto significa que em cada documento que representa qualquer uma das situações, podemos ter documentos embebidos para representar os outros níveis de hierarquia.

Para ficar com uma ideia mais clara de como as informações de um determinado evento são armazenadas na base de dados, vejamos o exemplo apresentado na Figura 5.10. Nesta figura estão representados três documentos que na base de dados estão em coleções separadas ("tipos de eventos", "países" e "eventos"). Conforme podemos constatar, cada um dos documentos tem um _id único que é criado automaticamente quando criamos o documento, mas também podemos criar IDs para os documentos embebidos, como é o caso dos subtipos

de ventos ("subtypeID") e dos distritos e cidades ("regionID" e "cityID"). No caso de um documento que representa um país, podemos optar por usar o ID gerado automaticamente, ou usar o códio ISO que representa cada país ("countryCode").

Neste exemplo admitamos que um utilizador cuja interface de aplicação está em espanhol, pretende inserir um evento na base de dados. Suponhamos ainda que como tipo de evento ele escolhe "*Deporte*", como subtipo "*Baloncesto*" e o local do evento escolhido foi: país - "*España*", região - "*Cataluña*" e cidade - "*Barcelona*". No momento de armazenar o evento, em vez de utilizar os nomes que o utilizador selecionou na sua interface, utilizam-se os IDs. Dessa forma garantimos que um utilizador que visualizar o evento por exemplo em português, verá como tipo de evento "Desporto" e como subtipo "Basquetebol". Da mesma forma, na informação do local onde o evento será realizado, ele verá "Barcelona", "Catalunha" e "Espanha".

O benéfico desta estrutura de armazenamento implica de certo modo algum custo a nível de aplicação, uma vez que será necessário uma *querie* mais extensa para obter a informação completa do evento. Neste caso em concreto, para além da coleção principal ("eventos") que armazena os dados do evento, precisávamos incluir mais duas coleções na consulta:

1. "tipos de eventos" – coleção onde íamos buscar o tipo e o subtipo de evento.
2. "países" – coleção onde está armazenado o nome do país, da região e da cidade.

No entanto, quando comparado ao Modelo Relacional, podemos observar que a coleção "tipos de eventos", originaria duas tabelas e a coleção "países", três tabelas. Ou seja, enquanto no MongoDB acrescentávamos na nossa consulta mais duas coleções, numa base de dados do Modelo Relacional normalizada, a consulta envolveria mais cinco tabelas:

1. "tipos de eventos" – tabela onde estaria armazenado o tipo de evento
2. "subtipos de eventos" – tabela que armazenaria o subtipo de evento
3. "países" – tabela com nome do país
4. "regiões" – tabela com nome da região/distrito
5. "cidades" – tabela onde estaria o nome da cidade

Deste modo, numa base de dados relacional, para obter a informação completa do evento, seria necessário vários *joins*, o que de certo modo influenciava na performance do sistema.
No capítulo seguinte e último, são apresentados conclusões e trabalhos futuros em relação ao projeto *LifeSpeeder*.

