Instituto Superior de Engenharia do Porto

SISTEMA PARA A MONITORIZAÇÃO ATIVA COM RECURSO AO CLOUD COMPUTING

"ChildSafe"

Ricardo Daniel Marques Pessoa

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Doutor Filipe Pacheco Paulo

Co-orientador: Mestre João Guimarães

Júri:

Presidente:

Doutor Luis Miguel Moreira Lino Ferreira, DEI/ISEP

Vogais:

Doutor Luis Miguel Pinho Nogueira, DEI/ISEP

Doutor Filipe De Faria Pacheco Paulo, DEI/ISEP

Porto, Outubro de 2014

Resumo

O desenvolvimento de aplicações para dispositivos móveis já não é uma área recente, contudo continua a crescer a um ritmo veloz. É notório o avanço tecnológico dos últimos anos e a crescente popularidade destes dispositivos. Este avanço deve-se não só à grande evolução no que diz respeito às características destes dispositivos, mas também à possibilidade de criar aplicações inovadoras, práticas e passíveis de solucionar os problemas dos utilizadores em geral.

Nesse sentido, as necessidades do quotidiano obrigam à implementação de soluções que satisfaçam os utilizadores, e nos dias de hoje, essa satisfação muitas vezes passa pelos dispositivos móveis, que já tem um papel fundamental na vida das pessoas.

Atendendo ao aumento do número de raptos de crianças e à insegurança que se verifica nos dias de hoje, as quais dificultam a tarefa de todos os pais/cuidadores que procuraram manter as suas crianças a salvo, é relevante criar uma nova ferramenta capaz de os auxiliar nesta árdua tarefa.

A partir desta realidade, e com vista a cumprir os aspetos acima mencionados, surge assim esta dissertação de mestrado. Esta aborda o estudo e implementação efetuados no sentido de desenvolver um sistema de monitorização de crianças. Assim, o objetivo deste projeto passa por desenvolver uma aplicação nativa para Android e um *back-end*, utilizando um servidor de base de dados NoSQL para o armazenamento da informação, aplicando os conceitos estudados e as tecnologias existentes. A solução tem como principais premissas: ser o mais *user-friendly* possível, a otimização, a escalabilidade para outras situações (outros tipos de monitorizações) e a aplicação das mais recentes tecnologias.

Assim sendo, um dos estudos mais aprofundados nesta dissertação de mestrado está relacionado com as bases de dados NoSQL, dada a sua importância no projeto.

Palavras-chave: Dispositivos móveis, base de dados, NoSQL, *Big Data*, *Internet of Things*

Abstract

The development of applications for mobile devices is no longer a new area, but continues to rapidly grow. It is remarkable the technological advances in last years and the growing popularity of these devices. This progress is due not only to the large developments about the characteristics of these devices, but also the ability to create innovative applications, practical solutions and solve problems of users in general.

Accordingly, the needs of everyday life require the implementation of solutions which satisfy users, and nowadays, that satisfaction often goes by mobile devices, which already has a key role in people's lives.

Attending to the increasing number of children abductions and insecurity nowadays makes it difficult for parents and caregivers to keep their children safe, therefore it is relevant to create a new tool, able to help parents and caregivers in this arduous task.

From this fact, and in order to accomplish the aspects above, this dissertation is presented. This thesis describes the study and the development of the monitoring system for children. The objective of this project is to develop a native application for Android and a back-end using NoSQL database server for storing information, summarizing the studied concepts and implemented of technologies. The solution has the following main assumptions: being as user-friendly as possible, optimization, scalability to other situations (other types of monitoring in different conditions) and the appliance of state-of-the-art software technologies.

Therefore, one of the detailed study in this dissertation is associated with NoSQL databases due to its importance in the project.

Keywords: Mobile Devices, Data Base, NoSQL, Big Data, Internet of Things

Agradecimentos

A primeira pessoa a quem quero prestar o meu mais sincero agradecimento é ao Eng.º Filipe Pacheco Paulo, orientador do ISEP, que me acompanhou ao longo da dissertação de mestrado. É um agradecimento especial pela sua prontidão nas respostas às minhas dúvidas, pelos seus conselhos e pela disponibilidade que sempre demostrou.

O meu agradecimento profundo ao Eng.º João Guimarães, orientador da instituição CeNTI, pelo apoio constante no decorrer do projeto. A sua atenção e preocupação foram inquestionáveis, uma vez que estava sempre disponível para discutir não só os progressos do projeto, mas também os obstáculos que surgiam a cada dia. Sendo a pessoa com quem trabalhei diretamente, caso algo não corresse como o planeado, juntos procurávamos encontrar uma solução corretiva. Deste modo, sempre se demonstrou disponível para me aconselhar, elucidar e ajudar, mesmo quando ele próprio se encontrava empenhado nas suas tarefas. Quero também agradecer ao Eng.º Vasco Machado, pelo apoio no desenvolvimento do dispositivo de monitorização e também ao *Designer* Rafael Vieira pela disponibilidade no desenvolvimento dos ícones a incluir na plataforma.

Ao CeNTI em geral, pela oportunidade de realização da dissertação de mestrado, e a todos os seus colaboradores o meu enorme obrigado. Além da receção calorosa, foram responsáveis pelo extraordinário ambiente de trabalho ao longo de todo o projeto, ambiente este que, sendo agradável e proveitoso, permitiu a minha integração na equipa sem qualquer problema. É indubitavelmente uma equipa que, diariamente, se esforça por alcançar os seus objetivos.

Por fim, e não menos importante, agradeço à minha família pela dedicação e esforço de todos os dias, para me proporcionarem uma vida melhor e por todo o apoio que me deram ao longo da vida. Em especial, à minha namorada pela sua paciência e apoio incondicional, ao longo de todo o projeto. Agradeço ainda a todos os meus amigos que me apoiaram durante a realização do projeto.

A todos eles, o mais sincero obrigado.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código

Acrónimos

ACID	*Atomicity, Consistency, Isolation, Durability*
ADT	*Android Development Tools*
AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
BASE	*Basically Available, Soft State, Eventual Consistency*
CAP	*Consistency, Availability and Partition-Tolerance*
CouchDB	*Cluster Of Unreliable Commodity Hardware Data Base*
CPU	*Central Processing Unit*
CRUD	*Create, Read, Update and Delete*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
DDMS	*Dalvik Debug Monitor Server*
GLONASS	*Global Navigation Satellite System*
GPRS	*General Packet Radio Service*
GPS	*Global Position System*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IT	*Information Technology*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
MAT	*Eclipse Memory Analyzer*
MB	*Megabyte*
MVCC	*Multiversion Concurrency Control*, também conhecido por MCC

NoSQL	*Not Only SQL*
OO	Orientado Objeto, do inglês *Object-Oriented*
OODBMS	*Object-Oriented Database Management System*
OTP	*Open Telecom Platform*
RDBMS	*Relational Database Management System*
RAM	*Random-Access Memory*
REST	*Representational State Transfer*
RFID	*Radio-Frequency Identification*
SCM	*Source Code Management*
SHA	*Secure Hash Algorithm*
SO	Sistema Operativo
SQL	*Structured Query Language*
SSD	*System Sequence Diagram*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
UUID	*Universally Unique Identifier*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*
WHATWG	*Web Hypertext Application Technology Working Group*
WSN	*Wireless Sensor Network*

1 Introdução

O crescente avanço tecnológico reflete-se na vida diária das pessoas e o progresso, no que diz respeito à aquisição e utilização dos dispositivos móveis em geral, não é exceção.

Os dispositivos móveis estão a mudar a forma como as pessoas interagem entre si e com o mundo ao seu redor. Pode inclusivamente dizer-se que se criou alguma dependência destes aparelhos. Além da melhoria contínua nas características e funcionalidades que apresentam, a usabilidade das inúmeras aplicações atualmente disponíveis satisfazem os utilizadores em geral, o que propicia a aquisição destes dispositivos.

A Internet of Things (IoT) vem ainda adicionar novos tipos de dispositivos, com capacidade de comunicação e partilha de informação, capacidade sensorial e captação de novos e interessantes tipos de informação. Com a necessidade de armazenar e processar grandes conteúdos de informação surge o movimento *Big Data*.

Sendo notória a importância dos movimentos acima referidos, e procurando obter com esta dissertação de mestrado um produto inovador e funcional, nasceu assim o projeto "ChildSafe", um sistema de monitorização de crianças.

1.1 Enquadramento

O projeto desenvolvido nesta dissertação de mestrado está inserido num contexto empresarial, uma vez que foi proposto ao CeNTI [1] - Centro de Nanotecnologia e Materiais Técnicos, Funcionais e Inteligentes - situado em Vila Nova de Famalicão, Portugal.

O CeNTI é um centro de investigação privado, sem fins lucrativos, fundado em 2006 pelo CITEVE (Centro Tecnológico das Indústrias Têxtil e do Vestuário em Portugal), pelas Universidades do Minho, Porto e Aveiro e pelo CTIC (Centro Tecnológico das Indústrias do Couro). Tem como

[1] http://www.centi.pt/

principal missão o desenvolvimento de novos materiais, de modo a contribuir para a criação de produtos inovadores e de elevado valor acrescentado. Os serviços prestados são abrangentes e integrados, disponibilizando apoio aos clientes, desde a conceção de um material até à respetiva valorização económica.

Assim sendo, uma vez desenvolvida no CeNTI, esta dissertação de mestrado teria obrigatoriamente de trazer algo novo, inovador, com valor acrescentado e com possibilidade de eventualmente chegar ao mercado.

Como já referido anteriormente, o uso de aplicações para dispositivos móveis continua a apresentar uma grande adesão e, nesse sentido, em parceria com o departamento de eletrónica do CeNTI, surgiu um projeto interno com o objetivo de criar um sistema genérico de monitorização e controlo de sensores remotamente.

O projeto "ChildSafe", que será objeto de estudo e implementação nesta dissertação, está assim associado ao desenvolvimento de uma plataforma capaz de receber diversos tipos de informação provenientes de diferentes dispositivos. Estes vão acoplar diversos sensores, de forma a monitorizar o utilizador. A plataforma será dinâmica, uma vez que, ao adicionar novos tipos de sensores, não será necessário reestruturar a base de dados do servidor, apenas será necessário adicionar as novas capacidades nos *end-devices*, ou seja, no *back-end* e no dispositivo móvel.

Como prova de conceito será disponibilizado um conjunto de dispositivos capazes de monitorizar as tarefas do quotidiano de um utilizador e enviar essa informação para um servidor de base de dados NoSQL. Este servidor será responsável por armazenar, filtrar e disponibilizar esta informação de acordo com as necessidades dos utilizadores. Para apresentar a informação recolhida pelos sensores serão desenvolvidos um *back-end* e uma aplicação móvel, onde esta será apresentada, de forma pertinente e simples, para os pais/cuidadores. O conceito tem enfoque na monitorização de crianças, com vista a aumentar a sua segurança e bem-estar.

1.2 Objetivos

A dissertação de mestrado tem como objetivo estudar os vários tipos de base de dados NoSQL, e encontrar o sistema mais adequado para o armazenamento da informação, tendo em conta os requisitos propostos.

Esta tese pressupõe a criação de uma plataforma de raiz. Assim, serão desenvolvidos: um servidor, um *back-end* e uma aplicação *mobile* [2] em Android. Nesse sentido, durante a dissertação é acompanhado e abordado o desenvolvimento de todos os componentes do projeto, apresentando todas as tecnologias utilizadas na conceção do mesmo, as ferramentas utilizadas e as soluções encontradas para a sua conclusão.

[2] A partir deste ponto, por aplicação *mobile* entenda-se aplicação móvel

O desenvolvimento do *back-end* prevê a existência de dois perfis de utilizador: o primeiro, do tipo administrador, tem como objetivo inserir dispositivos no sistema e o segundo, do tipo utilizador comum, adquire e utiliza o dispositivo para a monitorização. Este recurso tem enfoque na gestão e configuração dos dispositivos e sensores, não ignorando a apresentação e notificação da informação.

O desenvolvimento da aplicação nativa Android foca-se, essencialmente, na apresentação dos resultados da monitorização dos dispositivos e, consequentemente, na notificação ao utilizador do estado dos sensores. Ainda que estas sejam a principal missão da aplicação, esta também permite a gestão e configuração dos dispositivos.

Não obstante aos objetivos acima descritos, um dos requisitos do desenvolvimento da plataforma passa por criar uma estrutura transversal, que permita adaptar a solução criada (monitorização de crianças) a diferentes contextos, como por exemplo, monitorização de idosos, pessoas com deficiência, imóveis, objetos, etc. A plataforma deve ainda permitir a adição de novos sensores de forma simples, célere e sem comprometer a performance e funcionalidades já existentes.

1.3 Motivação

A motivação para a escolha do tema deste projeto reside essencialmente no objetivo de estudar e adquirir conhecimentos relacionados com bases de dados não relacionais. O facto de ter tido contacto com esta tecnologia, de forma indireta e em contexto profissional, despertou em mim bastante interesse. Esta tecnologia está a ser cada vez mais utilizada, devido à sua versatilidade em armazenar diversos tipos de dados, surgindo assim como uma solução para problemas tipificados.

Assim, a dissertação surgiu como um enorme desafio, dada a necessidade de aprender uma nova tecnologia que veio revolucionar o paradigma do desenvolvimento de bases de dados. Mostrou-se não só como um desafio intelectual, que visa a aquisição de novos conhecimentos e competências, mas assumiu-se, acima de tudo, como um desafio atual.

Também o movimento *Big Data* e *Internet of Things* motivaram a escolha deste tema. A procura de informação é cada vez mais desejada, tanto pelas empresas como pelas pessoas.

Paralelamente, com o crescente volume de informação e a chegada da *Web* 2.0, a consulta e o armazenamento de informação por parte das bases de dados aumentou de forma significativa. As redes sociais e as grandes empresas, como é o caso do Facebook e da Google, lidam cada vez mais, com elevados conteúdos de informação, e como tal necessitaram de encontrar uma solução que não comprometesse a sua performance. De acordo com os autores [Zikopoulos and Eaton, 2012], 80% da informação do mundo não é estruturada e esta cresce 15 vezes mais que a informação estruturada. Uma das soluções para colmatar este problema passou por escalar os sistemas de bases de dados horizontalmente [Hecht and Jablonski,2011].

Por outro lado, o interesse pelas aplicações *mobile*, assim como a curiosidade associada ao seu desenvolvimento e a perspetiva de obter um produto final funcional e útil foram decisivos na escolha deste projeto.

1.4 Estrutura da dissertação

A presente dissertação está dividida em 7 capítulos e 5 anexos, cujo teor se descreve de seguida:

Neste primeiro capítulo, para além da contextualização do tema da dissertação, é feita uma descrição dos principais objetivos e da motivação inerente à realização deste projeto.

No segundo capítulo é apresentado o estado da arte de mercado e tecnológico, onde são descritas as tecnologias alvo de estudo e implementadas durante o desenvolvimento do projeto. Nesse capítulo são também estudadas algumas aplicações e as suas interfaces. Por fim, são retiradas algumas conclusões do estudo efetuado.

O terceiro capítulo é destinado ao estudo de sistemas de bases de dados NoSQL, sendo feita uma pequena introdução relativa à evolução das bases de dados. Posteriormente são apresentados os tipos de bases de dados, as suas características e fundamentada a decisão de escolha do sistema de base de dados aplicado neste projeto. Durante este capítulo são também apresentados alguns temas importantes, como: teorema CAP, características ACID e BASE e os conceitos *Big Data* e *Internet of Things*.

No quarto capítulo é descrito o levantamento de requisitos, incluindo a descrição detalhada de todos os requisitos funcionais e não funcionais, aos quais a plataforma deverá responder. A partir destes são apresentados os diagramas de casos de uso e de sequência de sistema.

O quinto capítulo dá lugar à descrição detalhada da solução conseguida, desde o servidor, *back-end* e aplicação móvel. Nesse capítulo é também apresentada uma visão genérica do sistema desenvolvido, o modelo de dados utilizado, alguns excertos de código relevantes e algumas ilustrações da interface.

Não menos importante é a análise do desempenho da plataforma. Reservou-se então o capítulo sexto para descrever alguns dos testes efetuados.

Por último, no sétimo capítulo são apresentadas as conclusões, análise crítica e trabalho futuro na plataforma.

Relativamente aos anexos, estes encontram-se divididos da seguinte forma:

- Anexo 1 - *Mockup* do *back-end*;
- Anexo 2 - *Mockup* da aplicação móvel - Android;
- Anexo 3 - Modelo de dados - NoSQL orientado ao documento;
- Anexo 4 - Modelo de dados - relacional;
- Anexo 5 - Diagrama de classes simplificado.

2 Contexto

Neste capítulo apresentam-se os temas alvo de estudo, efetuados no sentido de aprofundar os conceitos e tecnologias existentes. Também é abordado o estudo realizado referente às soluções existentes no mercado, bem como os produtos que por algum motivo serviram de inspiração para o desenvolvimento do projeto.

2.1 Contextualização

O desenvolvimento de raiz de qualquer sistema pressupõe um período inicial de levantamento de requisitos e análise do problema. Esta análise teve como objetivo perceber e clarificar os requisitos do sistema em questão, permitindo não só interpretar alguns dos requisitos da plataforma, mas também delinear o processo de resolução do problema em mãos. O tipo de informação a armazenar foi um dos obstáculos mais difíceis de superar, uma vez que, a informação é bastante dinâmica.

Ao longo do desenvolvimento do *back-end* e da aplicação móvel, os requisitos foram continuamente refinados em várias reuniões, de forma evolutiva e iterativa. Do mesmo modo, os conhecimentos adquiridos através da interpretação pessoal e os diversos esclarecimentos por parte dos orientadores do projeto permitiram o aperfeiçoamento dos requisitos da plataforma.

2.2 Estado da arte de mercado

Antes de se iniciar a implementação de qualquer projeto é boa prática efetuar um estudo prévio, de soluções já existentes no mercado para conhecer os produtos atuais e perceber de que forma se poderia criar algo inovador. Este estudo constitui o estado da arte de mercado e permite uma maior clarividência sobre os objetivos que se podem atingir, e qual a melhor forma de os implementar.

Assim sendo, foram estudadas algumas soluções existentes no mercado, com objetivos e implementações semelhantes. Foram também estudadas aplicações com o objetivo de verificar e analisar os padrões utilizados nas suas interfaces, de forma a tornar a aplicação o mais *user-friendly* possível. Destas, apenas serão apresentadas duas aplicações, já que se verificou existir uma certa semelhança/tendência na sua estruturação, bem como na apresentação da informação.

2.2.1 GreatCall Link

A empresa GreatCall[3] disponibiliza um conjunto de produtos e serviços focados no público sénior, oferecendo soluções de saúde e segurança. O objetivo passa por proporcionar bem-estar aos seus utilizadores, de forma a que estes mantenham a sua independência, e ajudar os seus cuidadores, por exemplo, familiares, a terem conhecimento do seu estado atual e de situações de perigo em que seja necessário agir rapidamente.

Não se trata de uma empresa concorrente, contudo, apresenta produtos e serviços interessantes no contexto de monitorização. Além de disponibilizarem telemóveis personalizados e *smartphones* com o sistema operativo modificado, como consta nas imagens abaixo, simplificando a interface para o público a que se destinam, disponibilizam também uma série de aplicativos móveis com funcionalidades práticas e indicadas para a faixa etária em questão.

Figura 1 – Telefone e *smartphone* GreatCall

Dos aplicativos referidos anteriormente, destaca-se o GreatCall Link, uma vez que este permite:

- Enviar uma notificação quando o dispositivo é usado para pedir ajuda;
- Apresentar a atividade recente do utilizador;
- Mostrar se o dispositivo está ligado ou desligado e qual o nível de energia da bateria;
- Localizar o dispositivo no mapa.

Nas próximas figuras apresentam-se alguns *screenshots* da aplicação GreatCall Link:

[3] http://www.greatcall.com/

Figura 2 – Aplicação GreatCall Link

2.2.2 Life360

O Life360[4] é um projeto internacional, com o *slogan* "*Stay connected to the people that matter*". Além da plataforma *web*, também oferece, de forma gratuita, aplicações para dispositivos móveis, nomeadamente para iOS, Android e Windows Phone. A plataforma é constituída por um *front-end* simples, intuitivo e gratuito, sem necessidade de pagar pelas funcionalidades básicas, contudo, também existem algumas funcionalidades *premium* pagas.

Através da aplicação nativa, fornecida pela plataforma, é possível monitorizar a localização dos dispositivos móveis dos utilizadores e saber onde realmente se encontram as pessoas. Possibilita ao utilizador criar círculos de utilizadores, de forma a manter-se conectado com as pessoas que são realmente importantes. Este sistema também prevê a monitorização de dispositivos que não sejam *smartphones*, contudo esta funcionalidade é destinada apenas a utilizadores com conta *premium*.

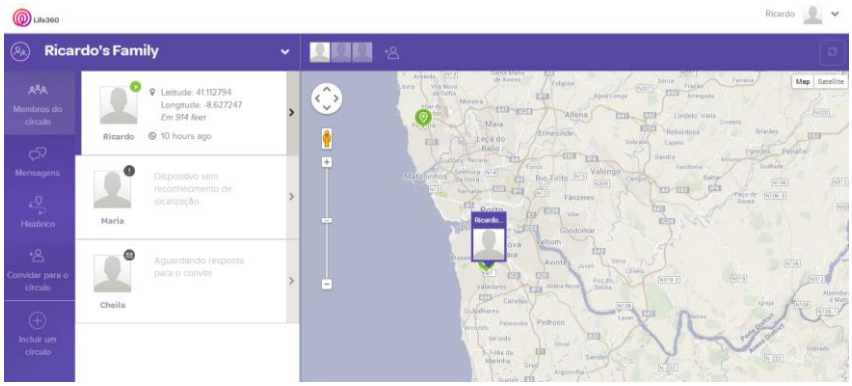


Figura 3 – *Back-end* Life360

Do estudo desta plataforma destaca-se o *design* simples e intuitivo do *back-end*, totalmente adaptado ao propósito final, e ainda a consolidação com as aplicações nativas.

[4] https://www.life360.com/

No que diz respeito a aplicações nativas salienta-se a aplicação Android, encontrando-se esta na versão 6.1.3, que regista uma maturidade no mercado com cerca de 5.000.000 - 10.000.000 *downloads*[5].

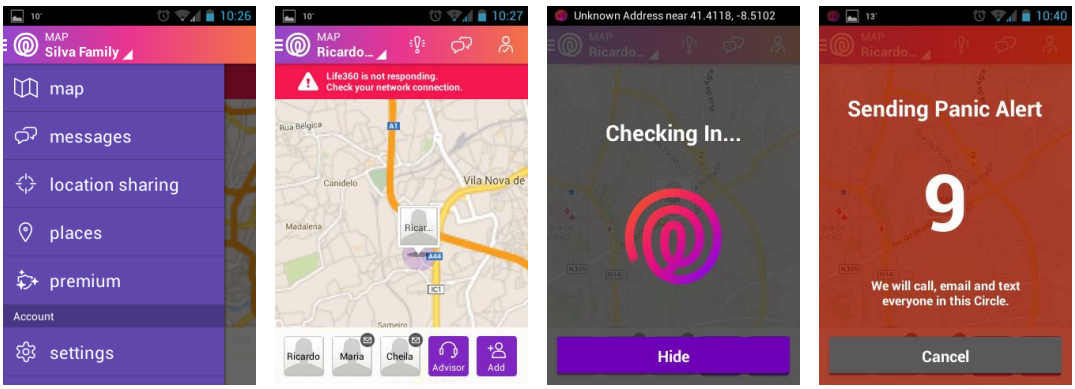


Figura 4 – *Screenshot* aplicação nativa Life360 (menu, mapa, *check-in* e *panic alert*)

Esta aplicação partilha a interface do *back-end*, colocando o utilizador confortável com as funcionalidades que esta implementa. Contudo, a aplicação é um pouco confusa no que toca a adicionar zonas de seguranças e círculos de utilizadores. Dois aspetos positivos e pertinentes estão relacionados com a possibilidade de trocar mensagens num círculo ou com um utilizador em particular e ainda a possibilidade de adicionar zonas de segurança temporárias.

2.2.3 AmberAlertGPS

A AmberAlertGPS[6] destaca-se pelo facto de se focar essencialmente no público-alvo crianças e adolescentes, tendo como objetivo promover a segurança através da monitorização. Para tal, disponibiliza um conjunto de produtos, todos eles pagos. As funcionalidades deste dispositivo são as seguintes:

- Monitorização e localização através do portal ou aplicação móvel para os pais;
- Alerta de bateria fraca;
- SOS - em caso de emergência, os pais são alertados;
- Envio da localização da criança por SMS ou por correio eletrónico, de forma automática;
- Configuração de velocidade máxima a que o dispositivo se desloca e, consequentemente, o alerta;
- Configuração de zonas de segurança, até um máximo de 20, e notificação quando estas são ultrapassadas;
- Possibilidade de comunicar diretamente com a criança através do dispositivo.

O dispositivo de monitorização e o *back-end* AmberAlertGPS exibem-se em seguida:

[5] Baseado no *site* https://play.google.com/store/apps/details?id=com.life360.android.safetymapd
[6] http://www.amberalertgps.com/

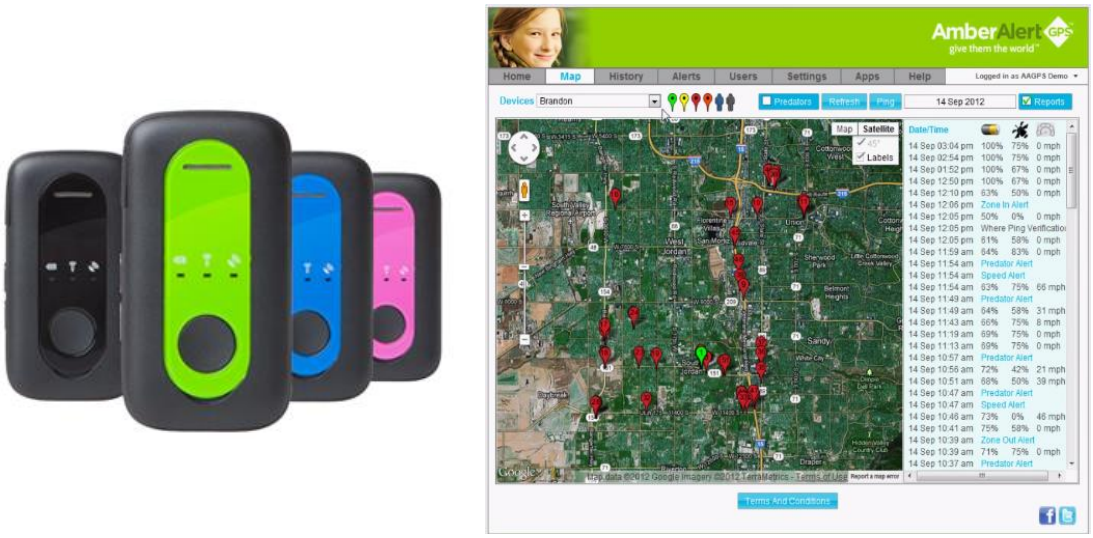


Figura 5 – Dispositivo e *Back-end* AmberAlertGPS

Apesar de funcional, esta plataforma é um pouco complexa. A informação é disposta de forma bastante detalhada e fatigante para o utilizador, não conseguindo, por isso, focar-se nas funcionalidades principais, ou seja, saber onde a criança se encontra.

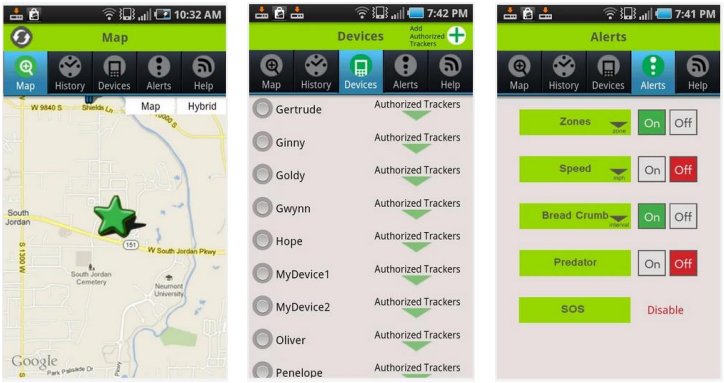


Figura 6 – Aplicação móvel Android AmberAlertGPS

A aplicação Android carece de uma interface moderna e apelativa. As principais funcionalidades que o sistema apresenta são apresentadas na aplicação, de forma sintetizada, ao contrário do que acontece no *back-end*.

2.2.4 Google Play

A aplicação Google Play[7], atualmente na versão para Android 4.8.22, foi estudada com o intuito de percecionar, em termos de interface, quais os padrões utilizados para uma melhor interação utilizador-aplicação e para a apresentação da informação.

[7] https://play.google.com

Google Play é uma aplicação desenvolvida pela Google, que proporciona aos utilizadores o acesso aos conteúdos disponíveis na loja *online*. É uma das aplicações mais usadas do sistema operativo Android, uma vez que, através desta, é possível aceder a novos e diversos conteúdos: aplicações, músicas, livros e filmes. Sendo uma aplicação desenvolvida pela Google é um bom exemplo de implementação de padrões, no que toca a diretrizes para o desenvolvimento de aplicações.

Durante o estudo das várias aplicações verificou-se a existência de algumas tendências na construção da interface. A aplicação Google Play é um ótimo exemplo de conceção e implementação de boas práticas na construção das aplicações e serve assim, como já referido anteriormente, como referência para as diversas aplicações.

Uma das tendências cada vez mais adotadas é a utilização do componente *Navigation Drawer*. Este é um painel de navegação normalmente situado no lado esquerdo do ecrã, com o intuito de apresentar um menu ao utilizador, sem que este ocupe espaço na *view* principal. Apresenta-se abaixo o painel *Navigation Drawer* da aplicação Google Play:

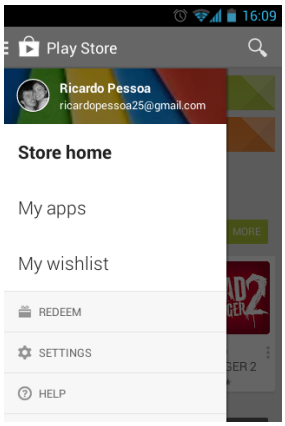


Figura 7 – Painel *Navigation Drawer* Google Play

O componente *Navigation Drawer* é usado em diversas aplicações, tais como: Google Maps, I/O 2013 e 2014, LinkedIn, Life360, Evernote, entre outras. É um componente já intrínseco em várias aplicações disponíveis para o sistema operativo e portanto já familiar aos utilizadores da plataforma Android.

Outro aspeto interessante e observado na aplicação é o uso do componente *Swipe Views Tabs*. Este componente tem como objetivo apresentar várias *views* e poder navegar entre elas com um gesto horizontal, também conhecido por paginação. Na imagem que se segue visualiza-se um exemplo de *Swipe Views Tabs*, extraído da aplicação Google Play.

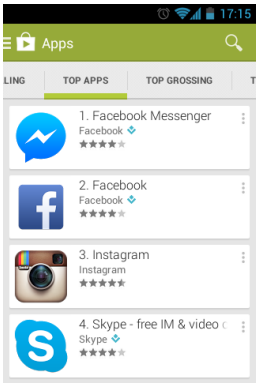


Figura 8 – *Swipe Views Tabs* Google Play

Por fim, destaca-se a divisão da informação por categorias, de forma a apresentar a informação mais organizada, motivo de inspiração no desenvolvimento do projeto "ChildSafe":

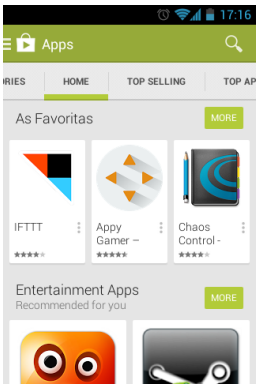


Figura 9 – Divisão em categorias Google Play

Com esta divisão por categorias, o utilizador terá uma maior facilidade em visualizar a informação e utilizar essas divisões de forma vantajosa. Como se pode analisar no exemplo anterior, o utilizador poderá ver as aplicações divididas por: favoritas, entretenimento, saúde e *fitness*, fotografia, entre outras. Se o utilizador pretender visualizar mais sugestões de uma categoria em concreto, poderá fazê-lo ao carregar no botão "*more*".

