



Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Diogo dos Santos Mendes

DESENVOLVIMENTO DE INFRAESTRUTURA DE SOFTWARE DO PROJETO VITASENIOR

Relatório de Projeto de Mestrado

Orientado por:

Professor Renato Panda, Instituto Politécnico de Tomar
Professor Luís Oliveira, Instituto Politécnico de Tomar

Relatório apresentado ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em
Engenharia Informática – Internet das Coisas

“Se houver esforço, há sempre realização”

Kanō Jigorō

RESUMO

O VITASENIOR-MT é uma plataforma de teleassistência, desenvolvida especificamente para a população idosa, que permite monitorar remotamente os dados biométricos e ambientais dos pacientes no seu conforto doméstico. Estes dados são transferidos para a infraestrutura na nuvem por intermédio de um dispositivo, Vitabox, que permite também a interação dos pacientes com o sistema através da televisão.

Este relatório foca-se na componente de computação em nuvem, onde se encontram os serviços de processamento e armazenamento dos valores recolhidos, bem como a emissão de alertas e gestão do equipamento.

A solução pretende transferir, processar e armazenar os valores recolhidos pelos sensores associados às Vitaboxes e aos respetivos pacientes, seguindo uma arquitetura de microserviços, que promove escalabilidade computacional, um melhor isolamento de falhas e fácil integração entre os serviços.

O sistema é também provido de um portal, onde podem ser registados médicos e cuidadores, garantindo o acompanhamento dos pacientes.

A abordagem apresentada garante a segurança de dados pessoais e uma forma simplificada de recolher e apresentar os dados aos diferentes atores sem comprometer a performance do sistema.

Os testes realizados comprovaram que esta solução é mais eficiente que uma abordagem monolítica, promovendo melhor acesso e controlo nos dados provenientes de equipamentos heterogêneos.

ABSTRACT

VITASENIOR-MT is a telehealth platform designed specifically for the elderly population, which allows to remotely monitor biometric and environmental data in their home comfort. These data are transferred through a device, the Vitabox, which also allows patients to interact with the system through the television.

This work is focused on the cloud computing component, where the processing and storage services are defined, as well as the alerts issuance and the equipment management.

The solution is able to transfer, process and store the data collected by the sensors related with the Vitaboxes and their patients, following a microservice architecture, which promotes computational scalability, better faults isolation and easier integration.

The system also provides a web portal, where doctors and caregivers can register and easily follow their patients' status.

This approach provides personal data security to the patients and a simplified way of collecting and presenting the data to the different actors without compromising the system's performance.

The tests performed demonstrated that this solution is more efficient than a monolithic approach, promoting better access and control to the data coming from heterogeneous equipment.

AGRADECIMENTOS

A finalizar esta etapa do meu percurso académico, marcado por momentos de superação e reinvenção de mim próprio, não poderia deixar de dar algumas palavras de apreço.

Em primeira instância gostaria de agradecer aos meus pais Paula Bastos dos Santos Mendes e Leonel Ferreira Mendes, que foram e continuam a ser os meus ídolos, símbolo de sacrifício e dedicação por algo em que acreditam. Durante estes anos acreditaram em mim e pretendo retribuí-los em dobro. A eles que foram a minha motivação.

Em segundo lugar à minha namorada Inês Silva Santos, por me acompanhar, ouvir os meus desabafos, partilhar as minhas conquistas e lidar comigo nos dias mais complicados. A ela que foi a minha casa.

Os meus sinceros agradecimentos aos meus orientadores, ao professor Renato Panda, pelo seu acompanhamento incensurável no meu trabalho e esforço em procurar respostas e descobrir soluções às minhas questões e ao professor Luís Oliveira por todas as questões que me colocou ainda antes de eu pensar que eram questões e pelos contactos estabelecidos que me agilizaram o desenvolvimento. A eles que foram o meu suporte.

Agradeço também aos meus colegas e amigos bolseiros Pedro Dias, Dário Jorge, Pedro Nunes e Ricardo António, que tornaram o ambiente de trabalho muito melhor. A eles que foram a minha alegria.

Por fim agradeço também a toda a equipa que falta do Vitasenior, ao professor Ricardo Campos e ao André Carvalho por se mostrarem dispostos a partilhar o seu conhecimento em momento de necessidade, ao professor Gabriel Pires pelo seu acompanhamento ao longo do projeto e pela ajuda na escrita do artigo, à professora Ana Lopes pelo seu acompanhamento ao longo do mestrado e esforço por atender aos meus pedidos mais complicados e ouvir as minhas críticas e a todos os meus colegas de Mestrado pela sua companhia e boa disposição durante as aulas e horas de trabalho que se realizavam tão tarde.

Este projeto foi suportado financeiramente pelo projeto IC&DT VITASENIOR-MT CENTRO-01-0145-FEDER-023659 com fundos do FEDER através dos programas operacionais CENTRO2020 e FCT.

ÍNDICE

| | |
|---|------|
| RESUMO | iii |
| ABSTRACT | v |
| AGRADECIMENTOS | vii |
| ÍNDICE..... | ix |
| ÍNDICE DE FIGURAS | xiii |
| ACRÓNIMOS | xv |
| Capítulo 1 Introdução | 1 |
| 1.1. Motivação | 1 |
| 1.2. Solução proposta..... | 6 |
| 1.2.1. Objetivo do projeto VITASENIOR-MT | 6 |
| 1.2.2. Visão geral do projeto..... | 6 |
| 1.2.3. Descrição da solução | 8 |
| 1.2.4. Organização do relatório | 11 |
| Capítulo 2 Contextualização tecnológica | 13 |
| 2.1. Computação em Nuvem..... | 13 |
| 2.2. Internet das Coisas | 14 |
| 2.3. Integração de Computação em Nuvem com a Internet das Coisas..... | 16 |
| 2.3.1. Desafios de integração..... | 16 |
| 2.3.1.1. <i>Cloudlet</i> | 18 |

| | |
|---|----|
| 2.3.1.2. <i>Fog Computing</i> | 19 |
| 2.3.1.3. Mobile Sink..... | 20 |
| Capítulo 3 Estado da arte | 21 |
| 3.1. Soluções de Computação em Nuvem para Internet das Coisas..... | 21 |
| 3.2. Desenvolvimento na área da e-saúde e teleassistência..... | 26 |
| Capítulo 4 Trabalho realizado | 31 |
| 4.1. Modo de funcionamento da solução..... | 31 |
| 4.2. Arquitetura Geral..... | 32 |
| 4.2.1. Microserviços desenvolvidos | 34 |
| 4.2.1.1. API..... | 35 |
| 4.2.1.2. WebSocket | 35 |
| 4.2.1.3. Worker | 35 |
| 4.2.1.4. Schedule..... | 36 |
| 4.2.1.5. Peer | 36 |
| 4.2.2. Estrutura de cada microserviço | 36 |
| 4.3. Ambientes de desenvolvimento e produção | 37 |
| 4.4. Escalabilidade..... | 40 |
| 4.5. Armazenamento..... | 45 |
| 4.6. Segurança..... | 48 |
| 4.7. Acesso..... | 51 |

| | |
|--------------------------------------|----|
| Capítulo 5 Testes e resultados | 59 |
| Capítulo 6 Conclusão | 63 |
| 6.1. Trabalho futuro | 64 |
| Bibliografia..... | 65 |

ÍNDICE DE FIGURAS

| | |
|---|----|
| Fig. 1 - Crescimento da percentagem de população com 65 anos ou mais entre 2007 e 2017 [2]. | 2 |
| Fig. 2 - Projeção da taxa de dependência idosa de 2017 a 2080 na União Europeia [2]..... | 3 |
| Fig. 3 - Causas de morte em 2014 [5]. | 4 |
| Fig. 4 - Percentagem de população com limitações na atividade diária [6]. | 5 |
| Fig. 5 - Arquitetura geral do VITASENIOR-MT..... | 7 |
| Fig. 6 - Esquema de acessos dos utilizadores..... | 9 |
| Fig. 7 - Esquema de relações entre as entidades do VITASENIOR-MT. | 10 |
| Fig. 8 - Adoção de IoT por segmento e região [12]. | 15 |
| Fig. 9 - Arquiteturas de camadas de abstração entre <i>Cloud</i> e dispositivos [18]..... | 18 |
| Fig. 10 - Arquitetura do UT-GATE [21]. | 20 |
| Fig. 11 - Google Cloud IoT construção em blocos [30]. | 24 |
| Fig. 12 - Arquitetura da ThingSpeak [33]. | 25 |
| Fig. 13 - Interface gráfica de projeto de monitorização de saúde na Índia [36]. | 27 |
| Fig. 14 - Arquitetura do CardiaQcloud [37]. | 28 |
| Fig. 15 - Fluxo de instalação da Vitabox na casa do utente. | 32 |
| Fig. 16 – Arquitetura geral da <i>Cloud</i> | 34 |
| Fig. 17 - Comparação entre máquinas virtuais e <i>containers</i> [48]. | 38 |
| Fig. 18 – Registos de falha de serviço por <i>Bad Gateway</i> | 39 |

| | |
|--|----|
| Fig. 19 - Evolução do paradigma da <i>Cloud</i> | 42 |
| Fig. 20 - Tipos de filas de mensagens do RabbitMQ [50]. | 43 |
| Fig. 21 – Associação de WebSockets a Filas de mensagens (Publish/Subscribe). | 44 |
| Fig. 22 - Limite de memória nas bases de dados. | 47 |
| Fig. 23 - Documentação da API. | 52 |
| Fig. 24 - Anotações do código fonte para documentação. | 53 |
| Fig. 25 - Dicionário de tradução..... | 54 |
| Fig. 26 - Instalação da PWA. | 56 |
| Fig. 27 - Execução da PWA. | 56 |
| Fig. 28 - Testes automatizados com Mocha. | 59 |
| Fig. 29 - Teste de carga à API (envio de dados de sensores) com abordagem monolítica . | 60 |
| Fig. 30 - Teste de carga em microserviços..... | 61 |
| Fig. 31 - Teste de usabilidade por utente com auxílio de enfermeira. | 62 |

ACRÓNIMOS

| | |
|--------|--|
| API | Application Programming Interface |
| CoAP | Constrained Application Protocol |
| GDPR | General Data Protection Regulation |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Secure HTTP |
| IDE | Integrated Development Environment |
| IoT | Internet of Things |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| JWT | JSON Web Token |
| LAN | Local Area Network |
| MQTT | Message Queuing Telemetry Transport |
| ODM | Object-Documento Model |
| ORM | Object-Relational Model |
| P2P | Peer-to-Peer |
| RBCA | Role-Based Control Access |
| REST | Representational State Transfer |
| RFID | Radio-Frequency Identification |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| WebRTC | Real Time Communication over the Web |
| WBAN | Wireless Body Area Network |
| WS | WebSocket |
| WSS | Wireless Sensor Network |
| XMPP | Extensible Messaging and Presence Protocol |

Capítulo 1

Introdução

Nas últimas décadas tem-se verificado um aumento da taxa de envelhecimento da população e um aumento do número de casos de idosos isolados, resultado da evolução da estrutura da família, em que os jovens são mais independentes dos patriarcas. Este fenómeno justifica a necessidade de soluções de monitorização e teleassistência desta população envelhecida e isolada. Este trabalho tem por objetivo contribuir com uma nova solução para este problema.

Neste capítulo é apresentada em a motivação e objetivos deste projeto, uma descrição geral do projeto e a solução proposta.

1.1. Motivação

Atualmente a Europa vive o maior envelhecimento de população da sua história, um fenómeno que se começou a sentir no fim do século passado. Se em 1950 nenhum país tinha uma taxa de população idosa (com 65 anos ou mais) superior a 11%, em 2000 esse valor atingiu os 18% e prevê-se que atinja os 28% em 2050 [1]. Segundo relatórios da Comissão Europeia de 2017, na última década, a taxa de envelhecimento cresceu em média 2,4%, sendo que esse efeito foi mais acentuado em Portugal com uma pontuação média de 3,6% (Fig. 1), em que a idade média da população aumentou em 4 anos [2].

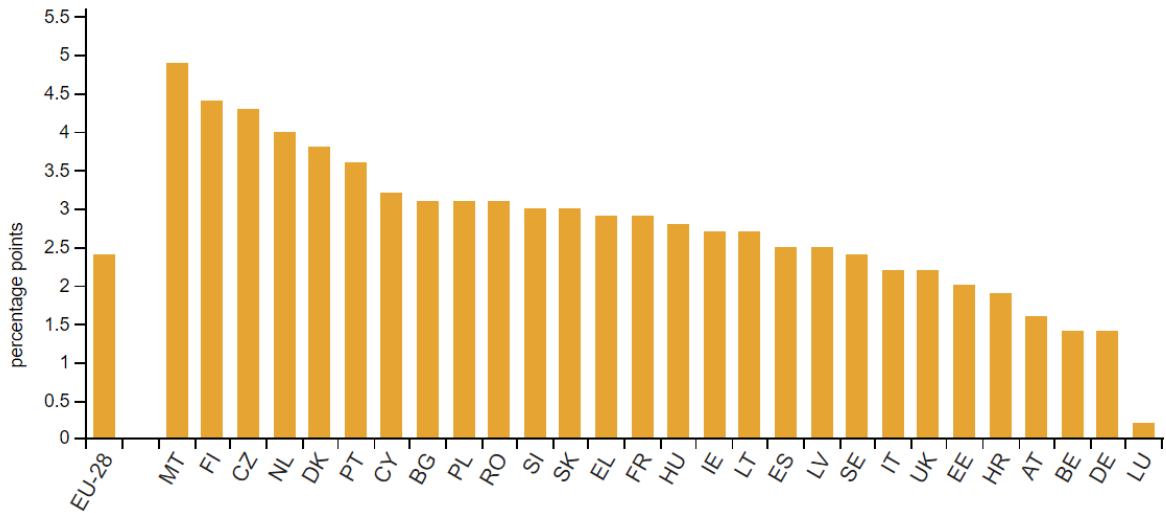


Fig. 1 - Crescimento da percentagem de população com 65 anos ou mais entre 2007 e 2017 [2].

Este fenómeno é causado pela diminuição das taxas de natalidade e pelo aumento contínuo da esperança média de vida, motivado pelos avanços da medicina e dos serviços nacionais de saúde, que em Portugal era inexistente até à segunda metade do século XX [3]. Desta forma a proporção de população dependente idosa em relação à população ativa atingiu 29,9% em 2017, ou seja, a responsabilidade social de cada idoso incide em aproximadamente 3 trabalhadores.

Considerando o ritmo atual, prevê-se que em 2050 seja atingido o pico de população na Europa (jovem, ativa e idosa), seguindo-se de um contínuo declínio, que levará ao agravamento da taxa de dependência idosa para 52,3% em 2080 (Fig. 2).

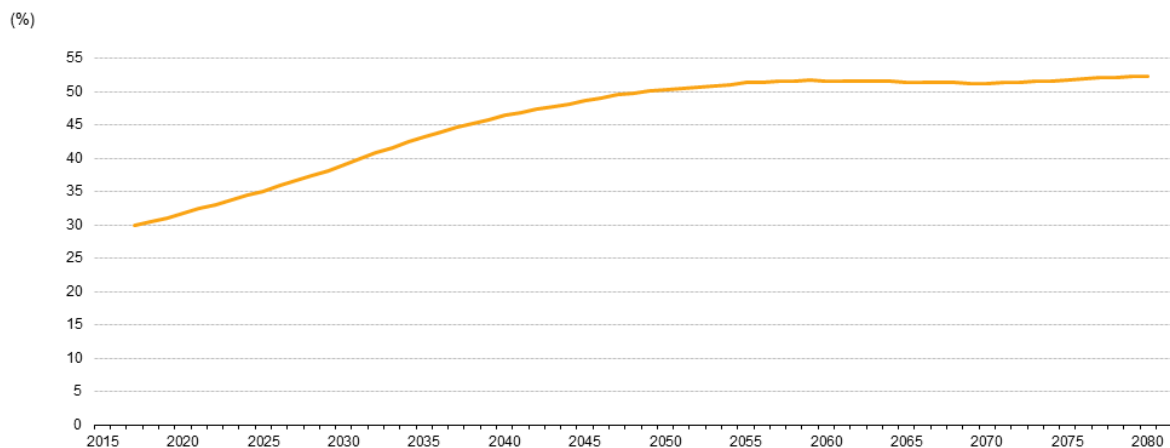


Fig. 2 - Projeção da taxa de dependência idosa de 2017 a 2080 na União Europeia [2].

Uma vez que os idosos serão mais numerosos que as crianças e que a responsabilidade social sobrecarregará um número cada vez menor de população ativa serão visíveis as alterações na natureza da sociedade [1].

Algumas das consequências desta tendência são já visíveis, tais como os crescentes casos de abandono e isolamento de idosos. Em 2017 existiam 28279 idosos a viver sozinhos, representando mais de metade da população idosa sinalizada e mais 8,2% que no ano anterior [4].