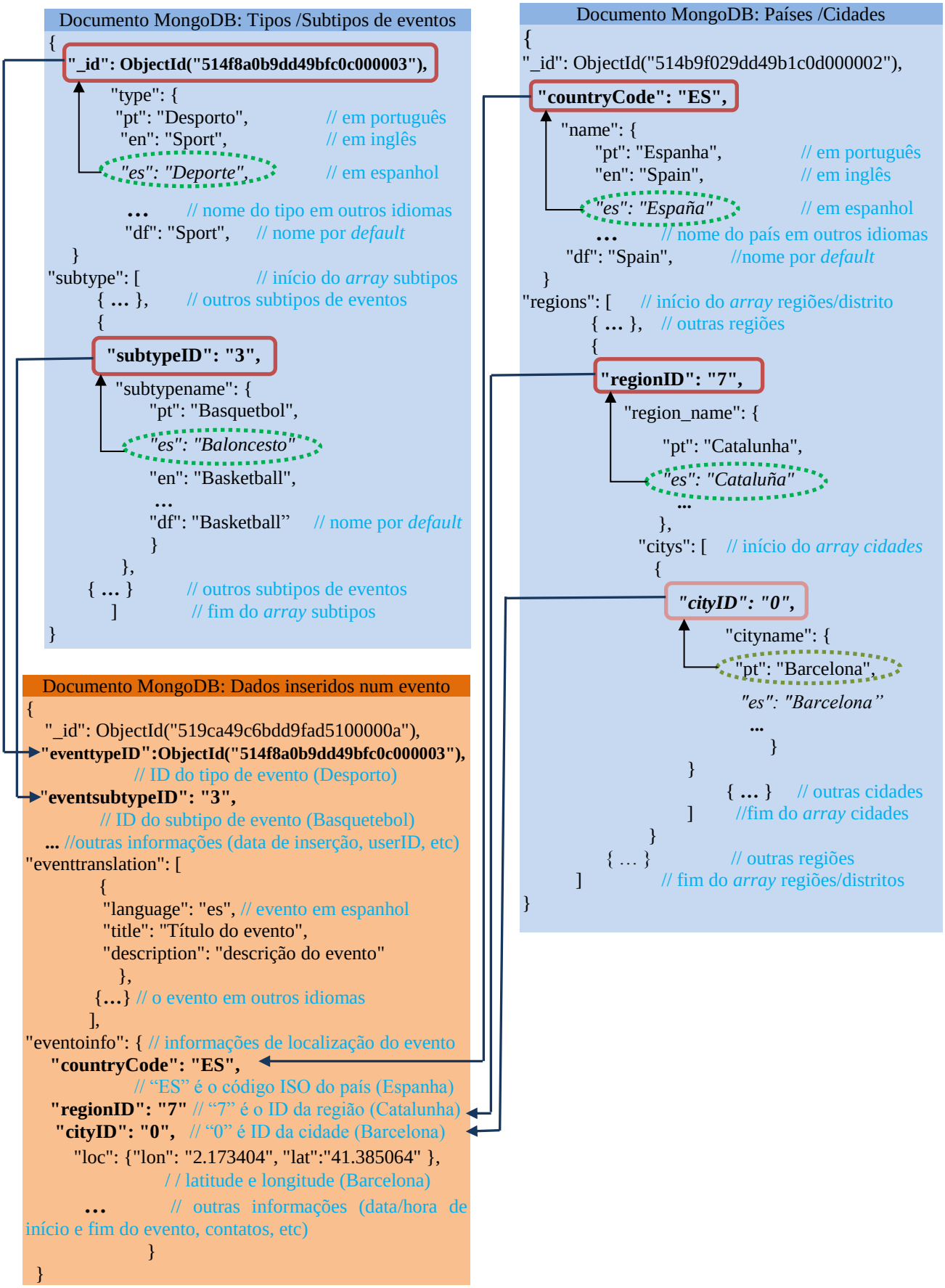


Figura 5.10: Exemplo de como são armazenados dados de um evento

6. CONCLUSÕES E TRABALHOS FUTUROS

Depois de ter apresentado a plataforma *LifeSpeeder* e referido as aplicações que a constituem, bem como algumas das tecnologias utilizadas no seu desenvolvimento, neste capítulo apresentam-se algumas conclusões relativamente ao projeto. Ir-se-á também fazer uma análise das melhorias que podem ser feitas para enriquecer as aplicações.

6.1 CONCLUSÕES

Tirando proveito dos vários recursos e tecnologias disponíveis atualmente, foi possível implementar uma ferramenta capaz de auxiliar e facilitar o utilizador nas suas pesquisas, de forma a permitir-lhe obter a informações (em particular, eventos) que lhe interessa, a partir de um computador ou de um telemóvel (ou outro dispositivo qualquer com SO Android), sem que ele tenha que andar em busca dessas informações por diversas páginas da Web.

Durante as tarefas desenvolvidas no âmbito desta plataforma, um dos focos principais foi a forma de armazenar os dados. O objetivo era encontrar uma forma eficaz e capaz de garantir performance na escrita e leitura dos mesmos, oferecer alguma liberdade na forma de estruturar os dados e facilitar a escalabilidade horizontal, caso o número de utilizadores do sistema cresça significativamente. Depois de analisar as soluções de armazenamento de dados atualmente existentes, conclui-se que uma solução de base de dados não relacional (NoSQL), seria uma boa opção. Uma vez que os eventos precisam conter informações suficientes, incluindo a posição geográfica, a escolha de um modelo de base de dados que suporte nativamente estes tipos de dados, mostrou ser uma mais-valia, permitindo obter boas performances principalmente nas consultas que envolvem pesquisas com informações de latitude e longitude.