2.2.5 LinkedIn

LinkedIn[8] é uma rede social profissional, bastante conhecida para quem pretende recrutar profissionais, procurar emprego ou simplesmente fazer prospeção de mercado. O LinkedIn disponibiliza uma aplicação para vários sistemas operativos móveis, contudo, este estudo incidiu essencialmente na versão 3.4 para o SO Android.

Para além da aplicação LinkedIn implementar o *Navigation Drawer*, o que se destaca é o mecanismo de apresentação das últimas novidades:

[8] https://play.google.com/store/apps/details?id=com.linkedin.android&hl=pt_PT

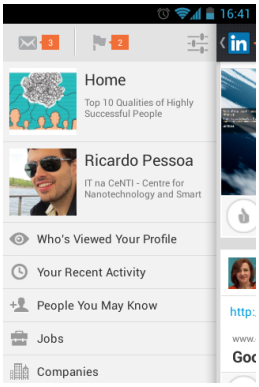


Figura 10 – Aplicação LinkedIn - últimas novidades

No interior do menu *Navigation Drawer* é possível visualizar, no topo, uma primeira linha onde aparecem as notificações de mensagens e eventos e ainda um botão para as configurações da aplicação. Na linha que se segue, com um título "*Home*" e uma pequena descrição "*Top 10 Qualities of Highly Successful People*" mostram-se as últimas novidades.

Foi assim ponderada a criação de algo semelhante, de forma a apresentar as últimas notificações de monitorização, para o utilizador ter noção dos últimos eventos associados aos seus dispositivos.

2.2.6 Conclusões e análise comparativa

No estado da arte de mercado foi possível verificar os principais concorrentes existentes no mercado, os seus objetivos, missões e a dimensão de cada projeto. Um dos pontos positivos verificados neste estudo é que atualmente não existe nenhum concorrente no mercado português, tornando-se assim uma vantagem na aceitação do projeto. As várias soluções estudadas podem ser expandidas para vários países num período relativamente pequeno, contudo, a aplicação desenvolvida no âmbito desta dissertação procurou responder aos principais requisitos dos utilizadores no nosso país, criando-se assim uma solução mais objetiva, o que é, consequentemente, uma vantagem para o projeto desenvolvido.

Infelizmente foi impossível estudar todas as tecnologias implementadas em cada solução. É algo que as plataformas tentam esconder, já que poderá ser o segredo de negócio o uso de determinada tecnologia. Contudo é possível verificar que alguns projetos implementam soluções inteligentes e eficientes, servindo como exemplo para o desenvolvimento deste projeto. Acredita-se que o uso da tecnologia NoSQL será uma vantagem para o projeto desenvolvido, uma vez que, oferece funcionalidades de escalabilidade, suporta grandes conteúdos de informação e acelera todo o processo de criação/alteração do modelo de dados, assuntos descritos no decorrer da dissertação.

Em suma, o estudo das aplicações existentes no mercado e das interfaces/padrões utilizados para apresentação da informação, bem como a análise da interação utilizador-aplicação foi sem dúvida benéfico, sendo possível percecionar as soluções mais adotadas no mercado e as mais

adequadas no projeto em questão, com vista a melhorar e acrescentar valor ao projeto. Também foi possível visualizar em que área cada solução se situa, que funcionalidades apresenta e qual o público-alvo a que se destina. Assim é possível aprender, evoluir e inovar num setor que apresenta poucas soluções no mercado nacional.

2.3 Estado da arte tecnológico

No estado da arte tecnológico são apresentados alguns estudos das tecnologias possíveis de implementação no desenvolvimento do projeto. Este estudo é de enorme importância, uma vez que, através dele é possível encontrar as melhores tecnologias para responder aos requisitos do projeto em si. Neste tópico é feita uma pequena introdução das tecnologias e as razões de tal escolha.

2.3.1 Linux Ubuntu

No mundo dos sistemas operativos *open-source,* o Ubuntu[9] não necessita de apresentação, sendo uma das distribuições Linux mais conhecidas do mundo. É reconhecido no seio dos programadores como um sistema operativo robusto e estável, ideal para o desenvolvimento de soluções sem custos de licenciamento a nível de *software*. De acordo com o *site* Ubuntu, é o SO e a distribuição *open-source* mais utilizados do mundo, e a Canonical oferece suporte para manutenção e segurança durante cinco anos. É um SO flexível, de fácil instalação, com linha de comandos para utilizações mais avançadas, e com uma vasta comunidade onde é possível encontrar suporte para diversos problemas.

Para o desenvolvimento do projeto, o sistema operativo Ubuntu 12.04 LTS possuía todos os requisitos necessários a nível de *software*, dando suporte ao desenvolvimento e implementação dos vários componentes, designadamente, a instalação do servidor para disponibilizar serviços *web* e do sistema de base de dados NoSQL. É também compatível com ferramentas de desenvolvimento do *back-end* e da aplicação Android.

2.3.2 Apache2 *Web Server*

O Apache[10] é um servidor *web* livre. Devido à sua versatilidade, possibilidade de configuração e mesmo à compatibilidade com diversas linguagens de programação foi determinado o uso do deste para disponibilizar o *back-end*. Apache disponibiliza para instalação, diversas bibliotecas públicas de *add-ons*, é um projeto bem documentado e no caso de existir alguma dificuldade ou problema é possível, com enorme facilidade, encontrar uma solução na sua enorme comunidade. É também muito conhecido na comunidade do sistema operativo Ubuntu. De

[9] http://www.ubuntu.com/
[10] http://www.apache.org/

acordo com o *site* W3Techs[11], Apache tem cerca de 60% de quota de mercado de utilização referente a servidores *web.* Com este resultado é o mais utilizado para *websites*.

2.3.3 REST

REST (*Representational State Transfer)* consiste num estilo de arquitetura de *software* para sistemas distribuídos *hypermedia*. Como tal, não é um método estritamente necessário para a construção de serviços *web*. O termo REST foi introduzido em 2000, na tese de doutoramento de *Roy Fielding*, um dos principais autores do protocolo *Hypertext Transfer Protocol* (HTTP).

De acordo com Fielding [Fielding,2000], REST é um conjunto de princípios de arquitetura de rede que definem como os recursos são definidos e resolvidos. A sua base é a interação entre clientes e servidores, começando com um pedido feito pelo cliente, que é processado pelo servidor e respondido pelo mesmo. Nos momentos em que o cliente se encontra em transição entre estados ("*at rest*") não faz pedidos ao servidor, não ocupando largura de banda. O cliente envia um novo pedido apenas quando está pronto para fazer a transição para um estado específico.

Por conseguinte, a utilização deste estilo de arquitetura foi utilizado na construção do *back-end* e na disponibilização dos serviços *web*, consumidos tanto pela aplicação como pelo *back-end*, dado que, facilita em termos de acesso e perceção da divisão dos recursos através do *url*. Este estilo é também usado pela API RESTful do servidor base de dados CouchDB.

2.3.4 NetBeans

NetBeans[12] IDE (*Integrated Development Environment*) é uma aplicação *open-source* para desenvolvimento de *software* em várias linguagens de programação nomeadamente: PHP, JAVA, Ruby, C, C++, entre outras. Este aplicativo tem suporte para vários sistemas operativos: Linux, Mac OSX e Windows, oferecendo as principais ferramentas para o desenvolvimento de aplicações.

O IDE facilita o desenvolvimento de aplicações através do seu editor, analisador de código e muitas outras ferramentas, criando um ambiente ideal para programação de aplicações.

Esta aplicação foi usada essencialmente para o desenvolvimento do *back-end*, na versão mais recente, ou seja, 8.0.

[11] http://w3techs.com/technologies/overview/web_server/all
[12] https://netbeans.org/

2.3.5 PHP

"PHP: *Hypertext Preprocessor"*[13] é uma linguagem de programação executada do lado do servidor. Desde o seu nascimento tem vindo a dar suporte às necessidades dos programadores e, consequentemente, evoluindo: desde a versão 1.0 até à última versão 5.5, com data de 20 de Junho de 2014. A evolução da linguagem de programação tem ocorrido de forma natural, devido às linguagens de programações concorrentes, às exigências das *frameworks* como é o caso da Zend Framework, Cake PHP, Symfony2, entre outras, e obviamente devido as exigências do mercado.

A linguagem PHP é uma linguagem de programação bastante ativa, isto é, no lançamento de uma nova *release* são corrigidos *bugs* a nível de segurança, erros ou mesmo performance. A versão instalada no servidor Apache é a PHP 5.5.9, considerada a mais recente e estável.

Prevaleceu o uso desta linguagem de programação, uma vez que é atual e bastante utilizada devido às suas diversas características: portabilidade, *frameworks* robustas, etc. Também a familiaridade com esta linguagem e o enorme suporte oferecido pela comunidade PHP foram fatores que pesaram nesta escolha. Esta linguagem de programação foi usada no desenvolvimento do *back-end* e para disponibilizar os diversos serviços *web* da plataforma.

2.3.6 Fligth

Fligth[14] é uma *framework* PHP, que permite criar uma aplicação *web* RESTful de forma simples e rápida. Esta *framework* foi implementada dada a sua simplicidade de instalação no servidor e devido à rápida aprendizagem. Com apenas três passos é possível criar uma arquitetura simples e bem estruturada, implementando as boas práticas de engenharia de *software*. De forma intuitiva é possível configurar as *routing* e as *views*, um processo moroso quando feito de raiz.

Assim, com recurso a esta *framework* foi possível criar o esqueleto do *back-end*, com o intuito de acelerar o processo de desenvolvimento do *site*. Para além disso, o uso desta *framework* não limitou o uso de qualquer outro tipo de bibliotecas, oferecendo assim uma total liberdade para a utilização de qualquer tecnologia disponível no mercado.

2.3.7 Sag

O projeto Sag[15] foi desenvolvido por Sam Bisbee, uma vez que este não estava satisfeito com as soluções existentes na linguagem PHP para conexão com CouchDB. Nasceu então esta biblioteca, com enfoque na simplicidade e com o objetivo de criar uma interface com pouco

[13] http://php.net/
[14] http://flightphp.com/
[15] http://www.saggingcouch.com/

overhead, de forma a possibilitar interligar o CouchDB com qualquer estrutura aplicacional. Esta biblioteca está disponível para as linguagens de programação PHP e para JavaScript.

A escolha desta biblioteca permitiu facilmente implementar a conexão com o servidor CouchDB, acoplando perfeitamente no projeto PHP e *framework* Fligth.

2.3.8 WebSockets

Com o aparecimento do HTML5, surgiu uma nova funcionalidade denominada WebSockets[16]. Através desta tecnologia é possível criar um canal bidirecional, sobre o protocolo TCP (*Transmission Control Protocol*), entre o cliente e o servidor, de forma a possibilitar a ambos o envio de informação. WebSockets foi desenvolvido e padronizado pelo W3C, com o intuito de criar uma conexão persistente entre o *browser* e o servidor. Tipicamente, o cliente faz a conexão com o servidor através do *browser* e o servidor fica responsável por gerir a ligação e redirecionamento da informação para o cliente.

Com recurso a esta tecnologia seria possível enviar para o cliente, através do *browser,* a monitorização dos sensores em tempo real, um dos principais requisitos do projeto.

Na figura que se segue consta o estudo efetuado relativo à compatibilidade da tecnologia WebSockets com os diversos navegadores de internet. Apresentam-se os vários *browsers* e versões correspondentes, assinalando a compatibilidade com WebSockets: verde-escuro, verde-claro e vermelho significando compatível, parcialmente compatível e não compatível, respetivamente.

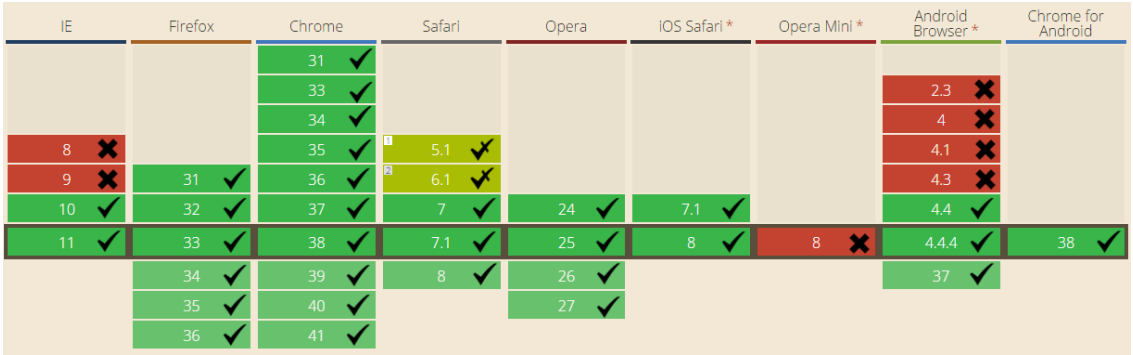


Figura 11 – Compatibilidade WebSockets com os diversos *browsers*[17]

Os *browsers* mais comuns são compatíveis com esta tecnologia, nomeadamente o Chrome, Internet Explorer, Firefox e Safari, etc. Assim, o uso desta tecnologia em nada compromete o funcionamento do sistema de monitorização no *browser*.

[16] https://www.websocket.org/
[17] Adaptado do *site*: http://caniuse.com/#feat=websockets

2.3.9 HTML

O HTML (*Hypertext Markup Language*) é a linguagem de marcação, interpretada pelos *browsers* para poder representar informação. Esta linguagem está presente em todas as páginas *web* e possuem extensão *.html* ou *.htm*.

O propósito do navegador *web* é interpretar um documento HTML recebido por um servidor e apresentar a informação contida no próprio documento. O navegador não apresenta as *tags* HTML, mas utiliza-as para interpretar e formatar o conteúdo da página. A última versão HTML, especificamente a HTML5, permite desenvolver aplicações *web* modernas.

O HTML5 não teve data oficial de lançamento. Depois de, em 2004, a Apple, Mozilla e Opera formarem a *Web Hypertext Application Technology Working Group* (WHATWG), com o objetivo de padronizar o HTML, em 2006, W3C mostrou interesse em juntar-se ao grupo WHATWG, e em 2011 lançou uma versão "final" do HTML5 [WHATWG, 2008].

Na construção do *website* para a plataforma era essencial o uso da linguagem HTML. Com a linguagem de programação PHP, apresentada anteriormente, é possível criar páginas com conteúdos dinâmicos, de forma a apresentar a informação referente ao utilizador.

Hoje em dia, o HTML está associado a duas outras tecnologias: CSS e JavaScript, apresentadas a seguir, que permitem que as páginas sejam ainda mais dinâmicas e tenham estilos associados e customizados.

2.3.10 JavaScript

JavaScript é uma linguagem de programação *web* dinâmica. A primeira versão, inicialmente denominada *livescript*, foi criada em Maio de 1995, por Brendan Eich, que trabalhava na Netscape, e é atualmente membro da equipa do Mozilla. A maior parte dos *sites* modernos usam JavaScript e todos os *browsers desktops*, *tablets*, *smartphones* incluem interpretadores JavaScript.

JavaScript faz parte das três tecnologias que todos os programadores *web* devem dominar: HTML para especificar o conteúdo, CSS para especificar a apresentação e JavaScript para especificar o comportamento das páginas *web*. A implementação *client-side* da linguagem de programação JavaScript permite uma melhor interação com o utilizador, controlar o *browser*, efetuar uma comunicação assíncrona e alterar o conteúdo do documento sem o carregamento da informação na íntegra.

Por conseguinte, através da linguagem de programação JavaScript foi possível criar uma interação mais *user-friendly* com o utilizador, implementar funcionalidades com ajuda da linguagem JavaScript como correção de formulários, utilização de APIs em JavaScript, etc.

2.3.11 jQuery

O jQuery[18] é uma biblioteca que permite simplificar o código JavaScript existente nas páginas HTML. Facilita a criação de animações, manipulação DOM, pedidos AJAX e outro tipo de processos, permitindo um simples e rápido desenvolvimento do código. O alto nível de abstração que oferece torna-se a principal vantagem do uso desta biblioteca.

jQuery permite abreviar tarefas comuns, que em JavaScript puro requerem várias linhas de código, em simples chamadas à biblioteca. Assim, muitas das *frameworks* JavaScript, analisadas nos próximos tópicos, dependem desta biblioteca que se tornou um elemento essencial no que toca ao desenvolvimento de funcionalidades que recorrem a código JavaScript.

2.3.12 CSS

O *Cascading Style Sheets* (CSS) é utilizado para especificar a aparência e a formatação de um documento HTML. O seu principal benefício é promover a separação entre a formatação e o conteúdo de um documento. No enquadramento do HTML5 e do JavaScript, o CSS não é um elemento indispensável, isto é, podem ser implementadas funcionalidades HTML5 e JavaScript sem recorrer ao CSS. No entanto, o CSS permite apresentar estas funcionalidades de forma mais agradável ao utilizador. Adicionalmente, facilita a mudança do estilo de apresentação e reduz o tamanho dos ficheiros HTML, já que os elementos relativos à apresentação são movidos do ficheiro .html para um ficheiro .css.

2.3.13 Bootstrap

O Bootstrap[19] é uma *framework* desenvolvida pelo Twitter para a criação de *front-ends* de forma simples e célere. Esta *framework* é constituída por um conjunto específico de ficheiros CSS, HTML e JavaScript, que utilizam uma série de padrões e/ou convenções para formulários, botões, tabelas, etc. Permite então que um programador *web* construa *front-ends* sofisticados e com *responsive design*, ou seja, oferecendo uma ótima experiência de visualização, navegação, fácil leitura e redimensionamento para uma vasta gama de dispositivos. Deste modo, é possível dar mais enfoque a questões de usabilidade e apresentar facilmente *designs* interessantes e bem desenvolvidos.

Com recurso a esta *framework* foi então possível desenvolver um *back-end* com uma interface agradável, e acelerar o processo de aparência do *site*.

[18] http://jquery.com/
[19] http://getbootstrap.com/2.3.2/

2.3.14 Mapstraction

Mapstraction[20] é uma biblioteca, em JavaScript, que disponibiliza uma interface comum com as APIs JavaScript fornecidas por serviços de mapas. Através desta é possível alterar o *map provider* com apenas algumas modificações no código. Este projeto é *open-source* e suporta a maior parte dos serviços de mapas conhecidos, como é caso do Google Maps v2 e v3, MapQuest e MapQuest Open, Microsoft Bing v6 e v7, Nokia Here e Ovi, OpenLayers, entre outros.

Uma vez que a plataforma *web* desenvolvida depende de serviços de mapas era boa prática a criação de uma camada de abstração para o caso de ser necessária a troca do serviço de mapas, se proceder a essa alteração de forma mais acessível e menos morosa.

2.3.15 Google Maps

A API JavaScript da Google Maps[21], mais precisamente a versão 3, permite incorporar os mapas da Google em qualquer página *web*. Esta versão está otimizada para dispositivos móveis, não esquecendo os navegadores tradicionais *desktop*. A versão em causa foi desenvolvida para ser mais rápida que a sua antecessora, com funcionalidades de interação e com a adição de conteúdos no mapa por meio de diversos serviços.

Foi decidida a implementação destes mapas para a apresentação da localização dos dispositivos, uma vez que seriam recebidas as coordenadas da localização por via de dispositivos com GPS. Assim, a API JavaScript Google Maps foi essencial não só para a apresentação das localizações dos dispositivos mas também na definição de zonas geográficas e na pesquisa do endereço através das coordenadas.

2.3.16 Highcharts

Highcharts [22] é uma biblioteca de gráficos em JavaScript, que oferece a possibilidade de adicionar facilmente gráficos interativos em páginas *web*.

A Highsoft AS é a empresa que desenvolveu Highcharts JS, lançado no final de 2009, tornando-se rapidamente num dos líderes na indústria de gráficos, baseados na linguagem de programação JavaScript.

Funciona em todos os navegadores de internet modernos, nas versões *desktop* e móveis, incluindo *smartphones, tablets*, nos sistemas operativos iOS e Android. No Internet Explorer, está disponível a partir da versão 6.

[20] http://mapstraction.com/
[21] https://developers.google.com/maps/documentation/javascript/
[22] http://www.highcharts.com

Highcharts suporta atualmente gráficos do tipo *line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange, bubble, box plot, error bars, funnel, waterfall e polar*.

Foi prevista a apresentação da informação em forma de gráfico, e, nesse sentido, foram estudadas algumas bibliotecas capazes de oferecer essas funcionalidades. A biblioteca Highcharts foi a que se apresentou capaz executar a tarefa da melhor forma: os gráficos são apelativos, editáveis e, portanto, indicados para o tipo de informação a apresentar.

2.3.17 jCryption

jCryption[23] é um *plugin open-source* para jQuery, utilizado para executar a criptografia em JavaScript do lado do cliente, podendo ser descodificado do lado do servidor. O projeto jCryption nasceu por intermédio de Daniel Griesser em 2009, para preencher a lacuna da inexistência de um *plugin* que possibilitasse a encriptação dos dados. Inicialmente só foi lançado para a linguagem de programação PHP, mas, devido ao elevado número de *downloads* e sua consequente utilização, o autor avançou com o desenvolvimento e melhoramento do projeto jCryption. A última versão lançada, ou seja, a versão 3.0.1, para além de utilizar o algoritmo RSA para a troca de chaves encriptadas, também usa o algoritmo AES (*Advanced Encryption Standard*) que torna a encriptação ainda mais otimizada. Recorrendo às duas bibliotecas Crypto-JS 3.1 e JSEncrypt, jCryption é totalmente compatível com OpenSSL tornando, deste modo, as tecnologias independentes do servidor.

A segurança da informação é um tema bastante sensível e, como tal, é extremamente necessário implementar na plataforma alguns mecanismos de segurança. Foi assim introduzido este *plugin*, com intuito de proteger a informação do utilizador.

2.3.18 Android Studio

Android Studio[24] é um IDE lançado pela Google para o desenvolvimento de aplicações Android, atualmente, ainda na sua versão beta (versão 0.8.8). Desde o lançamento da versão 0.1, em Maio de 2013, a Google tem-se esforçado no sentido de melhorar e desenvolver esta aplicação. Este IDE é baseado no ItelliJ IDEA e vem melhorar as ferramentas e funcionalidades já existentes no mercado, dado que o Eclipse, com o ADT (*Android Development Tools*) era a única alternativa para o desenvolvimento de aplicações nativas para Android.

Com o Android Studio, a Google pretende oferecer uma ferramenta mais completa para o desenvolvimento e depuração de aplicações para o sistema operativo móvel. Como é baseado ItelliJ IDEA oferece vantagens como: facilidade de escrita de código, através de sugestões

[23] http://www.jcryption.org/
[24] https://developer.android.com/sdk/installing/studio.html

inteligentes, analisa e sugere correções para o código e facilita a navegação entre o código do projeto, o que consequentemente torna os programadores mais produtivos.

Esta ferramenta é totalmente gratuita e a sua versão, ainda beta, trás enormes vantagens como é o caso da ferramenta Gradle. O objetivo desta ferramenta é facilitar a reutilização de código e recursos, a criação de várias versões da aplicação, a simplicidade em configurar, customizar e estender a construção da aplicação. Também permite efetuar *build* à aplicação, tanto através do IDE, como recorrendo à linha de comandos, o que possibilita uma boa integração com o IDE.

Outra vantagem do Android Studio é a capacidade de visualizar em tempo real as alterações efetuadas no *layout* da aplicação, de forma a ter noção do aspeto da interface nas várias *views*.

2.3.19 Java

Java[25] é a linguagem de programação lançada em 1995 pela Sun Microsystems. Esta linguagem de programação caracteriza-se, essencialmente, por ser orientada a objetos, simples, robusta e independente da plataforma. Para tal é necessária a instalação de uma máquina virtual denominada *Java Virtual Machine* (JVM), com o objetivo de interpretar o código do programa Java para o código da máquina específico da plataforma em questão. No processo de compilação, em vez de ser compilado para o código máquina do sistema desenvolvido é compilado para o *bytecode*. Assim, através da JVM, é possível executar qualquer aplicação Java em qualquer plataforma, o que lhe confere a característica portabilidade.

O desenvolvimento das aplicações nativas em Android implica a utilização da linguagem de programação Java. Deste modo, a utilização desta linguagem está intrínseca no desenvolvimento da aplicação *mobile*.

2.3.20 CouchDB

Apache CouchDB[26] é um sistema de base de dados NoSQL, que nos fins de 2009/inícios de 2010 rapidamente se tornou popular. Damien Katz, criador do CouchDB, trabalhava na Lotus Notes, numa *framework* de gestão de documentos e aproveitou o potencial do MapReduce do BigTable criando uma fusão destes dois projetos [Chandler, 2009]. Nasceu assim o CouchDB, desenhado para ser um novo tipo de base de dados, baseado nos desafios de escalabilidade e abrangendo completamente a *web*.

Apache CouchDB é um projeto *open-source*, armazena a informação em documentos JSON, e não força o uso de um esquema ou estrutura rígida da informação. Disponibiliza uma interface *web* via HTTP para o acesso aos dados e gestão da base de dados. Permite efetuar pesquisas e indexação dos documentos através de JavaScript. Funciona com aplicações *web* e aplicações

[25] www.java.com
[26] http://couchdb.apache.org/

móveis, e possibilita a distribuição e replicação dos dados, de forma eficiente, suportando configurações *master-master* com deteção automática de conflitos.

CouchDB trás diversas funcionalidades, facilitando o desenvolvimento de aplicações. É altamente disponível, tolerante a falhas e eventualmente consistente. Também implementa mecanismos de segurança.

Foi utilizado este sistema de base de dados para o armazenamento da informação porque se trata de um sistema à altura dos requisitos e de fácil instalação. Outro fator importante nesta decisão foi a funcionalidade Futon para visualização e manipulação das *views* e dados. A compatibilidade com CouchBase foi também um fator decisivo na escolha deste sistema. Foi então possível apresentar uma solução para o armazenamento da informação adequada aos requisitos do projeto.

Dada a importância desta tecnologia, esta é descrita mais detalhadamente no subcapítulo 3.8.

2.3.21 CouchBase

CouchBase[27] nasceu em 2011, da junção dos projetos CouchDB e Membase. A ideia por de trás da fusão era bastante simples: combinar o modelo de dados orientado ao documento, as capacidades de indexação e consulta do CouchDB com a alta performance, fácil escalabilidade, e *alway-on* do Membase, de modo a construir um sistema de base de dados superior.

CouchBase é igualmente 100% *open-source* sobre a licença Apache 2.0. É uma base de dados orientada a documento e também orientada a *key-value*, assunto abordado no decorrer da dissertação. CouchBase e CouchDB detêm bastantes características em comum devido à influência do CouchDB, que usa praticamente a mesma abordagem na indexação e consulta, apenas estendendo e otimizando para casos distribuídos.

Seguem-se as principais diferenças entre CouchDB e CouchBase:

[27] http://www.couchbase.com/

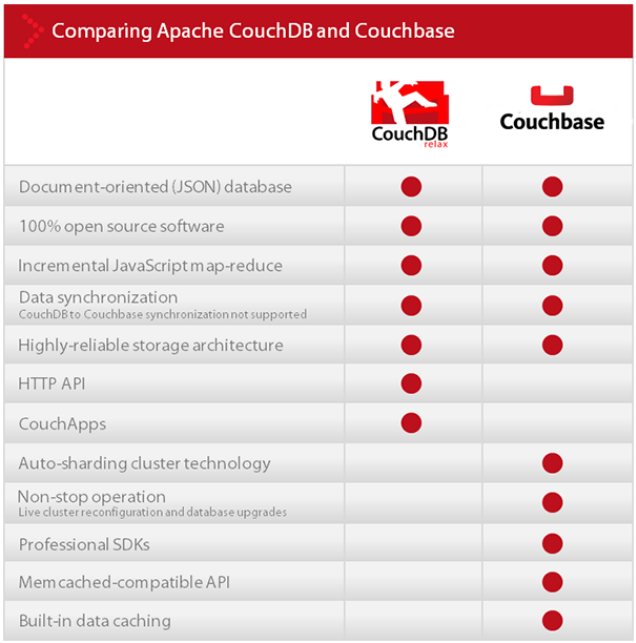


Figura 12 – Comparação Apache CouchDB com CouchBase

CouchBase *Server* disponibiliza uma *web* interface administrativa onde é possível adicionar ou remover servidores num *cluster*, através de um simples botão. Adicionando novos servidores é possível escalar horizontalmente o *cluster*.

Auto-sharding é também uma característica interessante do CouchBase, permitindo distribuir a informação de forma uniforme pelos servidores do *cluster*, para a distribuir a carga de trabalho e aumentando a disponibilidade sem necessidade de efetuar alterações na camada aplicacional.

Segundo Cardoso [Cardoso,2012], *sharding* consiste em dividir os dados em fragmentos (*shards*) por nós independentes, para que cada cliente que necessite dos dados seja redirecionado para o nó que os contém.

Apesar de ter sido escolhido o CouchDB, em detrimento do CouchBase, para o armazenamento da informação da plataforma, é possível migrar o sistema de base de dados para CouchBase com relativa facilidade, devido à compatibilidade entre os dois sistemas. Por conseguinte, se houver necessidade de escalar o sistema devido ao grande volume de dados, CouchBase torna-se a solução alternativa de fácil implementação.

2.3.22 MongoDB

MongoDB[28] deriva da palavra inglesa "*humongous*", com significado em português, gigantesco. Começou a ser desenvolvido em Outubro de 2007 pela empresa 10Gen, e a primeira versão

[28] http://www.mongodb.com

open-source foi lançada em Fevereiro de 2009, disponibilizando suporte comercial e serviços associados.

MongoDB é um sistema de base de dados NoSQL, desenvolvido na linguagem C++. É orientada ao documento, armazenando os documentos no formato JSON com *schema* dinâmico, oferecendo simplicidade. MongoDB também oferece *auto-sharding* para escalar horizontalmente, sem comprometer o funcionamento do sistema. É um sistema que oferece funcionalidades para efetuar pedidos baseados em documento e também possui a *framework* MapReduce.

MongoDB não fornece soluções para o armazenamento de dados em modo *offline* e o suporte para aplicações móveis ainda não apresenta grandes soluções, comparativamente ao projeto CouchDB e CouchBase.

2.3.23 CouchBase Lite Android

CouchBase Lite Android[29] é uma API disponibilizada pela CouchBase, referente ao segmento CouchBase *mobile*. CouchBase Lite é um motor de base de dados NoSQL, orientado ao documento, leve e indicado para dispositivos móveis. São disponibilizadas APIs para sistemas nativos destinados ao desenvolvimento de aplicações em iOS (Objective-C) e Android (Java). É também disponibilizado um *plugin* para PhoneGap e Xamarin (este último ainda beta) permitindo o desenvolvimento, através de técnicas familiares *web*, para os sistemas iOS e Android. Em termos de documentação, à exceção dos *plugins* PhoneGap e Xamarin, apresenta-se bastante completa e bem detalhada no que concerne às características, procedimentos e dependências necessárias para a instalação.

CouchBase Lite é ideal para dispositivos móveis, já que é uma solução leve e otimizada. O motor da base de dados é uma biblioteca incorporada na aplicação e não é criado nenhum serviço separado. Tamanho comprimido, rápida iniciação do serviço, baixo consumo de memória e boa performance são ainda características desejadas em qualquer biblioteca para os dispositivos móveis, características estas da CouchBase Lite.