Ao crescimento da taxa de envelhecimento está também associado o elevado número de acidentes cardiovasculares na população. Embora tenha vindo a diminuir nos últimos 30 anos por toda a União Europeia, no caso de Portugal representou quase 31% das causas de morte em 2014, sendo superada apenas pelo cancro no sexo masculino (Fig. 3).

| | | Todas as causas | Acidente vascular cerebral | Acidente vascular encefálico | Outros acidentes cardiovasculares | Cancro do estômago | Cancro colorretal | Cancro do pulmão | Cancro da mama | Outros cancros | Problemas respiratórios | Lesões e envenenamentos | Outras causas |
|----------|----------|-----------------|----------------------------|------------------------------|-----------------------------------|--------------------|-------------------|------------------|----------------|----------------|-------------------------|-------------------------|---------------|
| Portugal | Homens | 53498 | 4178 | 5117 | 5279 | 1382 | 2200 | 3084 | 22 | 9246 | 6308 | 3151 | 13531 |
| | Mulheres | 51721 | 3278 | 6691 | 7743 | 911 | 1608 | 853 | 1664 | 5772 | 5856 | 1667 | 15678 |
| UE | Homens | 2471355 | 335517 | 176504 | 331854 | 35631 | 83962 | 184659 | 943 | 442917 | 208509 | 147383 | 523476 |
| | Mulheres | 2499396 | 297411 | 249539 | 457649 | 23279 | 68535 | 84541 | 91892 | 322601 | 185143 | 82328 | 636478 |
| Europe | Homens | 4475990 | 862219 | 405415 | 535495 | 71251 | 118704 | 276598 | 1464 | 615505 | 325008 | 391957 | 872374 |
| | Mulheres | 4370306 | 877216 | 583158 | 677162 | 47276 | 105128 | 105679 | 137596 | 463554 | 251279 | 157217 | 965041 |

Fig. 3 - Causas de morte em 2014 [5].

No entanto não são apenas os casos de acidentes cardiovasculares que preocupam, pois embora a população na Europa, em média, viva hoje mais anos, estes têm cada vez mais doenças crónicas associadas aos, cada vez piores, hábitos alimentares. Entre estes, destacam-se o excessivo consumo de álcool e outras drogas, diminuição de atividade física e aumento da exposição à poluição. Assim, Portugal é um dos países com menos de 50% de população adulta reportada como saudável, sendo agravado pelo facto de mais de 25% dessa população ter limitações para a prática das suas atividades diárias [6].

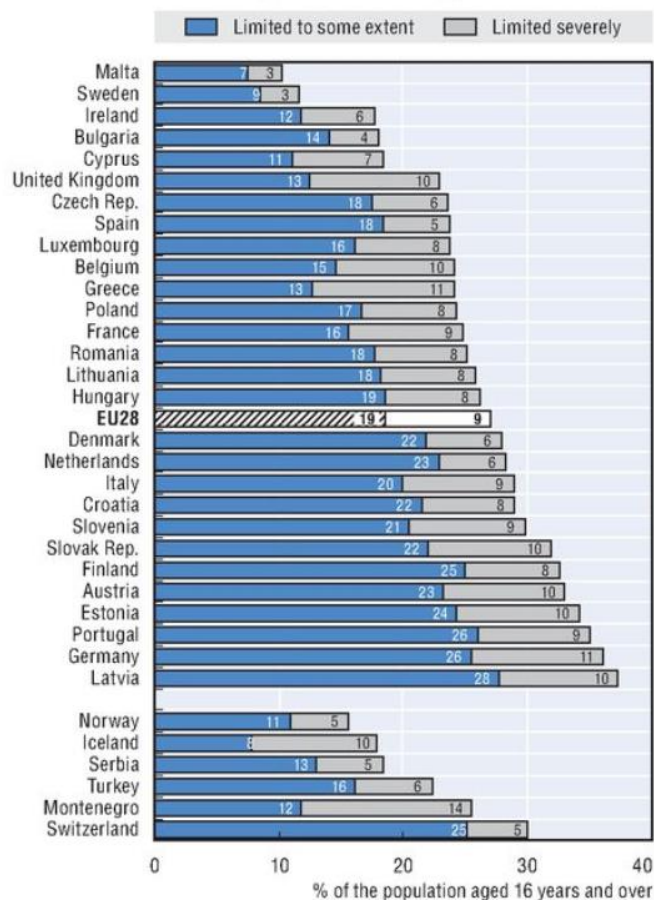


Fig. 4 - Percentagem de população com limitações na atividade diária [6].

Com base nestes estudos percebemos que os países desenvolvidos encontram-se cada vez mais envelhecidos e que esta população exige mais cuidados e acompanhamento, que por sua vez necessita de uma forte presença humana. No entanto, esta tende a escassear a longo prazo, pois serão cada vez menos os trabalhadores ativos e capacitados em relação à população dependentes.

Para responder aos problemas identificados tem sido realizado um investimento, associado ao plano de crescimento inteligente, sustentável e inclusivo da Europa 2020 [7]. Este investimento tem dado origem a diversos projetos com a missão de facultar melhores cuidados de saúde e acompanhamento social, a menor custo, através de metodologias de prevenção e monitorização suportadas por recursos tecnológicos.

1.2. Solução proposta

No sentido de atenuar os impactos sociais da evolução demográfica apresentada foi proposto o projeto VITASENIOR-MT, uma solução de teleassistência com foco na população idosa da região do Médio Tejo [8] descrita de forma geral nas secções seguintes.

1.2.1. Objetivo do projeto VITASENIOR-MT

O projeto VITASENIOR-MT pretende assegurar o acompanhamento de idosos isolados e pessoas incapacitadas para a prevenção de acidentes, que possam ocorrer em ambiente doméstico, de forma automatizada, através da monitorização do ambiente envolvente e do estado biométrico dos seus pacientes, oferecendo assim maior autonomia e segurança no seu quotidiano.

O projeto caracteriza-se pela sua natureza holística que se adequa a várias necessidades, podendo ser integrado em qualquer ambiente doméstico (por exemplo uma casa ou um lar) e para qualquer estrato social. Para além do idoso, o sistema poderá também ser usado por crianças, jovens e adultos, uma vez que a solução foi desenvolvida com vista ao pouco conhecimento tecnológico por parte do utilizador.

1.2.2. Visão geral do projeto

A solução final agrega os dados recolhidos por sensores ambientais organizados numa rede de sensores sem fios (WSN), de uma pulseira desportiva e de dispositivos médicos como uma balança, um glicosímetro ou um esfigmomanómetro por meio de *Bluetooth*. De seguida estes dados são modelados e guardados remotamente para posterior análise, que despoleta avisos em situações anómalas. O sistema permite também a visualização dos mesmos por parte de entidades competentes como um médico ou um familiar. Uma vez que se tratam de

dados pessoais, todo o processo de transporte, alojamento e acesso a estes é devidamente validado e seguro, de acordo com as normas vigentes para dados sensíveis.

O sistema está dividido em 3 componentes distintas (Fig. 5):

1. Rede de sensores;
2. Vitabox – dispositivo que interliga os sensores e televisão;
3. *Cloud* – infraestrutura de armazenamento e processamento remoto.

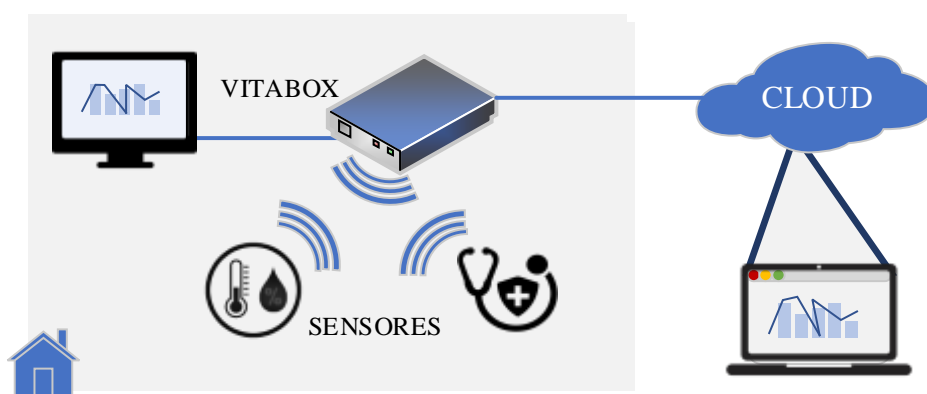


Fig. 5 - Arquitetura geral do VITASENIOR-MT.

Na componente referente à rede de sensores são implementados mecanismos de comunicação entre os sensores na WSN, de recolha e transporte dos dados obtidos e de aceitação/remoção de dispositivos na rede. Nesta componente são também interpretados comandos *Bluetooth* para a extração de dados dos equipamentos médicos e da pulseira.

A Vitabox apresenta-se como um o *gateway*¹ que suporta a interoperabilidade entre a *Cloud* e a rede de sensores. Isto é, estabelece a ligação com ambos os pontos e encaminha os dados, previamente modelados, para a *Cloud*, aplicando assim o conceito de *Fog Computing*. Esta exerce também a interação com o paciente por meio da televisão e do comando da mesma,

¹ Dispositivo que serve de interface entre duas redes distintas.

sendo esta funcionalidade de importância fulcral para aumentar a usabilidade da solução e com isto facilitar a aceitação do sistema por parte do idoso, pois é um dispositivo com o qual este já se encontra familiarizado.

Por fim, a *Cloud* guarda os dados, valida a autenticidade dos acessos, gere os equipamentos e pacientes definidos em cada Vitabox, procura anomalias nos valores recebidos e envia alertas para situações de risco. Na *Cloud* existe a interface gráfica de configuração de equipamento e visualização de dados.

1.2.3. Descrição da solução

O foco deste relatório será a componente de *Cloud*, apresentando a arquitetura desenvolvida, com a devida fundamentação para cada decisão tomada ao longo deste processo, apresentando as dificuldades encontradas e os resultados obtidos.

Um dos critérios para a implementação do projeto foi a utilização da plataforma IBM Cloud [9] para alojar os serviços do projeto VITASENIOR-MT. Esta opção é justificada pela cooperação entre o Instituto Politécnico de Tomar e a empresa Softinsa² para o desenvolvimento da referida solução. Deste modo será necessário conhecer as soluções oferecidas, os modelos de negócio aplicados e as implicações da integração do *software* na plataforma.

A *Cloud* deverá disponibilizar serviços para que seja possível à Vitabox obter a lista do equipamento e pacientes a ela associados e submeter os valores dos sensores. No momento da receção dos dados dos sensores, estes deverão ser imediatamente submetidos a análise, a fim de averiguar eventuais anormalidades. Caso se detetem valores fora do intervalo que se entende como aceitável, a Vitabox e os utilizadores relacionados com a própria (podendo ser um médico ou um familiar), ou o paciente em causa, deverão ser imediatamente alertados.

² Empresa do grupo IBM.

Para a visualização dos dados recolhidos, receção de alertas e configuração de equipamentos foi desenvolvida uma aplicação web, para que os utilizadores possam aceder remotamente. Os utilizadores da aplicação web são categorizados em 4 papéis possíveis (Fig. 6):

- Responsáveis clínicos (será usado o termo “médico” para esta função ao longo do relatório);
- Responsáveis da Vitabox (será usado o termo “patrono” para esta função ao longo do relatório);
- Cuidadores;
- Administradores.

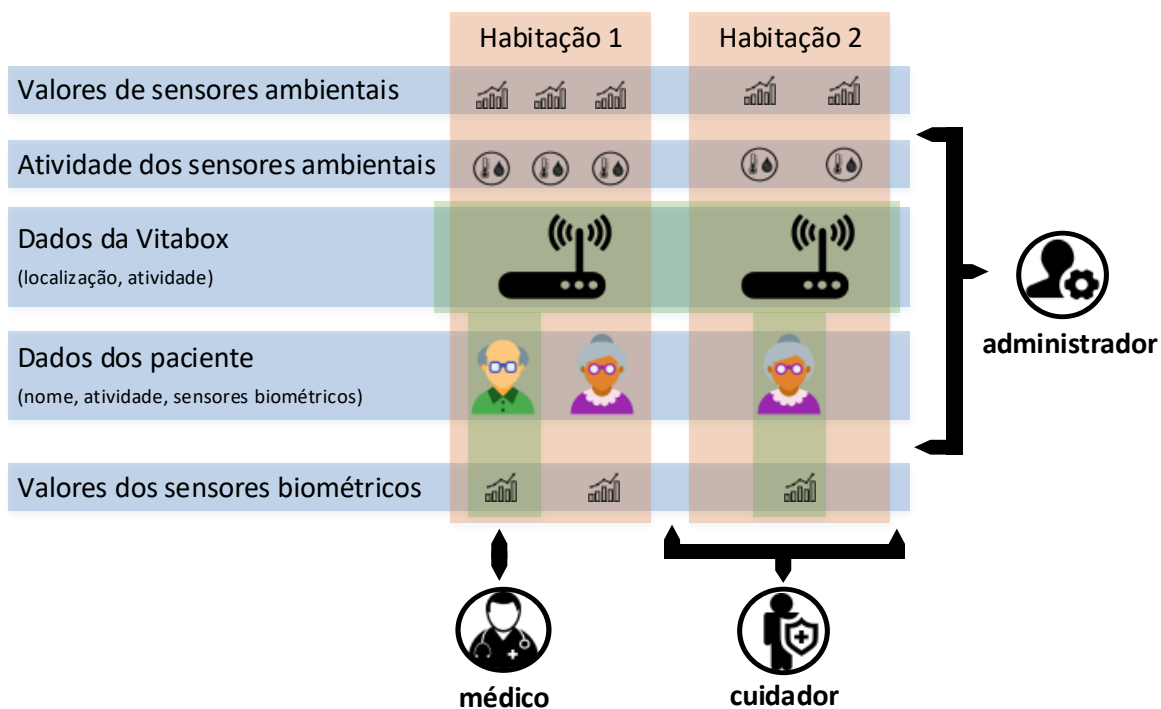


Fig. 6 - Esquema de acessos dos utilizadores.

Os médicos têm permissões para aceder aos dados dos pacientes a si atribuídos, receber alertas, agendar a utilização do equipamento pelos pacientes e definir os seus perfis clínicos. Já os cuidadores, além do acesso aos dados dos pacientes da Vitabox em causa e da receção

de alertas, podem também consultar o histórico do estado ambiental da habitação. Os patronos têm acesso a todas as ações dos cuidadores e também são responsáveis pela associação dos pacientes aos respectivos médicos, gerir o equipamento, registando-o na Vitabox e no caso das pulseiras, definido os pacientes a que pertencem e definir utilizadores como cuidadores da Vitabox. Adicionalmente o patrono, o médico e o cuidador poderão realizar videochamadas e enviar notificações em tempo real para a Vitabox em causa.

O administrador desempenha o papel de responsável pela manutenção da infraestrutura fornecido pela empresa que presta o serviço, estando impedido de visualizar os dados dos sensores. A sua função será apenas de monitorizar o estado de atividade do equipamento e de todo o sistema, tendo acesso apenas ao momento da última receção de valores de um equipamento e sendo notificado caso algum equipamento emita valores considerados inválidos, como por exemplo humidade negativa.

Para que seja possível a implementação das regras de acesso são definidas as relações entre as diferentes entidades do sistema, impondo uma estrutura hierárquica em que no centro estarão a Vitabox e o paciente (Fig. 7).

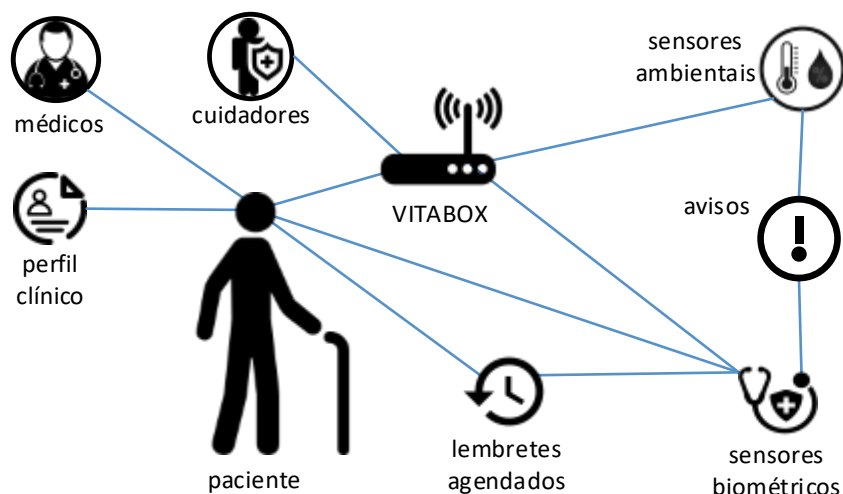


Fig. 7 - Esquema de relações entre as entidades do VITASENIOR-MT.

A solução desenvolvida cumpre boas práticas e contém mecanismos para garantir escalabilidade, disponibilidade e otimização, a fim de promover os menores tempos de resposta e de atuação possíveis sem comprometer a flexibilidade que o sistema deve ter de forma a, no futuro, integrar novas funcionalidades ou alterar as existentes. Para isso recorre a padrões de desenvolvimento já definidos e testados para comunicação cliente-servidor, envio de eventos e alojamento de dados.

Para concluir, o desenvolvimento bem-sucedido de toda a componente de *Cloud* do projeto VITASENIOR-MT teve como base 4 etapas específicas que são abordadas em detalhe nos capítulos seguintes:

- Planeamento da solução a desenvolver e escolha das tecnologias a utilizar;
- Desenvolvimento da solução de *backend* e API;
- Desenvolvimento de aplicação de *frontend*;
- Testes (integração) ao sistema desenvolvido.