Por outro lado, a plataforma desenvolvida pode ser facilmente adaptada a diferentes situações, o que abre a possibilidade de criação de *Rebrandings*. Por outras palavras, esta plataforma pode ser adaptada e personalizada por uma empresa que por exemplo organize eventos e queira manter os seus clientes informados sobre as datas e locais dos mesmos. Outro exemplo pode passar por uma empresa ou agência que trabalha com promoções, para informar os

clientes quais serão as regiões, bem como as datas e os produtos abrangidos por uma determinada promoção. Por outras palavras, a plataforma pode ser reutilizada em diversos cenários, basta fazer uma readaptação e personalização da mesma de acordo com as necessidades de cada cliente/empresa.

6.2 TRABALHOS FUTUROS

Muitos dos objetivos que levaram ao desenvolvimento da primeira versão das aplicações da plataforma *LifeSpeeder*, foram concretizados. No entanto, ainda há muita coisa que pode ser feita para enriquecer a plataforma.

No que toca à aplicação móvel por exemplo, atualmente é suportada apenas por dispositivos que têm o sistema operativo Android. Futuramente pretende-se alargar o horizonte de plataformas de suporte, isto é, desenvolver a aplicação móvel para outras plataformas, nomeadamente iOS, Windows Phone, entre outras. Atualmente a função principal da aplicação Móvel é pesquisar eventos, mas o que se pretende é que no futuro para além de pesquisar eventos, o utilizador consiga inserir, editar e eliminar eventos a partir dela. Alargar a quantidade de idiomas de suporte é um outro objetivo.

Quanto à aplicação Web, à semelhança da aplicação móvel, neste momento o número idiomas das interfaces são apenas três. Assim, pretende-se acrescentar mais idiomas no futuro, de forma a alargar o horizonte geográfico do seu uso. Mas para além de aumentar o número de idiomas, futuramente pretende-se:

- Transformar a plataforma numa "rede social de eventos", onde os utilizadores podem partilhar, recomendar, comentar e avaliar eventos de outros utilizadores.
- Permitir ao utilizador criar uma agenda pessoal de eventos, que dependendo das suas configurações, ele receberá um alerta (e.g., SMS, email) avisando-o da data e hora de início de um determinado evento, evitando deste modo que ele perca os seus eventos favoritos.
- Implementar um sistema de *ranking* e pesquisas inteligentes que reconhecem as preferências de cada utilizador e apresenta-o os eventos de a cordo com essas preferências

Essas foram algumas das melhorias que se pretendem implementar nas aplicações do *LifeSpeeder*. Muitas outras ideias ainda podem surgir para aumentar não somente as funcionalidades dessa ferramenta, mais também torná-la numa ferramenta indispensável no dia-a-dia do utilizador.

BIBLIOGRAFIA

[1] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes & Robert E. Grube. "Bigtable: A Distributed Storage System for Structured Data". ACM Transactions on Computer Systems, Vol. 26, No. 2, Article 4, June 2008

[2] Kevin Howard Goldberg. XML: Visual QuickStart Guide, Second Edition. Peachpit Press, 2009

[3] Eric Newcomer. Understanding Web Services - XML, WSDL, SOAP and UDDI. Addison Wesley, 2002

[4] Tapan K. Sarkar, Robert J. Mailloux & Arthur A. Oliner, Magdalena Salazar-Palma, Dipak L. Sengupta. HISTORY OF WIRELESS, Published by John Wiley & Sons, Inc., Hoboken. New Jersey, 2006

[5] Yan Zhang, Jijun Luo & Honglin Hu. WIRELESS MESH NETWORKING - Architectures, Protocols and Standards. Taylor & Francis Group, LLC, 2007

[6] James F. Kurose & Keith W. Ross, Computer Networking – A Top-Down Approach, 6th Ed. Pearson Education, Inc, 2013