Através desta API para Android foi possível criar, de forma ágil, uma aplicação com ligação à base de dados CouchDB/CouchBase e a sincronização da informação. Apesar dos projetos Apache CouchDB e CouchBase serem projetos diferentes, CouchBase derivou do projeto CouchDB o que levou à partilha de características. Assim, a compatibilidade das APIs disponibilizadas pela comunidade CouchBase com o CouchDB é totalmente conhecida e aprovada por ambas as partes. CouchBase continua a contribuir com o CouchDB, o que estreita os laços dos dois projetos.

CouchBase Lite confere às aplicações móveis um comportamento autónomo, como se se tratasse de uma extensão dos servidores de base de dados e faculta, principalmente, a possibilidade de funcionamento da aplicação em modo *offline* [CouchBase,2012a]. Suporta a

[29] http://www.couchbase.com/nosql-databases/couchbase-mobile

replicação com os servidores de base de dados compatíveis, oferecendo recursos à aplicação de sincronização de dados. Oferece baixa latência e acesso *offline* aos dados, uma vez que, na maior parte dos casos está sempre a trabalhar com os dados localmente e em *background* procede à sincronização da informação [CouchBase,2012a].

A arquitetura pressupõe as seguintes tecnologias:

- CouchBase Lite - com CouchBase Lite embebido na aplicação móvel é possível ter um controlo refinado sobre os dados e redirecionamento;
- *Sync Gateway* - este servidor (optativo) tem como objetivo gerir e controlar o acesso e redirecionamento da informação HTTP - *based* para cliente móveis;
- CouchBase *Server* - é um servidor de base de dados NoSQL, orientado ao documento de alta performance, escalável, servindo milhões de utilizadores.

Neste projeto, o CouchBase *Server* foi substituído pelo CouchDB. Abaixo é possível visualizar a estrutura *mobile*, de forma ilustrativa:

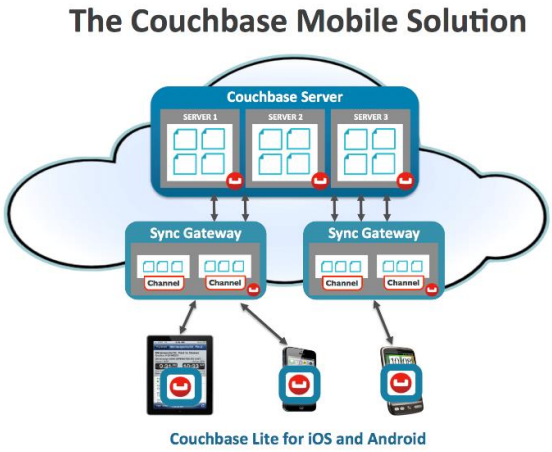


Figura 13 – Solução CouchBase *mobile*[30]

2.3.24 GitHub

GitHub[31] é um repositório Git, para controlo de versões e gestão de código fonte de *software* (em inglês, *Source Code Management* - SCM). O seu *slogan* "*Build software better, together*" caracteriza as suas funcionalidades: oferece as melhores ferramentas através da sua arquitetura de controlo de versão distribuída, o que possibilita a existência de uma cópia do repositório na máquina local de trabalho e no servidor remoto. GitHub disponibiliza uma interface *web*, *desktop* e recentemente *mobile* para um melhor controlo de acesso e colaboração nas tarefas de desenvolvimento de *software*. Lançou recentemente uma campanha[32] para estudantes, com a possibilidade de criação de repositórios privados.

[30] Retirado do *site*: http://docs.couchbase.com/couchbase-lite/cbl-concepts/
[31] https://github.com/
[32] https://education.github.com/

GitHub tornou-se o maior repositório de código do mundo, com mais de cinco milhões de programadores de *software* e mais de dez milhões de repositórios. Imensos projetos *open-source* como Ruby on Rails, Homebrew, Bootstrap, Django e jQuery, escolheram GitHub como repositório e migraram para este serviço [Gousios, Vasilescu, Serebrenik and Zaidman, 2014].

Este serviço disponibiliza funcionalidades como *wikis*, gestão de tarefas, controlo de *bugs*, pedido de novas funcionalidades, o que lhe confere algumas propriedades de funcionamento de rede social para desenvolvimento de projetos.

Sendo assim, a utilização deste repositório trouxe ao projeto enumeras vantagens. Sempre que alguma funcionalidade era adicionada, alguma alteração efetuada ou algum *bug* encontrado era documentado no repositório. De forma profissional foi possível proceder ao controlo de versões do desenvolvimento dos projetos *back-end* e aplicação móvel. Foi ainda disponibilizado o acesso ao orientador do ISEP, de forma a disponibilizar o estado atual do projeto e alterações ocorridas.

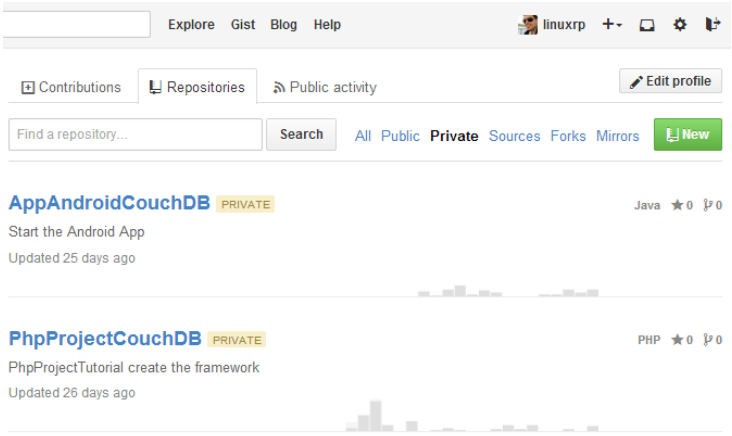


Figura 14 – GitHub repositórios

2.4 Aplicações nativas vs. aplicações web

Depois de analisado o estado da arte de mercado e ainda o estado da arte tecnológico é importante refletir sobre a escolha entre desenvolver uma aplicação nativa ou desenvolver uma aplicação *web*, devido às repercussões distintas que tal decisão pode deter ao nível dos requisitos, funcionalidades e público-alvo da aplicação.

Uma aplicação nativa é uma aplicação que é desenvolvida para uma determinada plataforma ou sistema operativo e que corre nativamente neste. Tem por isso de ser transferida e instalada para ser utilizada. Já uma aplicação *web*, como o próprio nome indica, é baseada numa linguagem *web*, tipicamente em HTML5, JavaScript e CSS3 [Viswanathan,2013]. Deste modo, é simplesmente uma página *web* que pode ser visualizada num *browser*, logo, não é necessário instalar nada para além deste.

Contudo, a distinção acima não é a única. Existem diversas vantagens e desvantagens que diferenciam as aplicações nativas das aplicações *web*, que serão descritas seguidamente.

As aplicações nativas são mais objetivas, uma vez que, em regra, o desenvolvimento destas segue as convenções e normas das plataformas destino, tornando a interface mais intuitiva e apelativa para os utilizadores que já estão familiarizados com determinado tipo de interfaces. Também é possível ter total controlo sobre o seu *layout* da interface de uma aplicação nativa, enquanto que numa aplicação *web*, o mesmo já não é possível. O programador deve ter em consideração o modo como a aplicação é visualizada nos diferentes *browsers* e dispositivos.

Também se a aplicação necessitar de aceder às várias funcionalidades do dispositivo, como por exemplo câmara fotográfica/vídeo, GPS, acelerómetro, bússola, lista de contactos, ou mesmo o modo de notificações em *background*, então a melhor opção será a aplicação nativa pois através desta é possível aceder sem restrições a todas essas funcionalidades [Viswanathan,2013]. Ainda a possibilidade de executar a aplicação em modo *offline* e aceder à informação armazenada no dispositivo restringe a escolha. Neste caso, as aplicações necessitam de estar preparadas para o funcionamento *offline*, o que significa que os utilizadores podem proceder à utilização da aplicação sem serem afetados pela conectividade ou velocidade da internet [CouchBase,2012a].

O desempenho da aplicação é também um aspeto crítico e tem um papel fundamental na experiência de utilização. De uma forma geral, as aplicações nativas são mais rápidas e fluídas do que as aplicações *web*, oferecendo também uma experiência de utilização mais sólida e integrada porque são desenvolvidas tendo em conta as especificações dos dispositivos e SO.

No que diz respeito a atualização da aplicação, as aplicações *web* apresentam vantagem, devido à apresentação da última versão, o que não se verifica numa aplicação nativa. Caso o utilizador não atualize a aplicação, esta apresentará a versão instalada previamente, e a falta de "manutenção" da aplicação pode desapontar o utilizador.

Concluindo, nenhuma das implementações é perfeita para todas as situações e atualmente tanto aplicações nativas como aplicações *web* devem ser igualmente consideradas. Em alguns casos até ambas as abordagens são válidas e fazem sentido. Noutras situações, como é o caso do projeto "ChildSafe", ainda se disponibiliza uma versão *mobile* do *site*, efetuada através de *responsive design*.

2.5 Conclusões

A decisão crucial durante o estudo tecnológico esteve relacionada com o sistema de base de dados, devido à importância que este tem no sistema global, uma vez que, a performance da plataforma dependerá das características da base de dados.

A escolha do sistema de base de dados recaiu sobre o CouchDB. É conveniente para o projeto a utilização de uma base de dados orientada ao documento, devido à sua versatilidade na

obtenção da informação e às enumeras vantagens detalhadas nos capítulos subsequentes. Conforme já referido anteriormente, o suporte deste sistema com aplicações móveis, a documentação completa e organizada, a compatibilidade entre o CouchDB e o CouchBase e a sua fácil instalação foram fatores decisivos.

Apesar da tecnologia NoSQL ser relativamente recente, os sistemas anteriormente estudados já possuem um certo peso no mercado. Qualquer ponto de partida para implementação do servidor de base de dados, na sua génese, seria uma boa opção devido à natureza dos dados da plataforma, ou seja, dinâmicos e o suporte para o sistema ser escalável no futuro.

Concluindo, o estudo efetuado providenciou um maior conhecimento sobre as tecnologias envolventes no projeto, especialmente no âmbito teórico. Por outro lado, auxiliou diretamente a escolha das tecnologias que mais se adequam aos requisitos, sem desrespeitar as decisões da organização.

De um modo geral, os estudos referidos neste capítulo foram fulcrais para a conclusão do projeto, e para a qualidade da solução apresentada.

3 Base de dados NoSQL

Base de dados NoSQL é um tema bastante atual. Apresenta-se como um movimento tecnológico em alternativa ao modelo relacional. Devido à sua importância é apresentado num capítulo independente nesta dissertação de mestrado, de forma a abordá-lo mais aprofundadamente.

Primeiramente é apresentada uma pequena introdução às bases de dados, a evolução das mesmas e os seus objetivos. Posteriormente são apresentadas as diversas características e tipos de bases de dados NoSQL. É também neste capítulo que se aprofundará o CouchDB, as suas principais funcionalidades e os motivos da escolha deste sistema para o armazenamento da informação na plataforma.

3.1 Introdução às bases de dados

O termo base de dados surgiu em meados dos anos 60, como um sistema que se destinava ao armazenamento de informação. Esse sistema passaria a representar uma camada, tornando-se assim um novo e interessante conceito, que propunha a divisão da aplicação com a informação, tornando assim as aplicações mais robustas e fáceis de manutenção.

Em 1970, Edgar Codd propôs uma nova forma, mais eficaz, de armazenar informação - o modelo relacional [Horrocks,2001] e [Codd,1970]. Este modelo usaria a linguagem de interrogação à base de dados, denominada SQL (*Structured Query Language*), servindo para efetuar pesquisas e gerir a própria base de dados.

Embora este modelo tenha sido amplamente aceite, não foi até meados de 1980, uma vez que, nesta altura não existiam grandes soluções, a nível de *hardware*, que acoplassem um sistema de base de dados [Dalakov,2011] e [Brown,2002]. A partir de meados de 1990, o modelo relacional tornou-se o método dominante para armazenamento de informação, já que, o *hardware* tinha conseguido acompanhar os requisitos necessários para sustentar um sistema de gestão de base de dados [Brown,2002].

Assim surge o termo *Relational Database Management System* (RDBMS), que não é nada mais do que um sistema de gestão relacional de base de dados. Alguns exemplos dos sistemas RDMBS mais populares são Oracle, Microsoft SQL Server, MySQL e PostgreSQL.

Com o aparecimento das linguagens de programação orientadas a objetos (OO) surgiram também os *Object-Oriented Database Management System* (OODBMS), na tentativa de acompanhar o enorme sucesso das linguagens OO [Erlingsson,1996]. Estes sistemas armazenam informação, através da representação do objeto, contudo, não tiveram o mesmo sucesso que as linguagens de programação OO, uma vez que não apresentaram grande adesão, devido à sua complexidade e performance para grandes volumes de dados. Assim, os sistemas RDBMS continuaram a ser dominadores nas soluções para armazenamento de dados.

A partir de 2000, as aplicações começaram a produzir grandes volumes de dados, dada a sua complexidade e utilização. É o caso das redes sociais e motores de busca: Facebook, Amazon e Google [Hecht and Jablonski,2011], que lidam com quantidades de informação na ordem dos *terabytes* e *petabytes*, massivos pedidos de leitura e escrita, que têm de ser respondidos sem os utilizadores notarem a latência.

Com as empresas a utilizarem grandes volumes de dados que, de dia para dia, continuavam a crescer, começaram a aparecer problemas com que nunca antes se tinham deparado. Consequentemente foi necessária uma reflexão, por parte das empresas, sobre os sistemas de gestão de base de dados que utilizavam e repensar a forma como eram geridas as vastas quantidades de dados [Chandler, 2009]. Assim, as empresas começaram a repensar a estrutura de dados, e os problemas de escalabilidade e disponibilidade dos dados, que o modelo relacional parecia não lidar da melhor forma.

Com a necessidade de gerir grandes volumes de dados em constante mudança, surgiu o termo NoSQL. NoSQL é um acrónimo de "*not only* SQL", ou seja, não segue o modelo relacional, nem usa a linguagem SQL para comunicação com a base de dados.

O resultado deste novo paradigma foi o projeto BigTable, produto de uma pesquisa levada a cabo pela Google. Chegaram à conclusão de que a utilização dos dados, na maioria das aplicações da internet, não apresentava os mesmos pressupostos das bases de dados relacionais, projetadas nos anos 70 e 80 [Chandler, 2009]. Os investigadores da Google perceberam que a grande maioria das aplicações *web* não necessitariam de mecanismos de bloqueio (ACID) e a integridade passaria a ser gerida do ponto de vista aplicacional. Ao desenvolverem um sistema de dados simples, com base no modelo de uma coluna, permitiram o crescimento do sistema para volumes de informação nunca antes vistos.

Neste sentido, as bases de dados surgem como uma possível alternativa para problemas tipificados, onde é expectável a existência de imensos dados, onde a informação não seja necessariamente rígida e o sistema possa ser escalável. Aumenta-se, assim, o leque de escolha no armazenamento de dados, de forma a que não seja sempre utilizada a mesma ferramenta para todas as soluções.

3.2 Teorema CAP

No mundo dos sistemas de bases de dados é comum abordar o teorema CAP (*Consistency*, *Availability*, and *Partition-Tolerance*), em português: consistência, disponibilidade e tolerância a falhas, respetivamente. Este teorema foi introduzido por Eric Brewer em 2000 [Brewer,2000] e mais tarde comprovado por Gilbert and Lynch [Gilbert and Lynch,2002].

Em suma, o teorema refere que ao criar um sistema, qualquer que ele seja, é necessário ter em conta as propriedades do teorema CAP e escolher apenas duas características das três, ou seja, não podemos construir um sistema disponível, consistente e tolerante a falhas ao mesmo tempo, sendo ele distribuído ou não.

Num sistema de base de dados, as propriedades do teorema CAP caracterizam-se por:

- Disponibilidade - indica que sempre que é efetuado um pedido, espera-se que esse pedido seja respondido com a informação pretendida. A alta disponibilidade geralmente é realizada através de vários servidores interligados entre si, de forma a atuar como um único servidor de base de dados;
- Consistência - significa que sempre que os dados são guardados, todos os que acedem à base de dados irão receber a versão mais recente. Por outras palavras, ao existirem vários nodos num *cluster*, os dados necessitam de estar consistentes, isto é, os dados no nodo A devem estar consistentes com os dados do nodo B. Então, para satisfazer a consistência deve assegurar-se que ao existirem dois pedidos para a mesma informação com destinos diferentes (no mesmo *cluster*), estes devem ter o mesmo resultado;
- Tolerância a falhas - um sistema de base de dados distribuído é tolerante a falhas quando o sistema continua em funcionamento, mesmo que a comunicação entre alguns servidores no mesmo *cluster* esteja temporariamente indisponível.

Figura 15 – Teorema CAP[33]

[33] Retirado do *site*: http://blog.nosqltips.com/

Brewer [Brewer,2000] constatou que em sistemas distribuídos apenas uma das características disponibilidade ou consistência pode ser garantida ao mesmo tempo. Já Pritchett [Pritchett,2008] refere que qualquer estratégia para escalar um sistema horizontalmente é baseada em tolerância a falhas, portanto os *designs* de sistemas são forçados a escolher entre consistência e disponibilidade. Quer isto dizer que, ao criar um sistema distribuído é necessário escolher a característica tolerância a falhas, ficando apenas disponibilidade ou consistência como optativas.

Um dos tópicos fortemente discutidos neste tema é a diferença entre consistência e disponibilidade. Existem algumas dúvidas sobre a diferença entre estas propriedades e as consequências de tal escolha. Seguidamente, apresenta-se um exemplo baseado na descrição feita pelos autores Martin e Pramod [Fowler and Sadalage,2012], comparando e apresentado um exemplo explicativo desta diferença e as consequências que a escolha tem para o modelo de negócios.

Duas pessoas estão a reservar o último quarto disponível num determinado hotel (o Martin que está nos USA e o Pramod que está na Índia). Cada utilizador está aceder a um servidor local no respetivo país, mas se a ligação entre os dois servidores por algum motivo fica inacessível, o que acontece?

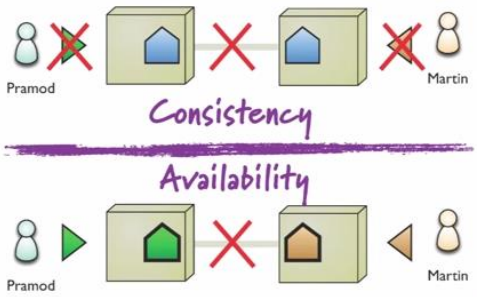


Figura 16 – Exemplo consistência e disponibilidade[34]

Existem dois cenários possíveis de desenlace desta operação. A escolha entre disponibilidade e consistência tem repercussões distintas no comportamento do sistema.

No primeiro cenário, optando pela consistência, como não existe comunicação entre os servidores, não é permitida de maneira alguma a transação, uma vez que, não há a certeza do quarto ter sido reservado previamente e coloca-se o hotel indisponível para reserva.

A segunda possibilidade, ou seja, a disponibilidade, passa por aceitar ambas as reservas, sem ter a certeza se o quarto foi previamente reservado e corre-se o risco do mesmo ter sido reservado por mais do que um cliente.

Cada solução tem as suas vantagens e desvantagens e ambos os cenários são válidos, dependendo das caraterísticas do negócio em questão. Se o modelo de negócios do hotel se sentir confortável com a possibilidade de lidar com a inconsistência das reservas dos quartos e

[34] Retirado de [Fowler and Sadalage,2012]

solucionar este tipo de problemas com a permuta de quartos sem reserva ou mesmo com desistências de clientes, poderá ser uma boa estratégia, uma vez que, ao permitir a disponibilidade do sistema está a permitir que o negócio esteja sempre ativo e que os clientes possam efetuar reservas.

Por outro lado existem casos em que optar pela consistência em vez da disponibilidade é o mais apropriado, como é o caso dos sistemas bancários. Nestes sistemas, que envolvem transações monetárias, faz todo o sentido primar pela consistência, de modo a ter certeza de que o dinheiro apenas foi levantado uma vez.

Concluindo, o aspeto mais importante a reter no que diz respeito à conceção de um sistema distribuído está relacionado com as três propriedades do teorema CAP. Só se poderá satisfazer duas das três características e estas devem ser escolhidas de forma ponderada, não devendo ser decididas apenas pelo programador, mas sim uma decisão de negócio.

3.3 ACID

ACID, acrónimo de atomicidade, consistência, isolamento e durabilidade (do inglês *Atomicity, Consistency, Isolation, Durability*) é um conjunto de características que definem as transações usadas pelas tradicionais bases de dados relacionais. Apesar das inúmeras vantagens que apresentam, as características ACID tornam a leitura e a escrita da informação um pouco mais lentas. Por fim, o modelo normalizado e as garantias ACID são o motivo das bases de dados não serem escaláveis horizontalmente [Hecht and Jablonski,2011].

Cada propriedade acima referida é caraterizada por:

- *Atomicity* - todas as transações devem ocorrer de forma correta, caso contrário, não é efetuada qualquer alteração;
- *Consistency* - para a informação permanecer consistente é necessário que esta passe por validações na base de dados, como por exemplo tipo de dados;
- *Isolation* - uma transação não afetará outra transação que está a decorrer;
- *Durability* - uma vez armazenada a informação com sucesso, essa permanece sem erros e sem falhas.

Como descrito anteriormente, estas propriedades fazem parte do modelo de base de dados relacionais, e pelas suas características devem fazer parte de qualquer tipo de base de dados. Contudo, a maior parte das bases de dados NoSQL não implementam as propriedades ACID, em detrimento das propriedades BASE apresentadas seguidamente.

3.4 BASE

Os programadores e pessoas influentes no seio das bases de dados perceberam a necessidade de criar bases de dados distribuídas. Pritchett questiona [Pritchett,2008]: se ACID oferece consistência para bases de dados distribuídas, então como é possível escolher a disponibilidade?

A resposta é BASE (*Basically Available, Soft State, Eventually Consistent*), em português, disponibilidade básica, estado simples, eventualmente consistente.

As características de um sistema BASE são as seguintes:

- *Basically Available* - indica que o sistema garante a disponibilidade apresentada no teorema CAP. Chapple [Chapple,2012] refere que as bases de dados NoSQL se centram na disponibilidade de dados, mesmo na presença de erros, através da abordagem altamente distribuída;
- *Soft State* - indica que o estado do sistema se altera ao longo do tempo, mesmo sem a necessidade de intervir. Chapple [Chapple,2012] expõe que as bases de dados BASE abandonam por completo os requisitos ACID e que a consistência dos dados é um problema do programador e não deve ser tratado pela base de dados;
- *Eventually Consistente* - significa que o sistema se torna eventualmente consistente ao longo do tempo. Na eventual consistência, Chapple [Chapple,2012] explica que esta é a única exigência que as bases de dados NoSQL tem sobre a consistência. Esta obriga que em algum momento os dados estarão num estado consistente.

O modelo BASE não é adequado para todas as situações, mas é certamente uma alternativa flexível ao modelo ACID no que toca a base de dados. Pritchett [Pritchett,2008] cita que BASE é completamente oposta a ACID. ACID é pessimista, forçando que todas as transações são completamente efetuadas com sucesso. Já BASE é otimista e aceita que a consistência da base de dados estará eventualmente num estado de fluxo. Apesar de parecer impossível lidar com a eventual consistência, na realidade, BASE é bastante flexível e concede níveis de escalabilidade que não poderiam ser obtidos com ACID.

3.5 Big Data

Quando se fala de sistemas de base de dados é inevitável falar do movimento *Big Data*. É um tema relativamente atual e esta relacionado com as bases de dados NoSQL. *Big Data* é um termo usado para descrever grandes volumes de dados que os sistemas de bases de dados nos dias de hoje lidam constantemente. Ao longo do tempo, a informação foi crescendo para volumes nunca antes visto. Por consequência, as empresas no ramo das redes sociais e *Big Data* como é o caso do Facebook, Yahoo, Google e Amazon foram as primeiras a percecionar que os modelos relacionais não eram uma boa solução para grandes volumes de dados [Reeve,2012].

O termo *Big Data* aplica-se a informações que não podem ser processadas ou analisadas por meio de processos ou ferramentas tradicionais [Zikopoulos and Eaton, 2012].

Na imagem abaixo, retirada do *white-paper* [CouchBase,2014b] é possível visualizar o grande crescimento da informação não estruturada/semi-estruturada, comparativamente com a informação estruturada. Mais de 80% da informação gerada nos dias de hoje é caracterizada por não estruturada ou semi-estruturada [CouchBase,2014b] e [Zikopoulos and Eaton, 2012].

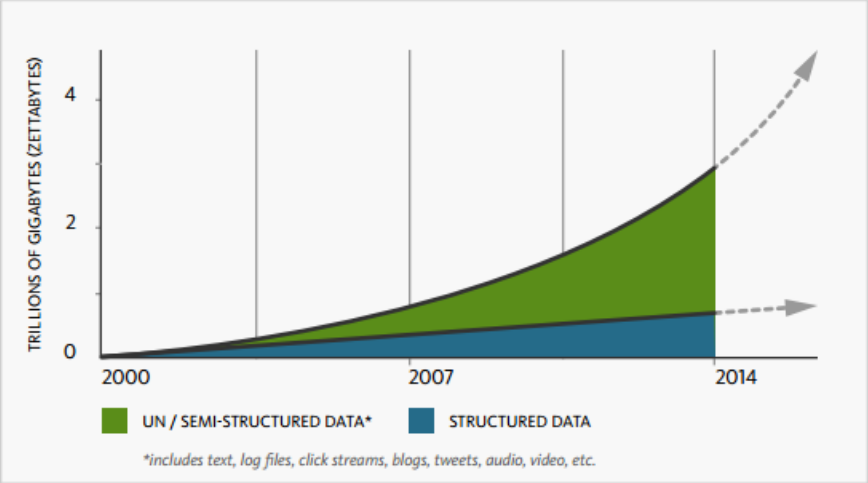


Figura 17 – Crescimento da informação estruturada vs. não estruturada/semi-estruturada[35]

Este volume de informação impulsionou-se devido a determinadas razões, tais como:

- Custo reduzido de dispositivos com capacidade de armazenamento;
- Dispositivos móveis que vieram possibilitar novos e interessantes tipos de informação: localização, preferências de utilização, publicidade direcionada, etc;
- Virtualização de máquinas e *Cloud Computing* vieram dinamizar e responder a necessidades de armazenamento e processamento adicional em picos de utilização;
- *Internet of Things* trouxe novos dispositivos com capacidade de comunicação e captação de variados tipos de dados;
- Bases de dados NoSQL nascem para acomodar grandes volumes de informação e com a possibilidade de escalabilidade em mente.

Em suma, um dos grandes responsáveis pelo surgimento da tecnologia de base de dados NoSQL foi o movimento *Big Data*, já que era necessário encontrar uma solução para o armazenamento de grandes volumes de informação. O mais importante era encontrar uma ferramenta que permitisse analisar e pesquisar, de forma eficaz e eficiente, com o intuito dessa informação estar acessível.

3.6 Internet of Things

Internet of Things representa a próxima evolução da internet, interligando dispositivos numa rede privada ou pública e recorrendo ao uso de RFID (*Radio-Frequency Identification*) e WSN (*Wireless Sensor Networks)*. O principal objetivo passa por monitorizar e partilhar dados e, posteriormente, converter esses dados em informação que permita retirar conhecimento e usufruir deste nas mais diversas situações.

[35] Retirado de [CouchBase,2014b]

Para Barret, a palavra "*thing*" representa objetos, máquinas, veículos, edifícios, animais e até mesmo pessoas. Para fazer parte da *Internet of Things* deve preencher determinados requisitos: ser reconhecido com um identificador único na internet, possuir habilidade para comunicar e monitorizar (através dos seus sensores) e opcionalmente ser controlado remotamente [Barrett,2012].

As vantagens e aplicações destes tipos de sistemas são inúmeras: desde simples monitorizações de objetos em casa, aplicações médicas para monitorização de doentes, aplicações em cidades para tráfego, eficiência energética, localização de dispositivos, etc.

De acordo com [CouchBase,2014b] e [Turner, Reinsel, Gantz and Minton,2014], a quantidade de dados gerados por máquinas está a aumentar com a proliferação de telemetria digital. Em 2020, 32 biliões de "coisas" estarão conectadas à internet; em 2020, 10% dos dados serão gerados por sistemas embebidos (*vs.* 2% à data da dissertação); em 2020, 20% *target rich data* são gerados por sistemas embebidos (*vs.* 8% à data da dissertação). A denominação *target rich data* é geralmente associada a informação IT, incluindo metadados. Estes dados vão continuar a crescer em paralelo com o projeto *Big Data* [Turner, Reinsel, Gantz and Minton,2014].

Já segundo a investigadora Berthelsen, os tradicionais sistemas relacionais de gestão de base de dados continuarão a ter um papel importante no movimento *Internet of Things*, ao processar conjuntos de dados altamente uniformes e estruturados em sistemas relativamente isolados [Berthelsen,2014]. Refere ainda que, ao longo do tempo, as bases de dados necessitarão de níveis de flexibilidade, agilidade e escalabilidade já que é necessário proceder à gestão de dados heterogéneos, gerados por milhões e milhões de sensores/dispositivos, cada um com a sua própria estrutura de dados e potencialmente conectados à internet. Nestes ambientes, os sistemas de base de dados NoSQL estão a provar o seu valor. Para enfrentar estes desafios, empresas inovadoras baseiam-se em tecnologias NoSQL possibilitando o acesso simultâneo à informação de milhões de "coisas" conectadas, o armazenamento de biliões de pontos de dados e atender aos requisitos de infra-estrutura e operações de missão de desempenho crítico [CouchBase,2014b].

Este movimento relaciona-se com o projeto de dissertação, uma vez que, são utilizados na monitorização de crianças dispositivos com as características referidas. É necessário recolher e filtrar essa informação, para a apresentar ao utilizador de forma objetiva.

3.7 NoSQL

NoSQL ou "*not only SQL*" é uma denominação para bases de dados não relacionais, realçando o facto de não usar SQL como linguagem de interrogação à base de dados. Para além dessas características, as bases de dados NoSQL também se destacam por ser *cluster-friendly*, *schemaless* e *open-source*.

Dentro dos sistemas NoSQL existem vários tipos de base de dados, tais como: base de dados baseadas em documentos "*document*", chave-valor "*key-value*", coluna família "*column-family*"

e grafo "*graph*". A diferença entre os vários tipos de base de dados relaciona-se, sobretudo, com a sua *data-modeling*, ou seja, existem quatro novos tipos de modelo de dados para problemas tipificados.

À data desta dissertação existem 150 sistemas de base de dados NoSQL[36]. Entre estes destacam-se os seguintes:

Tabela 1 – Tipos de bases de dados NoSQL[37]

Modelo de dados	Exemplo base de dados
Key-Value ("*Key-Value Database*")	BerkeleyDB LevelDB Memcached Project Voldemort Redis Riak
Document ("*Document Databases*")	CouchDB MongoDB OrientDB RavenDB Terrastore
Column-Family ("*Column-Family Stores*")	Amazon SimpleDB Cassandra HBase Hypertable
Graph ("*Graph Databases*")	FlockDB HyperGraphDB Infinite Graph Neo4J OrientDB

3.7.1 Caraterísticas das bases de dados NoSQL

Independentemente do tipo de base de dados NoSQL, estas são conhecidas por implementar determinadas características: *schemaless* e APIs.

3.7.1.1 *Schemaless*

Schemaless, também designada por *schema-free*, em português, livre de arquitetura, significa que a informação não segue uma estrutura rígida, não há uma arquitetura de dados para o armazenamento, ao contrário dos modelos relacionais, que recorrem a uma estrutura rígida, resultado da normalização, de forma a diminuir a redundância e inconsistência dos dados. Por outras palavras, os sistemas de base de dados relacionais armazenam a informação seguindo uma estrutura definida, desde o tipo de dados, relações entre tabelas, etc.

[36] http://nosql-database.org/
[37] Adaptado de [Fowler and Sadalage,2012]

Num sistema de base de dados com *schemaless*, a base de dados torna-se mais tolerante ao erro, com o principal objetivo de armazenar a informação independentemente da sua estrutura, o que consequentemente torna a base de dados mais disponível.