1.2.4. Organização do relatório

O relatório está organizado da seguinte forma: no capítulo 2 é apresentada uma contextualização tecnológica de metodologias para o alojamento e comunicação entre serviços dedicados à monitorização bem como os seus desafios e algumas soluções.

No capítulo 3 é apresentado o levantamento do estado de arte na área da integração de soluções *Cloud* com a Internet das Coisas (IoT), iniciando com algumas propostas de *Cloud* no mercado, bem como os seus pontos diferenciadores e finalizando com um levantamento sobre outros trabalhos e estudos já realizados na área da teleassistência e e-saúde na perspetiva de *Cloud*.

No capítulo 4 é detalhado todo o sistema desenvolvido, começando por apresentar a proposta de solução e uma vista geral da mesma, seguido de uma descrição dos componentes mais relevantes. Entre estes, destacam-se os ambientes utilizados para desenvolvimento e

produção, a escalabilidade da arquitetura, o armazenamento, a segurança, o acesso aos dados e por fim uma demonstração dos testes realizados à solução.

Para terminar, o capítulo 5 apresenta as conclusões do desenvolvido e aborda algumas ideias para trabalho futuro.

Capítulo 2

Contextualização tecnológica

Neste capítulo são apresentados os conceitos de computação em nuvem (*cloud computing*) e de Internet das Coisas, bem como os seus benefícios e desafios. De seguida são apresentados os mecanismos de integração entre estes conceitos e as arquiteturas tipicamente utilizadas.

2.1. Computação em Nuvem

O conceito de Computação em Nuvem (comumente referido como *Cloud Computing* ou simplesmente *Cloud* ao longo do relatório) refere-se à alocação de recursos de computação sem o conhecimento da localização física deles. Isto é, um conjunto de computadores ligados entre si pela infraestrutura de rede que permite acesso remoto, e disponibiliza processamento e armazenamento de grandes volumes de dados. Desta forma, é possível mover os serviços disponibilizados para um centro de processamento de dados mais próximo do cliente e configurar réplicas ou cópias de segurança (*backups*) no outro lado do mundo de forma automatizada, consoante as necessidades.

Com esta facilidade de alocação de recursos é possível ao provedor disponibilizar novos modelos de negócio como por exemplo o *Platform as a Service* (PaaS).

Neste modelo, o provedor fornece todo o ecossistema e ferramentas de desenvolvimento e implementação já configurados, cabendo ao programador desenvolver sobre a tecnologia fornecida. Assim, este delega a responsabilidade de *backups* de dados, balanceamento de serviços e atualizações de infraestrutura para o provedor do serviço. Este paradigma vem opor-se ao tradicional *Infrastructure as a Service* (IaaS), em que o provedor apenas oferece os recursos físicos de processamento, armazenamento e rede. Nesta segunda abordagem, o programador é responsável por configurar todo o sistema, garantir o seu correto funcionamento e assumir o risco de falha, tornando o processo mais moroso e propício a erros [10].

Com a oferta de novos modelos de negócio vem também a oferta de novos modelos de pagamento, pois ao invés do cliente alocar um valor fixo de recursos, é lhe agora possível pagar apenas os recursos utilizados, aplicando uma política de *pay-as-you-go* (PAYG). Uma vez que a plataforma oferecida é elástica, o cliente pode iniciar o serviço com os recursos provisionados ao mínimo e ir alocando mais recursos conforme a necessidade de utilização.

2.2. Internet das Coisas

A Internet das Coisas (IoT) pode ser definida como a interseção do mundo digital com mundo físico, por meio de dispositivos interligados com o objetivo de monitorizar, atuar e/ou disponibilizar informação de forma “natural” aos seus utilizadores, de forma não intrusiva e de fácil acesso. Estes dispositivos são maioritariamente caracterizados pelo seu baixo consumo energético, portabilidade e capacidade imersiva do utilizador, sendo já aplicados em diferentes áreas tais como a monitorização ambiental, gestão de energia, cuidados de saúde, automação industrial, venda a retalho, manutenção de equipamento, multimédia e transportes.

Sendo o projeto VITASENIOR-MT uma solução IoT aplicada à saúde e monitorização ambiental, serão encarados os desafios que daí provêm, como o tratamento de um extenso volume de dados provenientes dos sensores, a interoperabilidade entre os sensores e a infraestrutura de *Cloud* e também a segurança de qualquer informação que daí possa ser extraída.

O sentimento corrente do mercado e a expectativa a curto-prazo em relação à evolução da IoT é muito elevada, esperando-se que o desenvolvimento de plataformas e aplicações continuem a dirigir o mercado conforme os serviços são deslocados para a *Cloud* (Fig. 8). A perspetiva de crescimento da IoT nas suas diversas aplicações é tão elevada que se supõe um impacto no mercado de 3 a 11 mil milhões de dólares em 2025, o que poderá representar 11% da economia mundial [11].

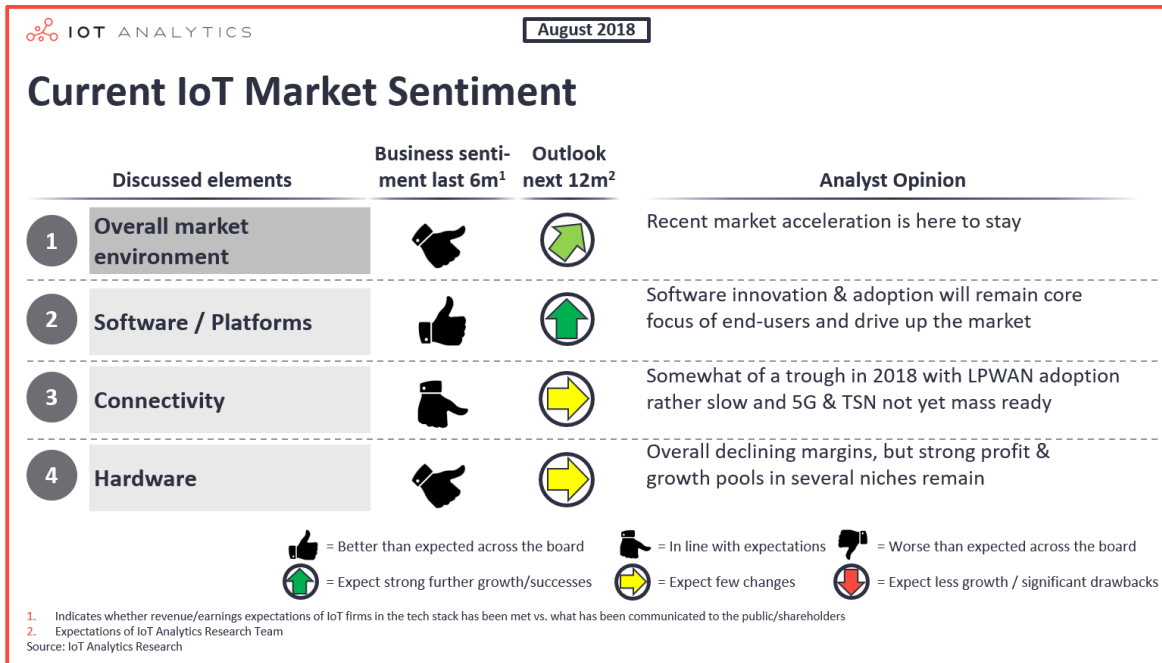


Fig. 8 - Adoção de IoT por segmento e região [12].

No entanto ainda existem diversas barreiras à adoção da IoT [13]. A primeira e mais óbvia é a adoção destas tecnologias, que implica uma alteração da posição do ser humano em relação ao mundo que o rodeia. Na perspetiva do consumidor final existe uma grande diferença geracional no que toca à adesão digital, em que os mais novos, que já nasceram num mundo digital, aceitam a integração de novas tecnologias no seu quotidiano com naturalidade, enquanto os mais velhos são desconfiados e muitas vezes adversos à alteração dos seus hábitos. Na perspetiva das empresas, o obstáculo está no investimento necessário para integrar estas soluções, obrigando a uma reformulação dos seus sistemas e à aquisição de novo equipamento, que apesar de a longo prazo se apresentar como opção lucrativa, a maioria continua a adiar a sua integração. Por fim existe a questão da segurança dos dados pessoais, em que muitos provedores não asseguram a confidencialidade dos mesmos, possibilitando a exposição de dados relativos aos utilizadores, edifícios ou empresas a ataques externos.

2.3. Integração de Computação em Nuvem com a Internet das Coisas

Embora os dispositivos IoT (ex. sensores, RFID *tags* ou câmeras) tenham um enorme leque de aplicações, estes encontram-se dependentes das suas restrições de recursos, tais como a autonomia (capacidade da bateria), memória, acessibilidade e processamento. Assim, é necessário delegar a computação de maior complexidade para os serviços da *Cloud*, onde as mesmas restrições não se aplicam e que podem ser facilmente acedidos pelos utilizadores finais [14].

Para que a funcionalidade disponibilizada na *Cloud* possa responder às necessidades da uma arquitetura IoT, esta deve suportar a escalabilidade das aplicações, alojamento de grandes volumes de dados e processamento distribuído de forma a melhor responder ao problema do “*Big Data*” [15].

Nas seções seguintes serão apresentados os desafios de integração da *Cloud* com a Internet das Coisas e algumas arquiteturas que têm vindo a ser adotadas.

2.3.1. Desafios de integração

A comunicação entre a *Cloud* e os clientes é geralmente realizada por meio de *web services*, seguindo a arquitetura RESTful, que opera sobre o protocolo HTTP, ou SOAP, trocando informação no formato JSON ou XML. No entanto, estes métodos operam sobre IEEE 802.3 (Ethernet) ou IEEE 802.11 (Wireless LAN), que é um obstáculo para muitos dos dispositivos IoT a operar sobre IEEE 802.15.4. Assim estes dispositivos necessitam de protocolos de transferência de dados alternativos, como o caso do CoAP e MQTT. O CoAP segue o estilo RESTful mas funcionando sobre UDP na camada de transporte, removendo o *overhead* do TCP. Outra diferença do CoAP é que, ao invés dos serviços normais RESTful em que a comunicação é iniciada por um cliente em *pull mode*, este usa *push mode* assíncrono para obter os dados do servidor seguindo o padrão de *observer* [15], permitindo ciclos de dormência ao servidor (que se encontra no sensor) otimizando a poupança energética do mesmo. Já o MQTT por outro lado funciona com operações *publish/subscribe*, utilizando o

sistema de filas de mensagens para encaminhar os valores para o *broker* (intermediário) que se encontra permanentemente à escuta. Este protocolo diferencia-se dos similares (ex. AMQP, XMPP) por ser mais básico, sem a nomeação de filas de espera, minimizando o consumo energético e o uso de rede [16]. Apesar do MQTT ter sido idealizado para dispositivos de baixos recursos é também verdade que muitas empresas começam a empregá-lo em aplicações na Internet, um exemplo disso é o *Facebook Messenger* [17].

Devido a esta diversidade de protocolos de comunicação e da heterogeneidade dos dados recolhidos é necessária uma articulação na *Cloud* para compreender os dialetos dos dispositivos que com esta comunicam. Este esforço tem sido cada vez mais transferido para camadas de abstração que se localizam entre os dispositivos e a *Cloud*, geralmente no limite (*edge*) das redes locais [18] por meio da utilização de um Cloudlet (abordado em detalhe na secção seguinte) ou de uma arquitetura de processamento distribuído na rede local. As arquiteturas mais adotadas são (ilustradas na Fig. 9): *Mobile Cloud Computing* (MCC); *Mobile Edge Computing* (MEC); *Edge Computing* (EC); *Dew Computing* (DC); *Fog Computing* (FC); *Fog Dew Computing* (FDC).

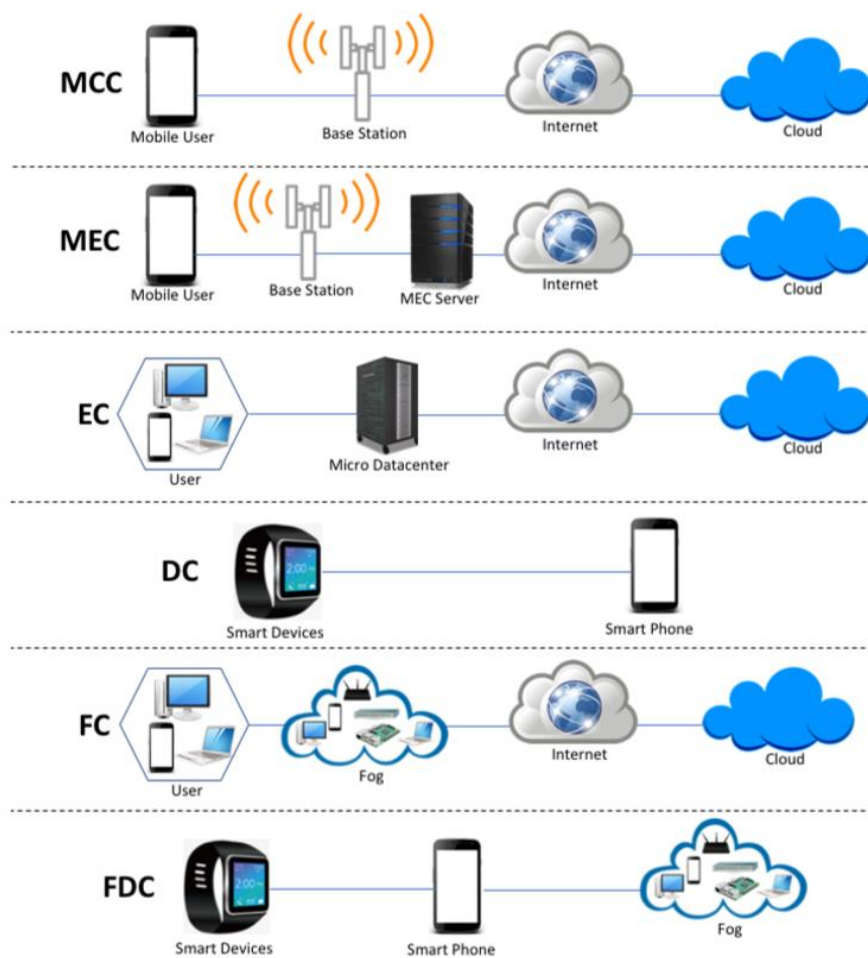


Fig. 9 - Arquiteturas de camadas de abstração entre *Cloud* e dispositivos [18].

2.3.1.1. *Cloudlet*

Cloudlet é um servidor básico, mais limitado em termos de capacidade de processamento e armazenamento que a *Cloud*, mas mais robusto quando comparado com os dispositivos IoT. Este normalmente localiza-se no *edge* da rede local e é utilizado em sistemas que não localizam o negócio na *Cloud*, mas que a utilizam apenas para tarefas realmente exaustivas em memória e processamento (ex. *machine learning*, analítica e *backups*) ou para serviços de agregação de dados de diferentes *Cloudlets* [18].

Esta abordagem é utilizada nas arquiteturas MCC, MEC e EC (referidas no capítulo 2.3.1). No caso do *Dew Computing*, o conceito é similar, sendo que o equipamento responsável pelo processamento é o *smartphone*, destinando-se geralmente apenas a aplicações pessoais.

A título de exemplo, em [19] é apresentada uma aplicação de *Cloudlet* utilizada na recolha de dados numa rede local de sensores corporais (WBAN). Nesta, um conjunto de sensores biométricos monitorizam os parâmetros de um paciente e transmitem os dados para o servidor que serão posteriormente encaminhados para um hospital, clínica ou provedor de serviço.

2.3.1.2. Fog Computing

Fog computing é um paradigma de computação distribuída, que consiste num módulo agregador dos sensores da rede local [18], a fim de lidar com as ligações dos dispositivos heterogéneos, com a modelação dos seus dados para um padrão homogéneo e interpretável pela *Cloud*, com o alojamento temporário dos dados e por fim com o encaminhamento dos mesmos para a *Cloud*. Desta forma é retirada a responsabilidade de autenticação de todos os sensores na *Cloud*, tendo apenas que autenticar o módulo. A infraestrutura de rede pública é também libertada do excesso de tráfego causado pelos sensores, uma vez que será contido na rede local. E por fim a *Cloud* foca os seus recursos, que embora extensos têm um preço, apenas na análise direta dos dados, sem necessidade de traduções, conversões e modelações dos dados vindos dos diversos sensores que por norma têm formatos díspares.

A *Fog Computing* promove também as condições ideais para resposta a situações em que é necessária uma atuação imediata, analisando os dados localmente e acionando o protocolo de emergência [20].

Um exemplo de um sistema que faz uso de *Fog Computing* é apresentado em [21], com o *gateway* inteligente, a que chamaram UT-GATE, a agregar diferentes topologias de redes de sensores e a realizar o reencaminhamento para a *Cloud* (Fig. 10).