[7] Charles K. Summers. ADSL: Standards, Implementation, and Architecture. CRC Press LLC, 1999

[8] Pierre Lecoy. Fiber Optic Communications. ISTE Ltd and John Wiley & Sons, Inc., 2008

[9] Loutfi Nuaymi, WiMAX – Technology for Broadband Wireless Access. John Wiley & Sons, Ltd, 2007

[10] Hung-Yu Wei, Jarogniew Rykowski, Sudhir Dixit. WIFI, WIMAX AND LTE MULTI-HOP MESH NETWORKS. John Wiley & Sons, Inc, 2013

[11] James Governor, Dion Hinchcliffe & Duane Nickull. Web 2.0 Architectures. O'Reilly Media, Inc. 2009

[12] Roger W.McHaney. Web 2.0 and Social Media for Business. Roger W. McHaney & Ventus Publishing Aps, 2012

[13] Richard C. Jaeger & Travis N. Blalock. MICROELECTRONIC CIRCUIT DESIGN. The McGraw-Hill Companies, Inc., 2011

[14] Friedhelm Hillebrand, Finn Trosby, Kevin Holley, Ian Harris. SHORT MESSAGE SERVICE (SMS): The Creation Of Personal Global Text Messaging. John Wiley & Sons Ltd, 2010

[15] Juha Korhonen. Introduction to 3G Mobile Communications, 3rd Ed. ARTECH HOUSE, INC. 2003

[16] Christopher Cox, An Introdution to LTE – LTE, LTE-Advanced, SAE and 4G Mobile Comunications. John Wiley & Sons Ltd, 2012

[17] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM, Volume 13, nº 6, Junho de 1970.

[18] Toby Teorey, Sam Lightstone, Tom Nadeau. Database Modeling & Design: Logical Design, 4th Ed. Elsevier Inc. 2006

[19] Shashank Tiwari. Professional NoSQL. John Wiley & Sons, Inc. 2011

[20] Pramod J. Sadalage & Martin Fowler. NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, Inc. 2013

[21] Paulo Heitlinger, O Guia prático da XML, 1ª edição, Centro Atlântico, Outubro de 2001

[22] Fonseca, R., & Simões, A. "Alternativas ao XML: YAML e JSON". 5ª Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas, XATA 2007

[23] Oren Ben-Kik, Clark Evans, Ingy döt Net. "YAML Ain't Markup Language (YAML™)",Version 1.2 3rd Edition, Patched at 2009

[24] Douglas Crockford. "JSON: The Fat-Free Alternative to XML" Presented at XML 2006 in Boston, December 6, http://www.json.org/fatfree.html, último acesso 27/08/2013

[25] Arno Puder, Kay Römer & Frank Pilhofer. Distributed Systems Architecture. Elsevier Inc. 2006

[26] Doug Tidwell, James Snell, Pavel Kulchenko. Programming Web Services with SOAP, 1st Ed. O'Reilly, 2001

[27] Leonard Richardson, Sam Ruby. RESTful Web Services. O'Reilly Media, Inc. 2007

[28] Daniel Jacobson, Greg Brail, & Dan Woods. APIs: A Strategy Guide. O'Reilly Media, Inc., 2012

[29] "The Google Maps Geolocation API" https://developers.google.com/maps/documentation/business/geolocation/, último acesso 12/09/2013

[30] Rick Cattell. Scalable SQL and NoSQL Data Stores. SIGMOD Record, December 2010

[31] Kai OREND. Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer. 2010

[32] Gavin Powell. Beginning Database Design. Wiley Publishing Inc. 2006

[33] Peter Rob & Carlos Coronel. Database Systems Design - Implementation And Management. 2007

[34] Seema Kedar. Database Management System, 1st Ed. Technical Publications Pune, 2009

[35] Toby J. Teorey, Sam S. Lightstone, and Thomas P. Nadeau. Database Modeling and Design: Logical Design, Fourth Edition. Elsevier Inc. 2006

[36] Carlos Coronel, Steven Morris, and Peter Rob. Database Systems: Design, Implementation, and Management,3rd Ed. Course Technology, Cengage Learning. 2013

[37] Clare Churcher. Beginning Database Design:From Novice to Professional. Apress, 2007