De acordo com [Fowler and Sadalage, 2012], o termo associado ao NoSQL foi acidental, uma vez que o *schema* está implícito. De certa forma, este termo é verdadeiro quando se pretende armazenar informação, como por exemplo um documento. Este *schema* implícito é transferido para a aplicação, o que implica que o programador conheça a estrutura dos dados, para saber como manipular a informação. Novamente [Fowler and Sadalage, 2012] afirma que, ao passar o *schema* implícito para a aplicação, apesar de à primeira vista ser uma má abordagem, esta característica trás certos benefícios, tais como aumentar a disponibilidade do sistema.

A característica *schemaless* é bastante apreciada pelos programadores no que toca à estrutura do modelo de dados, pois permite, sem grandes obstáculos, alterar o conteúdo da informação, agilizando o processo de desenvolvimento e manutenção do projeto, tarefas bastantes morosas e complexas no modelo relacional.

3.7.1.2 APIs

Os sistemas de base de dados NoSQL implementam APIs para a interação das várias aplicações com esses sistemas. API (*Application Programming Interface*) oferece interfaces para outras aplicações, com o intuito de aceder a certas funcionalidades do sistema implementado pela API. Normalmente uma API é documentada, definindo o *input* e/ou *output* para as aplicações que vão interagir com o sistema saberem os métodos implementados pela mesma.

Na seguinte tabela são apresentados alguns exemplos de APIs disponibilizadas pelos sistemas de bases de dados NoSQL.

Tabela 2 – APIs dos sistemas de bases de dados NoSQL

Sistema de base de dados	**API**
Cassandra	CQL3 & Thrift
CouchBase	HTTP/REST & JSON
CouchDB	HTTP/REST e JSON
MongoDB	Custom/BSON
Neo4J	HTTP/REST (or embedding in Java)
Riak	HTTP/REST e custom binary

3.7.2 Tipos de base de dados NoSQL

Cada sistema NoSQL tem o seu modelo de dados. Existem quatro categorias de base de dados:

- *Key-value*: em português chave-valor, armazena uma única chave com um valor. Este valor pode ter qualquer formato;
- *Column-family*: armazena a informação em colunas famílias, que possuem diversas linhas, que, por sua vez, podem possuir várias colunas;

- *Document*: armazena a informação em documentos XML, JSON, BSON ou formato binário;
- *Graph*: permite guardar entidades e relações entre elas. Entidades são também conhecidas por nodos, que possuem propriedades.

Na imagem que se segue é possível visualizar, tendo em conta o tipo de base de dados, como é que a informação será escalável em tamanho e complexidade:

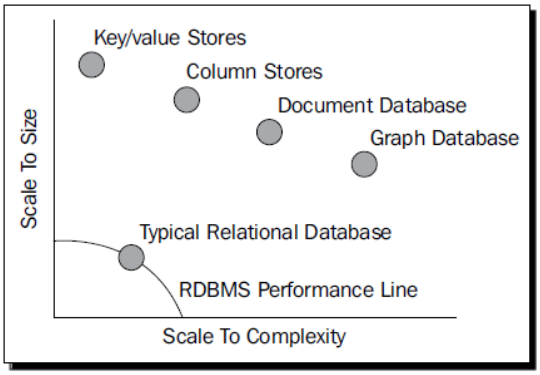


Figura 18 – Aumento do tamanho da informação *vs.* aumento da complexidade[38]

3.7.2.1 Base de dados *key-value*

As bases de dados do tipo *key-value* são as que apresentam a estrutura mais simples de base de dados NoSQL, lembrando aos programadores a ideia de uma *hashtable* ou *dictionary* para o armazenamento dos dados.

Na linguagem de programação Java, uma *hashtable* é constituída por uma *key* e um *value* como no exemplo que se segue:

```
Hashtable<String, Integer> numbers = new Hashtable<String, Integer>();
   numbers.put("one", 1);
   numbers.put("two", 2);
   numbers.put("three", 3);
```

Código 1 – Exemplo de *hashtable*[39]

No código anterior foi definido o valor do tipo *integer* mas poderia ser um objeto, uma lista, um *array*, etc.

Paralelamente às bases de dados relacionais, seria como proceder à criação de uma tabela com apenas duas colunas - *ID* e *VALUE*. Na coluna *ID* armazenava-se a chave e na coluna *VALUE* armazenava-se o valor. Todos os dados seriam armazenados nesta estrutura. Através desta simples estrutura de informação, a informação contida nesta base de dados é *schema-free*, ou

[38]Adaptado do *site*: http://en.citizendium.org/wiki/NoSQL
[39]http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html

seja, livre de esquema na sua arquitetura. Este tipo de estrutura permite efetuar pesquisas de forma mais eficiente [Pokorny,2011].

Na imagem que segue é apresentado um exemplo do modelo *key-value*:

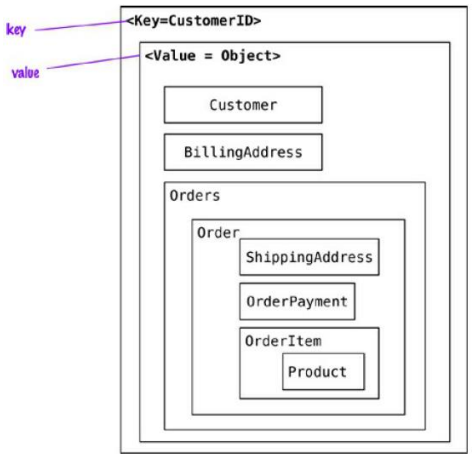


Figura 19 – Exemplo modelo *key-value*[40]

O uso desta nova abordagem aumenta a redundância da informação, contudo as vantagens são superiores às desvantagens, como é o caso do aumento de performance e facilidade de escalabilidade.

3.7.2.2 Base de dados *document*

As bases de dados do tipo *document* armazenam a informação estruturada em documentos do tipo XML, JSON, BSON ou mesmo em formatos binários. Na verdade, este modelo é muito parecido com o modelo de base de dados *key-value* e por vezes até confundido. A principal diferença é uso das vantagens do suporte de tipo de dados, como é o caso do JSON. O uso do JSON ou qualquer tipo de formato de serialização oferece a possibilidade de aproveitar, na maior parte das vezes, o suporte nativo das linguagens de programação em codificar/ descodificar este tipo de informação. Assim é possível usar a estrutura hierárquica de documentos, como é o caso de *maps* e *collections*. A familiaridade deste tipo de tecnologias torna as bases de dados orientadas a documento uma solução apreciada pelos programadores.

Na imagem que se segue é apresentado um exemplo do modelo documento em JSON.

[40] Retirado de [Fowler and Sadalage,2012]

Field	Value
_id	"819cb3f3c27b1de92dfce1f26c0071f1"
_rev	"1-03c9094bea172b21e23502b82cc6e7ca"
array	0 1 1 2 2 3
boolean	true
null	null
number	123
object	a "b" c "d" e "f"
string	"Hello World"

Figura 20 – Exemplo modelo *document*

3.7.2.3 Base de dados *column-family*

Numa base de dados do tipo *column-family,* a informação é armazenada de forma semelhante ao modelo *key-value*. Na verdade, o modelo *column-family* advém do modelo *key-value*, acrescentando uma nova funcionalidade: a coluna. Esta alteração surge devido à necessidade de estruturar a informação em agrupamentos mais funcionais, mantendo a informação relacionada e agrupada.

Por outras palavras, esta funcionalidade oferece a possibilidade de armazenar, dentro do valor, outra linha do tipo chave-valor, passando assim a existir dois níveis de agregação do tipo chave-valor. O segundo nível oferece a possibilidade de descrever o tipo de dados armazenados.

Na imagem que se segue é demonstrado um exemplo de modelo *column-family*:

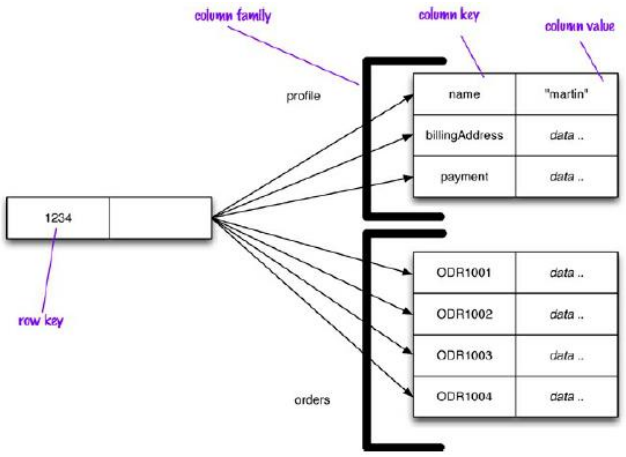


Figura 21 – Exemplo modelo *column-family*[41]

O modelo *column-family* apresenta então dois níveis de agregação. O primeiro é uma linha do tipo *key-value*, onde existe uma chave e um valor, o valor será então a coluna família. No segundo nível, nas várias linhas existe uma *columm-key*, com a denominação da informação e o

[41] Retirado de [Fowler and Sadalage,2012]

column-value com o valor real. Cada coluna deve fazer parte de uma família de colunas. Dessa forma, a informação é estruturada e agrupada logicamente.

3.7.2.4 Base de dados *graph*

As bases de dados do tipo *graphs* são baseadas na teoria dos grafos e permitem armazenar a informação em forma de vértices e ramos. A teoria dos grafos vem solucionar/desmitificar o problema da informação relacionada, com métodos de pesquisa eficazes e bastantes estudados, possibilitando armazenar informação dentro do nodo e as relações entre outros nodos através dos ramos. As bases de dados orientadas a grafos usam o conceito de vértices e ramos, em detrimento de tabelas ou *key-value* para armazenar e representar informação na base de dados [Allen,2010].

Deste modo, as bases de dados vêm solucionar o problema de lidar com pequenos conteúdos de informação mas com uma complexa relação [Fowler and Sadalage,2012]. Este tipo de base de dados é normalmente utilizada onde a informação é bastante relacionada, como é o caso da informação das redes sociais [Allen,2010].

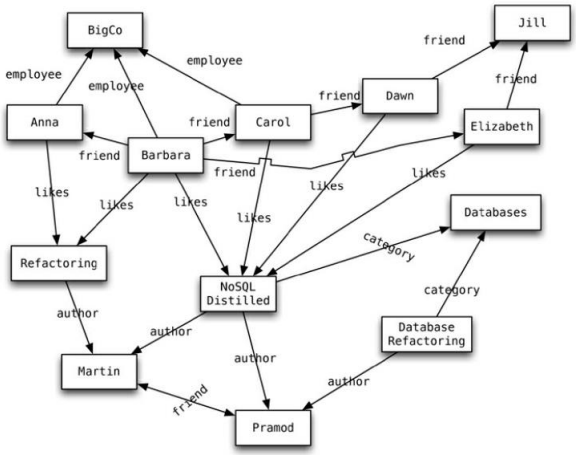


Figura 22 – Exemplo modelo *graph* [42]

3.7.3 Vantagens e desvantagens das bases de dados NoSQL

Genericamente, os sistemas de bases de dados NoSQL, como qualquer outra tecnologia, apresentam vantagens e desvantagens, restando ao gestor do projeto verificar se as vantagens destes sistemas são mais benéficas comparativamente a outros métodos de armazenamento.

A introdução das bases de dados NoSQL trouxe várias vantagens:

- Os dados são escaláveis e flexíveis, permitindo escalar o sistema rapidamente sem grandes alterações;
- Existem novos modelos de dados a considerar, não sendo obrigatório usar sempre o modelo relacional;

[42] Retirado de [Fowler and Sadalage,2012]

- Os tempos de escrita e leitura são mais rápidos comparativamente aos modelos relacionais, principalmente em grandes volumes de dados.

Existem obviamente algumas desvantagens no uso de bases de dados NoSQL, entre as quais:

- Ainda não existem padrões comuns devido à imaturidade da tecnologia e cada modelo de dados atua de maneira diferente;
- Ao existirem modelos de dados alternativos, a escolha da melhor solução pode tornar-se confusa e difícil;
- O não uso do SQL, linguagem fortemente enraizada por sistema de gestão de base de dados;
- As bases de dados NoSQL não garantem o modelo ACID, e por sua vez, não garantem que a informação esteja consistente.

Através da comparação das vantagens e desvantagens do uso de base de dados NoSQL, o ponto mais contraproducente é o não uso das propriedades ACID.

Deve evitar-se o uso de base dados NoSQL em sistemas que envolvam transações ou dinheiro, onde a inconsistência dos dados ou a falta dela é uma falha crítica do sistema. O sistema deve ser capaz de voltar/reverter todas as alterações aquando da ocorrência de uma transação mal sucedida.

Por outro lado, as bases de dados NoSQL podem e devem ser utilizadas quando:

- As aplicações escrevem muita informação na base de dados;
- A informação pode ser facilmente alterada ao longo do tempo;
- A essência da informação é tipicamente não estruturada;
- Se prevê um elevado número de informação.

3.7.4 Conclusões

Uma das tarefas mais importantes do projeto esteve relacionada com a decisão do sistema de base de dados NoSQL a usar no desenvolvimento da plataforma, uma vez que, essa decisão teria um forte impacto no futuro do projeto, a nível de limitações.

Existem diversos sistemas NoSQL com modelos de dados e características distintas, tornando difícil perceber qual é o sistema de base de dados que mais se adequa ao modelo de negócios e casos de uso. Foi por isso necessário analisar os requisitos do sistema a desenvolver e perceber quais seriam as características da informação.

O caso de uso apresentado para o projeto expõe a possibilidade de existirem diversos dispositivos e estes enviarem, com uma determinada cadência, os valores resultantes da monitorização dos sensores. Consequentemente, multiplicando pelos diversos utilizadores resulta um número elevado de pedidos de leitura e escrita na base de dados. Aliado a esse facto, o sistema de base de dados deve dar preferência à característica disponibilidade e tolerância a falhas, de forma a minimizar a informação perdida na indisponibilidade do serviço. Também a

possibilidade do sistema ser escalável no futuro para acomodar mais utilizadores/dispositivos é um fator muito importante para o sucesso do projeto.

Por outro lado, a informação rececionada pela plataforma não será propícia a grandes alterações depois de inserida. Também existe uma grande probabilidade de no futuro se acrescentarem mais sensores o que, naturalmente, origina implementação de novas funcionalidades. O suporte por parte dos sistemas de bases de dados para albergar aplicações móveis foi também tido em consideração.

A imagem que se segue clarificou a caracterização de cada sistema de base de dados no teorema CAP e ainda o modelo de dados correspondente.

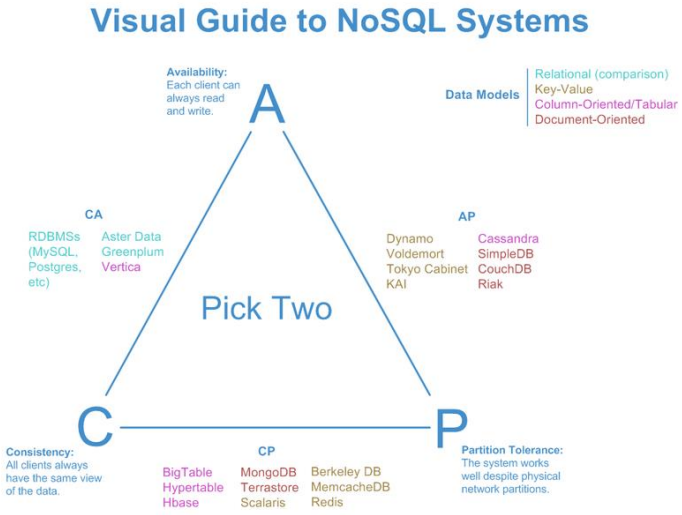


Figura 23 – Guia visual de sistemas NoSQL[43]

Assim, devido aos requisitos apresentados são colocadas de parte as bases de dados relacionais, uma vez que, esses sistemas de base de dados não são indicados para conteúdos de informação dinâmicos, ou seja, propícios de sofrerem alterações na sua arquitetura. Outro problema dos modelos relacionais está relacionado com o número de informação [Jacobs, 2009], uma vez que quando esta aumenta drasticamente, o desempenho do sistema diminui. Também Staveley [Staveley, 2012] refere que os sistemas de base de dados relacionais têm limitações na escalabilidade, já que à medida que a base de dados é distribuída, a complexidade dos *joins* com tabelas que não se encontrem na mesma máquina aumenta. Para além da complexidade, os *joins* e os *locks* influenciam negativamente o desempenho em sistemas distribuídos [Hecht and Jablonski, 2011].

Assim sendo, as bases de dados NoSQL surgem como uma alternativa viável para os requisitos apresentados.

[43] Adaptado do site: http://blog.nahurst.com/visual-guide-to-nosql-systems

Seguidamente foi importante escolher que modelo de dados mais se adequava. O modelo de dados orientado a grafos não seria uma boa solução para o problema, uma vez que, a natureza da informação não seria relacionada do tipo vértice e ramo. Ao descartar esta possibilidade, restam ainda as bases de dados do tipo *key-value*, *document* e *column-family*. Estes três tipos de base de dados denominam-se de *aggregate orientation* [Fowler and Sadalage,2012], o que significa que recorrem ao procedimento de agregação, de forma a operar em *cluster*, facilitando o processo de replicação, *sharding*, o que torna mais fácil trabalhar os dados.

Tipicamente, as bases de dados *key-value* tornam a agregação para a base de dados opaca, em contraste com as bases de dados do tipo *document*, em que a informação é mais legível. A vantagem que *key-value* apresenta relaciona-se com a possibilidade de armazenar qualquer formato de informação. No caso de *document*, há algumas restrições nas estruturas e tipo de dados, que em contrapartida tornam o acesso à informação mais flexível. Um fator importante nas base de dados *document* é que esta possibilita pesquisar tanto pela chave, como filtrar por algum atributo do valor. No caso das bases de dados *key-value* só é possível "olhar" para a chave.

Relativamente às bases de dados do tipo *column-family*, estas são mais indicadas para cenários onde a escrita na base de dados é rara, já que, a performance de escrita é mais baixa comparativamente com os outros tipos. Em contrapartida estas bases de dados são úteis quando é necessário ler várias colunas ou linhas de uma vez, podendo tirar partido de pesquisas mais complexas e dinâmicas [Fowler and Sadalage,2012].

Assim sendo, a opção mais viável passa pela implementação da base de dados NoSQL orientadas ao documento. Juntamente com o estudo efetuado no estado de mercado tecnológico para além do uso de base de dados NoSQL orientada ao documento foi escolhido o sistema CouchDB apresentado mais detalhadamente a seguir.

3.8 CouchDB

CouchDB[44], acrónimo de "*Cluster Of Unreliable Commodity Hardware Data Base*" foi lançado em meados de 2005 por Damien Katz. Tinha como objetivo criar uma base de dados orientada ao documento e tolerante a falhas. Inicialmente o sistema seria desenvolvido na linguagem C++ mas acabou por ser desenvolvido na plataforma Erlang[45] OTP. Foi concebido com base nos princípios *web*, fornecendo variadíssimos mecanismos contra falhas de sistema. É um sistema de base de dados orientado ao documento, fornecendo uma interface RESTful HTTP/JSON API.

CouchDB usa o *slogan* "*Relax"*, devido à sua facilidade de instalação. Em poucos passos é possível instalar um sistema base de dados distribuído NoSQL orientado ao documento passando logo ao desenvolvimento. É possível instalar este sistema num ambiente *cluster* em

[44] http://wiki.apache.org/couchdb/
[45] http://www.erlang.org/

diversos sistemas operativos, desde Linux, Mac OSX, Windows, e até mesmo em dispositivos móveis.

Como referido anteriormente, CouchDB é escrito na linguagem Erlang, uma linguagem desenvolvida pela Ericsson para suporte de aplicações distribuídas e tolerante a falhas, focada para ambientes em tempo real e processamento paralelo. CouchDB foi projetado como sendo "*shared nothing*", ou seja, cada nodo no *cluster* funciona de forma independente e replica a informação pelos pares do cluster, para que assim numa eventual falha individual não comprometa o funcionamento do *cluster*. CouchDB escolheu as propriedades disponibilidade e tolerância a falha do teorema CAP, mencionado previamente.

O sistema de armazenamento implementa as propriedades ACID. No disco, CouchDB nunca escreve "por cima" dados previamente armazenados, nem em estruturas associadas para garantir que a base de dados está num estado consistente. Durante o processo de leitura, esta nunca é bloqueada, nem necessita de esperar quando são feitas, em simultâneo, operações de leitura e escrita, mesmo sobre o próprio documento. CouchDB usa MVCC (*Multiversion Concurrency Control*). Este modelo de controlo de concorrência de versões tem por objetivo criar cópias consistentes na base de dados, desde o início até ao fim das operações. Os documentos são indexados em *b-trees*, através do identificador do documento (*_id*) e revisão (*_rev*). Sempre que o documento é alterado ou eliminado é gerada uma nova revisão incremental do documento na base de dados.

CouchDB é um sistema de base de dados distribuído *peer-based*, permitindo aos utilizadores e servidores aceder e editar os dados partilhados em modo *offline*, para mais tarde replicar (bidirecional) essas alterações pelos vários nodos.

Características como modelo de dados orientado ao documento, *views*, segurança, modelo de replicação bidirecional, linguagem de consulta para propósitos específicos, *layout* do disco eficiente e robusto, natureza concorrente da plataforma Erlang conceberam um sistema confiável e eficiente.

3.8.1 Futon

CouchDB oferece uma interface *web* nativa denominada Futon. Uma vez instalado o CouchDB e em execução na própria máquina, é possível aceder através do endereço "http://localhost:5984/_utils/" no *browser* à ferramenta Futon, apresentada na imagem a seguir.

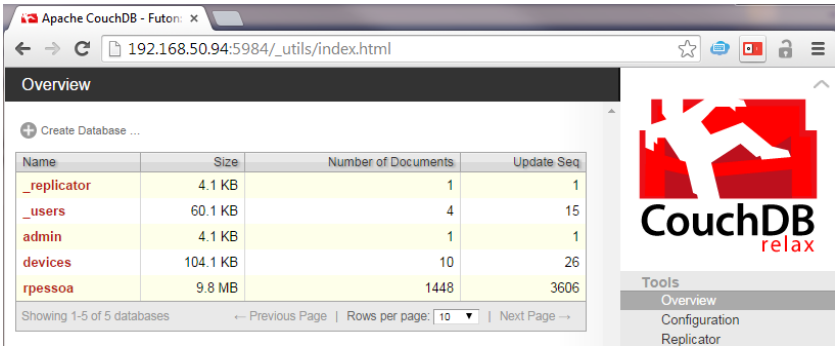


Figura 24 – CouchDB - Futon

Através da ferramenta Futon é possível configurar todo o sistema de forma intuitiva e simples. Na primeira utilização aparecerá uma notificação "*Welcome to Admin Party! Everyone is admin. Fix this!*" alertando que é necessário, por motivos de segurança, adicionar administradores e as suas permissões no sistema.

Outro aspeto importante, como é possível constatar na imagem anterior, é a existência de uma base de dados, por definição "*_users*". Nesta base de dados são armazenados todos os registos dos vários utilizadores do sistema. Deste modo, é possível configurar as permissões dos utilizadores no sistema, bem como quais as bases de dados a que têm acesso, de forma a implementar mecanismos de segurança.

3.8.2 RESTful HTTP/JSON API

Como foi referido anteriormente, os sistemas de base de dados NoSQL são caracterizados por disponibilizarem uma API para a interação com o próprio sistema. CouchDB não é exceção, disponibilizando a RESTful HTTP/JSON API.

A interação das várias aplicações com o sistema de base de dados CouchDB é efetuada através da arquitetura RESTful juntamente com o protocolo HTTP. É um estilo de arquitetura de *software* que descreve os seus métodos/serviços na *web*. Através do URI são identificados os recursos que se pretendem aceder/gerir usando os métodos *GET*, *POST*, *PUT* e *DELETE*. Quando se pretende aceder a um documento recorre-se ao método *GET*; para inserir um novo documento é usado o método *POST*; para editar um documento, o método *PUT* e, por fim, para eliminar um documento, o comando *DELETE*. CouchDB usa o paradigma CRUD (*Create, Read, Update and Delete*), já que se trata de um sistema de armazenamento persistente de informação.

Na tabela que se segue é possível ver um exemplo REST e HTTP:

Tabela 3 – Exemplo REST e HTTP

Recursos	***GET***	***PUT***	***POST***	***DELETE***
http://localhost/ collection	Lê/*read* uma lista de itens dentro da *collection*	Altera/*update* uma coleção dentro da *collection*	Cria/*create* uma nova *collection*	Elimina/*delete* a *collection*
http://localhost/ collection/abc123	Lê/*read* detalhe do item abc123 dento da *collection*	Altera/*update* o detalhe do item abc123 dento da *collection*	Cria/*create* um novo objeto acb123	Elimina/*delete* abc123 da *collection*

3.8.3 cURL

CouchDB também possibilita uma interação através do comando cURL. Através deste, é possível efetuar pedidos HTTP diretamente na linha de comandos para o CouchDB. Para isso, na *shell* podemos usar "curl -x" seguido do método HTTP (*GET*, *POST*, *PUT* ou *DELETE*). Não descrevendo o método a usar pelo comando cURL, por defeito é usado o método *GET*.

No exemplo seguinte consta um pedido *GET* ao servidor CouchDB, retornando o estado atual do sistema, versão e sistema operativo.

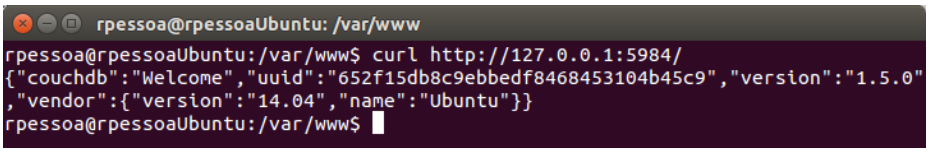


Figura 25 – Comando cURL no CouchDB

3.8.4 JSON

CouchDB usa JSON (JavaScript *Object Notation*) para armazenar os seus documentos e responder aos pedidos. JSON é conhecido pela portabilidade, compatibilidade com as várias linguagens de programação e tamanho reduzido comparativamente com o seu principal concorrente - XML (*Extensible Markup Language*).

Um objeto JSON é constituído por uma coleção de pares *key-value*. O valor de uma chave pode ser número, *string*, booleano (*true* ou *false*), *arrays* (como por exemplo ["Ricardo", "Inês", "Nomes..."]), *null* ou mesmo outro objeto JSON.

CouchDB reserva dois atributos *_id* e *_rev* para definição de um documento. O *_id* é um identificador único do documento, não podendo existir dois documentos iguais na mesma base de dados. Se o programador não atribuir um identificador ao documento, o CouchDB criará automaticamente um UUID. O atributo *_rev* é usado para controlo de versão do documento MVCC (*Multiversion Concurrency Control*). Recorrendo a este atributo, CouchDB sabe qual é a versão mais recente. Sempre que o documento é alterado o campo *_rev* é incrementado.

3.8.5 MapReduce

É importante realçar no CouchDB a funcionalidade MapReduce. Como o nome indica, MapReduce é a capacidade de mapear (*map*) uma série de documentos e reduzir/sumarizar (*reduce*) a informação com um determinado propósito. MapReduce é um padrão inspirado nas linguagens de programação, com o objetivo "pegar" em grandes conteúdos de informação e oferecer um processo de organização entre múltiplas máquinas de um *cluster*, quando são requeridas durante uma pesquisa/pedido [Fowler and Sadalage,2012].

Na imagem que se segue é possível visualizar, do lado esquerdo o funcionamento do *map* e do lado direito o funcionamento da funcionalidade *reduce*.

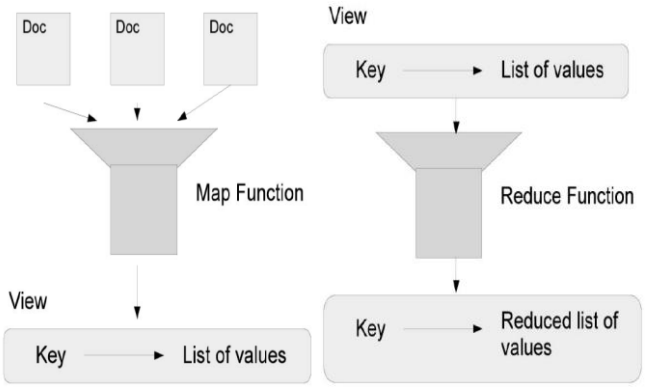


Figura 26 – Conceito MapReduce[46]

Todas as bases de dados *document* e *column-family* disponibilizam uma *framework* MapReduce que permite efetuar pesquisas/cálculos paralelos, em grandes volumes de dados e em várias máquinas [Hecht and Jablonski,2011]. Hecht e Jablonski referem ainda que os processos MapReduce são escritos em baixo nível de abstração, dificultando rescrever processos de forma personalizável.

À medida que os documentos são criados, atualizados e excluídos, CouchDB é suficientemente inteligente para atualizar apenas os documentos modificados no processo de MapReduce [Holt,2011]. Segundo Holt, apesar de se tratar de uma ferramenta bastante poderosa, não deve ser encarada como uma solução para todos os problemas. A indexação gerada pelo *map* é um mapa unidimensional e *reduce* não deve gerar grandes volumes de dados, já que há hipótese de degradar o desempenho do sistema.

MapReduce, mais especificamente no CouchDB, são funções escritas em JavaScript, que definem o comportamento de uma determinada *view*. Ao definir a *view* é facultativa a utilização da função *reduce*.

CouchDB implementa algumas funções *reduce* nativamente, apresentadas na seguinte tabela:

[46] Retirado de [Chandler, 2009]

Tabela 4 – Funções nativas *reduce* do CouchDB

Função *reduce*	Resultado
_count	Retorna o número de valores mapeados no conjunto;
_sum	Retorna a soma do conjunto mapeado;
_stats	Retorna estatísticas numéricas dos valores mapeados, incluindo soma, contagem, mínimo e máximo.

Também é possível escrever funções *reduce* personalizáveis, com intuito de responder a pedidos mais específicos, mas são necessários alguns cuidados, pois poderá reduzir a performance do pedido.

3.8.6 *Views*

As *views* possibilitam indexar e consultar os documentos no sistema CouchDB [Anderson, Lehnardt and Slater,2010]. De acordo com o *site* CouchDB Wiki[47]*, design documents* é um tipo especial de documento constituído por uma ou mais *views*. As *views* são definidas usando a linguagem JavaScript. Através das *views* é possível, filtrar documentos e apresentá-los de forma ordenada, construir índices eficientes para encontrar documentos através de uma determinada chave ou mesmo valor percentente à estrutura do documento e também usar indexação para representar a relação entre documentos, entre outras.

Ao criar uma *view*, procede-se à configuração da função MapReduce, apresentado anteriormente.

CouchDB possibilita criar *views* temporárias, através do Futon, de forma a facilitar o desenvolvimento e teste à mesma. As *views* temporárias só devem ser utilizadas durante o desenvolvimento, uma vez que, são bastante lentas na presença de vários documentos. Assim, é importante gravar as *views* e, posteriormente, torná-las permanentes com o intuito de obter a melhor performance possível.

A funcionalidade *map* tem como objetivo transformar um documento original numa estrutura do tipo *key-value*. Na implementação da *view*, pode escolher-se a *key* para mapear o *output* e o valor associado a chave.

Para criar uma *view* através do Futon é necessário selecionar "*temporary view*" no canto superior direito na *drop-down list* denominada *view*. Por omissão, surge uma estrutura básica de uma *view* como no seguinte excerto de código.

```
function(doc) {
  emit(null, doc);
}
```

Código 2 – Exemplo *view* por omissão

[47] http://wiki.apache.org/couchdb

No código anterior é possível visualizar na função a receção do parâmetro *doc*. Este atributo é a representação do objeto JSON do documento, que será invocado uma vez para cada documento na base de dados. O método *emit* é onde o mapeamento acontece, permitindo personalizar o *output key-value*. O *emit* tem como parâmetros dois argumentos: o primeiro é a *key* e o segundo argumento é o valor associado a essa chave. Se executar esta *view*, obtêm-se todos os documentos presentes na base de dados, uma vez que, não se efetuou nenhum tipo de filtragem de mapeamento.