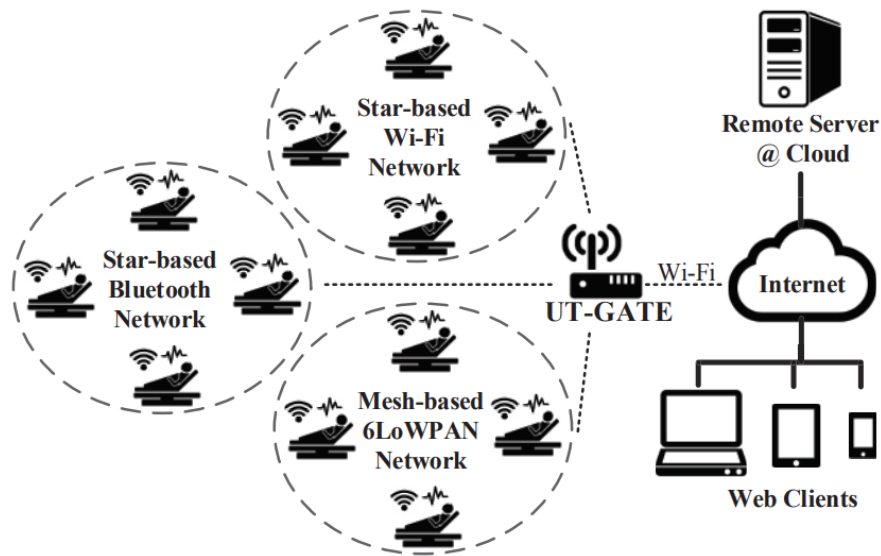


Fig. 10 - Arquitetura do UT-GATE [21].

2.3.1.3. Mobile Sink

Mobile Sink (MS) é apresentado em [22] como um agregador móvel que se desloca até ao alcance de cada sensor para receber os seus dados. Esta solução visa a remover os gastos excessivos das comunicações *multihop* e da construção da árvore de nós da rede de sensores sem fio (WSN), uma vez que os nós mais próximos da raiz consomem mais energia, que os das bordas, no reencaminhamento de dados. Esta solução pretende minimizar o número de transmissões de dados dos sensores, uma vez que cada sensor enviará apenas os seus próprios dados, e com isso aumentar o seu tempo de um ciclo de bateria. Além disso, desta forma os sensores não necessitam de ligação completa à rede, tendo apenas de estabelecer comunicação com o MS, sendo este o responsável por encaminhar os dados para a *Cloud*. Por fim esta solução reduz os erros de transmissão e as colisões que ocorrem em redes com muitos sensores a enviar dados em simultâneo.

Capítulo 3

Estado da arte

Neste capítulo são apresentadas algumas das soluções existentes no mercado de infraestruturas de *Cloud* com aplicação em sistemas *IoT*, assim como a sua arquitetura e os seus pontos diferenciadores. Por fim será abordada a aplicação prática de Computação em Nuvem para monitorização biométrica de pacientes

3.1. Soluções de Computação em Nuvem para Internet das Coisas

Sendo a expectativa de valorização do mercado da Internet das Coisas tão elevada, é natural o surgimento de soluções comerciais e abertas à comunidade que respondam a essa procura. Um exemplo disso é o OpenIoT [23], um *middleware* de código aberto, desenvolvido em Java que deverá comportar-se como uma extensão da *Cloud*, permitindo a agregação dos sensores e atuadores de diferentes plataformas (RFID, WSN, entre outras) que sejam reconhecidos pela plataforma Extended Global Sensors Network (XGSN) e fornecendo os recursos dos sensores numa plataforma sob o conceito *sensing-as-a-service*. Esta solução destaca-se pela sua estrutura semântica, baseada em ontologia, que permite a interoperabilidade com sistemas externos e com os utilizadores finais que podem visualizar, configurar dependências, agendar leituras.

O Stack4Things [24] é um projeto de código aberto que promove a gestão de um conjunto de dispositivos sem necessidade de conhecer a localização física dos dispositivos, a tecnologia utilizada e a sua configuração de rede. O projeto tem dois módulos desenvolvidos em Node.JS, o IoTronic, que permite aos utilizadores gerir os seus recursos através de uma interface gráfica, e o Lightning-rod que estabelece a ligação do dispositivo com a *Cloud* por CoAP ou AMQP. Atualmente o projeto está a migrar o IoTronic para um sistema baseado em OpenStack que permite criar máquinas virtuais para a gestão de grupos de sensores e realizar a orquestração das máquinas, ou seja, alocar recursos físicos para as máquinas

conforme as suas necessidades. As plataformas habilitadas a interagir com o Stack4Things são o Arduino, Android, Raspberry Pi, Orange Pi e Kitra.

Uma solução mais comercial e com uma oferta mais ampla, tanto nos serviços oferecidos como nos dispositivos é a CloudPlug [25], que consiste num serviço de *Cloud* de apoio às suas plataformas: a SmartPlug, o Edge One e o PicoPlug. O conceito da SmartPlug baseia-se no desenvolvimento de um agente seguro e robusto que funciona em qualquer dispositivo que corra Linux ou Arduino. Assim, o utilizador pode desenvolver em JavaScript ou C++ para a(s) sua(s) SmartPlug(s), conseguindo recolher valores, executar um atuador ou comunicar com outros dispositivos no seu alcance (ex. ZigBee, Bluetooth, ...) a funcionar como *bridge*. O Edge One funciona em *gateways* Intel x86, dedicando-se à computação no *edge* da rede, com um conjunto de módulos (*containers*) definidos pelo utilizador que podem ter funcionalidade de *accounting*, analítica e armazenamento. O PicoPlug destina-se a microcontroladores ESP8266 e ESP32, estando mais limitados nas suas funcionalidades. Na *Cloud* o utilizador poderá descarregar o seu script (conjunto de instruções a executar) para os equipamentos desejados, agendar tarefas, guardar os dados recebidos ou definir regras conforme os valores recolhidos. O plano de utilização oferecido pela CloudPlugs é de *pay-as-you-grow*, ou seja, paga conforme o número de equipamentos configurados com as suas *frameworks*.

Particle [26] apresenta-se no mercado como uma plataforma “*all-in-one*”, disponibilizando o sistema operativo dos dispositivos, a plataforma de desenvolvimento e a *Cloud* de gestão do equipamento e recolha de dados. A sintaxe de desenvolvimento é muito semelhante à do Arduino e a *Cloud* fornece uma interface de programação de aplicações (API) RESTful caso o cliente pretenda desenvolver aplicações suas sobre a plataforma. Este sistema pode ainda integrar com outras plataformas de *Cloud* como a Google Cloud Platform ou a Azure IoT Hub. Sendo o código do sistema operativo (Device OS) disponibilizado num repositório público [27], a *Cloud* fornecida gratuitamente e o equipamento de baixo custo, esta plataforma apresenta-se como uma boa solução para quem tiver intenção de começar a trabalhar na IoT.

Entre os gigantes tecnológicos inseridos no mercado de *Cloud* a IBM apresenta a solução mais simplista aos seus clientes, o IBM Watson IoT [28]. Este consiste num *hub* que permite

a ligação aos dispositivos por MQTT ou HTTP de modo a receber valores e enviar comandos para acionar as ações pretendidas. Ao contrário do que o nome sugere, esta plataforma não tem ligação direta ao serviço IBM Watson (solução de inteligência artificial da IBM). Após a receção de dados vindos dos dispositivos é despoletado um evento que pode ser escutado pelos clientes. Para este fim a IBM fornece uma biblioteca com suporte para várias linguagens de programação, que torna possível aos clientes desenvolver as suas aplicações sobre os valores recebidos. A solução mais comum é a implementação de um serviço de Node-red [29], uma ferramenta de programação em Node.JS, orientada a fluxos, com base numa interface gráfica, que permite escutar eventos, receber e enviar pedidos HTTP, estabelecer WebSockets e guardar os dados em bases de dados. Esta ferramenta inicialmente foi desenvolvida pela IBM Emerging Technology Services e agora é detida pela JS Foundation.

Ao contrário da IBM, a Google Cloud IoT [30] apresenta uma das construções em blocos mais completas entre os provedores de *Cloud* (Fig. 11). Começa por oferecer a aplicação Cloud IoT Edge que pode ser executada em dispositivos no *edge* da rede local com distribuições Linux. Este bloco é capaz de realizar operações que requeiram *machine-learning* em tempo-real através da *framework* de código-aberto TensorFlow Lite com suporte da sua solução Edge TPU (*tensor processing unit*), um circuito integrado capaz de acelerar as operações de inteligência artificial executando as mesmas por *hardware*. Os dados recolhidos são enviados para a Cloud IoT Core que permite ligações por HTTP ou MQTT e reencaminha os dados para o bloco de *publish/subscribe* (Cloud Pub/Sub). De seguida os dados são recolhidos pelo módulo Cloud Functions, onde o cliente pode definir um conjunto de funções a realizar com esses dados. Em alternativa, pode encaminhar os mesmos usando a Cloud Dataflow, em que os dados poderão ser submetidos a um conjunto de ferramentas disponibilizadas para análise, extração de valores chave, aprendizagem e previsão de tendências.

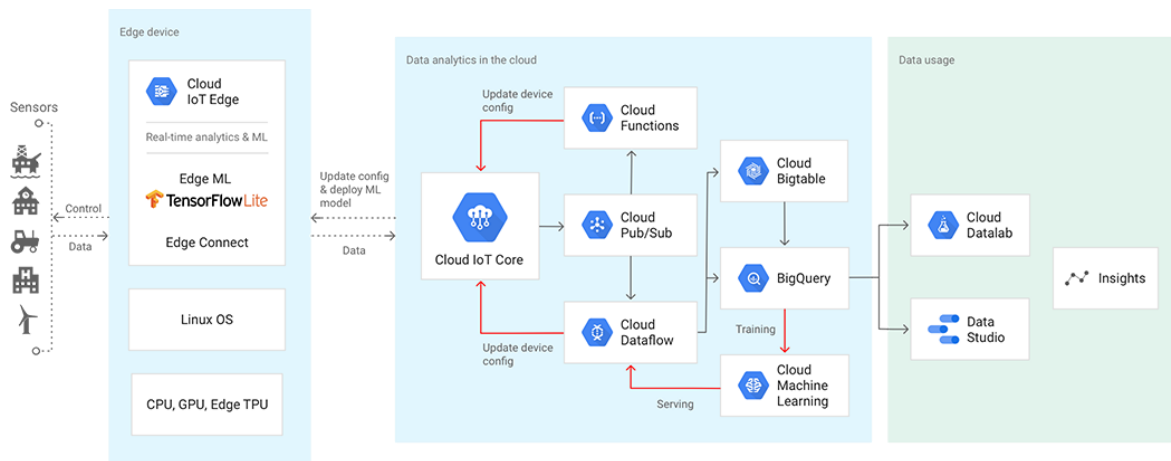


Fig. 11 - Google Cloud IoT construção em blocos [30].

À semelhança do modelo de negócio apresentado pela Google, também a Microsoft adotou o modelo de construção em blocos e de *Edge Computing* (à exceção do acelerador de *hardware*). O que diferencia a plataforma Azure IoT Hub [31] é a sua integração com ferramentas da própria empresa que já estão fortemente enraizadas em diversas empresas. Alguns exemplos são os casos do Active Directory, que permite a autorização de acessos a determinados sensores/atuadores mediante o utilizador da organização, e o Power BI que permite a submissão dos dados para análise imediata.

O modelo de negócio apresentado pela Amazon é muito semelhante ao apresentado pela Google e pela Microsoft, optando também pela estrutura de construção em blocos e de computação no *edge*. O potencial da AWS IoT [32] está na disponibilização de um sistema operativo para os *end devices*, o Amazon FreeRTOS, uma evolução do FreeRTOS dedicada e otimizada para as ligações com os serviços da Amazon, incluindo o AWS IoT Greengrass (software que corre no *edge* da rede local).

Por fim, o ThingSpeak [33] é um serviço de *Cloud* fortemente adotado pela comunidade de MATLAB, motivado pela sua facilidade de integração com a ferramenta. Esta plataforma fornece um conjunto de bibliotecas para dispositivos como Arduino, ESP8266, ESP32 e Raspberry Pi, permitindo que estes se possam ligar à *Cloud*. Após estabelecer a ligação, é possível a um utilizador partilhar os canais dos sensores com outros utilizadores da *Cloud*,

integrar serviços de MATLAB configurados por si para analisar os dados e despoletar eventos (Fig. 12).

Pelo facto do MATLAB ser uma ferramenta de ampla utilidade nas diferentes engenharias, são diversos os projetos académicos desenvolvidos com o suporte da ThingSpeak. Alguns exemplos disto são a implementação de um sistema de monitorização de batimentos cardíacos [34], ou o repositório de canais públicos em que os utilizadores da plataforma divulgam os resultados obtidos com os seus estudos [35].

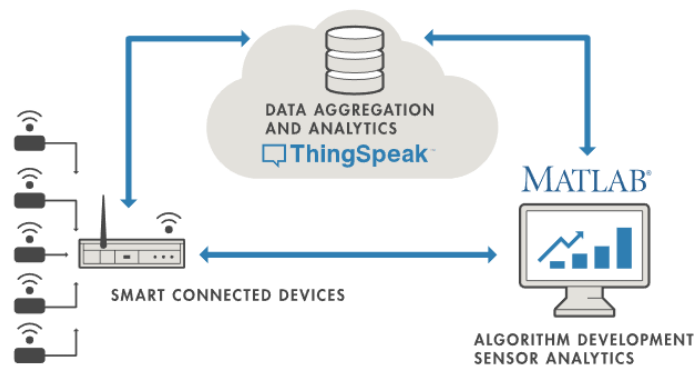


Fig. 12 - Arquitetura da ThingSpeak [33].

O estudo efetuado permitiu concluir que a oferta de soluções de *Cloud* para ambientes IoT é vasta e diversificada. Existem arquiteturas robustas em que o cliente aloca as funcionalidades que pretende utilizar, como o caso das ofertas de Google Cloud IoT, da AWS IoT e da Azure IoT Hub. Por outro lado, temos soluções comerciais em que o utilizador deve configurar o equipamento e a plataforma detidos pela marca, como o caso da Particle e da CloudPlugs. Existem ainda soluções para fases específicas na transmissão de dados, como o caso da OpenIoT que funciona como *middleware* para a estruturação e encaminhamento dos valores, da IBM Watson IoT o do Stack4Things que funcionam como um *hubs* de receção. Por fim, existem ainda soluções com foco em fins académicos como o caso da ThingSpeak. Por estas ferramentas serem tão divergentes quanto às suas funcionalidades é difícil compará-las diretamente, sendo que cada uma deve ser considerada consoante as especificidades do problema que for abordado.

3.2. Desenvolvimento na área da e-saúde e teleassistência

Apesar de a oferta de soluções IoT no mercado ser abundante a verdade é que as implementações de *Cloud* na área da e-saúde e teleassistência continuam fechadas. Uma das causas é o facto de os proprietários preferirem desenvolver os seus próprios sistemas em vez de utilizar os existentes. Possíveis causas são a falta de confiança nos serviços fornecidos pelos provedores ou porque estes não lhes proporcionam as condições favoráveis ao seu negócio, como por exemplo a gestão de utilizadores para os seus sistemas. Esta realidade implica, na maioria dos cenários, a escassez de inovação (na componente do servidor) e desaproveitamento dos recursos da *Cloud*.

Um exemplo simples disso é a proposta em [36], uma solução para monitorização de saúde, para regiões do interior da Índia, em que diversos dispositivos (ex. eletrocardiogramas, tensiómetros, entre outros) foram desenvolvidos de forma a se ligarem à plataforma para o envio dos valores lidos por intermédio de aplicações móveis (*Dew Computing*). Nessa plataforma os utilizadores são associados a responsáveis clínicos que monitorizam os parâmetros através de uma aplicação web (Fig. 13). O serviço foi desenvolvido com Java Servlets³, suportadas por um servidor web Apache Tomcat, sendo os dados alojados num servidor de base de dados MySQL. Sendo a solução apresentada uma prova de conceito, o projeto foca-se apenas na recolha de valores associados a um paciente, na captura de valores fora do intervalo aceitável, na notificação ao respetivo médico e no envio de mensagens aos pacientes. O sistema apresenta assim uma arquitetura simplista que impede a sua própria evolução para monitorizar diferentes equipamentos (estando limitados aos próprios), dependendo dos funcionários das PCU (centros de cuidados de saúde primários na Índia) para o manuseamento do equipamento (abrindo exceções para situações pontuais em que pode estar no ambiente doméstico) e não prevê sistemas de gestão de utilizadores, autenticando os mesmos pela identificação do *smartphone*.