[38] Mark D. Hill, What is Scalability?, Computer Sciences Department, 1210 West Dayton St., University of Wisconsin, Madison, Wisconsin

[39] André B. Bondi, "Characteristics of Scalability and Their Impact on Performance" ACM New York, USA 2000

[40] Eric Redmond, Jim R. Wilson. Seven Databases in Seven Weeks - A Guide to Modern Databases and the NoSQL Movement. Pragmatic Programmers, LLC. 2012

[41] Eben Hewitt. Cassandra: The Definitive Guide. O'Reilly Media, Inc., 2011

[42] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall & Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. SOSP'07, Stevenson, Washington, USA, October 2007

[43] J. Chris Anderson, Jan Lehnardt, & Noah Slater. CouchDB: The Definitive Guide. O'Reilly Media, Inc., 2010

[44] Kristina Chodorow and Michael Dirolf. MongoDB: the definitive guide. OReilly, Media, Inc., 2010

[45] "CAP theorem". http://en.wikipedia.org/wiki/CAP_theorem, último acesso 20/08/2013

[46] 10gen. MongoDB Documentation. March 02, 2013 (http://docs.mongodb.org/manual/)

[47] Kristina Chodorow. Scaling MongoDB. O'Reilly Media, Inc., 2011

[48] Kyle Banker. MongoDB in Action. Manning Publications Co, 2012

[49] Anthony T. Holdener III. HTML5 Geolocation. O'Reilly Media, Inc. 2011

[50] StatCounter GlobalStats. http://gs.statcounter.com/, último acesso 13/09/2013

ANEXOS

ANEXO A : A BASE DE DADOS DO *LIFESPEEDER* NO MODELO RELACIONAL

O diagrama entidade-relacionamento (DER) abaixo, representa o *layout* das tabelas com os respetivos relacionamentos, daquilo que seria a base de dados do *LifeSpeeder*, caso se tivesse optado pelo Modelo Relacional.