A seguir é apresentado o retorno da *view* temporária.

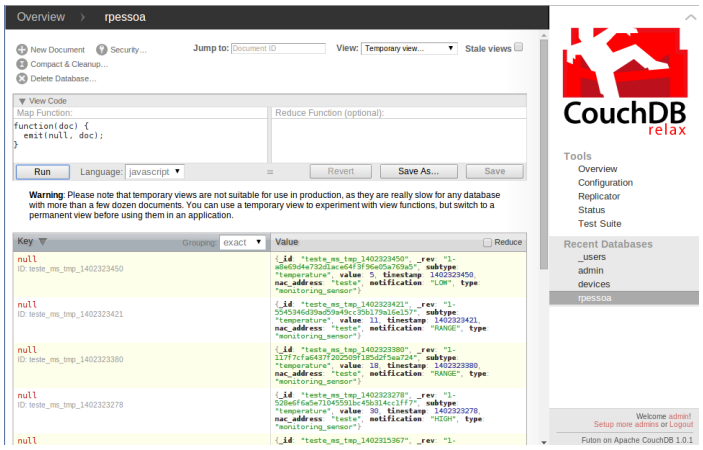


Figura 27 – *Output* da *view* temporária

Na *view* temporária, apesar de não ter sido definida nenhuma chave, *key* a *null*, por definição, a chave também pode ser indexada pelo *_id* do documento. Deste modo, qualquer valor pode ser indexado por uma chave definida e também pelo *_id* do documento.

Para uma melhor compreensão das potencialidades das *views*, apresenta-se um dos exemplos de *views* utilizada no projeto, a *view* getSensors com o seguinte código:

```
function(doc) {
if(doc.sensors){
for(var i in doc.sensors)
  emit(doc._id,doc.sensors[i]);
 }}
```

Código 3 – *View* getSensors

O objetivo desta *view* é retornar para cada dispositivo, os sensores associados ao mesmo. Logo, a primeira condição do documento é ser do tipo "*device*" e possuir um *array* de sensores *(sensors)*. Depois de satisfeita essa condição é percorrido esse *array* e através do método *emit*, é possível mapear o resultado com o *_id* do documento, que neste caso, será o *mac address*, e retornar o conteúdo do *array*. Esta *view* quando executada no Futon tem o seguinte resultado:

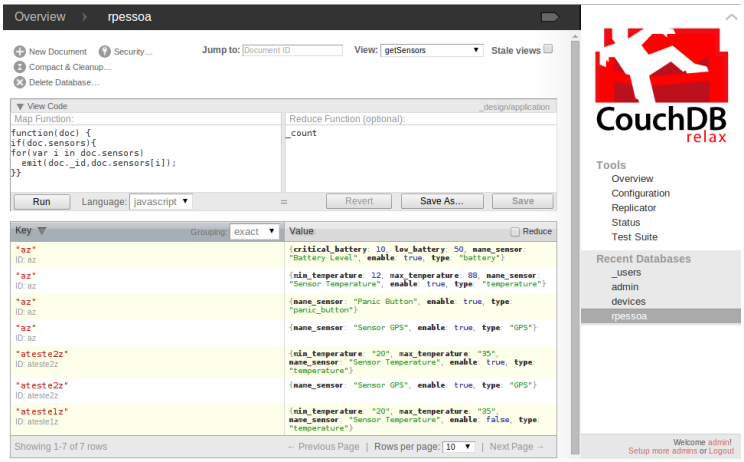


Figura 28 – Resultado da *view* getSensors Futon

Por outro lado também é possível efetuar o pedido através do terminal e do comando cURL:

```
rpessoa@rpessoaUbuntu: ~
rpessoa@rpessoaUbuntu:~$ curl http://127.0.0.1:5984/rpessoa/_design/application/_view/get
Sensors?reduce=false
{"total_rows":7,"offset":0,"rows":[
{"id":"ateste1z","key":"ateste1z","value":{"min_temperature":"20","max_temperature":"35",
"name_sensor":"Sensor Temperature","enable":true,"type":"temperature"}},
{"id":"ateste2z","key":"ateste2z","value":{"min_temperature":"20","max_temperature":"35",
"name_sensor":"Sensor Temperature","enable":true,"type":"temperature"}},
{"id":"ateste2z","key":"ateste2z","value":{"name_sensor":"Sensor GPS","enable":true,"type
":"GPS"}},
{"id":"az","key":"az","value":{"critical_battery":"15","low_battery":"25","name_sensor":"
Battery Level","enable":true,"type":"battery"}},
{"id":"az","key":"az","value":{"min_temperature":"20","max_temperature":"35","name_sensor
":"Sensor Temperature","enable":true,"type":"temperature"}},
{"id":"az","key":"az","value":{"name_sensor":"Panic Button","enable":true,"type":"panic_b
utton"}},
{"id":"az","key":"az","value":{"name_sensor":"Sensor GPS","enable":true,"type":"GPS"}}
]}
```

Figura 29 – Resultado da *view* getSensors cURL

Na figura anterior, ao fazer o pedido da *view* é necessário colocar também *reduce=false*, para retornar apenas o *map* da *view*. Esta *view* possibilita também fazer *reduce* do *output*. Neste exemplo foi definido o uso da função *_count*, com o objetivo de retornar o número de sensores que cada dispositivo possui. Na imagem que se segue é possível ver o resultado do uso do *reduce* na *view*.

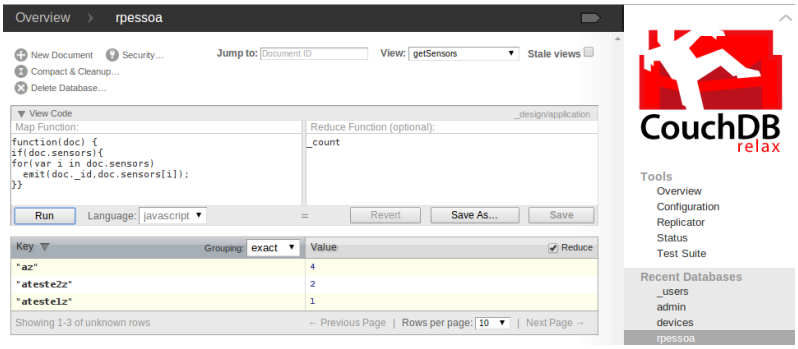


Figura 30 – *View* getSensors com *reduce*

4 Levantamento de requisitos

Face à importância que o levantamento de requisitos tem na elaboração de um projeto, segue-se um capítulo reservado a este tema, apresentando-se o processo levado a cabo na definição de todos os requisitos da plataforma. É essencial que esta tenha uma estrutura sólida em termos de interface, robustez e desempenho. No caso do servidor de base de dados, é necessário que este consiga responder aos requisitos e oferecer o melhor serviço e suporte tanto para o *back-end*, aplicação Android ou qualquer tipo de aplicações futuras.

No projeto de dissertação pressupõe-se o desenvolvimento de um *back-end* e aplicação *mobile* para o sistema Android. Uma vez que o projeto foi desenvolvido de raiz foi impossível calcular com exatidão, o número de utilizadores interessados no uso deste tipo de plataformas e serviços em Portugal. Expecta-se, no entanto, que o número de utilizadores interessados neste tipo de soluções, ou seja, monitorização de dispositivos cresça nos próximos anos. Independentemente do número de utilizadores foram estudados os vários sistemas de base de dados, com objetivo de encontrar um sistema que responde-se aos requisitos de escalabilidade.

Neste capítulo serão apresentados os requisitos funcionais e não funcionais, os principais diagramas de casos de uso e diagramas de sequência.

4.1 Levantamento de requisitos não funcionais

Os requisitos não-funcionais são aqueles que afetam a aplicação em termos de desempenho, segurança, usabilidade, entre outros. Estes requisitos influenciam os requisitos funcionais, através de restrições e adicionam valor e qualidade à aplicação.

4.1.1 Desempenho

Uma das premissas do projeto é que o desempenho tanto da aplicação, como do *back-end* não comprometa a experiência do utilizador no momento da sua utilização, pois o utilizador final

pretende utilizar serviços rápidos e cómodos. Dado que todos os serviços disponibilizados dependem da conexão com a internet para efetuar a troca de informação, o desempenho é influenciado pela qualidade da mesma.

Contudo, a aplicação não deve ser morosa no processamento e sincronização da informação, e esse mesmo processamento deve ser transparente na ótica do utilizador. No caso do *back-end*, este deve ser rápido e conciso nos pedidos de informação e deve apresentá-la ao utilizador. O serviço mais preponderante na performance da plataforma é o servidor de base de dados. Este deve responder aos pedidos de informação, de forma eficaz e eficiente.

4.1.2 Usabilidade

A interface gráfica das aplicações *web* e *mobile* devem apresentar uma boa interação pessoa-máquina, sendo intuitiva e fluida, procurando melhorar a experiência do utilizador. A interface deve adequar-se ao utilizador final, que no geral são utilizadores familiarizados com as novas tecnologias, mas deve também ser de fácil utilização, de modo a ser eficiente no seu propósito final, ou seja, apresentação da informação.

Como tal, a solução deve apresentar uma interface amigável, intuitiva e de fácil compreensão e utilização, garantindo uma boa comunicação entre o utilizador e o sistema. As ações devem ser transparentes, de modo a que o utilizador compreenda todos os seus efeitos.

4.1.3 Confiabilidade

A aplicação móvel e o *back-end* estão dependentes de serviços, e como tal, devem possuir mecanismos de controlo a falhas, nomeadamente falha no acesso à internet, falha de resposta dos serviços ou apresentação de informação inconsistente. Desta forma, no caso de existirem falhas, dependendo de qual se tratar, é importante garantir a contínua execução da aplicação e do *back-end*.

4.1.4 Segurança

Um dos aspetos mais importantes que se esperam em sistemas desta natureza é a segurança dos dados. A plataforma deve ser capaz de efetuar a autenticação do utilizador e a base de dados deve estar protegida contra terceiros. Para isso foram utilizados mecanismos de segurança, tanto no servidor de base de dados CouchDB, como nas aplicações, para proteção da informação face a utilizadores não autorizados.

Outro mecanismo de segurança que pode ser facilmente aplicado é a cópia de segurança periódica, através do mecanismo de replicação do sistema de base de dados CouchDB. Através deste é possível restaurar o sistema para o último ponto de restauro, caso seja necessário, de forma rápida e fácil.

4.2 Levantamento de requisitos funcionais

Nos requisitos funcionais são descritas as principais funcionalidades que o sistema deve disponibilizar. Correspondem, portanto, aos requisitos básicos propostos no projeto. Por outras palavras, representam aquilo que o utilizador espera que o sistema ofereça, atendendo aos propósitos para qual o sistema é desenvolvido.

As funcionalidades a seguir descritas foram continuamente analisadas e refinadas em diversas reuniões de acompanhamento com o orientador externo, com o objetivo de decidir o comportamento que o sistema deveria apresentar, que informação seria exposta e possível de gerir pelos diferentes tipos de utilizadores.

As principais funcionalidades oferecidas ao utilizador *caregiver*/cuidador são as seguintes:

- Registo;
- Autenticação;
- Terminar sessão;
- *Dashboard* com os diversos dispositivos e resumo das últimas monitorizações;
- Visualização dos dispositivos;
- Monitorização em tempo real;
- Gerir dispositivos:
    - Adicionar novos dispositivos;
    - Eliminar dispositivos;
    - Configurar sensores;
        - Definir máximos e mínimos;
        - Definir zonas de segurança;
    - Ativar e desativar sensores.

Relativamente às funcionalidades disponibilizadas ao administrador, estas são as seguintes:

- Autenticação;
- Terminar sessão;
- *Dashboard* com o resumo dos dispositivos ainda disponíveis;
- Gerir dispositivos:
    - Adicionar novos dispositivos no sistema;
    - Eliminar dispositivos;
    - Definir sensores.

Existe uma ligeira diferença na gestão dos dispositivos para cada perfil de utilizador. O administrador adiciona novos dispositivos no sistema, enquanto o *caregiver* apenas associa os dispositivos já existentes à sua lista de dispositivos. Na edição dos dispositivos, o administrador só consegue editar dispositivos não associados, enquanto o responsável pela criança só consegue gerir a configuração dos sensores. Por fim, quanto à eliminação dos dispositivos, o administrador só consegue efetuar essa operação a dispositivos não associados, enquanto o *caregiver* não elimina de verdade o dispositivo, apenas o torna invisível para o próprio.

Por convenção foi decidido o uso do *mac address* definido pelos fabricantes dos dispositivos para identificação do mesmo, uma vez que, era uma forma de assegurar que o identificador do dispositivo era único. Foi também colocado em questão o uso de um *UUID* interno, mais reduzido para a identificação, uma vez que, o *mac address* é bastante extenso e pode causar certas dificuldades e consequentemente levar o utilizador a cometer erros ao adicionar os vários caracteres. Essa abordagem também seria facilmente alterada. Contudo, é utilizado o *mac address* sem o uso do caracter ":" (dois pontos) passando de um endereço convencional como, por exemplo, "12:34:56:78:9a:bc" para "123456789abc".

4.2.1 Casos de uso

Neste subcapítulo são apresentados os diagramas de casos de uso. Após o levantamento de requisitos, é necessário elaborar uma representação da informação de forma mais detalhada e técnica, apresentando o comportamento do sistema e a interação do utilizador com o mesmo. Nos diagramas de casos de uso que se seguem são apresentadas as funcionalidades do sistema, para uma melhor compreensão das características do sistema desenvolvido.

4.2.1.1 Atores do sistema

A plataforma tem quatro tipos de atores do sistema: o primeiro, o administrador do sistema que tem responsabilidades administrativas; em segundo, o *caregiver* o elemento mais ativo na plataforma, sendo responsável pela criança; em terceiro, o sistema que efetua ações na plataforma e, por fim, a criança que tem um papel passivo na plataforma, sendo apenas utilizador dos dispositivos.

Na tabela que se segue é apresentada uma descrição mais detalhada de cada ator:

Tabela 5 – Atores caso de uso

Nome	Descrição
Admin	O administrador tem como responsabilidades: adicionar dispositivos no sistema, podendo também editar e eliminar quando estes ainda não tenham proprietário. Este utilizador define os sensores que cada dispositivo possui no sistema.
Caregiver	O *caregiver* é o responsável pela criança. É o utilizador que pretende monitorizar e receber as notificações. Também é responsável por configurar os valores dos sensores, como por exemplo definir o intervalo de máximo e mínimo da temperatura, definir as zonas de seguranças, etc.
System	O *system* tem como objetivo efetuar a monitorização do dispositivo usado pela criança, filtrar essa informação e armazená-la na base de dados. Caso essa monitorização apresente valores inesperados, notifica o ator *caregiver*.
Child	Este ator é aquele que transporta o dispositivo, e consequentemente é monitorizado pelo sistema.

4.2.1.2 Diagrama de caso de uso “Geral”

A figura seguinte apresenta o diagrama de caso de uso “Geral”. Como o próprio nome indica é o caso de uso genérico que demonstra as principais funcionalidades da plataforma. Estas funcionalidades serão explicadas individualmente de seguida.

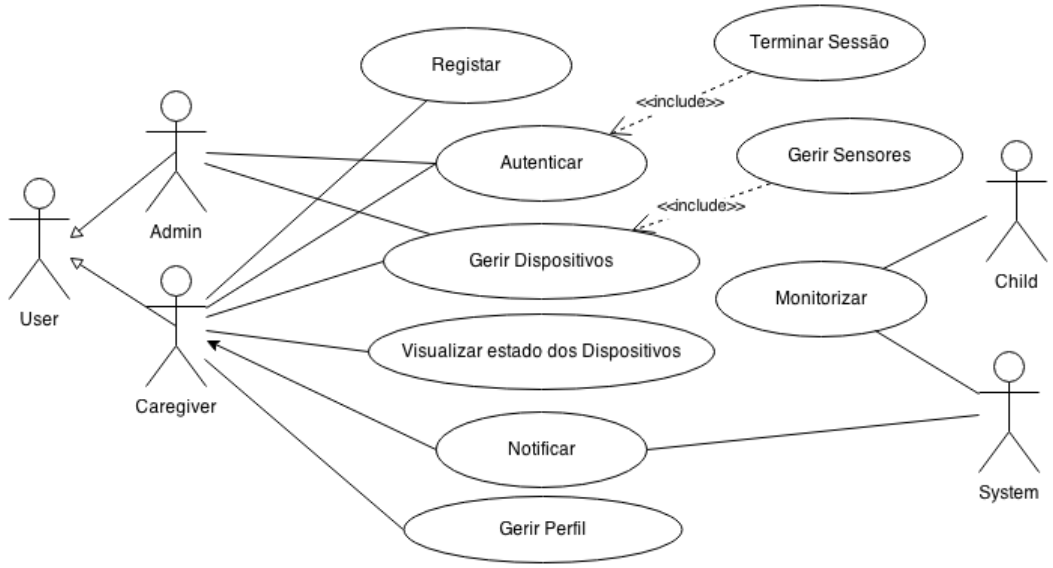


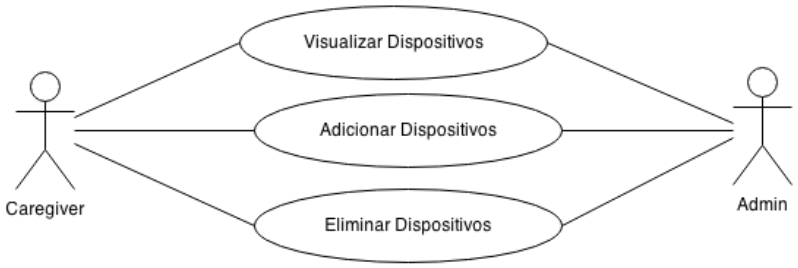
Figura 31 – Diagrama de caso de uso “Geral”

Na tabela que se segue é possível ver a descrição do diagrama de caso de uso “Geral”.

Tabela 6 – Descrição caso de uso “Geral”

UC ID	Nome	Descrição
1	Registar	O *caregiver* para aceder à plataforma terá que se registar através do *back-end* ou da aplicação *mobile*.
2	Autenticar	Para o *caregiver* ou o *admin* acederem e gerirem a informação na plataforma necessitam de efetuar a autenticação. Esta ação carece de registo prévio do UC ID 1, no caso do *caregiver*.
3	Terminar sessão	Depois da autenticação, ou seja, do UC ID 2, o utilizador pode terminar a sessão no *back-end* ou na aplicação *mobile* assim que desejar.
50	Gerir dispositivos	Permite ao utilizador gerir os seus dispositivos (ver diagrama caso de uso gerir dispositivos).
60	Gerir sensores	Permite ao utilizador gerir dentro dos dispositivos, os sensores associados (ver detalhes no diagrama de caso de uso gerir sensores).
4	Visualizar estado dos dispositivos	Permite ao utilizador visualizar na página principal - *dashboard* - todos os dispositivos e notificações dos sensores, bem como visualizar a monitorização em tempo real.
5	Notificar	O sistema notifica o *caregiver* sempre que recebe um valor inesperado.
6	Monitorizar	O sistema monitoriza, através dos dispositivos e sensores, o ator *child*.

4.2.1.3 Diagrama de caso de uso “Gerir dispositivos”



Segue-se a descrição do diagrama de caso de uso “Gerir dispositivos”:

Tabela 7 – Descrição caso de uso “Gerir dispositivos”

UC ID	Nome	Descrição
51	Visualizar dispositivos	O ator *caregiver* pode visualizar os seus dispositivos armazenados. O ator *admin* pode visualizar todos os dispositivos armazenados no sistema, sabendo ainda quem os possui.
52	Adicionar dispositivos	Possibilita ao *caregiver* adicionar um dispositivo novo na sua lista de dispositivos. Este dispositivo tem que existir previamente no sistema. Quem adiciona os dispositivos no sistema é o ator a*dmin*.
53	Eliminar dispositivos	O *caregiver* pode eliminar os seus dispositivos. No caso do *admin*, este só pode eliminar dispositivos que não tenham utilizador associado.

4.2.1.4 Diagrama de caso de uso “Gerir sensores”

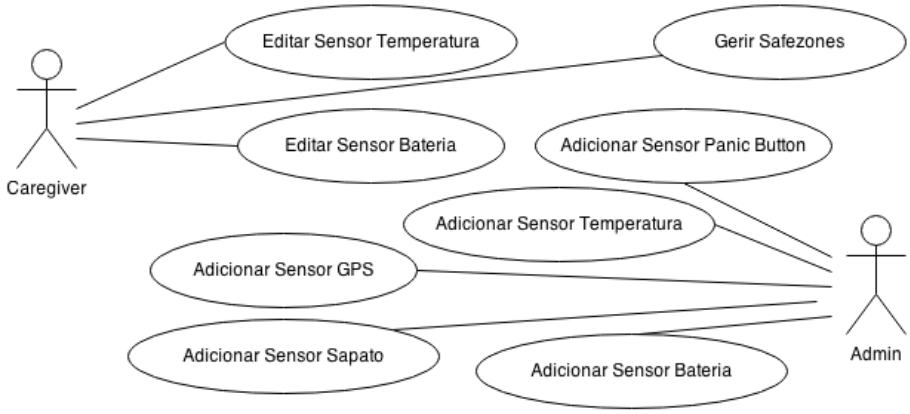


Figura 32 – Diagrama de caso de uso “Gerir sensores”

Na tabela abaixo consta-se a descrição do diagrama de caso de uso “Gerir sensores”.

Tabela 8 – Descrição caso de uso “Gerir sensores”

UC ID	Nome	Descrição
61	Editar sensor temperatura	O *caregiver* pode configurar o limite máximo e mínimo do sensor de temperatura, para emissão de alertas.

70	Gerir *safezones*	Permite ao *caregiver* gerir *safezones* (ver detalhes no diagrama de caso de uso gerir *safezones*).
62	Editar sensor bateria	O nível de bateria também é configurável para receber notificação para níveis baixo e crítico.
63	Adicionar sensor *panic button*	Permite ao administrador adicionar um sensor de botão de pânico a um dispositivo.
64	Adicionar sensor temperatura	Permite ao administrador adicionar um sensor de temperatura a um dispositivo.
65	Adicionar sensor GPS	O administrador poderá adicionar no dispositivo um sensor de GPS.
66	Adicionar sensor bateria	Permite ao administrador adicionar um sensor de bateria a um dispositivo.
67	Adicionar sensor sapato	Possibilita ao administrador adicionar um sensor a um dispositivo capaz de monitorizar um sapato, com o intuito de saber se este foi descalço.

4.2.1.5 Diagrama de caso de uso "Gerir *safezones*"

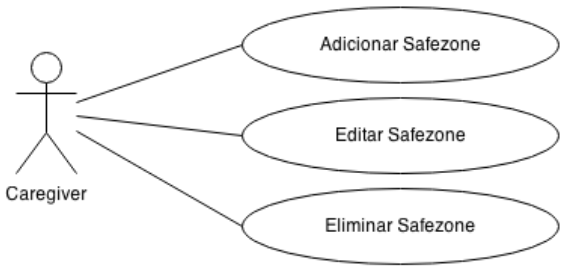


Figura 33 – Diagrama de caso de uso "Gerir *safezones*"

Na tabela seguinte apresenta-se a descrição do diagrama de caso de uso "Gerir *safezones*":

Tabela 9 – Descrição caso de uso "Gerir *safezones*"

UC ID	Nome	Descrição
71	Adicionar *safezone*	Permite ao *caregiver* adicionar uma zona de segurança para o dispositivo.
72	Editar *safezone*	O *caregiver* poderá editar uma zona de segurança sempre que desejar.
73	Eliminar *safezone*	Permite ao *caregiver* eliminar uma zona de segurança.

4.2.2 Diagrama de sequência de sistema

Depois de apresentados os diagramas de casos de uso expõem-se os diagramas de sequência de sistema (SSD, do inglês *System Sequence Diagram*). Com estes é possível visualizar o fluxo de dados das principais funcionalidades do sistema. Serão apresentados apenas os diagramas de sequência de sistema mais relevantes do ponto de vista de funcionamento.

4.2.2.1 SSD Registar

Para o utilizador aceder à plataforma necessita previamente de efetuar o registo. Na imagem que se segue é apresentado o fluxo de informação necessário para o caso de uso "Registar".

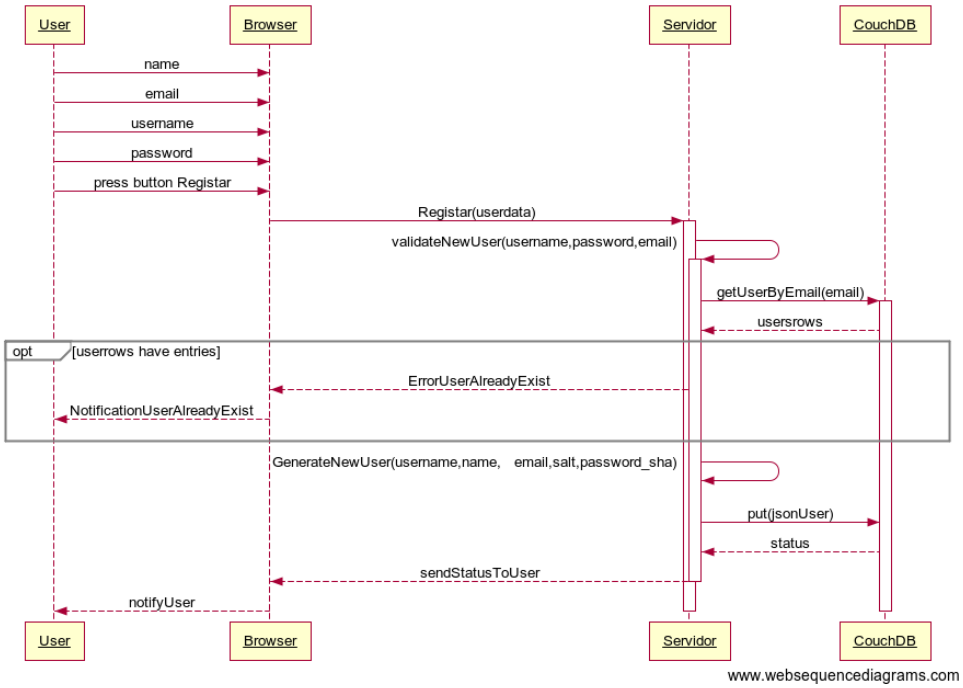


Figura 34 – SSD para o caso de uso "Registar"

Para o armazenamento da palavra-chave do utilizador, de forma segura, foi implementado um mecanismo de segurança. Este mecanismo pressupõe o não armazenamento da *password* em *plaintext*, ou seja, na sua forma original. A *password* em texto simples é combinada com um valor único e aleatório e processado sobre o algoritmo de encriptação SHA-1, uma versão posterior do algoritmo SHA (*Secure Hash Algorithm*). Com este processo é apenas armazenado o valor aleatório *salt* e a encriptação da combinação do valor *salt* com a *password*, denominado na base de dados por *password_sha*.

4.2.2.2 SSD Autenticar

O processo de *login* necessita do *username* e *password* do utilizador, para que este se possa autenticar na plataforma. Através desta autenticação é possível validar se o utilizador já existe e caso exista, e se a *password* corresponder à armazenada na base de dados, o acesso é validado.

Este processo é bastante similar em ambas as plataformas, ou seja, na aplicação móvel e no *back-end*, tendo o seguinte comportamento:

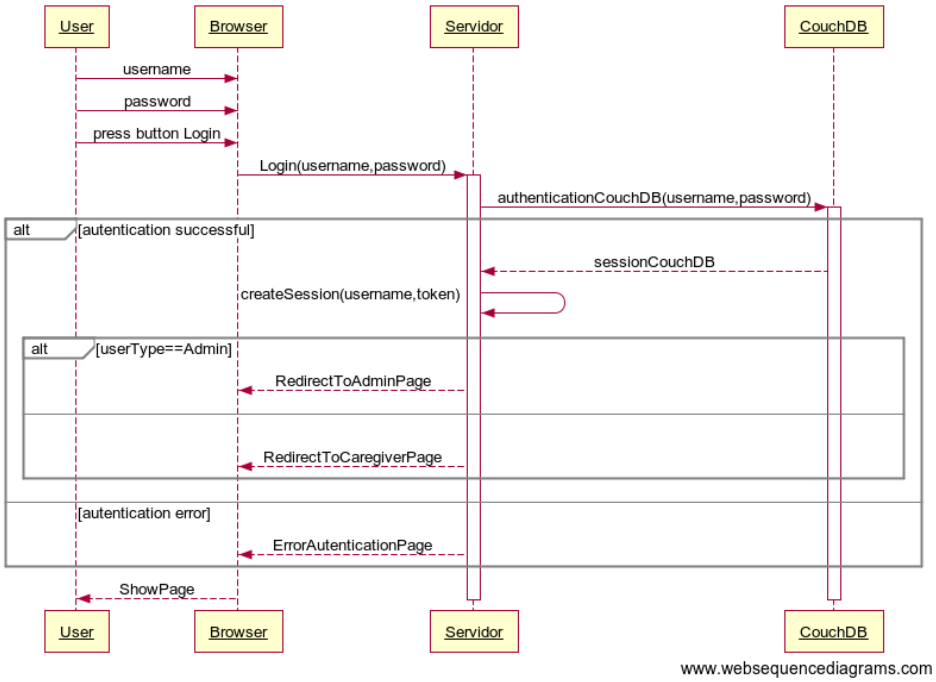


Figura 35 – SSD para o caso de uso "Autenticar"

4.2.2.3 SSD Monitorizar

Outro fluxo de dados cujo funcionamento é importante compreender é o caso de uso "Monitorizar". O ator *child* é monitorizado pelo dispositivo, que envia diretamente a informação para o servidor responsável pela receção, processamento e armazenamento da informação na base de dados. Caso este encontre informação fora dos parâmetros configurados para um determinado sensor, deverá notificar o ator *caregiver* quando autenticado no *browser* ou aplicação móvel. O processo é apresentado no diagrama de sequência de sistema que se segue:

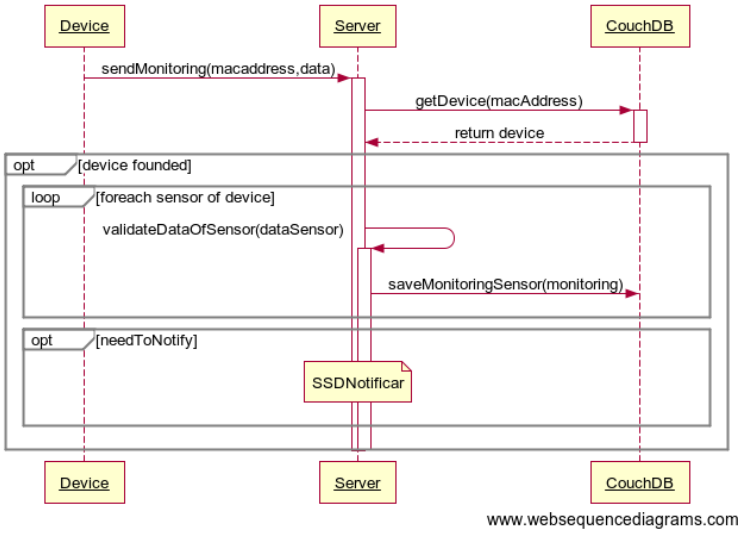


Figura 36 – SSD para o caso de uso "Monitorizar"

4.2.2.4 SSD Notificar

O processo de notificar o ator *caregiver* levado a cabo pelo sistema tem como objetivo a apresentação dos valores anómalos ao utilizador para um determinado sensor. Na imagem que se segue é apresentado o fluxo de dados principal do caso de uso SSD "Notificar".