³ Componente de Java para desenvolvimento de serviços web

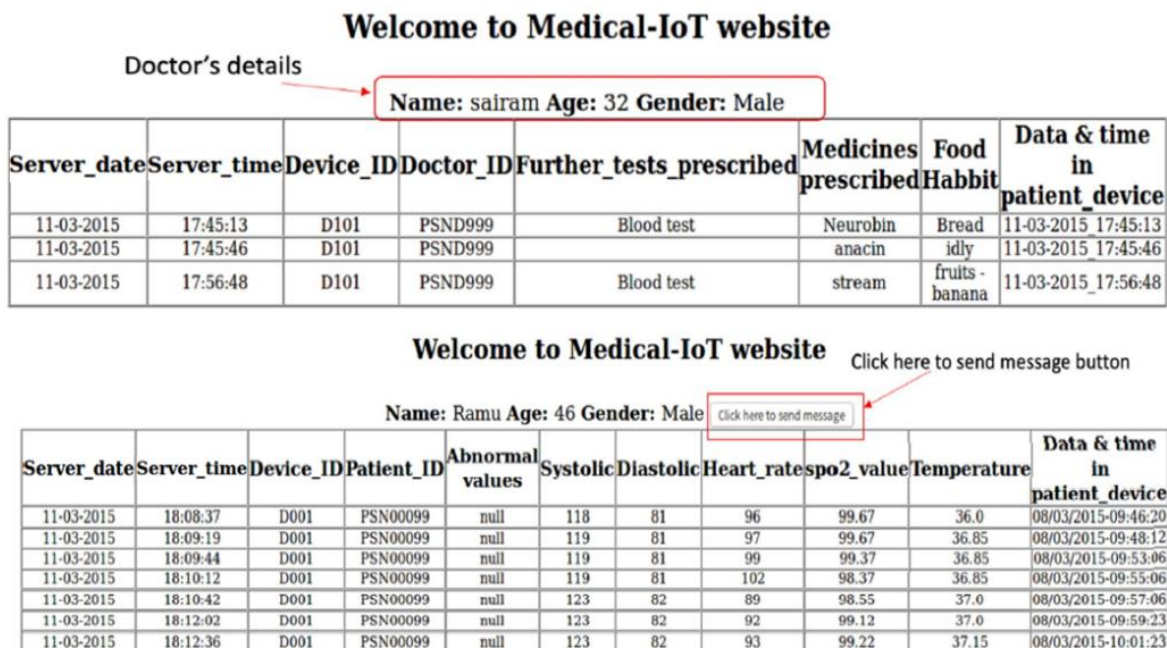


Fig. 13 - Interface gráfica de projeto de monitorização de saúde na Índia [36].

Outra abordagem similar é apresentada no projeto CardiaQloud [37], em que utilizam a e-Health Sensor Shield V2.0, na plataforma Arduino, para extrair valores de eletrocardiografia. Após os dados serem extraídos pela *shield*, são recolhidos por um cliente local e encaminhados para a *Cloud* (Fig. 14). Nesta solução já é utilizada uma plataforma escalável (Amazon EC2), onde desenvolveram um *web service*, em Node.JS, para visualização em tempo real dos dados. Sendo este projeto apenas um protótipo, não existe qualquer modelo de negócio associado, servindo apenas para demonstrar uma abordagem à recolha de parâmetros biométricos e uma comparação relativa a outros sistemas similares.

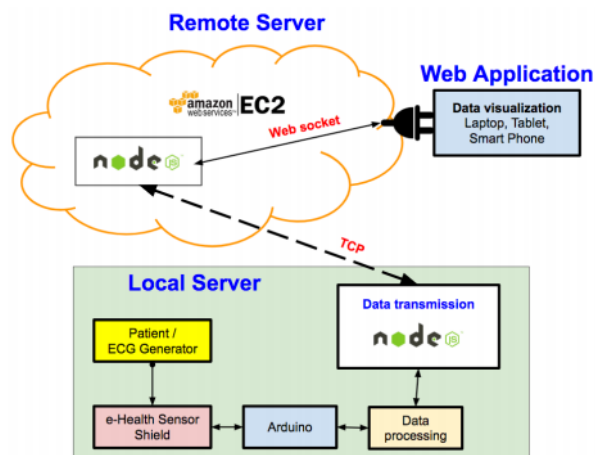


Fig. 14 - Arquitetura do CardiaQcloud [37].

Um dos grandes obstáculos apontados para a implementação de sistemas IoT na saúde é a capacidade de resposta dos sistemas, pois quando se lida com valores que podem assinalar uma situação fatal é necessária uma atuação no momento. Desta forma estes sistemas necessitam sempre de uma camada de tratamento dos dados recolhidos localmente (computação na névoa) [38]. Para este cenário ser realmente efetivo seria necessário o desenvolvimento de equipamento ajustado a esta realidade e o que se verifica é a reduzida oferta comercial destes dispositivos, pois o foco atualmente é na centralização dos serviços numa *Cloud*.

No mercado existem já algumas soluções fechadas como o caso do MySignals [39] e o Apple Health [40], no entanto estas requerem a utilização de equipamento específico da marca, que não se adequa a todos os públicos (por exemplo os *smartphones* não correspondem aos requisitos de usabilidade para a população idosa em Portugal). Para além disso, na sua maioria estas soluções estão limitadas à extração de valores e apresentação dos mesmos a um único utilizador, não sendo possível o relacionamento com familiares, a parametrização e acompanhamento remoto por cuidadores. Sendo que este é um ponto relevante ao sucesso de uma solução para a saúde, pois é essencial a adesão e a compreensão destes sistemas por parte dos profissionais especializados na área, como os médicos de família ou fisioterapeutas, para que estes consigam monitorizar a realização de exercícios de

reabilitação, a medicação ou atividades de prevenção, diminuindo assim a hospitalização e o número de cuidadores necessários [38].

Por fim, em [41] é referido outro motivo para a (ainda) pouca oferta de soluções e-saúde: a falta de padrões de desenvolvimento e arquitetura para a uma e-Health *Cloud*. Ou seja, não foi ainda definido um formato aceite de como os dados devem ser recolhidos, os intervalos de tempo para as recolhas e como devem ser alojados estes dados de forma segura. Isto acontece porque existe um conjunto de entidades distintas a desenvolver padrões (ex. DICOM, ISO/ 215, HL7) sem se conseguirem alinhar, impedindo a interoperabilidade entre os sistemas.

Capítulo 4

Trabalho realizado

4.1. Modo de funcionamento da solução

Conforme referido na seção 1.2 Solução proposta, a plataforma VITASENIOR-MT deve permitir o acesso a vários utilizadores com diferentes permissões, bem como recolher dados de sensores biométricos e ambientais oriundos das Vitaboxes que se encontram nas habitações. Deve ainda validar os valores e averiguar alguma anormalidade que justifique a emissão de um alerta.

Para que estas ações sejam possíveis existe um conjunto de processos que foram previamente definidos, como o registo dos modelos de equipamento a utilizar no sistema, das Vitaboxes ainda inativas que poderão ser posteriormente associadas a uma habitação pelo próprio proprietário, dos pacientes e dos equipamentos a associar à Vitabox e a transferência de todos estes parâmetros para a própria Vitabox (Fig. 15 - Fluxo de instalação da Vitabox).

Para cada equipamento foram definidos os intervalos de valores considerados normais e os limites que cada sensor é capaz de medir (podendo a observação destes significar uma avaria do equipamento). Foi também definido o intervalo de envio de valores tanto para efeitos de histórico do estado ambiental da habitação ou da evolução dos parâmetros vitais dos pacientes como para o rastreio de equipamento inativo (podendo significar também uma avaria do equipamento). Desta forma é possível tanto aos administradores perceberem quando o equipamento se encontra com anomalias e intervirem como aos cuidadores e médicos serem alertados de situações anómalas com os pacientes em tempo real, agilizando o processo de prevenção e intervenção.

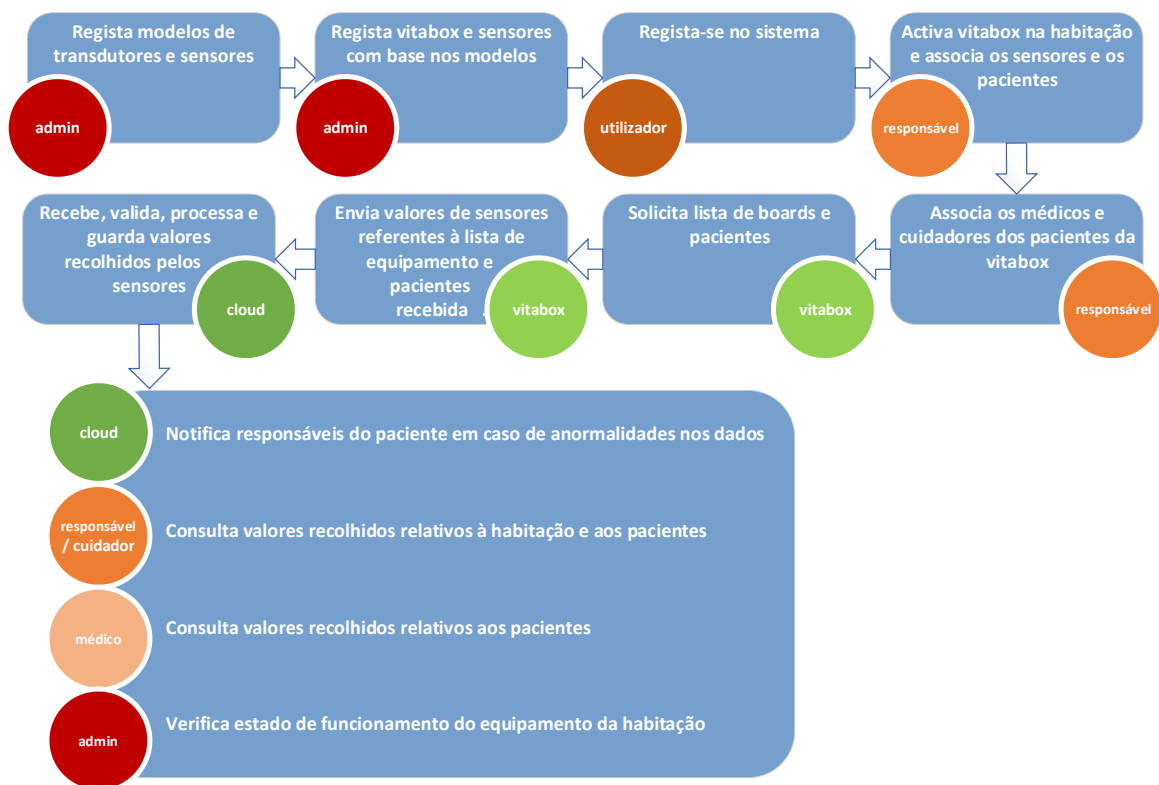


Fig. 15 - Fluxo de instalação da Vitabox na casa do utente.

4.2. Arquitetura Geral

A infraestrutura de *Cloud* foi inicialmente projetada e implementada na IBM Cloud. No entanto, o sistema foi desenhado da forma mais modular e genérica possível, contemplando desde o início a possibilidade de mudança de provedor, podendo mesmo ser alojada *in-house*. Para este fim, foi escolhida uma abordagem de arquitetura em microsserviços.

Uma arquitetura em microsserviços define-se pela distribuição de tarefas de um sistema por diversos serviços web, ao contrário do tradicional serviço web que aglomera todas as funcionalidades necessárias ao funcionamento do negócio. Desta forma é possível obter melhor isolamento e tratamento de problemas, desenvolvimento independente de cada serviço, maior distribuição de processamento e melhor gestão de recursos entre serviços. A principal desvantagem de uma arquitetura em microsserviços é o aumento de complexidade

do sistema, proporcional ao número de serviços desenvolvidos, sendo necessário manter a coordenação e comunicação entre eles.

Cada serviço foi desenvolvido em Node.js [42], um ambiente de execução (*runtime*) assíncrono de JavaScript (JS) orientado a eventos. O facto desta tecnologia ser não bloqueante permite diminuir os tempos de execução das tarefas. Isto é, enquanto o serviço espera uma resposta de outro serviço externo ou a execução de outro servidor (ex. uma pesquisa na base de dados) ele vai executando outros pedidos, nunca ficando bloqueado no meio de pedidos. Isto, aliado ao facto de ter uma classe de HTTP na sua arquitetura base torna-o uma ferramenta poderosa para o desenvolvimento de serviços de *streaming* e de baixa latência.

Apesar de o JavaScript ser uma linguagem interpretada, o Node.js utiliza o motor V8 da Google para compilar o código JS diretamente em código máquina (neste caso, a máquina virtual Java usada) e otimizar partes do código em tempo real [43]. O processo de compilação do V8 passa por duas fases, uma inicial, mais rápida mas menos eficiente, que traduz diretamente o código JS, permitindo um arranque mais rápido. Nesse momento o motor começa a instanciar todas as chamadas realizadas e recompilar o código mais usado pelo seu compilador *just-in-time* (JIT) de forma mais otimizada. Todos os métodos chamados serão guardados em cache (*inline caching*) e será guardado em memória uma tradução de todos os objetos dinâmicos em *hidden classes*, uma forma de abstração de protótipos (uma analogia semelhante à das classes). Assim, após duas chamadas bem-sucedidas do método com os parâmetros de entrada da mesma *hidden class* o V8 vai assumir que a estrutura não se alterou e vai saltar diretamente para o endereço de memória da respetiva estrutura aplicando o *offset* obtido das chamadas anteriores [44]. Este processo diminui muito os tempos de execução, no entanto é muito exigente em memória quando comparado com outras linguagens que não têm uma tipagem dinâmica⁴ como o Java. Por fim o JIT vai procurar as *hot functions* (métodos mais frequentemente chamados) que se encontram em cache e vai recompilar com base nas *hidden classes* fornecidas, procurando um código mais otimizado para obter a resposta já apresentada nos pedidos anteriores. É importante salientar que o JIT pode incidir

⁴ Permite alterar o tipo de dados de uma variável ao lhe atribuir um valor.

várias vezes sobre o mesmo método, e que o mesmo método pode ter várias *hidden classes* associadas a ele, por isso é importante mantê-las pelo mínimo para que o JIT obtenha uma melhor otimização.

4.2.1. Microserviços desenvolvidos

Os microserviços desenvolvidos são: Api, WebSocket, Worker, Schedule e Peer. Estes microserviços trocam mensagens entre si por meio de um servidor de fila de mensagens e comunicam para a rede pública por meio de um servidor *reverse proxy* que se encontra no limite da rede local criada pelo gestor de *containers*, tal como ilustrado na figura seguinte (Fig. 16 – Arquitetura geral da *Cloud*).

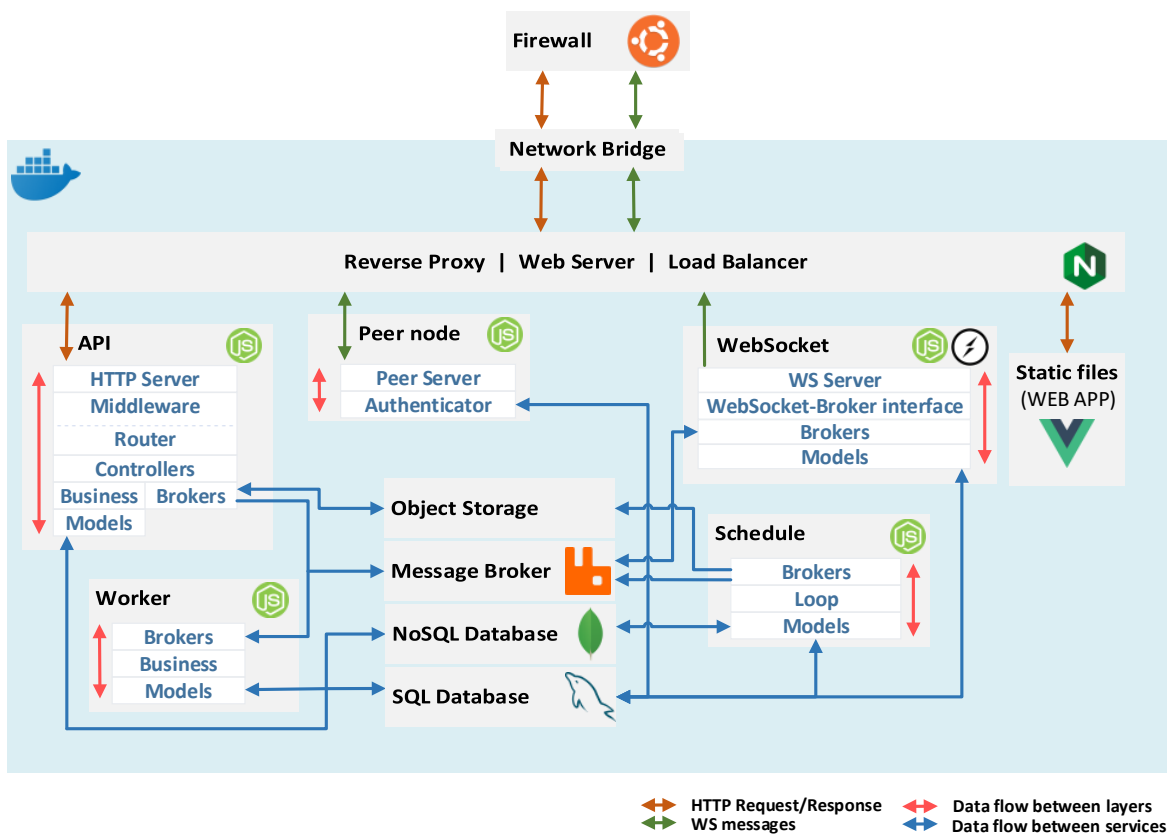


Fig. 16 – Arquitetura geral da *Cloud*.