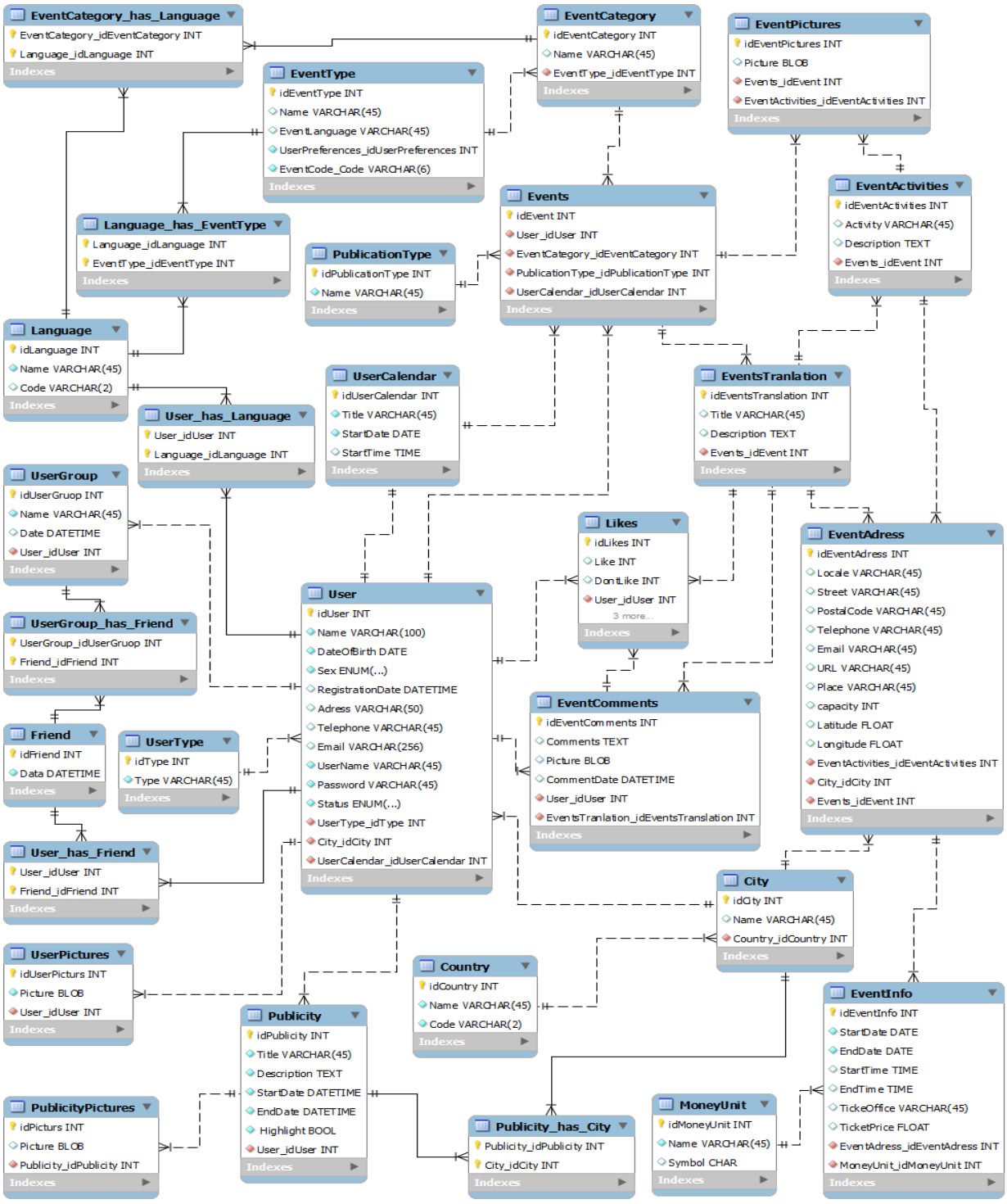


Figura A-1: Estrutura da base de dados LifeSpeeder no Modelo Relacional

ANEXO B : UM EVENTO COM VÁRIAS ATIVIDADES

Neste anexo é apresentado um documento em MongoDB referente a um evento. Neste caso, o evento seria um evento composto, ou seja, um evento com mais de uma atividade, realizadas em datas e locais diferentes. Podemos constatar alguma flexibilidade que o MongoDB oferece na estruturação dos dados neste tipo de base de dados. A estrutura do documento não é uma estrutura fixa, sendo que a qualquer momento se podem remover algum dos subdocumentos embebidos, ou acrescentar um novo subdocumento.
Segue abaixo uma pequena descrição, das informações que alguns subdocumentos embebidos podem ter:

- Subdocumento "likes" (linha 09) – guarda a informação de todos os utilizados que gostaram ou não de um determinado evento.
- Subdocumento "*comments*" (linha 16) – todos os comentários de um determinado evento ficam, armazenados neste subdocumento.
- Subdocumento "*eventtraslation*" (linha 24) – contém as traduções de um determinado evento.
- Subdocumento "*eventinf*o" (linha 35) – contém informação de data, hora, local e informações de contato do evento.
- Subdocumento "*eventactivities*" (linha 57) – subdocumento que guarda as informações de todas as atividades de um evento. Este subdocumento tem outros subdocumentos embebidos, como por exemplo "*activityinfo*" (linha 70), que contém informações de data, local, capacidade do espaço, etc.; "*ticke*t" (linha 100), que contém informação de locais onde pode-se comprar bilhetes para o evento.