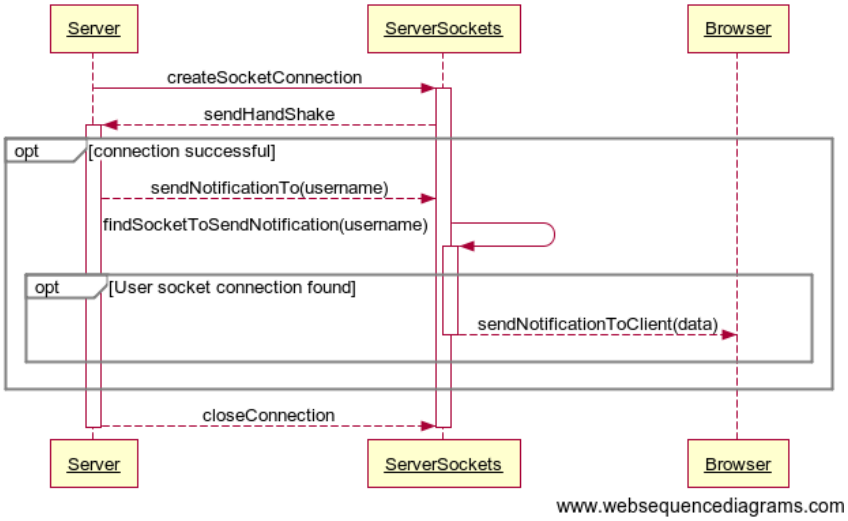


Figura 37 – SSD para o caso de uso "Notificar"

Neste exemplo apenas são apresentadas as autenticações realizadas no *browser*, uma vez que, nos dispositivos móveis o funcionamento é relativamente diferente, não existindo servidor de *sockets* entre a aplicação móvel e o servidor CouchDB. Na aplicação móvel é utilizado diretamente o motor de replicação do CouchDB com a API CouchBase Lite Android para a sincronização da informação e posteriormente a notificação do cliente. No SSD que se segue é apresentado o fluxo de dados do processo de notificação com a aplicação móvel.

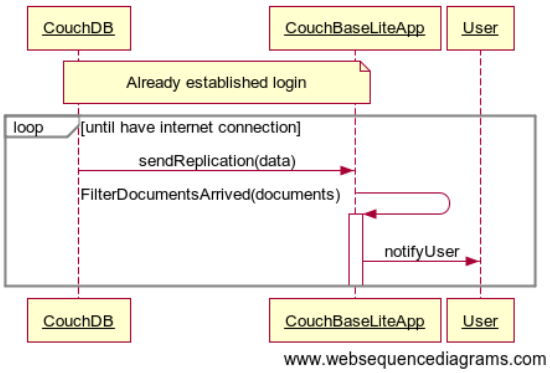


Figura 38 – SSD para o caso de uso "Notificar" (Aplicação Android)

5 Descrição técnica

Este capítulo aborda, detalhadamente, a solução final conseguida, com especial ênfase no aspeto técnico da solução e dos vários elementos que a compõem. Encontra-se dividido em várias secções para apresentar uma estrutura mais organizada. Primeiramente é feita uma breve introdução ao sistema desenvolvido e, posteriormente, será apresentado o desenvolvimento propriamente dito, nomeadamente ao nível do servidor, *back-end* e aplicação.

5.1 Visão geral do sistema

O sistema desenvolvido no âmbito desta dissertação de mestrado foi concebido de raiz, logo foi possível desenvolver uma solução mais objetiva, com o intuito de responder da melhor forma possível aos requisitos inicialmente propostos. Um dos principais requisitos era a necessidade de armazenar grandes volumes de informação já que o sistema teria um número elevado de utilizadores. Assim sendo, era fundamental construir um sistema que fosse escalável no futuro.

Nesse sentido, o período de investigação teve como principal objetivo encontrar um sistema de base de dados que solucionasse, da melhor forma possível, o requisito de escalabilidade. Para além de ser escalável, o sistema deveria também ser *open-source*, de forma a reduzir os custos associados a licenças.

Devido à natureza dinâmica da informação que seria armazenada, foi logo espectável a existência de uma grande probabilidade de alterações futuras na informação, o que consequentemente exigia que a base de dados permitisse o armazenamento da informação o mais flexível possível – *schemaless*. Outra característica necessária para o sistema passava pelo desenvolvimento ágil, devido ao tempo disponível para a concretização do projeto.

O sistema em geral deveria suportar diversos dispositivos com sensores acoplados. Estes dispositivos enviam informação referente ao estado dos sensores para o sistema, que por sua vez, armazena a informação na base de dados. Para efetuar a receção da informação, o sistema

teria que oferecer serviços através de *webservices*. Além dos dispositivos, o sistema teria que disponibilizar um *back-end* e serviços para a plataforma *mobile*.

5.2 Diagrama de sistema

Na imagem que se segue é apresentado o diagrama de sistema:

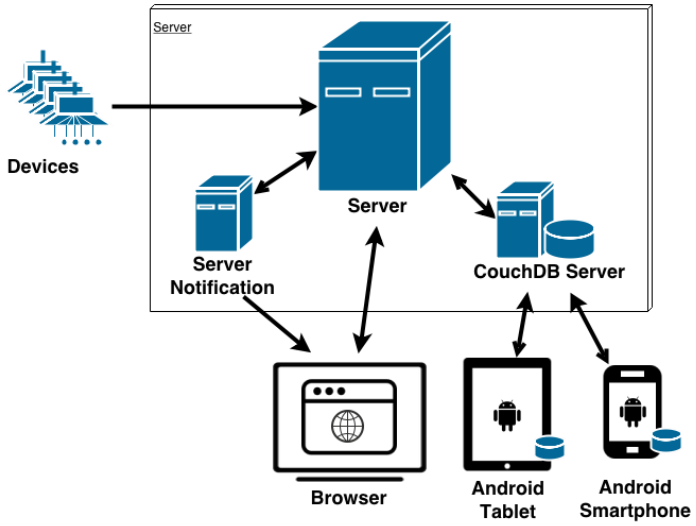


Figura 39 – Diagrama de sistema

Os dispositivos têm como objetivo monitorizar, através dos seus sensores e enviar por GPRS a informação para o servidor. Para isso recorrem aos *webservices* disponibilizados pelo mesmo.

Para prova de conceito, foi disponibilizado pela instituição um dispositivo eletrónico compacto para integração num sapato português com enfoque na monitorização de crianças. Tem funcionalidades de ativação/desativação, de forma automática, através de um acelerómetro; possibilita monitorizar a localização via GPS/GLONASS; envia informação relativa ao estado da bateria; informa caso detete a remoção do sapato, e, por último, efetua comunicação via GPRS. Assim sendo, contém relevância nesta dissertação de mestrado e, como tal, apresenta-se o seu protótipo, que futuramente será inserido no referido sapato:

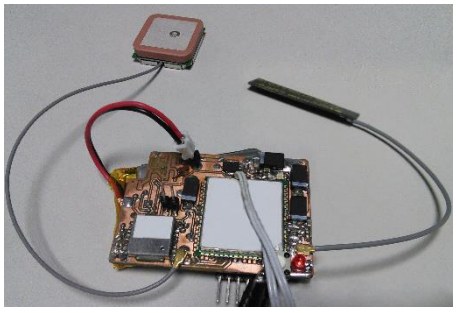


Figura 40 – Protótipo do dispositivo de monitorização CeNTI

O servidor encontra-se dividido em três componentes distintos: o primeiro, *server*, é o que implementa os vários serviços, através de *webservices*, aloja o *back-end*, é responsável pela receção da informação referentes aos *devices*, filtra a informação e armazena essa informação no servidor de base de dados. O segundo componente, CouchDB *server*, representando o sistema de base de dados NoSQL, é responsável pelo armazenamento da informação, pela resposta às *queries* efetuadas na plataforma e pela replicação da informação para os dispositivos móveis. Por último, o *server notification*, responsável por notificar os clientes aquando da monitorização dos seus dispositivos. Este servidor implementa um WebSocket, estabelecendo assim um canal bidirecional com os clientes autenticados no *browser*.

Existem ainda os sistemas de interação com os utilizadores, a única forma destes interagirem com a plataforma. Estes sistemas são representados no diagrama de sistema pelo *browser* e pelas aplicações móveis do sistema Android. Como é possível visualizar na figura anterior, as aplicações nativas possuem uma base de dados interna, proporcionando o funcionamento em modo *offline*.

5.3 Modelo de dados

O modelo de dados desenvolvido para o armazenamento de informação numa base de dados NoSQL é ligeiramente diferente do convencional. Contudo, seria interessante perceber, do ponto de vista didático, as diferenças de implementação, entre o modelo relacional e o modelo NoSQL orientado ao documento, tendo em conta os requisitos propostos no âmbito do projeto.

No anexo 8.4 Modelo de dados – Relacional é apresentada uma possível solução para o armazenamento da informação. Ao desenvolver o modelo de dados relacional concluiu-se que o armazenamento das configurações dos sensores e a monitorização iria ser um processo complexo devido à natureza dinâmica da informação. Esse problema passaria pelo armazenamento de diferentes tipos de conteúdos, como por exemplo, a configuração do sensor no que toca à informação do tipo *true* ou *false*, máximos e mínimos (valores numéricos), e zonas de segurança. Além disso, o inevitável volume de informação, a obrigatoriedade do uso de arquiteturas rígidas e o *overhead* necessário para o processo de alteração, como por exemplo, numa configuração de um sensor, foram aspetos que colocaram o modelo relacional de parte.

O modelo de dados NoSQL orientado ao documento é apresentado no anexo 8.3 Modelo de dados – NoSQL *document*. Como é possível ver neste anexo, existem três divisões lógicas da informação, representando cada uma destas uma base de dados, nomeadamente: *_usersDB*, *devices* e *usernameDB*.

Como já referido anteriormente, a base de dados *_usersDB* é uma base de dados pré-definida do CouchDB, que permite armazenar a informação relativa aos utilizadores registados no sistema de base de dados. Aproveitando a sua existência e funcionalidade, foi também utilizada para o armazenamento da informação do perfil do utilizador. No excerto de código seguinte demonstra-se a estrutura do documento de um utilizador:

```
{
"_id": "org.couchdb.user:rpessoa",
"_rev": "revision of document",
"name": "rpessoa",
"email": "rpessoa@centi.pt",
"full_name": "Ricardo Pessoa",
"salt": "33dc7b0e7ee6725a49cb869d2e01cf4c",
"password_sha": "00dc78463826ad698ab196f65f751e54936de6f1",
"roles": [
],
"type": "user",
"country": "Portugal",
"mobile_phone": "919999999"
}
```

Código 4 – Exemplo documento utilizador

No documento utilizador são armazenados os atributos *_id* onde consta o *username* do utilizador, que é obrigatoriamente único na base de dados. Ao nível da segurança, os atributos *salt* e *password_sha* armazenam a *password* na base de dados, de forma segura. Como já referido, o atributo *salt* é gerado de forma aleatória e seguidamente é combinado com a *password* escolhida pelo utilizador. Origina assim a *password_sha* que é armazenada na base de dados. O principio básico de encriptação usado foi SHA-1, que faz com que a *password* não seja distinguida e visível. O *array roles* está vazio, o que significa que este utilizador não tem privilégios sobre as restantes bases de dados, sendo apenas detentor de privilégios na sua própria base de dados, neste caso, utilizador rpessoa.

A base de dados *devices* serve para armazenar todos os dispositivos do sistema e só é acedida pelo administrador, com o objetivo de gerir, caso seja necessário, os diversos dispositivos no sistema. Ao receber a monitorização proveniente de um dispositivo é necessário encontrar o seu proprietário. Para isso, é utilizada a base de dados *devices*, que após encontrar o proprietário, armazena a informação da monitorização na base de dados desse utilizador. Em seguida é apresentado um exemplo de um dispositivo com os vários tipos de sensores:

```
{
   "_id": "id_mac_address",
   "_rev": "revision of document",
   "name_device": "name of device",
   "sensors": {
       "0": {
           "name_sensor": "Panic Button",
           "type": "panic_button",
           "enable": true
       },
       "0": {
           "name_sensor": "Shoe",
           "type": "shoe ",
           "enable": true
       },
       "1": {
           "name_sensor": "Sensor GPS",
```

```
            "type": "GPS",
            "enable": true

        },
        "2": {
            "min_temperature": "23",
            "max_temperature": "25",
            "name_sensor": "Sensor Temperature",
            "type": "temperature",
            "enable": true
        },
        "3": {
            "low_battery": "30",
            "critical_battery": "15",
            "name_sensor": "Battery",
            "type": "battery"
            "enable": true
        }
    },
    "timestamp": 1397147488,
    "owner": "rpessoa",
    "type": "device",
    "deleted": false,
    "monitoring": true
}
```


Código 5 – Exemplo documento dispositivo

O dispositivo tem um *array* de sensores: botão de pânico, estado do sapato (calçado ou não), localização, temperatura e bateria. Cada sensor possui o atributo *enable*, do tipo booleano, que permite saber se o sensor está ou não a ser monitorizado. Se o dispositivo for eliminado, o atributo *deleted* ficará a *true*, na verdade não será eliminado, apenas não ficará visível ao utilizador, nem receberá notificações referentes à sua monitorização.

Por fim, existe a base de dados *_usernameDB*, criada sempre que um novo utilizador se regista na plataforma, designando o *username* do registo para o nome da base de dados. Através deste comportamento, cada utilizador tem a sua própria base de dados, onde é possível armazenar a informação de forma independente, obtendo um maior controlo da informação, facilitando e melhorando significativamente a performance das *queries.*

Como exemplo de estrutura de documento para a base de dados *usernameDB* salienta-se o documento *safezone*. Cada dispositivo que tenha a funcionalidade GPS poderá associar a esse dispositivo zonas de segurança, com a seguinte estrutura:


```
{
    "_id": "safezone_timestamp",
    "_rev": "revision of document",
    "address": "Address Safezone,
    "name": "Safezone Name",
    "latitude": 41.2285226,
```

```
    "longitude": -8.6987819,
    "radius": 5000,
    "notification": "CHECK-IN||CHECK-OUT||ALL",
    "timestamp": timestamp,
    "type": "safezone",
    "device": "mac_address of device"
}
```

Código 6 – Exemplo documento zona de segurança

No excerto de código anterior identificam-se as propriedades necessárias para armazenar uma *safezone*. A *safezone* é uma *geo-fence* circular, ou seja, uma zona de segurança constituída por um ponto central definido por coordenadas geográficas, latitude e longitude, e por um raio, em metros. No atributo *notification* define-se o tipo de notificações que o utilizador receberá, entre *Check-in*, *Check-out* ou *All* (*Check-in* e *Check-out)*. Para associar uma determinada *safezone* a um determinado dispositivo, o atributo *device* armazena o identificador *mac_address* do dispositivo.

As particularidades dos documentos referentes à monitorização dos sensores botão de pânico, estado do sapato (calçado ou não), localização, temperatura e bateria são detalhadas no anexo 8.3 Modelo de dados – NoSQL *document*.

Uma vez inseridos os vários documentos, é necessário fornecer mecanismos de pesquisa dessa informação. Para tal, cada base de dados do utilizador necessita de várias *views* para implementar essa funcionalidade. As *views* são documentos com características especiais, apresentadas no excerto de código seguinte.

```
{
    "_id": "_design/application",
    "_rev": "revision of document",
    "language": "JavaScript",
    "views": {
        "getSafezones": {
            "map": "function(doc){ if(doc.type == 'safezone') emit(doc.device,
doc); }",
            "reduce": "_count"
        },
        "getDevices": {
            "map":  "function(doc){  if(doc.type  ==  'device')  emit(doc._id,
doc); }",
            "reduce": "_count"
        },
        "getSensors": {
            "map": "function(doc){ if(doc.sensors){ for(var i in doc.sensors)
emit(doc._id,doc.sensors[i]); } }",
            "reduce": "_count"
        },
        "getMonitoringSensor": {
            "map":        "function(doc)        {        if(doc.type        ==
'monitoring_sensor'){ emit([doc.mac_address,doc.subtype], doc); } }"
        }
    }
}
```

Código 7 – Exemplo documento *view*

As *view* tal como os nomes indicam getSafezones, getDevices, getSensores e getMonitoringSensor facultam a informação referente às zonas de segurança, dispositivos, sensores e monitorização de sensores.

5.4 Desenvolvimento

O projeto pressupõe o desenvolvimento de um servidor, um *back-end* e uma aplicação móvel. Este conteúdo é apresentado em três subsecções para uma melhor compreensão da dimensão e diversidade das tecnologias utilizadas.

Na imagem que se segue é possível visualizar os três componentes e as principais tecnologias que o decompõem:

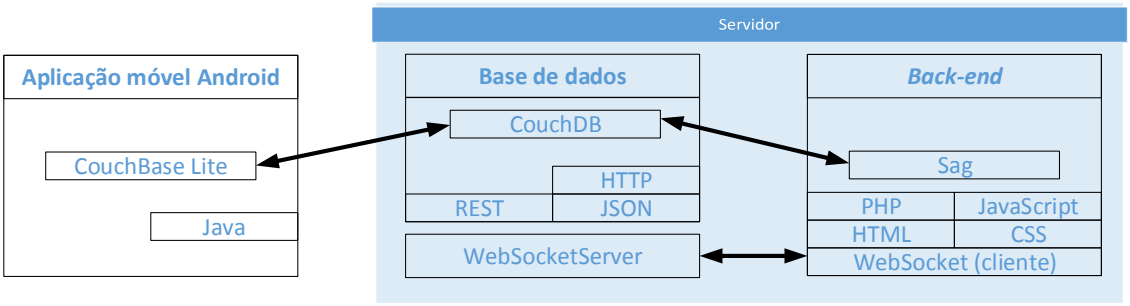


Figura 41 – Estrutura das tecnologias

5.4.1 Desenvolvimento do servidor

O servidor tem um papel fundamental no projeto, já que sem ele, não seria possível o armazenamento da informação, o alojamento do *back-end*, nem oferecer os serviços consumidos pela aplicação *web* e *mobile*. Na implementação do servidor foram instaladas as seguintes tecnologias:

Tabela 10 – Tecnologias utilizadas no servidor

Tipo de tecnologias	Plataforma utilizada
Sistema operativo	Linux Ubuntu 12.04 LTS
Servidor base de dados	CouchDB
Servidor *web*	Apache2 *web server*
Serviços *web*	REST
WebSockets	PHP WebSockets

No desenvolvimento do servidor propriamente dito foi efetuada a instalação e posterior configuração, com o objetivo de colocar o sistema em funcionamento. Para isso foi disponibilizada pela empresa uma máquina e posteriormente procedida à instalação do sistema operativo Ubuntu 12.04 LTS. Depois de instalado o SO foram instalados os vários componentes, através do terminal.

O Apache2 foi instalado através do comando *sudo apt-get install apache2*. Seguidamente editado o ficheiro *httpd.conf*, tipicamente localizado em */etc/apache2/conf-available/httpd.conf* com a seguinte linha: *LoadModule rewrite_module libexec/apache2/mod_rewrite.so*. Esta linha permite ao servidor compreender *urls*, de forma mais limpa. Assim, no *back-end* é possível utilizar o estilo REST no endereço. No Apache2 foi também necessário editar o ficheiro *apache2.conf* normalmente localizado no diretório */etc/apache2/apache2.conf* com a seguinte informação:

```
<Directory />
  Options FollowSymLinks
  AllowOverride All
  Order deny,allow
  Allow from all
</Directory>
<Directory /var/www/>
  Options Indexes FollowSymLinks
  AllowOverride All
  Order deny,allow
  Allow from all
</Directory>
```

Código 8 – Configuração *apache2.config*

Na máquina foi efetuada a instalação do PHP e alguns componentes, recorrendo ao comando *sudo apt-get install php5 php5-dev libapache2-mod-php5 php5-curl php5-mcrypt*. Por último foi instalado o CouchDB com recurso ao comando *sudo apt-get install couchDB* e configuração de um *admin* para limitar o acesso ao sistema de base de dados.

Depois de todas estas configurações efetuadas, o servidor estava praticamente pronto para o desenvolvimento da aplicação *web* e *mobile*.

5.4.2 Desenvolvimento do *back-end*

A primeira aplicação a ser desenvolvida foi a *web*, ou seja, o *back-end*. Na tabela que se segue encontram-se detalhadas as tecnologias utilizadas para o seu desenvolvimento:

Tabela 11 – Tecnologias usadas no desenvolvimento do *back-end*

Tipo de tecnologias	**Plataforma utilizada**
Software de desenvolvimento	NetBeans
Linguagem de programação	PHP, JavaScript, HTML e CSS
Framework	Fligth, Bootstrap
Bibliotecas	Sag, Mapstraction, Highcarts, jCryption 3.0.1
API	Google Maps

As tecnologias apresentadas na tabela anterior foram já introduzidas no estado da arte tecnológico. Contudo é relevante apresentar alguns excertos de código importantes no desenvolvimento do projeto, bem como alguns *screenshots* das principais funcionalidades disponibilizadas pelo *back-end*.

5.4.2.1 Interface do *back-end*

A interface do *back-end* foi estudada num processo paralelo ao desenvolvimento do *site*. Primeiramente foram apresentados *mockups* contendo os aspetos base da interface e o que seria planeado implementar em termos de funcionalidades. É possível visualizar a última versão do *mockup* no anexo 8.1 *Mockup back-end*.

Na figura seguinte consta a página *MyDashboard,* que apresenta ao utilizador de forma sumarizada as últimas notificações dos sensores para cada dispositivo. No painel principal é possível visualizar os seguintes estados de cada sensor: botão de pânico, estado do sapato, coordenadas GPS, temperatura e estado da bateria.

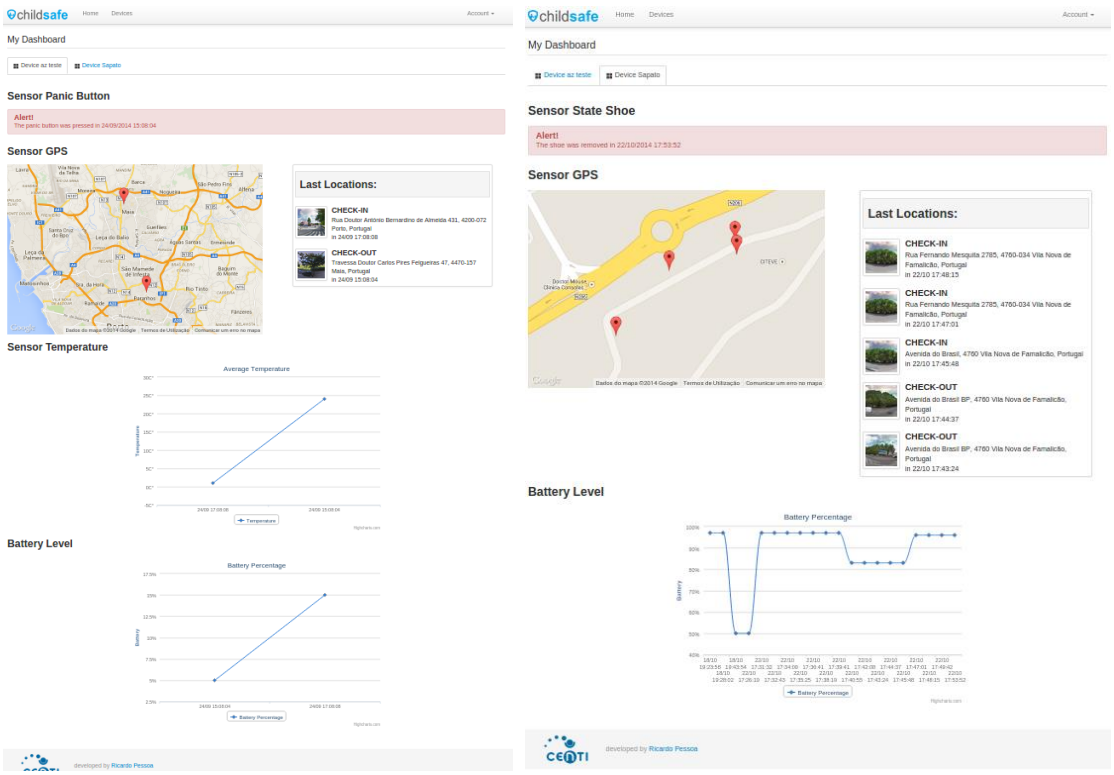


Figura 42 – *Screenshot* painel principal

No *back-end* também se implementou a possibilidade de adicionar novos dispositivos, visualizar lista de dispositivos, monitorizar um determinado dispositivo em tempo real e gerir dispositivos e sensores individualmente.

Na imagem que se segue é apresentada a página "*Devices*":

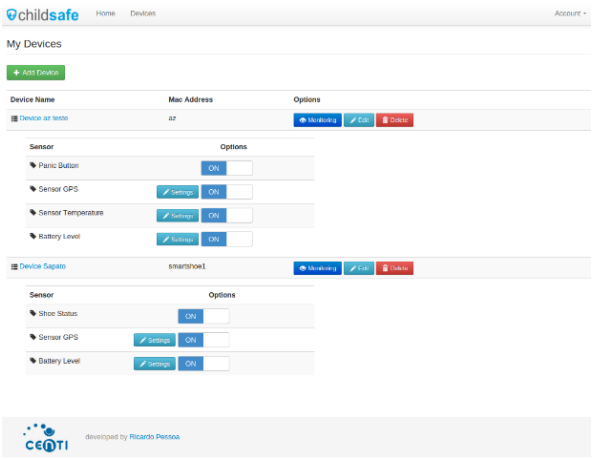


Figura 43 – *Screenshot* meus dispositivos

Uma funcionalidade relevante na plataforma corresponde à configuração das zonas de segurança. A qualquer dispositivo que possua uma antena GPS podem associar-se zonas de segurança. Na imagem que se segue é possível visualizar a lista de zonas de seguranças de um dispositivo.

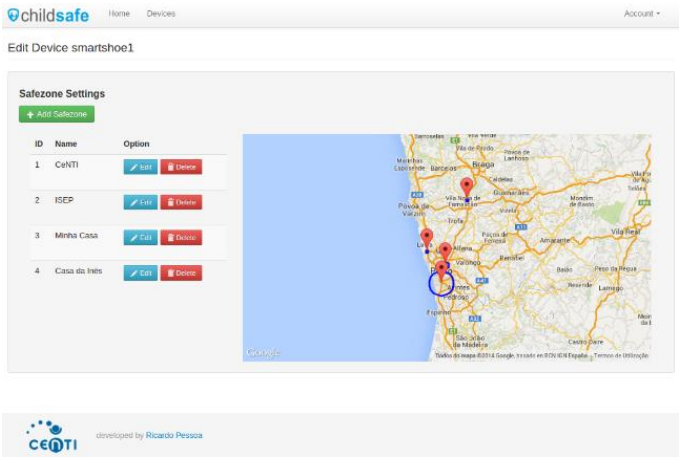


Figura 44 – *Screenshot* lista de zonas de segurança

No *back-end*, as novas zonas de seguranças são criadas através do botão "*Add Safezone*" que redireciona o utilizador para a página responsável por inserir e editar zonas de segurança. Inicialmente é efetuada a pesquisa pelo endereço e posteriormente, define-se o raio, nome e tipo de notificação. É também possível mover o marcador para uma localização mais precisa. As duas imagens seguintes refletem o processo de adicionar ou gerir zona de segurança.

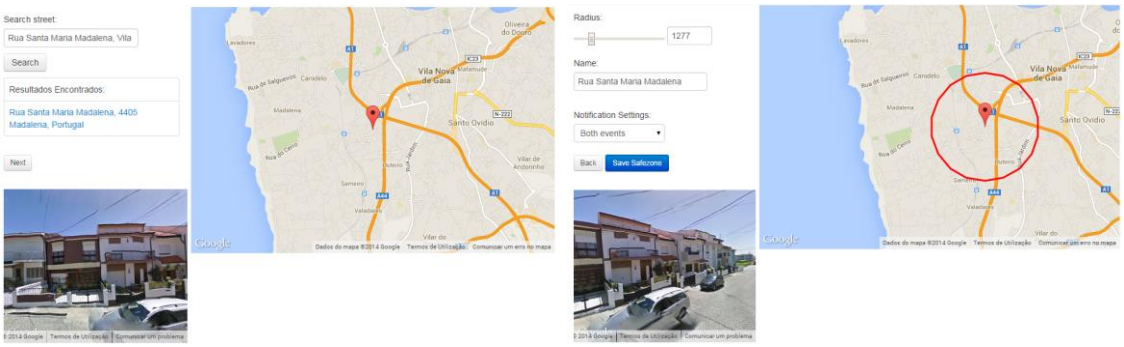


Figura 45 – *Screenshot* gerir zona de segurança

Além das zonas de segurança, a plataforma apresenta também a possibilidade de monitorização em tempo real, através do *back-end*. Esta funcionalidade, visível na imagem seguinte, permite visualizar a localização atual do dispositivo, o estado atual dos diversos sensores, bem como o trajeto efetuado durante o período de monitorização.

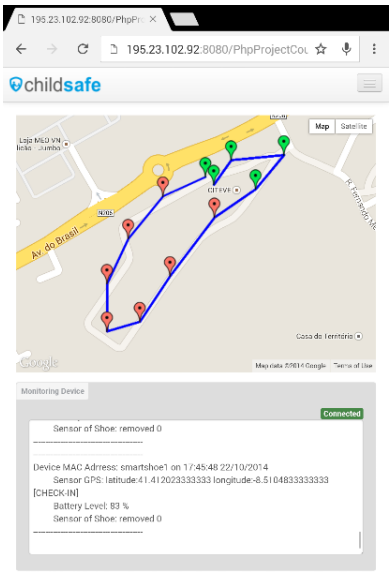


Figura 46 – *Screenshot* monitorização de dispositivos

5.4.2.2 Exemplo de pedido à base de dados

As *views* têm um papel fundamental em todo o projeto, já que permitem realizar pesquisas, filtrando a informação armazenada na base de dados. Assim é relevante apresentar um excerto de código com uma *view* que visa retornar as monitorizações dos sensores armazenados. Esta *view* recebe como parâmetro um *array* com as chaves *mac address* e subtipo, filtrando a informação pelo dispositivo e tipo de sensor. Outra particularidade desta *view* é que limita o número de resultados através do *limit=numberOfResults* e apresenta a informação de forma descendente, através do atributo *descending=true.*

```
public function getMonitoringSensorByKeys($usernameDB, $macAddress, $subtype,
$numberOfResults) {
   $sensorsGps = array();
```

```
   try {
      foreach(Base::getViewToIterateBasedInUrl($usernameDB,
'_design/application/_view/getMonitoringSensor?key=[“'.$macAddress.'“,”'.
$subtype . '“]&limit=$numberOfResults&descending=true')as $_monitoringGPS) {
         $monitoringSensorGPS = new MSGPS();
         $monitoringSensorGPS->_id = $_monitoringGPS->id;
         $monitoringSensorGPS->_rev = $_monitoringGPS->value->_rev;
         $monitoringSensorGPS->type = $_monitoringGPS->vavlue->type;
         $monitoringSensorGPS->subtype = $_monitoringGPS->value->subtype;
         $monitoringSensorGPS->latitude = $_monitoringGPS->value->latitude;
         $monitoringSensorGPS->longitude= $_monitoringGPS->value->longitude;
         $monitoringSensorGPS->timestamp=date('d/m H:i:s',$_monitoringGPS->
value->timestamp);
         $monitoringSensorGPS->mac_address    =    $_monitoringGPS->value->
mac_address;
         $monitoringSensorGPS->address = $_monitoringGPS->value->address;
         $monitoringSensorGPS->notification    =    $_monitoringGPS->value->
notification;
         array_push($sensorsGps, $monitoringSensorGPS);
        }
     } catch (Exception $exc) {
         return $exc;
     }
   if (sizeof($sensorsGps) > 0)
       return $sensorsGps;
   return NULL;
}
```

Código 9 – Pesquisa descendente da monitorização de um dispositivo, subtipo e limite de resultados de um determinado utilizador

O método estático “*getViewToIterateBasedInUrl*” é capaz de efetuar um pedido *GET* a uma determinada base de dados através do parâmetro “*usernameDB*”, que efetua o pedido respetivo do utilizador retornando o *output* da *view*.