4.2.1.1. API

A *Web Application Programming Interface* (API) é o serviço que permite o acesso aos dados pelos utilizadores através de uma arquitetura *Representational State Transfer* (REST), promovendo as ações de escrita, leitura, atualização e remoção sobre os modelos de dados do sistema, aqui também é realizada a autenticação dos utilizadores e a atribuição de permissões aos mesmos.⁵

4.2.1.2. WebSocket

O serviço de WebSocket estabelece as ligações por WebSockets (WS) entre este e as Vitaboxes ou o *browser* (navegador) utilizado pelos utilizadores para o envio de mensagens em tempo real. Estas mensagens são enviadas por iniciativa do servidor para diversas tarefas, tais como a atualização das listas de pacientes ou equipamento na Vitabox, para avisos aos cuidadores e médicos perante alguma irregularidade nos valores biométricos do paciente ou para o envio de lembretes em tempo real para os pacientes através da Vitabox.⁶

4.2.1.3. Worker

O serviço de Worker é responsável pela execução de tarefas intensivas como a aplicação de filtros aos dados recolhidos, por comparar os valores dos sensores com os limites estabelecidos para o equipamento e os perfis clínicos dos pacientes, atualizar tempos de receção nos equipamentos e gerar avisos a ser enviados aos respetivos utilizadores.⁷

⁵ https://github.com/Vitasenior-MT/cloud_RESTfulAPI

⁶ https://github.com/Vitasenior-MT/cloud_WebsocketServer

⁷ https://github.com/Vitasenior-MT/cloud_analytics

4.2.1.4. Schedule

O serviço de Schedule executa todas as ações que se encontrem agendadas (em *loop*), isto é, que ocorrem sistematicamente num período de tempo definido, como verificar se determinados exames foram realizados pelos pacientes, gerar conjuntos de dados (*datasets*) de valores anonimizados e limpar os avisos já demasiado antigos da base de dados.⁸

4.2.1.5. Peer

O serviço de Peer permite o registo e a negociação de utilizadores que pretendam estabelecer uma comunicação em tempo-real *peer-to-peer* (P2P) através do mecanismo *Real Time Communication over the Web* (WebRTC).⁹

4.2.2. Estrutura de cada microserviço

Para obter o máximo partido do motor V8 o código foi dividido em camadas, seguindo o padrão *layered* de arquitetura de *software*. Desta forma é possível reaproveitar código e impor uma estrutura de chamada das funções direcionando de forma mais eficiente o JIT do V8 e a criação de *hidden classes* mais eficientes. Este padrão permite também o isolamento de problemas no serviço, sendo facilitado o rastreio do mesmo, melhor gestão dos pedidos dos utilizadores e aplicação de políticas de acesso, uma vez que toda ela é realizada nas camadas superiores da pilha e melhor organização do código e da equipa de desenvolvimento, uma vez que permite a distribuição de tarefas por camada [45].

As camadas definidas, considerando que algumas não são exclusivas de um único microserviço, estando presentes em vários com exatamente a mesma funcionalidade, foram:

⁸ https://github.com/Vitasenior-MT/cloud_scheduler

⁹ https://github.com/Vitasenior-MT/cloud_PeerServer

- HTTP Server: representa o serviço de arranque que escuta os pedidos dos clientes;
- Middleware: Primeiro código executado em cada pedido, definindo os cabeçalhos aceites, valida o *token* de autenticação e extraí o idioma do cliente;
- Router: interpreta a rota e a versão requerida e reencaminha para o respetivo *controller*;
- Controller: filtra os acessos e acordo com as permissões do cliente, recolhe os parâmetros enviados e chama os métodos necessários à lógica do negócio;
- Business: executa todas as ações de CRUD e analítica dos dados;
- Broker: estabelece ligação com o servidor de filas de mensagens e efetua a transmissão e receção de mensagens;
- Model: estabelece a ligação com os servidores de base de dados e define os modelos dos dados persistentes;
- WebSocket server: inicia o servidor de WebSockets e estabelece a ligação com os clientes;
- Broker-WebSocket interface: Associa a cada canal de WebSockets a fila de mensagens correspondente;
- Peer Server: Suporta o registo e descoberta de clientes que pretendam iniciar comunicações WebRTC.

4.3. Ambientes de desenvolvimento e produção

Um dos maiores obstáculos no desenvolvimento de *software* está ligado à complexidade da configuração do ambiente de desenvolvimento, sendo necessário instalar uma panóplia de diferentes tecnologias. Para além disso, cada membro da equipa tem por norma um ambiente de desenvolvimento diferente (por exemplo utilizando diferentes versões de sistemas operativos, bibliotecas e serviços) o que vai gerar problemas difíceis de replicar de máquina para máquina. Este problema acentua-se no momento de colocar o sistema em produção, pois o sistema foi desenvolvido e testado num ambiente de desenvolvimento que é habitualmente muito diferente do ambiente de produção, com sistemas operativos, serviços e configurações mais robustas e restritas.

De forma a evitar esse constrangimento foi montado um ambiente de desenvolvimento suportado pelo Vagrant [46], uma camada de abstração para a criação de máquinas virtuais. Com esta ferramenta é possível definir num só documento de configurações, denominado Vagrantfile, o sistema operativo (OS), as portas partilhadas entre o *host* e a máquina virtual, os recursos físicos alocados (CPU e RAM), os comandos a executar no momento da criação e a cada vez que a máquina é iniciada. Desta forma é possível partilhar o Vagrantfile pela equipa de desenvolvimento e assim todos os elementos estarão a desenvolver no mesmo ambiente virtual, com as mesmas configurações.

Apesar de o Vagrant ser uma ferramenta útil para auxiliar o desenvolvimento, não é a solução adequada para ambientes de produção, pois implica utilizar uma máquina virtual completa e conseqüentemente mais pesada. Numa arquitetura em microsserviços que se pretende escalável e robusta, é necessária a orquestração dos serviços e a recuperação imediata de um destes em situações de falha, sendo por isso mais eficaz utilizar um sistema de *containers*, em que cada *container* terá um microserviço associado.

A virtualização baseada em *containers* destaca-se das máquinas virtuais pela sua performance de execução, eficiência de recursos e portabilidade do sistema, pois, ao invés das máquinas virtuais que requerem instâncias de sistema operativo independentes assentes sobre um emulador (*hypervisor*), os *containers* partilham o mesmo *kernel* do sistema operativo anfitrião (Fig. 17), tornando a sua arquitetura mais simplista e possibilitando a aplicação da imagem de um *container* para diferentes *hosts* [47].

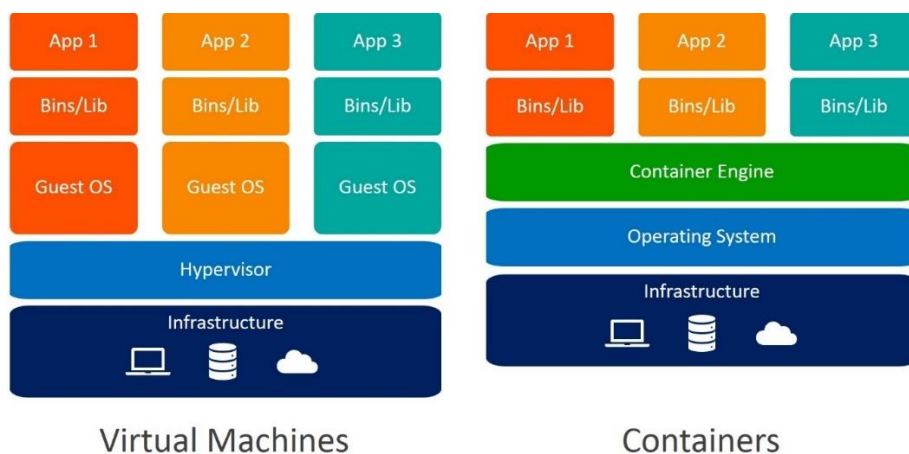
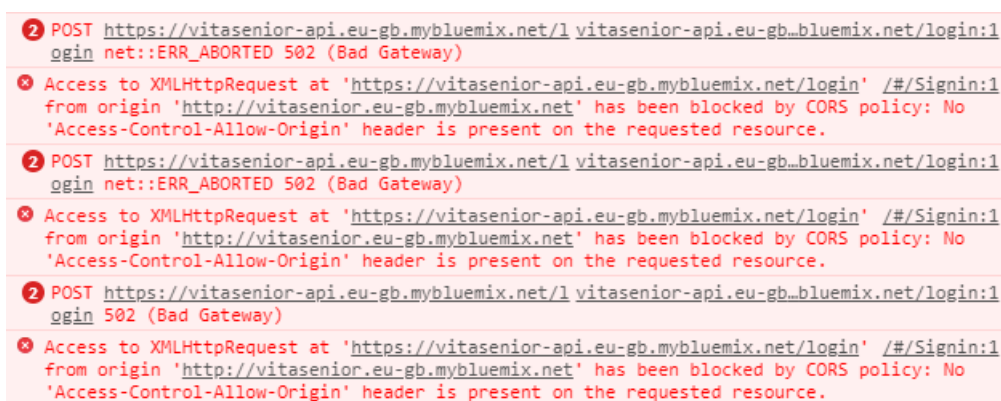


Fig. 17 - Comparação entre máquinas virtuais e *containers* [48].

Esta flexibilidade fomenta a utilização de técnicas de Agile e DevOps, aplicando os conceitos de testes e integração contínuos, uma vez que a alocação de recursos é mais eficiente, os ambientes de produção são os mesmos e os tempos de aplicação são mais reduzidos.

A infraestrutura de *Cloud* do projeto VITASENIOR-MT, devido à colaboração com uma empresa do grupo IBM, foi implementada na plataforma IBM Cloud que, pelos motivos apresentados anteriormente, é baseada na arquitetura em *containers*, facilitando assim o processo de contabilização e promovendo a oferta do modelo de negócio PaaS.

Apesar de todos os benefícios da metodologia PaaS, há certos cuidados que o programador deve ter em atenção, nomeadamente aos requisitos impostos pela plataforma, uma vez que muita responsabilidade é passada para o lado do provedor. Este tipo de solução PaaS pode tornar o processo semelhante a lidar com uma *blackbox*, pois o programador submete o código, espera o resultado, mas não sabe exatamente o processo no seu interior. No caso do VITASENIOR-MT ocorreu após uns meses de utilização uma falha do serviço, que tudo indicava ser uma questão de encaminhamento de pedidos entre o *reverse proxy* e a aplicação (Fig. 18), no entanto, após contacto com a equipa de suporte, percebeu-se que o erro ocorria pela incompatibilidade de versões entre o ambiente fornecido e o definido para a aplicação, pois a plataforma sofreu (sem controlo da nossa parte) uma atualização que inviabilizou a correta execução do serviço.



```
2 POST https://vitasenior-api.eu-gb.mybluemix.net/1 vitasenior-api.eu-gb.bluemix.net/login:1
  ogin net::ERR_ABORTED 502 (Bad Gateway)
x Access to XMLHttpRequest at 'https://vitasenior-api.eu-gb.mybluemix.net/login' /#/SignIn:1
  from origin 'http://vitasenior.eu-gb.mybluemix.net' has been blocked by CORS policy: No
  'Access-Control-Allow-Origin' header is present on the requested resource.
2 POST https://vitasenior-api.eu-gb.mybluemix.net/1 vitasenior-api.eu-gb.bluemix.net/login:1
  ogin net::ERR_ABORTED 502 (Bad Gateway)
x Access to XMLHttpRequest at 'https://vitasenior-api.eu-gb.mybluemix.net/login' /#/SignIn:1
  from origin 'http://vitasenior.eu-gb.mybluemix.net' has been blocked by CORS policy: No
  'Access-Control-Allow-Origin' header is present on the requested resource.
2 POST https://vitasenior-api.eu-gb.mybluemix.net/1 vitasenior-api.eu-gb.bluemix.net/login:1
  ogin 502 (Bad Gateway)
x Access to XMLHttpRequest at 'https://vitasenior-api.eu-gb.mybluemix.net/login' /#/SignIn:1
  from origin 'http://vitasenior.eu-gb.mybluemix.net' has been blocked by CORS policy: No
  'Access-Control-Allow-Origin' header is present on the requested resource.
```

Fig. 18 – Registos de falha de serviço por *Bad Gateway*.

Apesar do sistema ter sido desenvolvido com o intuito de ser executado na infraestrutura da IBM Cloud, foi também implementada uma versão de produção do sistema em ambiente local (servidor *in-house*). Nesse sistema foi utilizado como gestor de *containers* o Docker [49]. Para a implementação da arquitetura com o Docker foi necessário para cada serviço de Node.js definir um ficheiro (Dockerfile), que constrói a imagem do serviço a correr, contendo a referência da imagem base do ambiente de execução, os comandos de instalação de dependências, construção da aplicação e de início de atividade. Para os servidores de base de dados, filas de mensagens e *reverse proxy* foram utilizadas imagens disponibilizadas no repositório do Docker (Docker Hub). Com as imagens de cada serviço definidas, foi necessário configurar o método de arranque de todo o sistema através do Docker-Compose, onde se define a rede partilhada entre os serviços, as diretorias partilhadas entre os *containers* e o *host* OS, as variáveis de ambiente que definem a autenticação dos serviços, a ordem de arranque dos mesmos e as políticas de reinicialização. Com esta configuração é possível, em qualquer momento, escalar a solução para outras máquinas, por meio de Docker Swarm ou migrar o sistema para outra máquina, sendo o único requisito ter o Docker instalado.

4.4. Escalabilidade

Sendo a infraestrutura de *Cloud* do projeto VITASENIOR-MT responsável pela agregação e disponibilização dos dados, a garantia da comunicação com os seus clientes e os tempos de resposta são tópicos vitais ao funcionamento da solução.

Para isso foi desenvolvida uma API RESTful, que disponibiliza um conjunto de *endpoints* que são acessíveis tanto à Vitabox como aos clientes da aplicação web (no *browser*), desta forma foi possível abstrair a *Cloud* da diferença entre os clientes, lidando com eles da mesma forma.

Pelo facto de o serviço ser *stateless*, isto é, a API não guardar informação da sessão dos utilizadores e não rastreia a sua atividade, facilita o desenvolvimento do componente *frontend* de forma independente do servidor, tornado o cliente responsável pelo fluxo da comunicação, distribuindo melhor o serviço e libertando grande parte do processamento do

servidor, uma vez que o *render* das vistas é realizado no *browser*. Assim os dados são enviados no formato de anotação de objectos em JavaScript (JSON), que apresenta menor *overhead* que reenviar a página HTML completa a cada novo pedido. Por outro lado, a complexidade de processamento necessária para a composição de tabelas e gráficos é transferida para o cliente, limitando o servidor às ações mais simples de CRUD.

Apesar de grande parte da complexidade poder ser transferida para o cliente a verdade é que não se pode descartar a responsabilidade de filtrar e validar os valores recolhidos pelos sensores e encaminhados pela Vitabox. Por esse motivo cabe à *Cloud* a responsabilidade de validar os valores, aplicar os limites definidos como aceitáveis ou inválidos, no caso de parâmetros biométricos, aplicar o perfil clínico do respetivo paciente, atualizar as datas de receção dos equipamentos e gerar alertas caso se justifique.

Durante uma primeira versão do sistema, todo este processo estava a prejudicar os tempos de resposta do serviço na *Cloud*. Apesar do ambiente de execução, neste caso o Node.js, ser assíncrono, existe um limite para os processos que consegue lidar no seu *pipeline*. Neste caso, se um processo se tornar demasiado lento, todos os pedidos seguintes ficarão congelados à espera de tempo de processamento para serem executados. Para colmatar o problema em causa, comum em arquiteturas monolíticas, foram estudadas e testadas várias abordagens distintas (Fig. 19) até chegar à solução final.

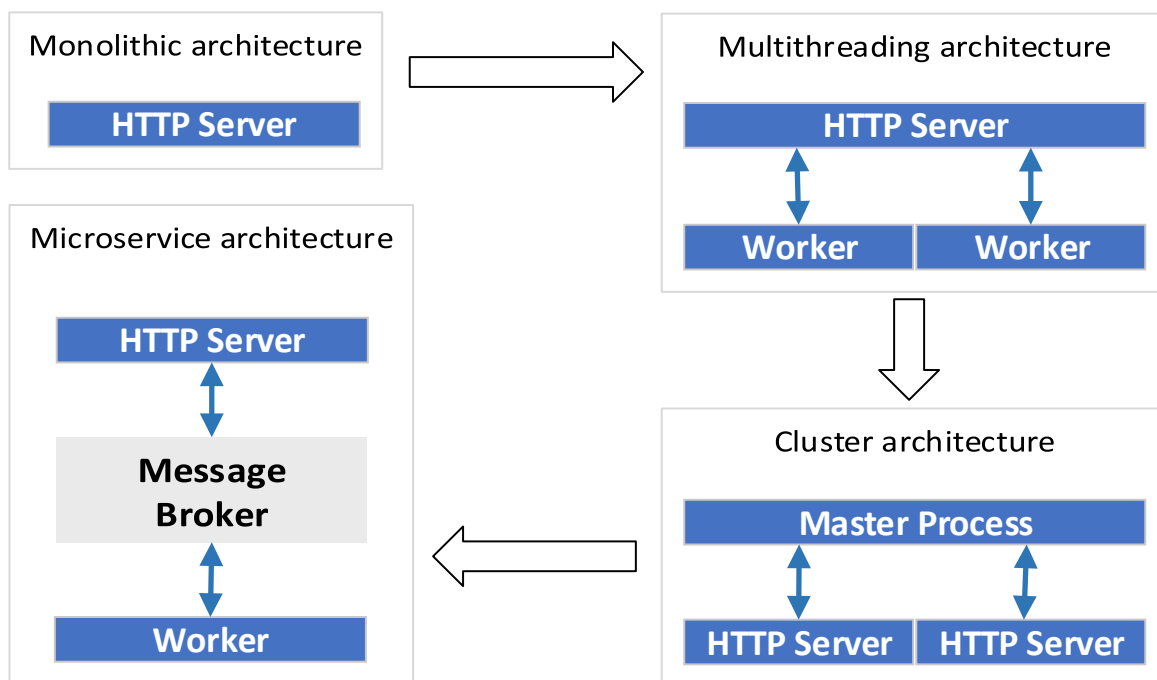


Fig. 19 - Evolução do paradigma da *Cloud*.