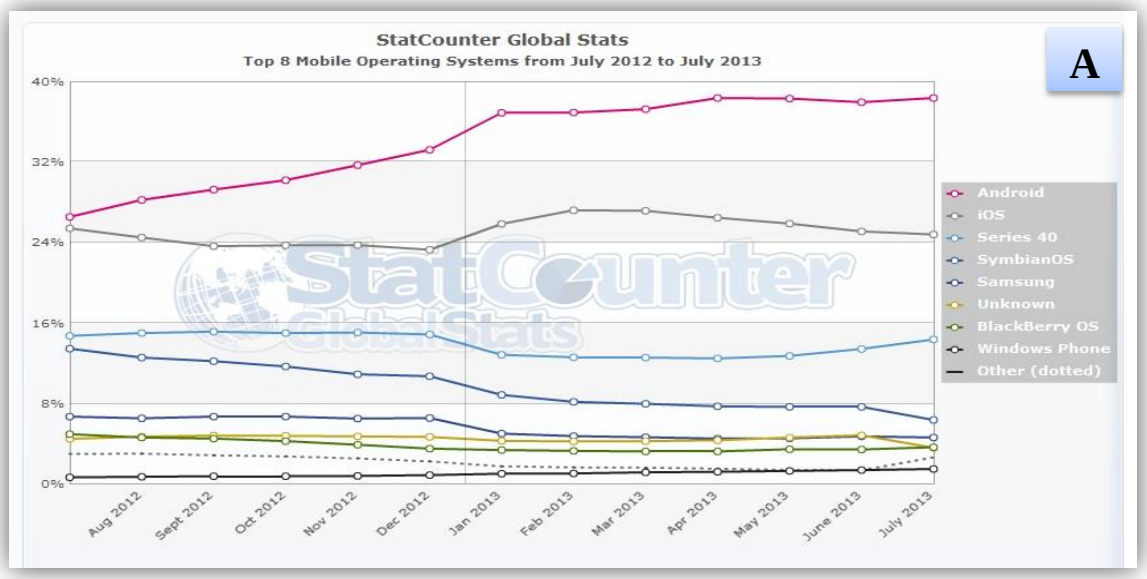
```
{
  "event": {
    "eventtype_id": ObjectId("51027913f6e24c09e4a4780d"),
    "insertdate": ISODate("2013-01-31T21:03:44.721Z"),
    "user_id": ObjectId("50ae9b9418978c889cca8d71"),
    "publicationtype": {
      "public": true,
      "private": false},
    "likes": {
      "user_id_like": [
       ObjectId("50b3fb2f3e5a89cc0a000003"),ObjectId("50b3eb843e5a89cc0a000002"),...
      "user_id_nolike": [ObjectId("50ae9b9418978c889cca8d71"),...],
      "yeslike": NumberInt(5),
      "nolike": NumberInt(3)
    },
    "comments": [{
        "user_id": ObjectId("50b3fb2f3e5a89cc0a000003"),
        "comment": "Gosto muito da semana cademica...",
        "language_id": ObjectId("50b371083e5a89f401000007"),
        "commentdate": ISODate("2013-02-11T21:03:44.722Z")
      },

    ],
    "eventtranslation": [
      {
         "language_id": ObjectId("50b371083e5a89f401000007"),
        "title": "Semana académica",
        "description": "Realiza-se em Faro  mais uma semana académica..." },
      {
        "language_id": ObjectId("50b370e53e5a89f401000005"),
        "title": "Academic week",
        "description": "Event held every year..."
      }
    ],
    "eventinfo": [
      {
        "country_id": ObjectId("50b197999930f937435ae4a6"),
        "city": "Faro",
        "location": {
          "long": -7.923337,
          "lat": 37.028681
        },
        "startdate": ISODate("2013-04-15T00:00:01.994Z"),
        "enddate": ISODate("2013-04-22T00:00:01.987Z"),
        "starttime": "10:00",
        "endtime": "00:00",

        },
        "contact": {
          "phone": ["932456568", "912098754"],
          "email": ["nome1@gmail.com","nome2@gmail.com"],

          "site": "www.assciacao.ualg.pt"
        }
      }
    ],
  "eventactivities": [ {
      "activitytype": ObjectId("51027913f6e24c09e4a4780d"),
      "activitytranslation": [ {
            "language_id": ObjectId("50b371083e5a89f401000007"),
             "title": "Concerto",
             "description": "concerto com o grupo... ..."
          },
        {
          "language_id": ObjectId("50b370e53e5a89f401000005"),
          "title": "Concert",
          "description": "Concert with the group ..."
        }
```

```
        ],
        "activityinfo": [{
            "country_id": ObjectId("50b197999930f937435ae4a6"),
            "city": "Faro",
            "local": "Largo da Sê",
            "capacity": NumberInt(5000),
            "location": {
              "long": -7.935346,
              "lat": 37.013156 },
            "startdate": ISODate("2013-04-15T00:00:01.994Z"),
            "enddate": "",
            "starttime": "21:00",
            "endtime": "03:00",
             },
            "likes": {
              "user_id_like": [ObjectId("50b192733e5a898405000001"), ObjectId("50b18f6a
              "user_id_nolike": [ObjectId("50b3fb2f3e5a89cc0a000003"),...]
              "yeslike": NumberInt(5),
              "nolike": NumberInt(2)
            },
            "contact": {
                  "phone": ["923412546", "964553452"],
                  "email": ["email1@ualg.pt","email2@ualg.p"]
                  "site": "www.ualg.pt" }
          ],
        "comments": [
               { "user_id": ObjectId("50b18f6a3e5a891404000000"),
                 "comment": "quero ir a este concerto...",
                 "language_id": ObjectId("50b371083e5a89f401000007"),
                 "commentdate": ISODate("2013-02-14T19:40:44.215Z")},
              ]
            "ticket": [
              {
                "ticketoffice": "AAUALG",
                "city": "Faro",
                "street": "Estrada da Penha",
                "location": {
                  "long": -7.923337,
                  "lat": 37.028681
                },
                "ticketprice": NumberInt(5),
                "contact": {
                  "phone": [
                            "213456654",
                            "92345675"
                            ],
                  "email": ["aaualg@ualg.pt"],
                  "fax": "21345632345",
                  "site": "www.aaualg.pt"
                }
              },
            ]
          }
        ],
      }
    ]
}
```