5.4.2.3 Algoritmo cálculo de distância entre coordenadas geográficas

O algoritmo determinante na monitorização é o cálculo da distância entre dois pontos geográficos. Este método recebe como parâmetro duas coordenadas geográficas em decimal e retorna a distância, em metros, entre os dois pontos. Depois de retornada a distância entre a zona de segurança e a coordenada do dispositivo atual é necessário compará-la com o raio das zonas de segurança do dispositivo. Deste modo é possível apurar se o dispositivo está dentro ou fora de uma determinada zona de segurança. O algoritmo responsável pelo cálculo das distâncias é o seguinte:

```
public function haversineGreatCircleDistance($latitudeFrom, $longitudeFrom,
$latitudeTo, $longitudeTo, $earthRadius = 6371000) {
        $latFrom = deg2rad($latitudeFrom);
        $lonFrom = deg2rad($longitudeFrom);
        $latTo = deg2rad($latitudeTo);
        $lonTo = deg2rad($longitudeTo);
        $latDelta = $latTo - $latFrom;
        $lonDelta = $lonTo - $lonFrom;
        $angle = 2 * asin(sqrt(pow(sin($latDelta / 2), 2) + cos($latFrom) *
cos($latTo) * pow(sin($lonDelta / 2), 2)));
```

```
    return $angle * $earthRadius;
}
```

Código 10 – Algoritmo cálculo de distância entre dois pontos de coordenadas[48]

O algoritmo por de trás de todo o processo de verificação do tipo de coordenada *check-in* ou *check-out* é feito através de um ciclo que percorre todas as zonas de segurança de um determinado dispositivo e caso seja *check-in* termina o ciclo, caso contrário é do tipo *check-out*. Se o dispositivo não possuir nenhuma *safezone* associada, o tipo de notificação será apenas *location* uma vez que, não existe nenhum critério de comparação.

5.4.3 Desenvolvimento da aplicação móvel

Depois de desenvolvido o *back-end*, procedeu-se ao desenvolvimento da aplicação móvel para o sistema Android. As tecnologias que a decompõem são:

Tabela 12 – Tecnologias usadas no desenvolvimento da aplicação móvel

Tipo de tecnologias	Plataforma utilizada
Software de desenvolvimento	Android Studio
Linguagem de programação	Java
API	CouchBase Lite Android, Google Maps

5.4.3.1 CouchBase Lite Android

A instalação da API CouchBase Lite Android em particular permitiu acelerar o processo de desenvolvimento da aplicação móvel. A estrutura da plataforma desenvolvida é a que se segue:

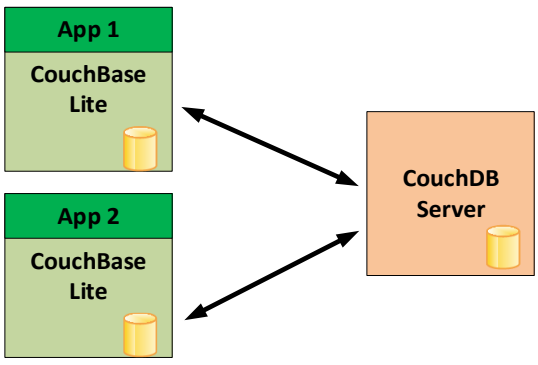


Figura 47 – Arquitetura CouchBase Lite e CouchDB *Server*

Para utilizar a API CouchBase Lite Android foi necessário instalar as dependências do projeto na aplicação. No Android Studio existe uma ferramenta denominada Gradle com intuito de configurar e instalar, de forma automática, as bibliotecas necessárias durante o processo de compilação do projeto. No excerto de código que se segue apresentam-se, com detalhe, as dependências instaladas:

[48] https://stackoverflow.com/questions/14750275/haversine-formula-with-php/14751773#14751773

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:0.12.+'
    }
}
apply plugin: 'android'
repositories {
    mavenCentral()
    maven {
        url “http://files.couchbase.com/maven2/”
    }
    mavenLocal()
}
dependencies {
    compile fileTree(dir: 'libs', include: 'td_collator_so.jar')
    compile fileTree(dir: 'libs', include: 'nineoldandroids-2.4.0.jar')
    compile 'com.android.support:support-v4:19.+'
    compile 'com.android.support:appcompat-v7:19.+'
    compile 'com.couchbase.cblite:CBLite:1.0.0-beta2'
    compile 'com.google.android.gms:play-services:4.2.42'
}
```

Código 11 – Dependências aplicação móvel

Depois de instaladas todas as dependências foi necessário criar as classes e os métodos necessários para a interligação do sistema CouchBase Lite com a aplicação. Apresenta-se no anexo 8.5 o diagrama de classes simplificado.

5.4.3.2 LiveQuery

LiveQuery é uma funcionalidade disponibilizada pela API CouchBase Lite, com o objetivo de receber notificações na aplicação sem que o utilizador necessite de efetuar qualquer pedido. Através desta funcionalidade é possível receber em tempo real, as alterações efetuadas durante a sincronização da base de dados local com o servidor. É também possível filtrar, através das *views*, os tipos de documentos dos quais se pretender receber notificação. Para implementar esta funcionalidade foi primeiramente necessário criar uma *view* (as *views* criadas na aplicação não são sincronizadas com a base de dados no servidor CouchDB) e posteriormente criar e associar ao objeto LiveQuery. Este processo ocorre da seguinte forma.

```
public static View viewGetDevices;
private LiveQuery liveQueryDevice;
String allDevicesViewName = “getAllDevices”;
viewGetDevices   =   database.getView(String.format(“%s/%s”,   designDocName,
allDevicesViewName));
        viewGetDevices.setMap(new Mapper() {
            @Override
            public void map(Map<String, Object> document, Emitter emitter) {
                Object objDevice = document.get(“type”);
                if (objDevice != null && objDevice.equals(“device”)) {
                    emitter.emit(objDevice.toString(), document);
                }
            }
```

```
        }, "1.0");
startLiveQuery(liveQueryDevice, viewGetDevices);
private void startLiveQuery(LiveQuery liveQuery, com.couchbase.lite.View
view) throws Exception {
        if (liveQuery == null) {
            liveQuery = view.createQuery().toLiveQuery();
            liveQuery.addChangeListener(new LiveQuery.ChangeListener() {
                @Override
                public void changed(LiveQuery.ChangeEvent event) {
                    sendNotificationBasedInType(event.getRows());
                }
            });
            liveQuery.start();
        }
    }
```

Código 12 – Algoritmo para a criação de uma LiveQuery

A *view "viewGetDevices"*, como o nome indica, foi desenvolvida para proceder à realização de pesquisas relacionadas com os dispositivos. Esta *view* foi associada, de forma dinâmica, à LiveQuery através do método *startLiveQuery*. Sempre que é recebido um evento de alteração na base de dados local, relacionado com uma determinada *view* é invocado o método *sendNotificationBasedInType*. Este método redireciona os eventos, através de um *Broadcast Receiver*, para a *Activity* responsável. Deste modo, através do *Broadcast Receiver*, a *Activity* sabe da ocorrência de uma alteração e proceda à atualização da *view*.

5.4.3.3 *BroadcastReceiver*

Uma vez que os dados são rececionados e tratados numa *thread* à parte, é necessário, de alguma forma, notificar as *views* e a *mainthread* (*thread* responsável pela interface gráfica) no processo de sincronização da informação da base de dados local com a do servidor CouchDB. Para isso foi utilizada uma funcionalidade nativa do Android, denominada *BroadcastReceiver*. Todas as *views* passiveis de receber alterações, como é o caso do painel principal, listas de dispositivos, zonas de segurança, etc, implementam o *BroadcastReceiver*.

No excerto de código que se segue apresenta-se um exemplo capaz de notificar a *view* dos dispositivos:

```
public static final String notify =
"mei.ricardo.pessoa.app.notifyDevice.devices";
private DeviceBroadcastReceiver deviceBroadcastReceiver = null;
@Override
  public void onResume() {
    super.onResume();
    deviceBroadcastReceiver = new DeviceBroadcastReceiver();
    registerReceiver(deviceBroadcastReceiver, new IntentFilter(notify));
  }
@Override
  public void onPause() {
    super.onPause();
    if (deviceBroadcastReceiver != null)
    unregisterReceiver(deviceBroadcastReceiver);
  }
private class DeviceBroadcastReceiver extends BroadcastReceiver {
```

```
    @Override
      public void onReceive(Context context, Intent intent) {
        deviceListAdapter.updateDeviceList(Device.getAllDevicesNotDeleted());
      }
  }
```

Código 13 – Exemplo evento de alteração da informação *BroadcastReceiver*

Sempre que a *Activity* está "*onResume*", ou seja, em operação e visível para o utilizador, esta regista o *BroadcastReceiver*. Por outro lado, quando a *Activity* está "*onPause*", ou seja, passa de primeiro para segundo plano, esta anula o *BroadcastReceiver*. O método "*OnReceiver"* é invocado sempre que a base de dados encontra uma alteração da informação através da funcionalidade LiveQuery que, por sua vez, notifica a *Activity* da ocorrência de uma alteração e força o *adapter* a atualizar-se.

5.4.3.4 Exemplo de pedido à base de dados

O pedido à base de dados é sempre efetuado sobre a base de dados local. Este comportamento aumenta a disponibilidade, diminui a latência dos pedidos e funcionamento da aplicação mesmo em modo *offline*. No exemplo consta um pedido à base de dados, utilizando a *view* "*viewGetDevices*". Este código tem como objetivo preencher a lista de dispositivos do utilizador.