Na primeira fase optou-se pela utilização de *worker threads* para executar os métodos que estavam a atrasar todo o sistema em background, aplicando os mecanismos de *multithreading*, enquanto o processo *main* do *event loop* do Node.js continuava a processar os restantes pedidos. A solução em teoria era plausível, mas depois de alguns testes foi encontrado novo problema: a impossibilidade de utilização de bibliotecas externas no código criado para ser executado pelos *workers*. Isto acontece pois a biblioteca não foi preparada para esse fim, podia apenas importar *scripts* em JavaScript. A utilização de *workers* torna-se assim uma solução inviável uma vez que no nosso sistema os referidos processos *workers* necessitam de bibliotecas (*packages*) para interagir com as bases de dados e que estas são tipicamente desenvolvidas com módulos em C++ (por questões de performance).

De seguida optou-se por tirar o máximo partido do facto do serviço ser *stateless* e implementou-se uma arquitetura em *cluster*, ou seja, levantavam-se tantos serviços homogêneos da arquitetura monolítica quantos os recursos da máquina permitissem e balanceava-se os pedidos dos clientes entre as várias instâncias. A solução funcionou temporariamente, no entanto o que estava a acontecer era a propagação do problema, além

de escalar a solução estava também a ser escalado o problema. Por outro lado, a longo prazo tornar-se-ia insustentável a utilização da arquitetura, pois continuava a prejudicar a disponibilização de todos os serviços em prol de uma minoria que exigia mais processamento

Por fim foi decidido resolver o problema partindo o serviço em serviços menores, isolando todas as tarefas que necessitavam de mais recursos computacionais num serviço à parte. Desta forma a sua execução ficava independente da execução da API, que assim não atrasava os pedidos dos clientes pois delegava as tarefas exaustivas para o serviço Worker. Para a comunicação entre os serviços é utilizado um servidor de filas de mensagens, no caso o RabbitMQ [50]. Desta forma a API publica numa fila uma mensagem com os valores dos sensores a processar e do outro lado os Workers que se encontram à escuta vão requerendo mensagens conforme os processam.

O RabbitMQ permite a criação de diferentes filas de mensagens, sendo que as utilizadas pelo VITASENIOR-MT são apenas as “Work Queues” e as “Publish/Subscribe” (Fig. 20). As “Work Queues” são as mais simples, é criada uma fila de mensagens com vários consumidores e no momento da publicação o RabbitMQ aplica um Round-Robin¹⁰ para definir qual é o consumidor que vai receber a mensagem, sendo enviada apenas para um. Este método foi o aplicado para a comunicação entre os serviços API e Worker de modo a poder escalar o nó de Worker. O outro método consiste na publicação da mesma mensagem em várias filas e todos os que se encontram à escuta nas respetivas filas vão receber a mesma mensagem.

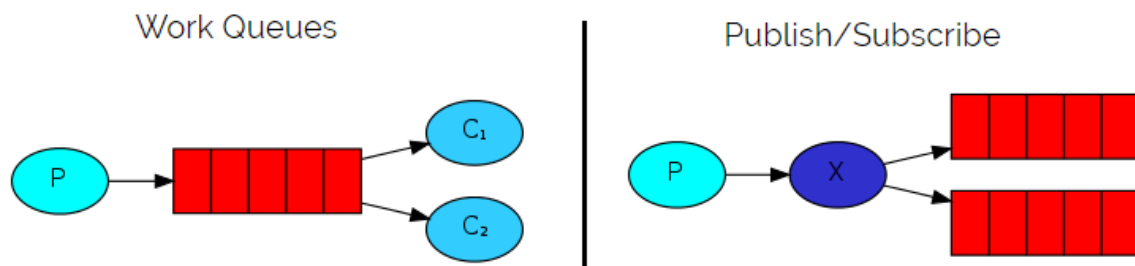


Fig. 20 - Tipos de filas de mensagens do RabbitMQ [50].

¹⁰ Algoritmo de distribuição de tarefas/recursos de forma circular entre consumidores

Após a questão do processamento dos valores dos sensores resolvido foi necessário um mecanismo de alerta para as situações em que os valores reportados pelos sensores ambientais ou biométricos não se encontravam no intervalo aceitável.

Continuando no caminho dos microserviços a solução mais óbvia foi a criação de um serviço de WebSockets. Esse serviço vai registrar cada cliente a uma fila de mensagens apenas deste (pessoal). Caso este seja um cuidador, vai também ser associado a uma fila de mensagens da cada Vitabox a que pertence. No caso do utilizador ligado ter o papel de médico, será associado a uma fila de mensagem de cada paciente (Fig. 21). O mecanismo de filas aplicado aos WebSockets é o de “Publish/Subscribe”, desta forma sempre que é gerado um alerta todos os cuidadores da respetiva Vitabox serão notificados. Se o alerta for de um parâmetro biomédico, também os médicos do paciente serão notificados. Esta solução permite escalar os nós do serviço de WebSocket, porque como as ligações estão associadas a filas de espera, independentemente do nó a que o cliente se encontre registado vai receber o alerta que vem da fila.

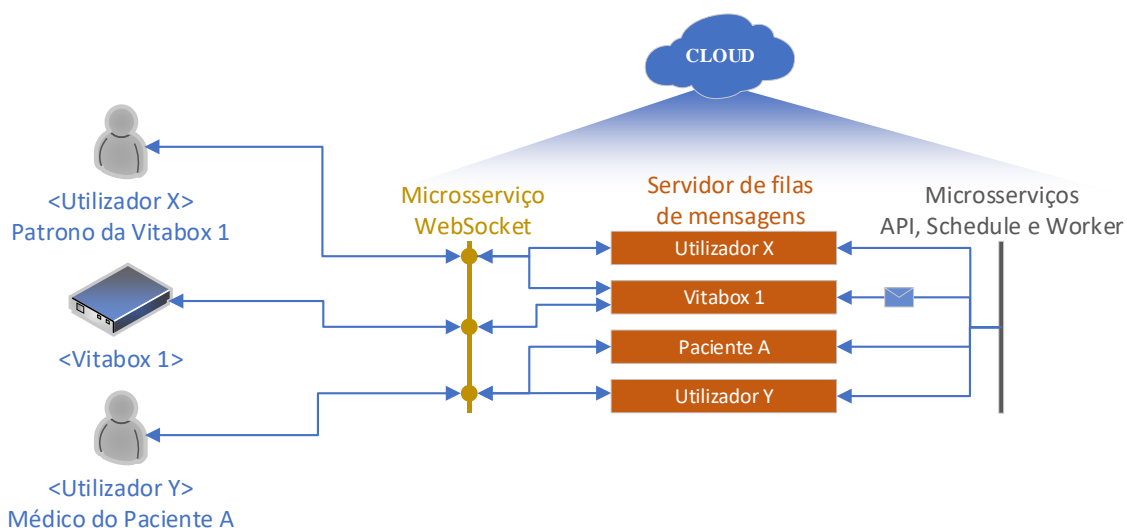


Fig. 21 – Associação de WebSockets a Filas de mensagens (Publish/Subscribe).

Posteriormente sentiu-se a necessidade de realizar chamadas *peer-to-peer* entre os pacientes (por meio da sua Vitabox) e os respectivos cuidadores e médicos clientes e também de agendar tarefas em períodos rotativos, provando mais uma vez que a arquitetura de microsserviços permite escalar em funcionalidades, pois o desenvolvimento destes serviços não influenciou o funcionamento dos já existentes.

Esta arquitetura permite ao sistema não só escalar em funcionalidades, mas também em número de instâncias por microsserviço de acordo com os tipos de acesso. Isto é, se por exemplo for gerada uma quantidade elevada de dados de sensores, mas não houver muitos utilizadores a consultar os mesmos, será necessário alocar mais recursos (escalar instâncias) do nó de Worker, sem necessidade de alterar o da API. Se por outro lado forem gerados poucos dados ambientais ou biológicos, mas desses, todos eles geram alertas, então o nó Worker não necessitará de tantos recursos, mas o nó de WebSocket sim, pois terá que lidar com o encaminhamento de um grande número de alertas. Isto significa que o sistema está preparado para se adaptar a todos os tipos de cenários.

4.5. Armazenamento

A infraestrutura de *Cloud* está constantemente a receber dados gerados pelos sensores das várias Vitaboxes instaladas. Dessa forma, é necessário um sistema de armazenamento de informação capaz de suportar uma enorme quantidade leituras e escritas. Por essa razão optou-se por utilizar uma base de dados não relacional para o alojamento dos valores recebidos, dos alertas gerados pelos mesmos e dos *logs* dos utilizadores. O sistema de bases de dados não relacional escolhido para o efeito foi o MongoDB [51], uma base de dados orientada a documentos, em que as pesquisas são de complexidade linear e que permite escalar horizontalmente.

O MongoDB, apesar de apresentar grande performance no tratamento de um grande volume de dados, tem uma característica que limita a sua utilização exclusiva em toda a solução: o facto de não ser ACID. Isto é, não disponibiliza um ambiente transaccional que garanta a atomicidade, consistência, isolamento e durabilidade dos dados. Este facto para processos

como autenticação de utilizadores, registo de Vitaboxes e criação de perfis clínicos de pacientes seria um risco que não poderia ser posto em consideração. É importante referir que a partir da versão 4.0, lançada em meados de 2018, o MongoDB passou a suportar transações multidocumento [52], sendo que à data já o projeto estava em desenvolvimento há aproximadamente 10 meses e, para além de ser uma funcionalidade nova e ainda não devidamente testada em ambientes reais, não era exequível refazer trabalho já desenvolvido.

Outro fator que pesou na tomada de decisão da não utilização do MongoDB para toda a solução é a característica dominante de ser *schemaless*¹¹. Não sendo necessariamente um ponto negativo, tendo a sua utilizada em cenários específicos, no caso do VITASENIOR-MT apresenta-se como um obstáculo aos critérios necessários para definir os modelos dos dados que suportam as regras de um sistema tão restrito em políticas de acesso.

Dessa forma foi também utilizada uma base de dados relacional, o MySQL [53], já bem estabelecida no mercado e reconhecida pela sua capacidade de transações seguras. Esta ferramenta foi utilizada para armazenar os dados de utilizadores, Vitaboxes, equipamentos, pacientes, perfis clínicos e todas as relações entre os mesmos. É importante salientar que o MySQL também suporta armazenamento não relacional, com base de dados orientada a documentos [54], no entanto por estar ainda num nível de pouca maturidade optou-se por não tirar partido dessa característica.

Como o sistema estava implementado na plataforma IBM Cloud, foram alocadas instâncias dedicadas para as bases de dados pretendidas. Uma lição retirada desta opção é a que devemos tirar partido da propriedade elástica que a *Cloud* tem para oferecer, pois na implementação do VITASENIOR-MT foi utilizado um plano fixo de armazenamento e a certo momento as aplicações deixaram de responder nos tempos desejados. A razão para tal foram os limites fixos de memória alocados para cada instância (Fig. 22). O plano de utilização em vigor não estava habilitado a alocar recursos conforme a exigência de utilização (escalar automaticamente) e desta forma o serviço encontrava-se limitado.

¹¹ Não obriga a seguir uma estrutura definida, sendo flexível a alterações mediante o contexto dos dados.

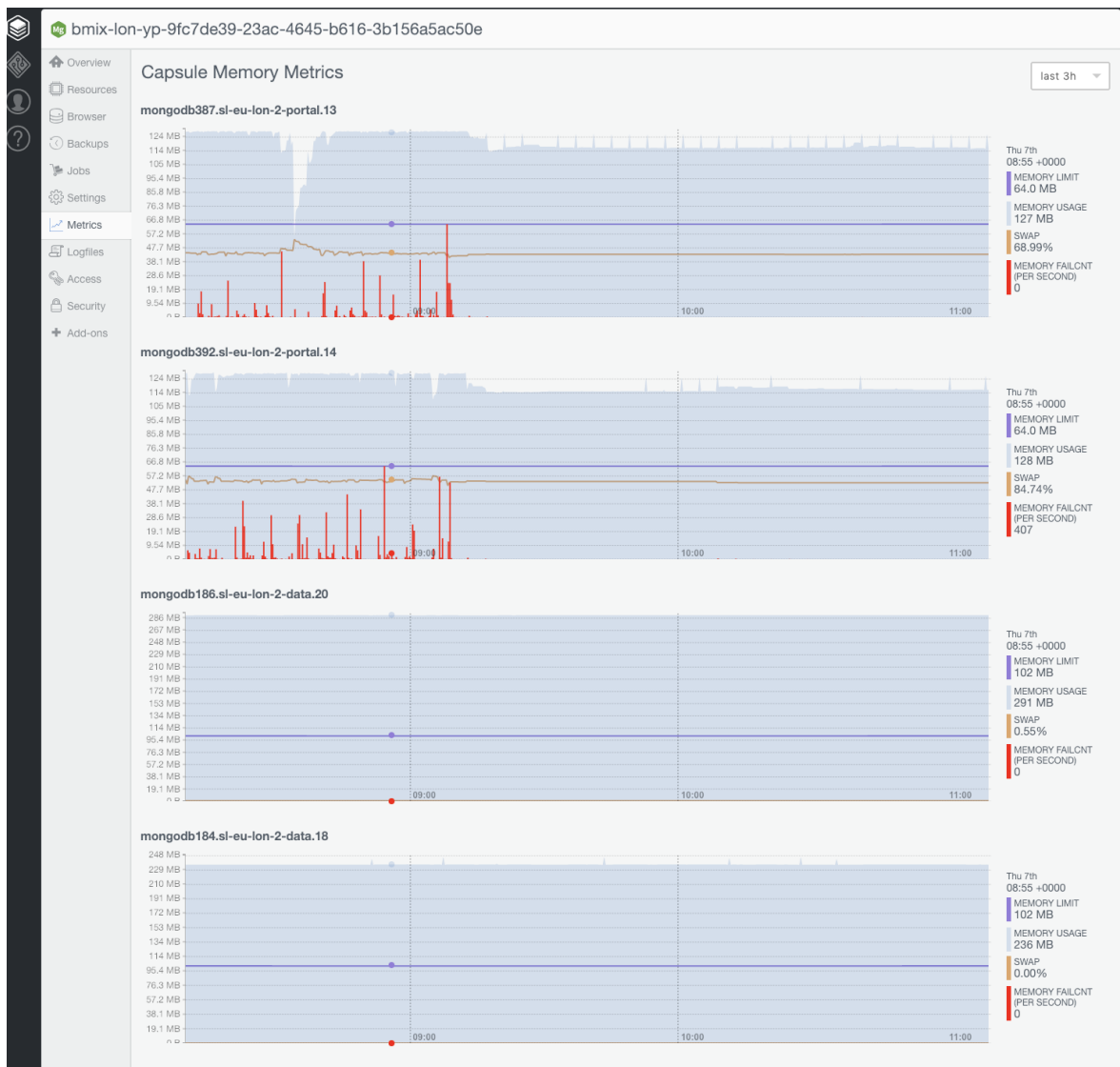


Fig. 22 - Limite de memória nas bases de dados.

Por fim, existia também a necessidade de ter um repositório de ficheiros, em que fosse possível alojar as imagens dos utilizadores e os dados anonimizados extraídos para análise. A solução passou pela utilização de um *object storage*, uma tecnologia que permite separar os ficheiros em repositórios (locais de armazenamento) que podem ser atribuídos a diferentes aplicações, sendo possível ainda definir uma chave de acesso que é renovada num intervalo definido de tempo, dando acesso aos ficheiros apenas a quem essa chave for transmitida.

Este é o modelo de alojamento para ficheiros que são partilhados na rede, utilizado pelos gigantes da indústria tecnológica como o caso do Facebook, Google e LinkedIn.

Com o *object storage* garante-se a consistência e duração dos ficheiros armazenados, uma vês que os dados dos serviços que se encontram em *containers* são voláteis. Permite também ao cliente aceder diretamente ao ficheiro, desde que lhe tenha sido transmitido previamente o caminho do ficheiro e a sua chave, sem necessitar se sobrecarregar a API com o envio de ficheiros.

4.6. Segurança

De acordo com o Regulamento Geral de Proteção de Dados (GDPR), todos os dados que diretamente ou indiretamente possam identificar um utilizador devem seguir um conjunto de medidas tecnológicas e organizacionais para garantir a segurança e os níveis de acesso aos dados [55]. Para isso foram implementados diversos mecanismos de forma a garantir que ninguém acede indevidamente à informação pessoal dos utilizadores e, mais importante, aos dados dos pacientes recolhidos através das Vitaboxes. As medidas aplicadas vão desde a comunicação, ao armazenamento e às políticas de acesso.