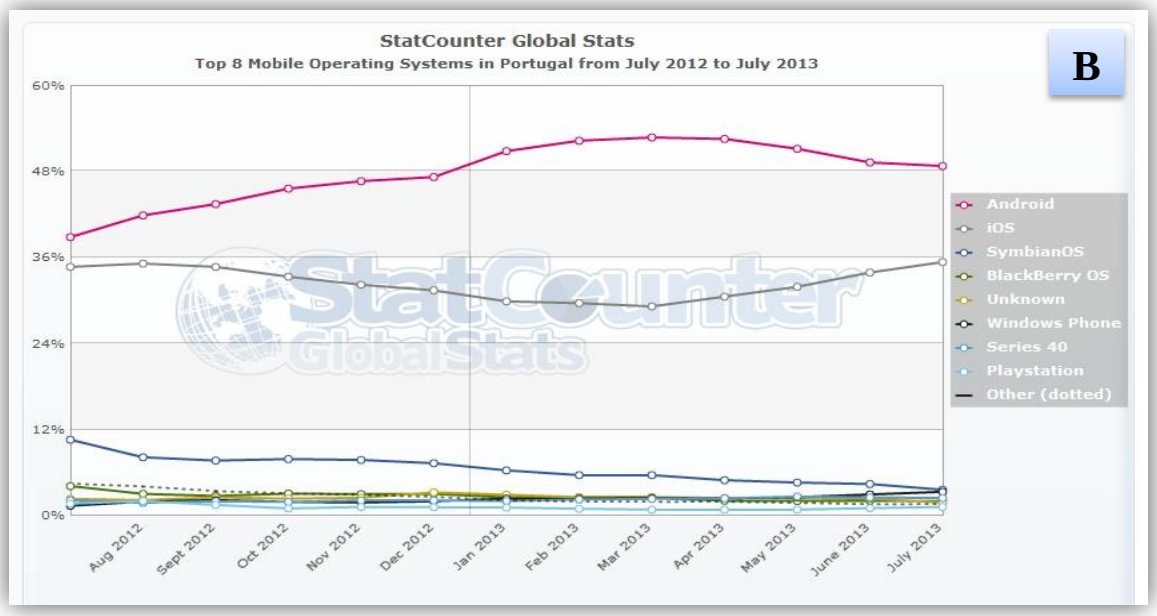
Figura B-2: Exemplo de um evento com mais de uma atividade no MongoDB

ANEXO C: SISTEMAS OPERATIVOS MÓVEIS MAIS UTILIZADOS

Os dois gráficos abaixo mostram os oito sistemas operativos para dispostos móveis mais utilizados atualmente. O primeiro gráfico (A) é referente a nível mundial e o segundo (B) referente a Portugal. Conforme podemos constatar, em ambos os casos é o sistema operativo móvel da Google (Android) que domina o mercado, seguido pelo sistema operativo da Apple (iOS)[50].



(a)



(b)

Figura C-3: Percentagem de utilizadores dos vários SO móveis (a) a nível mundial e (b) em Portugal.