```
public static ArrayList<DeviceRow> getAllDevicesNotDeleted() {
  ArrayList<DeviceRow> deviceRowsList = new ArrayList<DeviceRow>();
  com.couchbase.lite.View view = CouchDB.viewGetDevices;
  Query query = view.createQuery();
  query.setDescending(true);
  QueryEnumerator rowEnum = query.run();
  for (Iterator<QueryRow> it = rowEnum; it.hasNext();) {
    QueryRow row = it.next();
    DeviceRow deviceRow = new DeviceRow();
    deviceRow.deviceID = row.getDocumentId();
    String nameDevice = "";
    HashMap<Object, Object> numbSensors = new HashMap<Object, Object>();
    Document document = row.getDocument();
    nameDevice = document.getProperty("name_device").toString();
    if (document.getProperty("deleted").equals(true))
      continue;
numbSensors=(HashMap<Object,Object>)row.getDocument().getProperty("sensors
");
    deviceRow.deviceName = nameDevice;
    if (numbSensors == null)
      continue;
    deviceRow.deviceDescription = "Number of Sensors" + numbSensors.size();
    deviceRowsList.add(deviceRow);
  }
  return deviceRowsList;
}
```

Código 14 – Exemplo de pedido à base de dados aplicação móvel

Este método efetua um pedido à base de dados, somente sobre a informação necessária para a apresentação dos dispositivos numa lista: *id*, nome e número de sensores do dispositivo. Este

excerto de código apresenta uma versão simples do algoritmo, tendo sido omissas as validações, devido à sua elevada extensão.

5.4.3.5 Interface da aplicação móvel

A aplicação móvel tem como principal objetivo receber informação acerca da monitorização e, posteriormente, notificar o utilizador aquando de algum resultado inesperado. Assim, a aplicação dá um maior destaque ao processo de notificação e detalhes da informação. No desenvolvimento da aplicação foi tida em consideração a interface já desenvolvida no *back-end*, bem como o estudo efetuado sobre as aplicações, de forma a criar uma interface uniformizada sem que o utilizador necessite de aprender o funcionamento da mesma.

Segue-se o *layout* do *My Dashboard*, com uma aparência semelhante à versão do *site*.

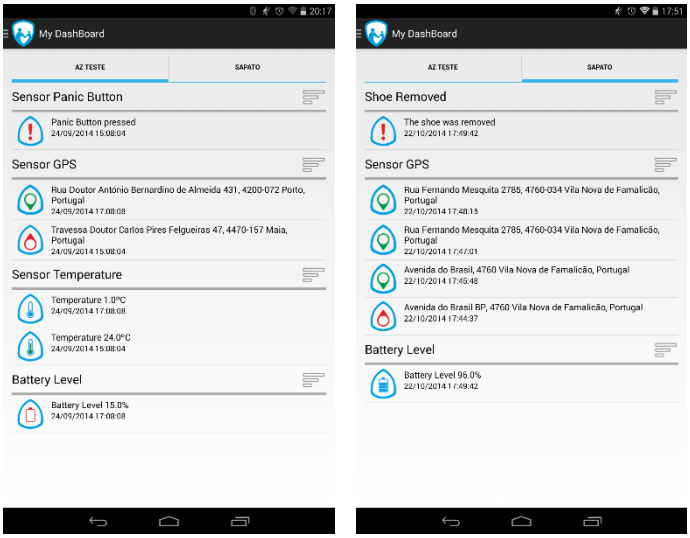
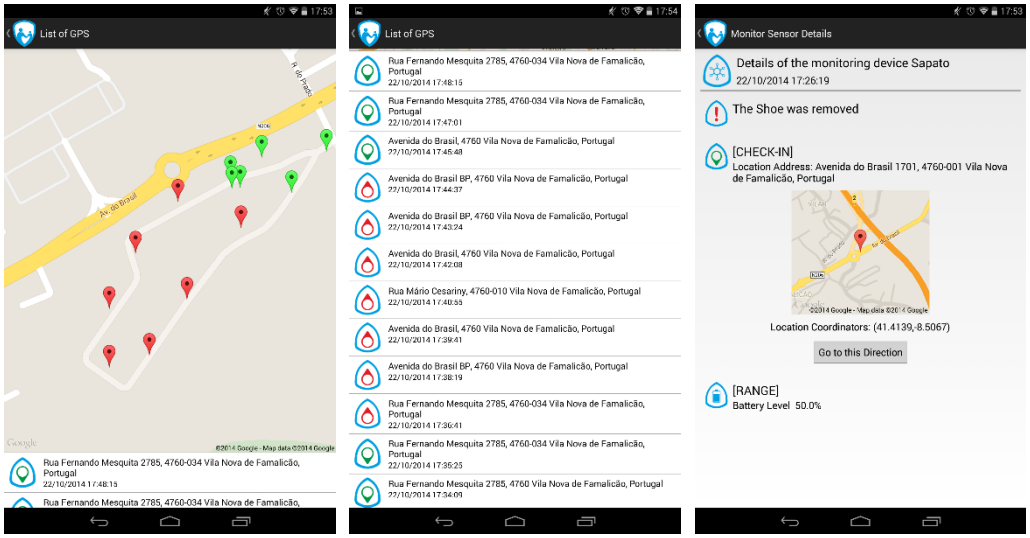


Figura 48 – *Screenshot* painel principal da aplicação móvel

Para apresentar os diversos dispositivos no painel principal e para que cada dispositivo mostrasse as últimas monitorizações dos sensores, de forma sintetizada, foi necessário implementar o componente *Swipe Views Tabs*. Em cada separador surge, no extremo direito, um botão para a visualização dos detalhes de cada sensor.

No caso de visualizar os detalhes das coordenadas GPS (nas duas primeiras imagens abaixo) é possível visualizar no topo um mapa com os pontos *check-in* e *check-out*. Na visualização dos detalhes da monitorização (na terceira imagem), apresenta-se um exemplo de detalhe de mecanismo de alerta (sapato retirado) onde é agrupada toda a informação associada ao mesmo.



Na navegação da aplicação foi utilizado o padrão *Navigation Drawer*, anteriormente abordado, no estado da arte. Exibe-se ainda o resultado da implementação do menu e no topo a *fragmentview*, responsável por apresentar as últimas novidades da monitorização.

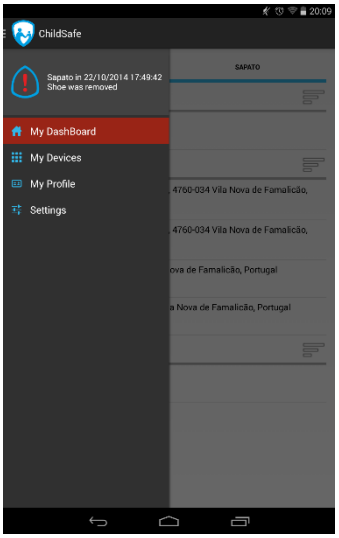


Figura 49 – *Screenshot* menu e notificação da aplicação móvel

No *Navigation Drawer* é possível navegar pelos vários menus (*My Dashboard, My Devices, My Profile* e *Settings*). Além disso, no topo são apresentadas as últimas novidades, resultantes das monitorizações recebidas dos dispositivos.

No caso das zonas de segurança, a aplicação permite a sua visualização e gestão, de forma simplificada. Nas imagens seguintes apresenta-se: do lado esquerdo, a lista das zonas de segurança e do lado direito as funcionalidades de inserção e edição da zona de segurança em termos de localização e raio.

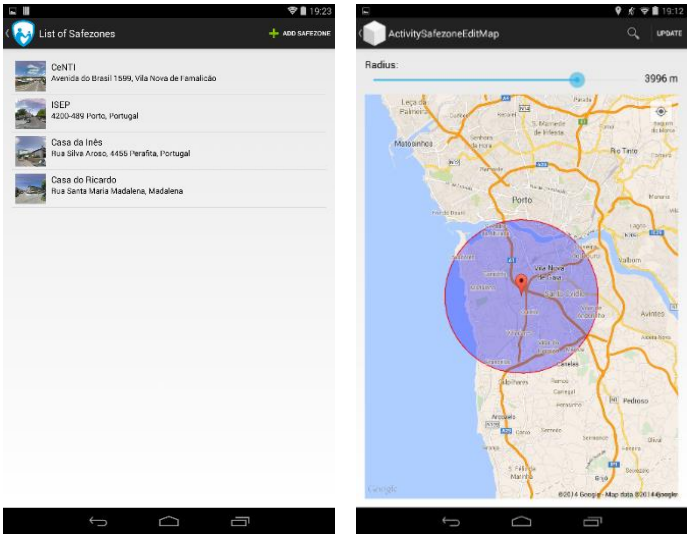


Figura 50 – Lista de *safezones* e gerir zonas de segurança

A aplicação implementa vários mecanismos de notificação, tanto em modo *background* como *foreground*. Na ativação de algum mecanismo de pânico (botão de pânico ou sapato retirado) a aplicação notifica o utilizador através de um alerta sonoro e um *dialog*, exibindo o alerta. No caso das notificações temperatura, bateria ou localização geográfica, a aplicação avisa do sucedido através do aviso sonoro definido no sistema e adiciona uma notificação na barra de notificações do sistema. Apresenta-se abaixo o sistema de notificações da aplicação:

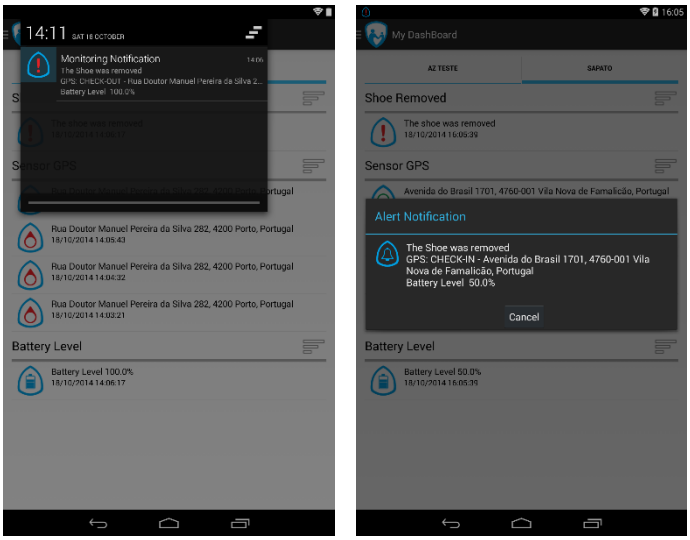


Figura 51 – Sistema de notificação da aplicação

A aplicação implementa inúmeras funcionalidades que infelizmente são impossíveis apresentar de forma detalhada. Contudo, apresentam-se alguns *screenshots* da aplicação das restantes funcionalidades e potencialidades da aplicação.

A *view* com perfil do utilizador e as definições da aplicação são as seguintes:

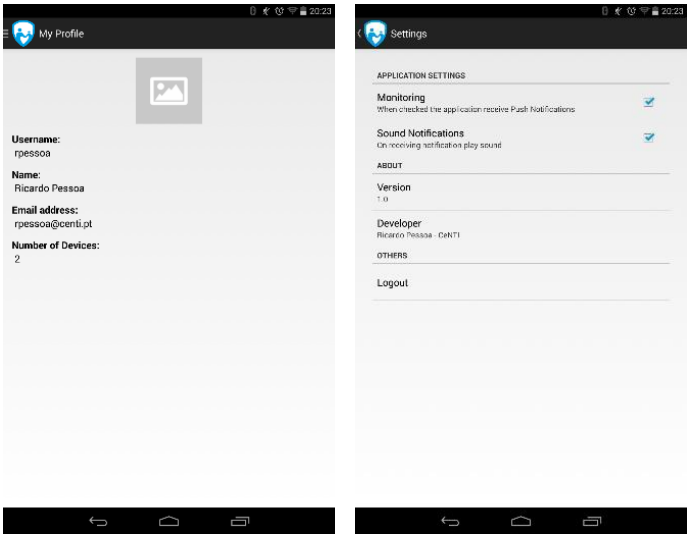


Figura 52 – *Screenshots* diversos da aplicação móvel

Através da *activity Settings*, o utilizador poderá ativar ou desativar o serviço que efetua a monitorização quando a aplicação está "desligada" ou em *background*, ativar ou desativar os sons para não incomodar e ainda possibilita fazer *logout* na aplicação. Esta funcionalidade apaga toda a informação do utilizador.

6 Testes

Este capítulo surge da necessidade de abordar os vários testes realizados durante o desenvolvimento da plataforma, bem como as conclusões retiradas dos mesmos. Uma vez que o conceito deste projeto obriga a implementações reais, é estritamente necessário validar e proceder a diversos testes, de forma a verificar e comprovar o comportamento da plataforma nestas situações.

6.1 Testes realizados

Durante o processo de desenvolvimento, e principalmente, na fase final do projeto foram realizados diversos testes à plataforma. Estes testes tiveram como objetivo verificar se as funcionalidades estavam a ser desenvolvidas de acordo com os objetivos a que a plataforma visava responder.

Inicialmente, o envio de informação foi simulado através de um cliente, que permitia verificar, principalmente, a receção ou não da informação, e também o comportamento do servidor no armazenamento dessa informação e a apresentação da mesma por parte das aplicações. Numa fase final do projeto, com vista a permitir o teste real, foi disponibilizado pela empresa um dispositivo com as seguintes funcionalidades: envio de coordenadas geográficas, capacidade de monitorização do estado do sapato (calçado ou não) e estado da bateria. Recorrendo a este dispositivo foram efetuados diversos testes reais na plataforma. Foram então planeados alguns trajetos com o objetivo de percorrer determinadas zonas de segurança, de forma a passar pelos estados de entrada e saída da *safezone*.

Na imagem que se segue consta um dos percursos planeados. O trajeto delineado por pontos azuis representa o percurso efetuado, enquanto o círculo azul representa a zona de segurança.

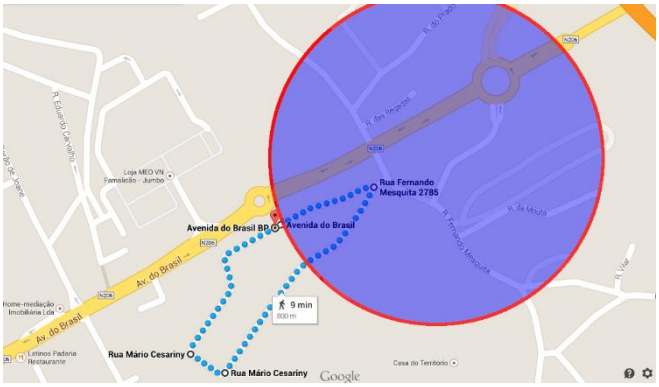


Figura 53 – Teste percurso planeado

Uma vez relizados os testes neste percurso foi possível obter os seguintes resultados:

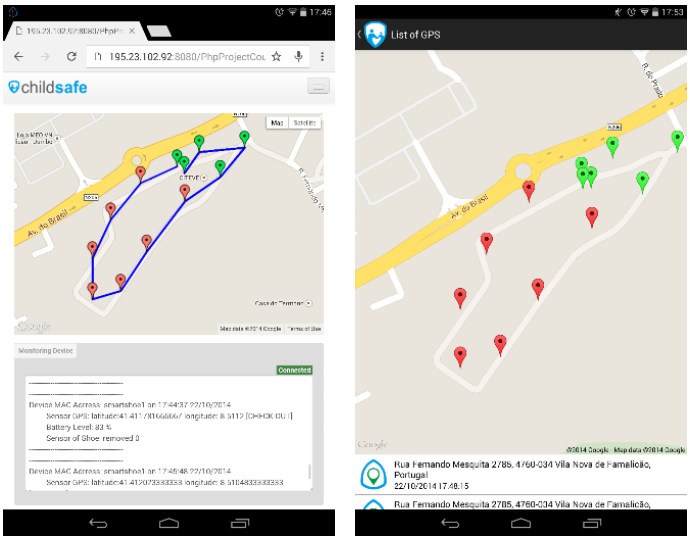


Figura 54 – Resultado do teste *back-end* e aplicação móvel

A monitorização foi realizada em simultâneo no mesmo dispositivo móvel, através do *back-end* e da aplicação, em modo *background*. Na versão *web*, as localizações são interligadas para o utilizador visualizar facilmente o trajeto efetuado durante a monitorização em tempo real. Na versão *mobile*, apenas são apresentados os pontos, já que na verdade o objetivo é visualizar as últimas localizações e não necessariamente o trajeto.

De forma a facilitar a interpretação da localização, esta apresenta-se de cor vermelha e verde representando *check-out* e *check-in*, respetivamente. Durante a monitorização, surgindo alguma informação relevante, a aplicação móvel alerta o utilizador, através dos mecanismos de notificação anteriormente apresentados.

Os testes realizados visaram validar todo o sistema: desde o envio da informação por parte do dispositivo, à receção dos dados e posterior processamento e armazenamento da informação no servidor e, por fim, a validação da informação recebida pela aplicação *web* e *mobile*.

A aplicação foi testada em vários dispositivos, entre os quais: Nexus 7 (KitKat 4.4.4), Samsung Galaxy Ace Plus (Gingerbread 2.3.6), Sony Xperia J (Jelly Bean 4.1.2), BQ Aquaris E5 (KitKat 4.4.2), com o intuito de validar o comportamento da aplicação, em diferentes dispositivos, com características distintas, quer ao nível de processamento, memória e em diferentes tamanhos de ecrã. Nestes testes, verificou-se que o comportamento do *back-end* e da aplicação móvel não ficaram comprometidos em nenhum dos níveis acima referidos.

Para proceder à realização dos testes recorreu-se não só à rede *Wi-Fi* mas também à rede móvel. Foram testadas diversas cadências de envio por parte do dispositivo de monitorização, com vista a encontrar um valor apropriado na monitorização dos vários sensores e também no sentido de validar o comportamento do servidor com diferentes fluxos de informação.

Nos testes efetuados notou-se um certo *delay,* na ordem de um/dois segundos, na receção dos dados entre o *back-end* e a aplicação móvel. Trata-se de um atraso totalmente aceitável, dado que a aplicação móvel utiliza um processo distinto, onde existe uma maior sobrecarga de processamento, comparativamente ao *back-end*.

Foram ainda realizados testes, desta vez utilizando ferramentas como: Dumpsys, Dalvik Debug Monitor Server (DDMS) e Eclipse Memory Analyser (MAT). Através destas ferramentas foi possível testar o comportamento da aplicação ao nível do desempenho, velocidade de transferência, uso de memória RAM e utilização do CPU. Estes testes foram realizados no dispositivo Nexus 7, e o procedimento efetuado foi o seguinte: iniciar pela primeira vez a aplicação, sincronizar cerca de 1450 documentos, 2 dispositivos e cada um com uma zona de segurança. Neste teste, em particular, verificou-se uma demora um pouco acentuada na sincronização da informação dado o volume da mesma.

O Android Studio lançou, na versão 0.8.10, a ferramenta Memory Monitor que possibilitou analisar a utilização da memória RAM ao longo do tempo de execução da aplicação. O teste consiste em iniciar pela primeira vez a aplicação, efetuar autenticação, sincronizar os dados, navegar pela aplicação, visualizar os detalhes e, por fim, percecionar o desempenho do serviço em *background*. Na figura que se segue consta o resultado da monitorização da aplicação.

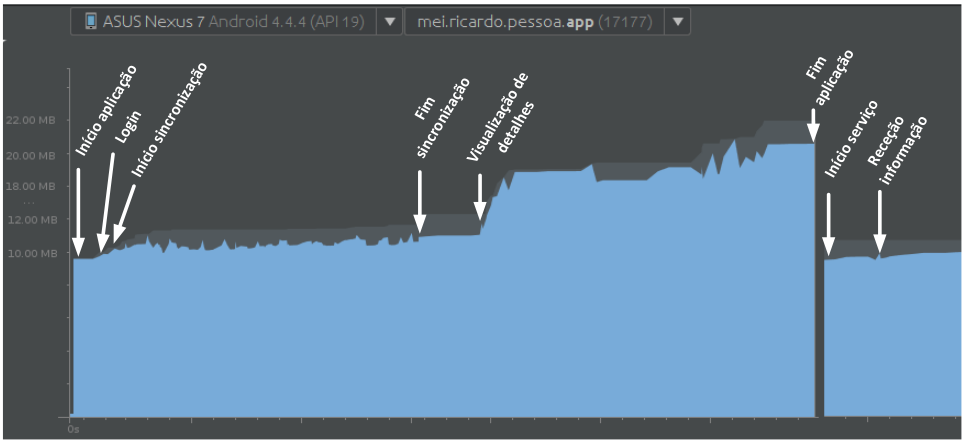


Figura 55 – Memory Monitor: teste à memória RAM

Analisando a figura anterior conclui-se que o início da aplicação consome quase 10 MB de memória RAM. O aumento do consumo de memória não é acentuado na autenticação, contudo, na sincronização da informação verificam-se picos de utilização entre os 10 e 11 *megabytes*. Por outro lado, na visualização dos detalhes da informação verifica-se um aumento significativo, relacionado com a visualização das zonas de seguranças, que implicam carregamento de mapas e imagens. Este consumo situa-se no intervalo dos 15 aos 22 MB. Ao terminar a aplicação, o consumo de memória do serviço em *background* situa-se nos 10 *megabytes*, e no caso de se rececionar alguma nova informação verifica-se um pequeno pico dada a necessidade de validar a informação e notificar o utilizador.

É de ressalvar que este teste tem aplicabilidade em diferentes dispositivos, contudo, os resultados obtidos serão diferentes tendo em consideração as características de cada dispositivo. Assim sendo, por exemplo as dimensões do ecrã ou a capacidade da memória RAM do dispositivo influenciam o resultado final.

As funcionalidades Update Heap e Dump HPROF do Android Device Monitor permitiram verificar quais os objetos mais utilizados e a memória consumida pelos mesmos. Posteriormente, com a ferramenta Eclipse Memory Analyser foi possível criar um relatório com o intuito de visualizar os possíveis erros na gestão de memória. O gráfico seguinte demonstra o consumo de memória RAM dos principais componentes da aplicação.

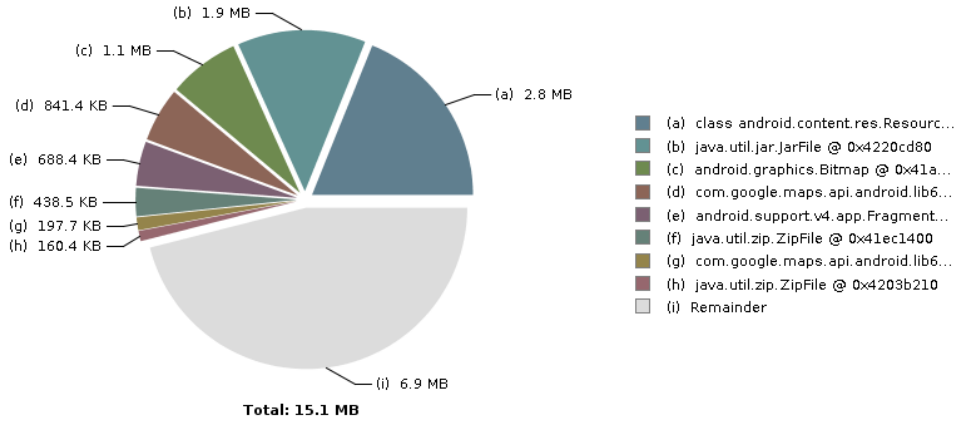


Figura 56 – Teste Eclipse Memory Analyser (MAT)

Através deste gráfico é possível visualizar de onde advém o peso da utilização de memória sendo possível retirar as seguintes conclusões:

- 3,9 MB (alíneas a) e c)) são usados para a apresentação de recurso como imagens, sons, animações, cores, etc;
- 1,9 MB (alínea b)) dizem respeito às várias bibliotecas utilizadas no projeto, principalmente CouchBase Lite;
- 1 MB (alíneas d) e g)) referente à apresentação dos mapas;
- 7 MB (alínea i)) representam os restantes componentes (*remainder*), que apesar de representarem a maior fatia do bolo são demasiado pequenos para serem enumerados, englobando os objetos, *arrays*, classes, entre outros.

Os testes relativos à interface da aplicação não foram esquecidos. Através da ferramenta UI Automator Viewer foi possível visualizar os vários componentes e propriedades que constituem as interfaces, de modo a inspecionar a existência de algum erro.

Na imagem que se segue é possível visualizar a aplicação e a estrutura hierárquica da *view* relativamente ao painel principal.

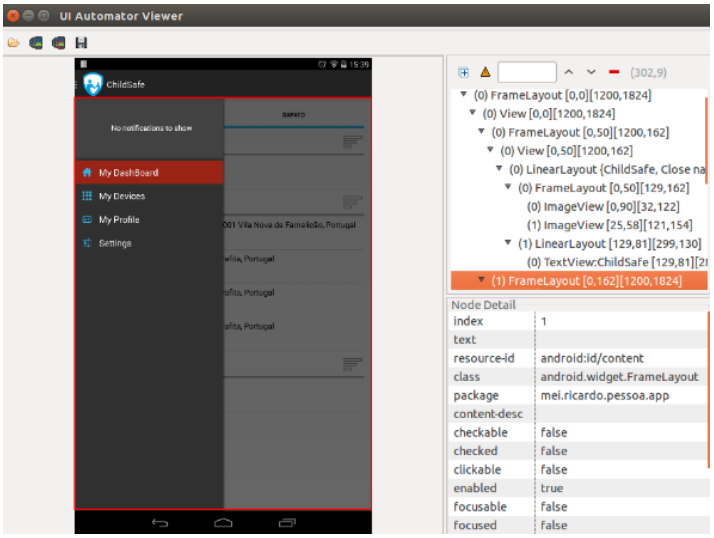


Figura 57 – UI Automator Viewer teste

Em jeito de conclusão, relativamente à interface da aplicação móvel, com o uso dos padrões já intrínsecos nos utilizadores Android é possível arriscar dizer que os utilizadores não sentiram qualquer problema em usar a aplicação desenvolvida no âmbito da dissertação. Do mesmo modo, a interface do *back-end*, está uniformizada em relação à aplicação *mobile*, o que facilita o uso da mesma.

6.2 Conclusões

À medida que os testes foram realizados e o volume de dados armazenado aumentava, notou-se um certo atraso no arranque da aplicação, no entanto, nada muito acentuado, já que, depois deste a aplicação se mostrava fluída durante a sua utilização e a sincronização da informação. Assim sendo, perante estes resultados, de certa forma esperados inicialmente, não se considera a existência de uma limitação por parte da aplicação, mas sim uma otimização a implementar no futuro, com o decorrer do sucessivo aumento do volume de informação.

Por outro lado, quando a aplicação era iniciada pela primeira vez, já com bastantes dados no servidor, notou-se que o processo de sincronização da informação era um pouco moroso. Este teste demonstrou uma limitação que será revertida com a implementação do *Sync Gateway*. Nesse sentido, será possível restringir a quantidade de documentos a sincronizar para os dispositivos móveis. Devido ao pouco tempo disponível decidiu-se não proceder a esse desenvolvimento, contudo será um requisito para o futuro desenvolvimento do projeto.

Relativamente a outros testes, a aplicação mostrou-se bastante estável, sem falhas, recebendo as notificações tanto em *background* como em *foreground* e permitindo visualizar os detalhes da monitorização em modo *online* e *offline*.

Concluindo, ainda que seja necessário refinar algumas funcionalidades, a aplicação cumpre os requisitos do conceito que lhe foi atribuído. Tem-se, assim, em mãos uma solução promissora, servindo como base para futuras implementações e totalmente expansível para qualquer solução.

7 Conclusões

O presente capítulo é reservado às conclusões finais do projeto. Nesse sentido, será feito o balanço dos objetivos cumpridos e não cumpridos, bem como, serão abordadas as limitações que o projeto apresenta e os possíveis desenvolvimentos futuros.

Por último, a apreciação final do projeto de dissertação de mestrado permitirá compreender até que ponto este foi proveitoso.

7.1 Conclusões e análise crítica

O facto de contactar com equipas multidisciplinares no CeNTI, principalmente com pessoas ligadas à eletrónica, enriqueceu os meus conhecimentos ao nível da conceção de *hardware* e desenvolvimento de *firmware,* bem como me despertou uma maior sensibilidade nos problemas relacionados com as limitações do próprio *hardware*.

Durante o projeto de dissertação também estive inserido em vários projetos na empresa, onde tive a possibilidade de trabalhar com várias tecnologias como é o caso de BLE (*Bluetooth Low Energy*), NFC (*Near Field Communications*), entre outras. Por outro lado, o facto de ter trabalhado em vários projetos em paralelo melhorou significativamente as minhas competências de organização e gestão de tempo, análise critica e capacidade de investigação de soluções.

Uma conclusão evidente é que os sistemas de bases de dados NoSQL não vão de maneira alguma substituir os tradicionais modelos relacionais ou mesmo estes cair em desuso. O NoSQL vem complementar em certos casos e, eventualmente, criar alternativas para o armazenamento de dados. Os vários modelos de bases de dados devem coexistir e começam cada vez mais a aparecer soluções híbridas, ou seja, o uso de base de dados relacionais para certas tarefas e não relacionais para outras, na mesma plataforma.

Assim sendo, o aparecimento das bases de dados NoSQL aumentou as alternativas para o armazenamento de informação, ao invés de ser usada sempre a mesma solução, que anteriormente passava pelas bases de dados relacionais.

Esta tecnologia surge no seio dos programadores como uma ferramenta ágil, que acelera todo o processo de desenho do modelo de dados. Esta característica veio verificar-se fulcral para o desenvolvimento da solução apresentada. Apesar de ser uma tecnologia relativamente recente, existem já diversos projetos de sistemas de bases de dados NoSQL, com algum peso de mercado e são cada vez mais adotadas soluções baseadas nesta tecnologia.

No que diz respeito aos obstáculos sentidos, destaca-se o tempo despendido nas sucessivas alterações solicitadas para o *back-end* e alguns desafios que foram surgindo durante o desenvolvimento, que limitaram o tempo necessário ao desenvolvimento da aplicação Android.

Em suma, todos os objetivos propostos no âmbito da dissertação foram realizados com sucesso, havendo obviamente a necessidade de algumas optimizações, como trabalho futuro. Foi então desenvolvida uma plataforma capaz de responder aos requisitos propostos à instituição, servindo como plataforma para futuros desenvolvimentos para dispositivos com sensores integrados, com vista à monitorização remota. A plataforma desenvolvida no âmbito da tese de mestrado é rica em termos tecnológicos, possibilitando a aquisição e aprofundamento de conhecimentos tecnológicos. Assim, de uma maneira global faz-se um balanço positivo do desenvolvimento obtido.

7.2 Trabalho futuro

O trabalho desenvolvido no âmbito da dissertação de mestrado é suscetível a otimizações, bem como adição de novas funcionalidades, devido à pretensão deste projeto ser uma base para o desenvolvimento de *software* com aplicação em casos reais.

Uma funcionalidade bastante importante a ser desenvolvida é a comunicação bidirecional entre os dispositivos de monitorização e o servidor, de forma a ser possível ao utilizador fazer diretamente pedidos ao dispositivo, com o intuito de receber o seu estado atual ou mesmo atuar sobre eles. Também seria interessante adicionar nesses mesmos dispositivos um módulo capaz de realizar chamadas de voz, possibilitando a comunicação dos pais/cuidadores com as crianças, através da aplicação móvel ou mesmo do *browser*.

Seria também benéfico, do ponto de vista da utilização, a hipótese de adicionar zonas de segurança com geometrias mais complexas – polígonos. As zonas de segurança atualmente existentes são circulares, o que acarreta algumas limitações ao delinear zonas de segurança mais específicas. Seria assim também possível adicionar zonas de segurança com forma de retângulos, quadrados, triângulos, entre outros, com intuito de delinear zonas de segurança como por exemplo: rios, ruas perigosas, entre outros. Ainda no seguimento das *safezones*, seria de extrema relevância a caracterização de zonas de seguranças dinâmicas, isto é, temporais e adaptativas.

No caso de zonas de segurança temporais deveria ser possível adicionar zonas de segurança para determinados dias e/ou horas. Caso a criança tivesse um *hobbie* em determinada altura da semana deveria ser possível aos pais/cuidadores saber se a criança chegou em segurança nesses períodos. No caso de zonas adaptativas/inteligentes seria interessante o sistema perceber certos padrões, ou seja, locais habituais. Inicialmente poderia emitir alertas devido a esses desvios não previstos, no entanto, ao longo do tempo adicionaria uma zona de segurança dinâmica.

Possibilitar adicionar a habitação e locais de interesse, funcionalidades de visualização mais detalhadas e interativas como pesquisa por dias, fazendo um histórico de localização, os locais mais visitados, através de um *heat map* por exemplo, são também implementações interessantes a aplicar no futuro.

Outro trabalho essencial no futuro é a implementação de mecanismos de segurança ainda mais fiáveis, nomeadamente, a implementação do protocolo HTTP sobre os protocolos de segurança SSL/TLS. Através do protocolo HTTPS é possível encriptar a informação trocada entre o cliente e servidor e além do mais verificar a autenticidade do servidor e do cliente.

Como foi referido anteriormente, também a implementação do *Sync Gateway* para restringir o número de documentos a sincronizar na aplicação móvel, para melhorar a performance desta.

O tratamento da interface é outro aspeto que deverá ser melhorado. Por outras palavras, seria uma mais-valia explorar e tratar as capacidades gráficas tanto da aplicação móvel como do *back-end*, principalmente no que diz respeito a um conjunto de cores mais atrativas e apelativas dando uma identidade mais forte à plataforma.

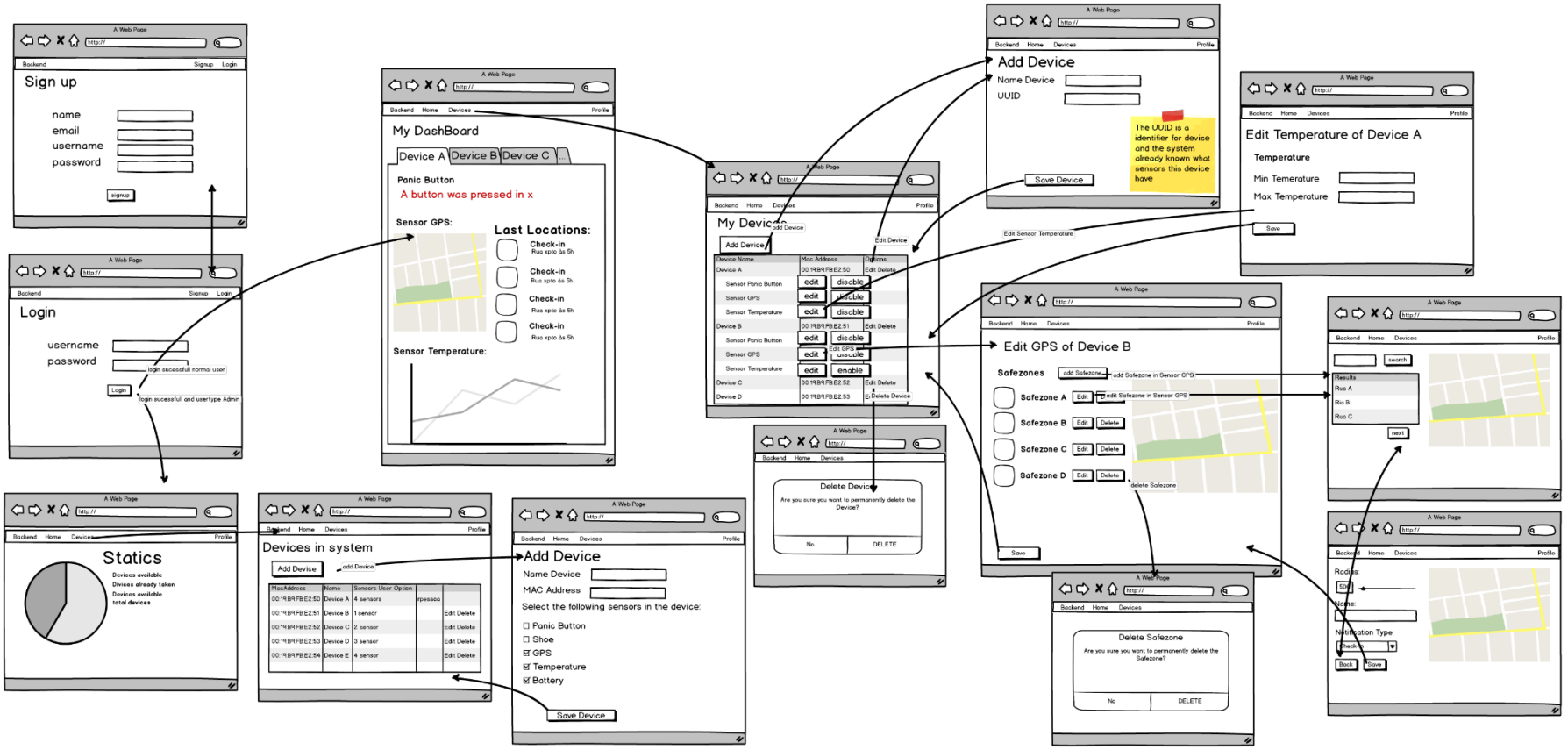
Referências

[Allen,2010]	Allen, J., 2010. InfoQ: A Case for Graph Databases, http://www.infoq.com/news/2010/09/Graph-Databases [último acesso: Julho 2014]
[Anderson, Lehnardt and Slater,2010]	Anderson J., Lehnardt J., Slater N., CouchDB: The Definitive Guide, OReilly Media, Inc., January 26, 2010
[Barrett,2012]	Barrett J., The Internet of Things, Nimbus Centre for Embedded Systems Research, Cork Institute of Technology, 2012, http://www.engineersirelandcork.ie/uploads/The%20internet%20of%20things%20abstract.pdf
[Berthelsen,2014]	Berthelsen E., Why NoSQL databases are needed for the Internet of Things, April 2014
[Brewer,2000]	Dr. Brewer E., Towards Robust Distributed Systems, Professor, UC Berkeley Co--Founder & Chief Scientist, Inktomi 2000, www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf [último acesso: Julho 2014]
[Brown,2002]	Brown F., A Brief History of Modern RDBMS IT Management Article 2: Emergence of the DBMS,2002, http://www.mountainman.com.au/software/history/it2.html [último acesso: Outubro 2014]
[Cardoso,2012]	Cardoso R., Bases de Dados NoSQL, Instituto Superior de Engenharia do Porto, Outubro 2012
[Chandler, 2009]	Chandler C., CouchDB in Action, Early access edition, Manning Publications April 2009
[Chapple,2012]	Chapple M. Abandoning ACID in Favor of BASE, Changes to the long-held relational model http://databases.about.com/od/otherdatabases/a/Abandoning-Acid-In-Favor-Of-Base.htm [último acesso: 16 Agosto 2014].
[Codd,1970]	Codd E., A Relational Model of Data for Large Shared Data Banks , IBM Research Laboratory, San Jose, California, 1970
[CouchBase,2012a]	Couchbase Mobile, Building Always-Available, Always-Responsive Apps http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/couchbaseMobileWhitepaper.pdf [último acesso: 20 Outubro 2014]
[CouchBase,2014b]	Couchbase Mobile, Why NoSQL? http://info.couchbase.com/rs/northscale/images/whyNoSQL.whitePaper.FINAL.pdf [último acesso: 22 Outubro 2014]
[Dalakov,2011]	Dalakov G., The Relational Databases of Edgar Codd, http://history-computer.com/ModernComputer/Software/Codd.html [último acesso: 23 Outubro 2014]
[DuVander,2010]	DuVander A., Map Scripting 101, San Francisco, 2010
[Erlingsson,1996]	Erlingsson Ú., Chapter 28: Object-Oriented Query Languages, 1996, http://www.cs.cornell.edu/home/ulfar/oodbms/ooqlang.html [último acesso: 23 Outubro 2014]
[Fielding,2000]	Fielding T., Architectural Styles and the Design of Network-based Software Architectures, university of california, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [último acesso: 13 Junho 2014]
[Fowler and Sadalage,2012]	Fowler M., Sadalage P., NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence, August 2012.
[Gilbert and Lynch,2002]	Gilbert S., Lynch L., Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, June 2002.

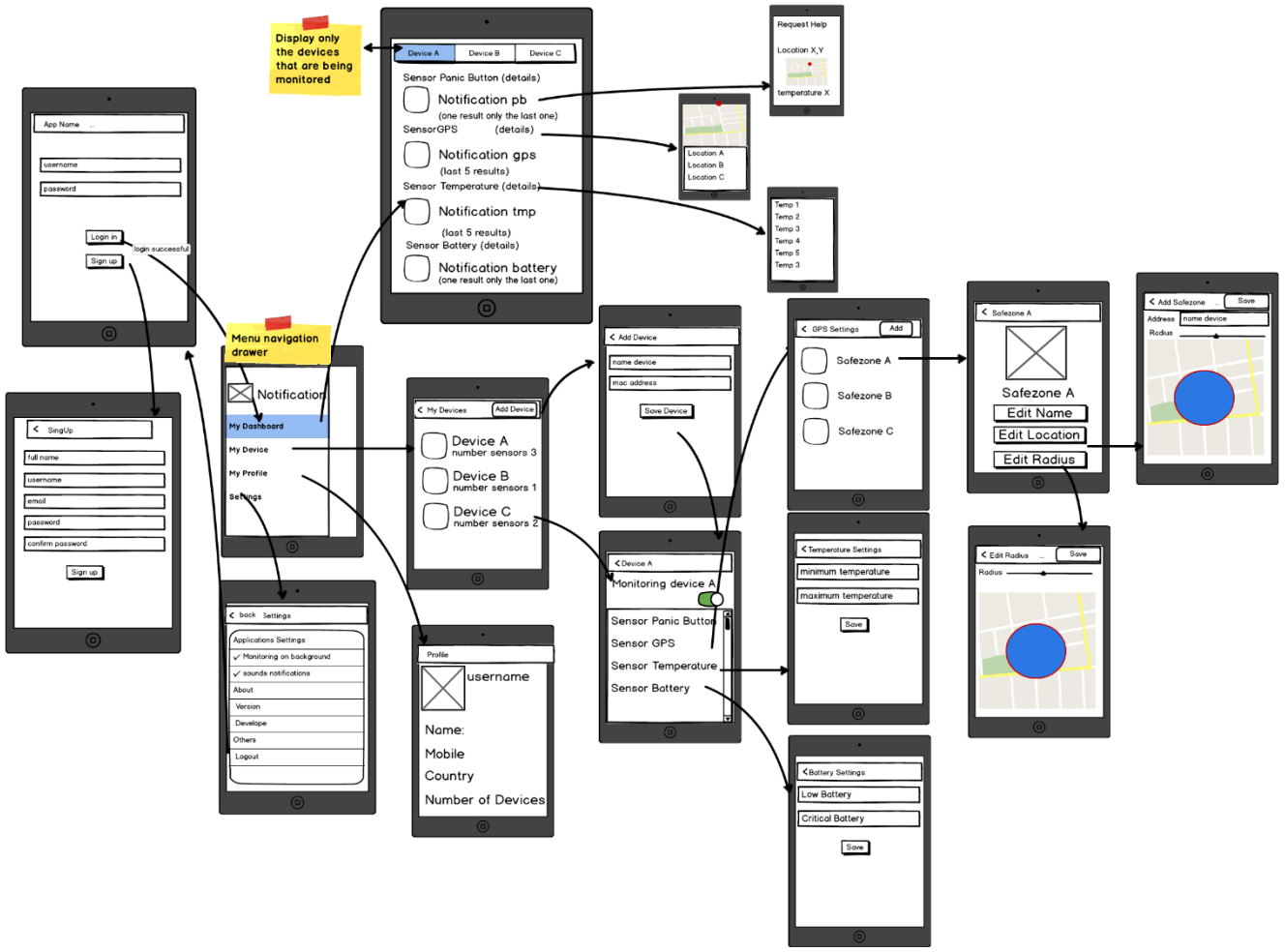
[Gousios, Vasilescu, Serebrenik and Zaidman, 2014]	Gousios G., Vasilescu B., Serebrenik A., Zaidman A., Lean GHTorrent: GitHub Data on Demand, http://flosshub.org/sites/flosshub.org/files/lean-ghtorrent.pdf [último acesso: 20 Outubro 2014]
[Hecht and Jablonski,2011]	Hecht R., Jablonski S., NoSQL Evaluation A Use Case Oriented Survey, 2011
[Holt,2011]	Holt B. Writing and Querying MapReduce Views in CouchDB, Published by O'Reilly Media, Inc., February 2011
[Horrocks,2001]	Horrocks I., Relational Data Model, http://www.cs.man.ac.uk/~horrocks/Teaching/cs2312/Lectures/Handouts/Relational.pdf [último acesso: 23 Outubro 2014]
[Jacobs,2009]	Jacobs A., The Pathologies of Big Data, Communications of the ACM, August 2009
[Pokorny,2011]	Pokorny J., NoSQL Databases: a step to database scalability in Web environment, Charles University, Malostranske, Czech Republic,2011
[Pritchett,2008]	Pritchett D., BASE: An Acid Alternative, Magazine Queue - Object-Relational Mapping, May/June 2008
[Redmond and Wilson, 2012]	Redmond E., Wilson J.,Seven Databases in Seven Weeks, A Guide to Modern Databases and the NoSQL Movement, The Pragmatic Bookshelf, 2012
[Reeve,2012]	Reeve A., Big Data and NoSQL: The Problem with Relational Databases, September 7, 2012, https://infocus.emc.com/april_reeve/big-data-and-nosql-the-problem-with-relational-databases/ [Acedido em 25 Agosto 2014].
[Turner, Reinsel, Gantz and Minton,2014]	Turner V., Reinsel D, Gantz J., Minton S., The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, EMC Corporation, Abril 2014
[Viswanathan,2013]	Viswanathan P., Mobile Devices Expert Native Apps vs. Web Apps – What is the Better Choice?, 2013, http://mobiledevices.about.com/od/additionalresources/a/Native-Apps-Vs-Web-Apps-Which-Is-The-Better-Choice.htm [último acesso 25 Outubro 2014]
[WHATWG, 2008]	HTML Living Standard, http://www.whatwg.org/specs/web-apps/current-work/#history-2 [último acesso 20 Outubro 2014]
[Zikopoulos and Eaton, 2012]	Zikopoulos P., Eaton C., Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data,2012

8 Anexos

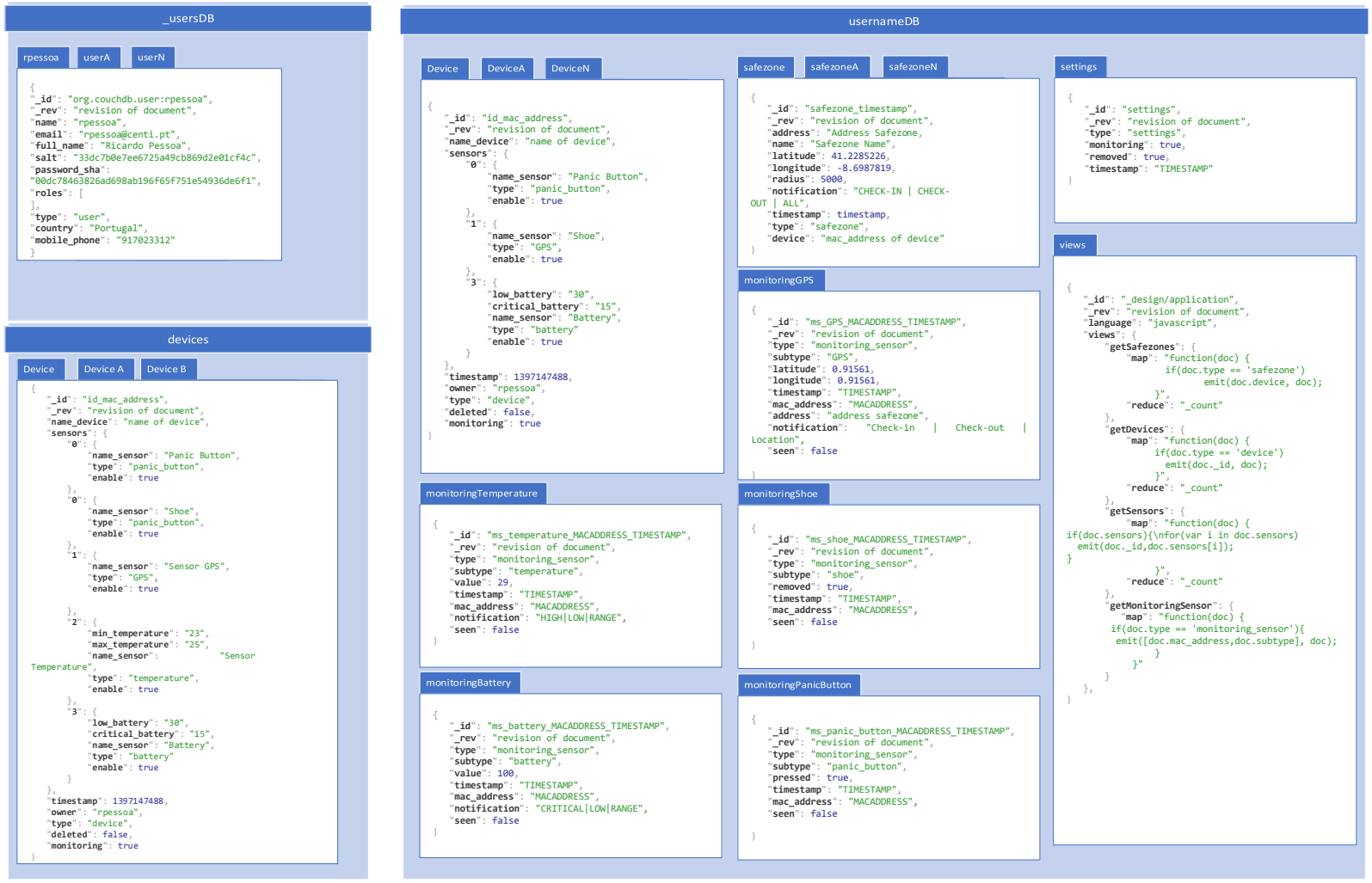
8.1 Mockup back-end



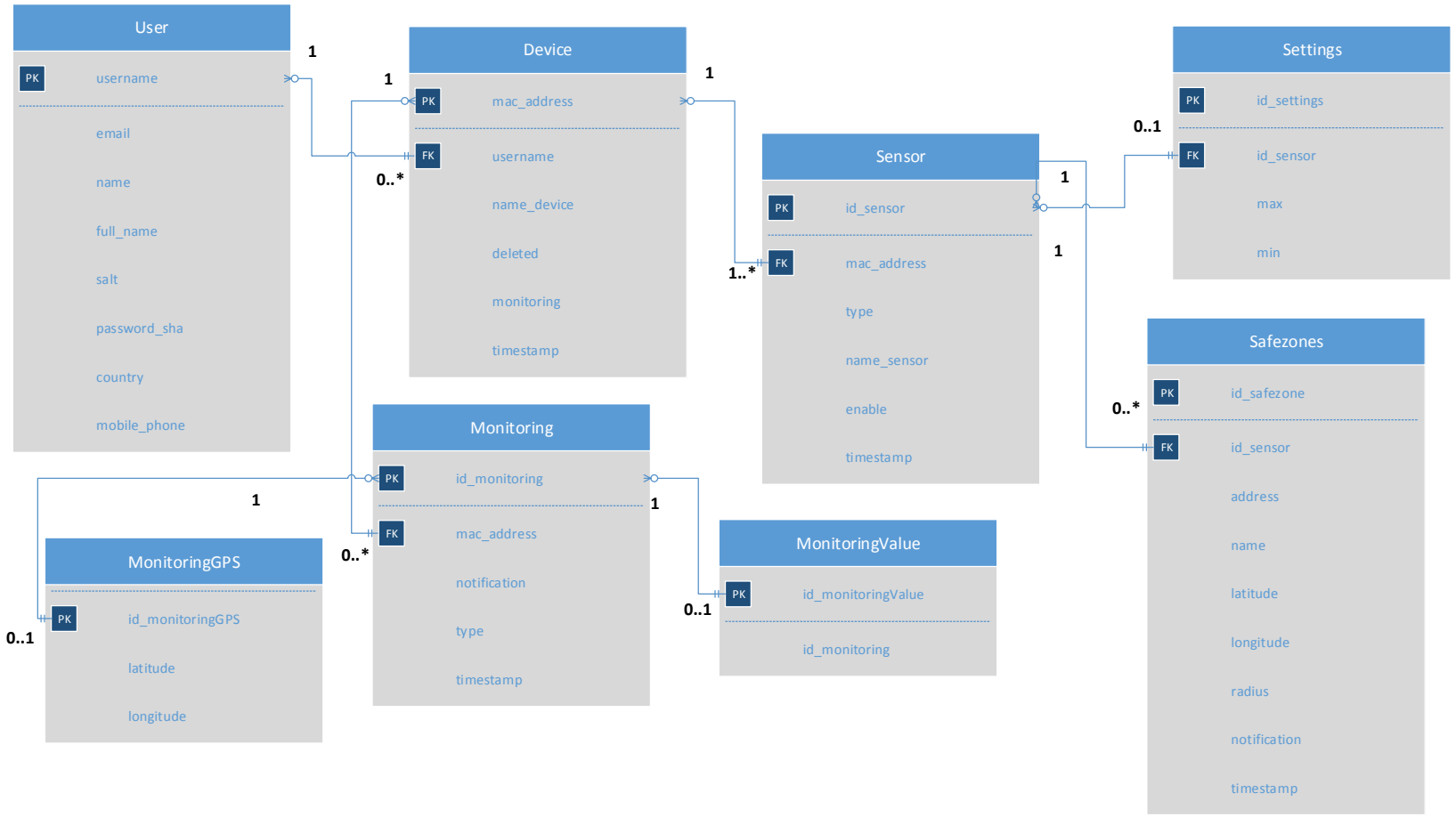
8.2 Mockup aplicação móvel - Android



8.3 Modelo de dados - NoSQL document



8.4 Modelo de dados - Relacional



8.5 Diagrama de classes simplificado