De forma a garantir a segurança da comunicação entre os clientes e a *Cloud* são utilizados canais seguros (HTTPS e WSS), desta forma toda a comunicação é cifrada por uma chave simétrica negociada entre o cliente (Vitabox ou aplicação *web*) e a *Cloud*. Para que a negociação seja possível, isto é, para que o cliente confirme que o servidor da *Cloud* é realmente quem se anuncia, existe um certificado assinado por uma autoridade certificadora (CA) já previamente reconhecida pelo cliente. Na implementação na IBM Cloud a responsabilidade de fornecer o certificado e garantir a comunicação segura coube ao provedor. No caso da implementação *in-house*, foi gerado um certificado através da Let's Encrypt [56], uma CA livre e automatizada, colaboradora da Linux Foundation. Para o Let's Encrypt atribuir um certificado ao servidor é necessário instalar um *script* no *host* e definir um *hostname*, de seguida esta ferramenta levanta um serviço de negociação na porta 80 para que seja autenticado o *hostname* e transferido o certificado. Estes certificados têm uma

validade de 90 dias, por isso é possível agendar a renegociação de certificado no agendador de tarefas do sistema *host*. Após a primeira negociação a ferramenta já conhece o *hostname*, daí o processo de renegociação não necessitar de intervenção do utilizador e ocorrer automaticamente. Com o certificado criado, este é definido no servidor de *reverse proxy*, no caso foi utilizado o NGINX [57], que pode forçar todas as ligações a serem seguras, isto é, todos os pedidos em HTTP ou WS progredirem para HTTPS/WSS.

Para a comunicação entre os microsserviços e os servidores de bases de dados foi utilizada autenticação, sendo criadas contas apenas para o acesso aos mesmos pelos serviços. Na camada responsável pelos dados (Modelos ou Models) dos microsserviços foram ainda utilizados sistemas de mapeamento objeto-relação (ORM) e mapeamento objeto-documento (ODM), que além de facilitarem a organização da lógica do negócio, contêm também mecanismos de proteção a possíveis ataques como o caso de injeção de SQL.

No caso da solução *in-house*, com todo o ambiente montado em *containers* de Docker, foi criada uma rede local dedicada apenas aos serviços, e não foi atribuída nenhuma porta partilhada entre o *host* e os *containers* que suportavam os servidores de base de dados, sendo apenas possível aceder aos mesmos dentro do domínio privado criado pelo Docker, onde se encontram também os serviços. Isto força a que todas as interações entre os utilizadores e as bases de dados ocorram por meio dos microsserviços, não deixando *backdoors* de acesso.

Por cima de todo este sistema foi ainda implementada uma *firewall* que deixa apenas passar pedidos aos portos 80 e 443 (para os microsserviços), 20 (para ligar por SSH) e 15672 para o cliente de gestão do RabbitMQ. Para aceder ao cliente de gestão do RabbitMQ é necessária autenticação, tendo sido alterados os utilizadores padrão e as ligações por SSH utilizando *password* foram desabilitadas, sendo apenas possível ligar as máquinas que já tenham previamente registado um certificado de acesso no *host*.

Sendo o propósito base do projeto implementar o sistema na IBM Cloud, foi necessário assumir um comportamento “honesto mas curioso” do respetivo provedor [58], isto é, embora o provedor tenha boas intenções, pode ter curiosidade em consultar os dados. Dessa forma é necessário cifrar o conteúdo das bases de dados de forma a mascarar o seu significado. No entanto, o processo de cifragem implica um custo extra de computação,

podendo atrasar o sistema desnecessariamente. Para colmatar esse problema foram analisados os campos que realmente necessitavam de ser cifrados, isto é, que pudessem identificar os utilizadores e pacientes, e concluiu-se que o grosso do volume de dados estava nos valores das leituras dos sensores, mas esses valores só por si não identificam inequivocamente um paciente, pois são apenas séries de números muito semelhantes entre cada utilizador. Assim, por uma questão de otimização de recursos, sem prejudicar a segurança do sistema, optou-se por cifrar apenas os campos que relacionavam diretamente os pacientes e os utilizadores, como os nomes, contactos, moradas e nomes de ficheiros. Desta forma, eventuais atacantes poderão eventualmente ler valores numéricos (ex: temperatura de uma divisão, batimento cardíaco), mas não conseguem descortinar a quem estes pertencem.

Como referido anteriormente foram implementadas diferentes permissões no sistema (cargos), sendo elas as de administrador, patrono, cuidador e médico. Uma outra permissão não perceptível aos restantes utilizadores é a da própria Vitabox, que para a *Cloud* apresenta-se como um cliente com o mesmo comportamento que os restantes. A validação das permissões dos utilizadores é obtida a partir do JSON Web Token (JWT) presente no cabeçalho de autenticação dos pedidos HTTP. Este *token* é assinado por uma chave privada da *Cloud* de forma a garantir a autenticidade do mesmo. No *token* estão presentes as permissões do utilizador bem como o seu identificador único. O JWT permite tirar o máximo potencial de um serviço RESTful uma vez que a identificação do cliente vem em cada pedido evitando guardar e ter de consultar a informação sobre cada sessão no servidor e permite também a autenticação para os WebSockets, uma vez que o *token* é facilmente validado por um serviço com acesso à chave privada do serviço emissor do mesmo.

Por fim outra medida de segurança implementada foi o pré-registo de todo o equipamento a ser utilizado pelo sistema. Desta forma, além de facilitar a sua gestão e a usabilidade, permite também garantir que não haverá equipamento fantasma no sistema a emitir valores falsos em grande escala com o objetivo de congestionar os serviços.

4.7. Acesso

Para libertar algum processamento da *Cloud*, em vez de realizar a renderização das páginas a apresentar ao utilizador do lado do servidor (*server-side-view*), foi desenvolvida uma aplicação *web* utilizando Vue.js [59], uma *framework* progressiva de JS para desenvolvimento de *frontend*. Esta abordagem permite uma melhor estruturação do projeto, tornando o desenvolvimento do *frontend* independente dos microsserviços. Desta forma todo o processamento para apresentar gráficos e interfaces mais dinâmicas é transferido para o cliente, sendo os dados fornecidos pela API em JSON. Este processo reduz o gasto desnecessário de recursos computacionais do lado da infraestrutura de *Cloud*, passando parte deste processamento para o *browser* utilizado no dispositivo do lado do cliente. Para além disto, diminui o tráfego necessário na interação cliente-servidor, uma vez que os pacotes apenas com os dados em formato JSON são menores que transferir toda a informação e formatação desta em HTML a cada novo pedido.

Para que o desenvolvimento do *frontend* e da Vitabox fosse facilitado e possa ser feito por equipas independentes no futuro, foi criada a documentação de todas as rotas disponibilizadas pela API (Fig. 23). Esta informação, disponível numa página HTML, está também aberta ao público, abrindo a possibilidade de terceiros poderem contribuir para o projeto.

Esta documentação é gerada pela ferramenta apiDoc [60], que interpreta as anotações do código fonte de uma determinada diretoria (Fig. 24) e gera um conjunto de ficheiros estáticos a serem apresentados por um servidor web. Nesta documentação estão definidas as rotas a serem usadas, as permissões de utilizadores necessárias ao acesso, os parâmetros e cabeçalhos a enviar e as possíveis respostas em caso de sucesso ou erro, bem como exemplos demonstrativos daquilo que a API espera receber.

← → ↻ <https://vitasenior-mt.ipt.pt/docs/>

Filter... x

- Authentication**
 - 01) Register user
 - 02) Validate Email
 - 03) Login user
 - 04) Verify Token
 - 05) Change password
 - 06) Forgot Password
 - 07) Reset password
- Sensormodel**
 - 01) Create
 - 02) List
 - 03) Update
 - 04) Delete
- Boardmodel**
 - 01) Create
 - 02) List
 - 03) Update
 - 04) Delete
 - 05) Add Sensor
 - 06) Get Sensors
 - 07) Remove Sensor
- Profilemodel**
 - 01) Create
 - 02) List
 - 03) Update name
 - 04) Remove
 - 05) Add measure
 - 06) Remove measure
- User**
 - 01) Update photo
 - 02) List
 - 03) Get Logs
 - 04) Get patients
 - 05) Count patient requests
 - 06) Get patient requests
- Vitabox**
 - 01) Create
 - 02) Register
 - 03) Request Token
 - 04) Get info
 - 05) Get Settings
 - 06) Set Settings
 - 07) Update
 - 08) Delete
 - 09) Add User

Authentication - 01) Register user 1.0.0

POST

`/register`

Header

| Field | Description |
|-----------------|--|
| Accept-Version | Default value: <code>1.0.0</code> |
| Accept-Language | Default value: <code>pt</code> |
| Content-Type | Default value: <code>application/json</code> |

Parameter

| Field | Type | Description |
|----------|--------|--|
| email | string | valid email |
| name | string | valid name |
| password | string | must have at least one uppercase letter, one lowercase, one digit and a minimum 8 characters |

Success 200

| Field | Type | Description |
|-----------|---------|--|
| token | string | jwt valid for 8 hours and must be placed at "Authorization" header |
| id | string | user id |
| name | string | user name |
| email | string | user email |
| is_admin | boolean | flag indicating if is admin |
| is_doctor | boolean | flag indicating if is doctor |
| photo | string | user photo |

Response example:

```
{
  "token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImR4b251c-f1ca-4ecd-893e-56ae404ef8ef",
  "id": "84bb251c-f1ca-4ecd-893e-56ae404ef8ef",
  "name": "Administrator Exemple",
  "email": "admin@some.thing",
  "is_admin": true,
  "is_doctor": false,
  "photo": "8b2fe0d0-0311-494a-8e27-522407d21b0e44fe0662-1271-4f42-a764-eeb0ba87cd87a2d6f862-c7e9-43a1-8066-87f157da"
```

Fig. 23 - Documentação da API.

```

/**
 * @apiDefine auth
 *
 * @apiHeader Accept-Version="1.0.0"
 * @apiHeader Accept-Language="pt"
 * @apiHeader Content-Type="application/json"
 * @apiError {number} statusCode http status code: 500 to business logic errors and 401 to unauthorized
 * @apiError {string} statusMessage error description
 */

/**
 * @api {post} /register @1 Register user
 * @apiGroup Authentication
 * @apiName userRegister
 * @apiVersion 1.0.0
 * @apiUse auth
 * @apiParam {string} email valid email
 * @apiParam {string} name valid name
 * @apiParam {string} password must have at least one uppercase letter, one lowercase, one digit and a minimum 8 characters
 * @apiSuccess {string} token jwt valid for 8 hours and must be placed at "Authorization" header
 * @apiSuccess {string} id user id
 * @apiSuccess {string} name user name
 * @apiSuccess {string} email user email
 * @apiSuccess {boolean} is_admin flag indicating if is admin
 * @apiSuccess {boolean} is_doctor flag indicating if is doctor
 * @apiSuccess {string} photo user photo
 * @apiSuccessExample {json} Response example:
 * {
 *   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ijg0YmIyNTFjLWYxY2EtNGVjZC04OTNlTU2YWU0MDRlZjhlZiIsInJvbGUiOi
 *   "id": "84bb251c-f1ca-4ecd-893e-56ae404ef8ef",
 *   "name": "Administrator Exemple",
 *   "email": "admin@some.thing",
 *   "is_admin": true,
 *   "is_doctor": false,
 *   "photo": "8b2fe0d0-0311-494a-8e27-522407d21b0e44fe0662-1271-4f42-a764-eeb0ba87cd87a2d6f862-c7e9-43a1-8066-87f157da71
 * }
 */
exports.register = (req, res) => {
  business.user.tempRegister(req.body.email, req.body.password, req.body.name).then(
    temp_user => business.user.sendEmail("confirm_email", temp_user.email, temp_user._id).then(
      () => res.status(200).json({ result: true }),
      error => res.status(error.code).send(error.msg)),
    error => res.status(error.code).send(error.msg));
}

```

Fig. 24 - Anotações do código fonte para documentação.

Esta ferramenta permite ainda distinguir as configurações das rotas entre versões da API. Para que o cliente possa aceder a uma versão específica de uma rota deve definir no cabeçalho “*Accept-Version*” a versão que solicita, senão será encaminhado para a mais recente.

Outro cabeçalho importante de mencionar é o de internacionalização, pois tanto no *frontend* como na API foram desenvolvidos mecanismos de internacionalização. Estes permitem ao cliente requerer que os dados sejam disponibilizados numa determinada língua e a API devolve a informação na linguagem requerida. De modo a isso ser possível o cliente deverá definir o cabeçalho “Accept-Language” com o código do país seguindo a norma ISO 3166-1 alpha-2 [61]. Para a tradução tanto do *frontend* como da API foi seguido padrão *i18n* com a definição de uma estrutura chave-valor para cada linguagem suportada, em que a chave é uma etiqueta única (*tag*) e o valor corresponde ao texto traduzido (Fig. 25).

```
{
  "unauthorized": "Não autorizado",
  "invalid_thresholds": "Não foi possível validar os limites do sensor",
  "trespassing_thresholds": "Os valores do sensor estão fora dos limites possíveis",
  "cannot_insert_record": "Não foi possível encontrar os registos do sensor",
  "cannot_remove_record": "Não foi possível remover os registos da board",
  "cannot_find_sensor": "Não foi possível encontrar o sensor",
  "cannot_update_profile": "Não foi possível atualizar o perfil",
  "cannot_update_sensor": "Não foi possível atualizar o sensor",
  "cannot_update_board": "Não foi possível atualizar a board",
  "cannot_find_board": "Não foi possível encontrar a board",
  "warning_up_limit": "o valor de %s do(a) %s está acima do recomendado",
  "warning_down_limit": "o valor de %s do(a) %s está abaixo do recomendado",
  "warning_up_tending": "o valor de %s do(a) %s está a aumentar mais que o esperado",
  "warning_down_tending": "o valor de %s do(a) %s está a diminuir mais que o esperado",
```

Fig. 25 - Dicionário de tradução.

Toda a tradução poderia ser transferida para o cliente, definindo previamente os dicionários de tradução no cliente e assim a API respondia apenas com as *tags* necessárias à tradução. No entanto, com a perspectiva de evolução do sistema, em que poderão existir vários clientes distintos, ou *frontends*, (que na verdade já existem, uma vez que para a API a Vitabox é outro cliente), e para evitar replicação de dicionários em diversos clientes, implementou-se a tradução de conteúdo fornecido pela API do lado do servidor, deixando para o cliente apenas a tradução do conteúdo estático das suas interfaces. Exemplificando este conceito, tanto a descrição dos *logs* dos utilizadores, como a descrição dos alertas, são guardadas em

forma de *tag* na base de dados e é definido um dicionário com as traduções para cada linguagem na API. No momento do pedido o serviço verifica a linguagem requerida, traduz os dados e envia.

O *frontend* desenvolvido suporta também WebRTC [62], permitindo videochamadas Peer-to-peer (P2P) entre clientes, o que garante uma latência mais reduzida que no típico cenário da utilização de um servidor intermédio. Para isto os clientes apenas precisam de se registar no servidor de *Peer*, que coordena a ligação entre clientes e lida com a tradução do endereço de rede (NAT) e com as *firewalls*, pois de outra forma os clientes não conseguiriam encontrar-se.

Outros aspetos positivos desta abordagem de desenvolvimento do *frontend* são a facilidade de integração com APIs externas como o caso do serviço de mapas Google Maps, ter acesso ao um armazenamento local, como a *localStorage* e a *indexedDB* e a possibilidade de desenvolver uma aplicação web progressiva (PWA).

Uma PWA distingue-se das aplicações nativas pelo facto de poder ser executada em qualquer plataforma, uma vez que a sua base é o *browser*. Para que ela possa ser utilizada, a comunicação tem de ser obrigatoriamente sobre HTTPS. Na primeira utilização o utilizador pode instalar a aplicação (Fig. 26) e a partir de então abri-la em qualquer momento como outra aplicação nativa (Fig. 27).

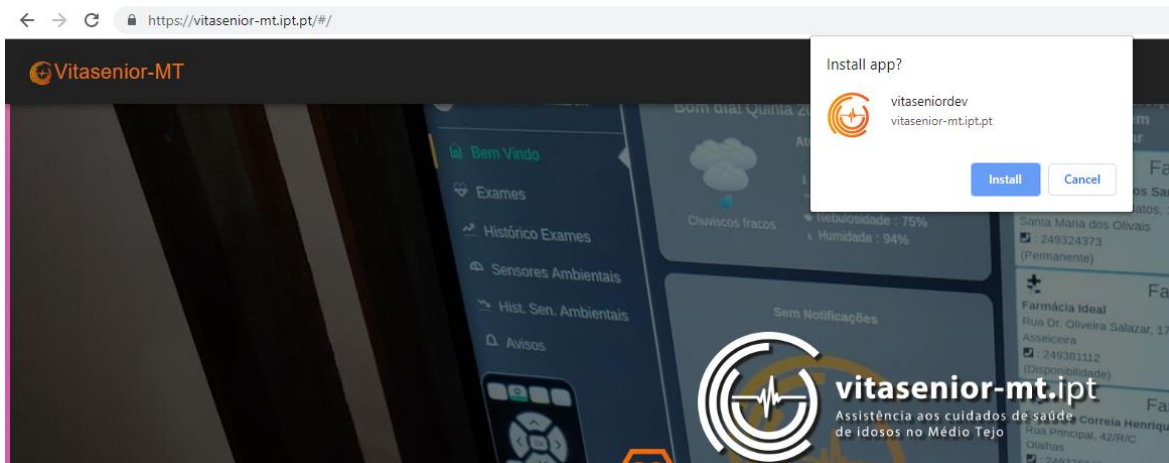


Fig. 26 - Instalação da PWA.

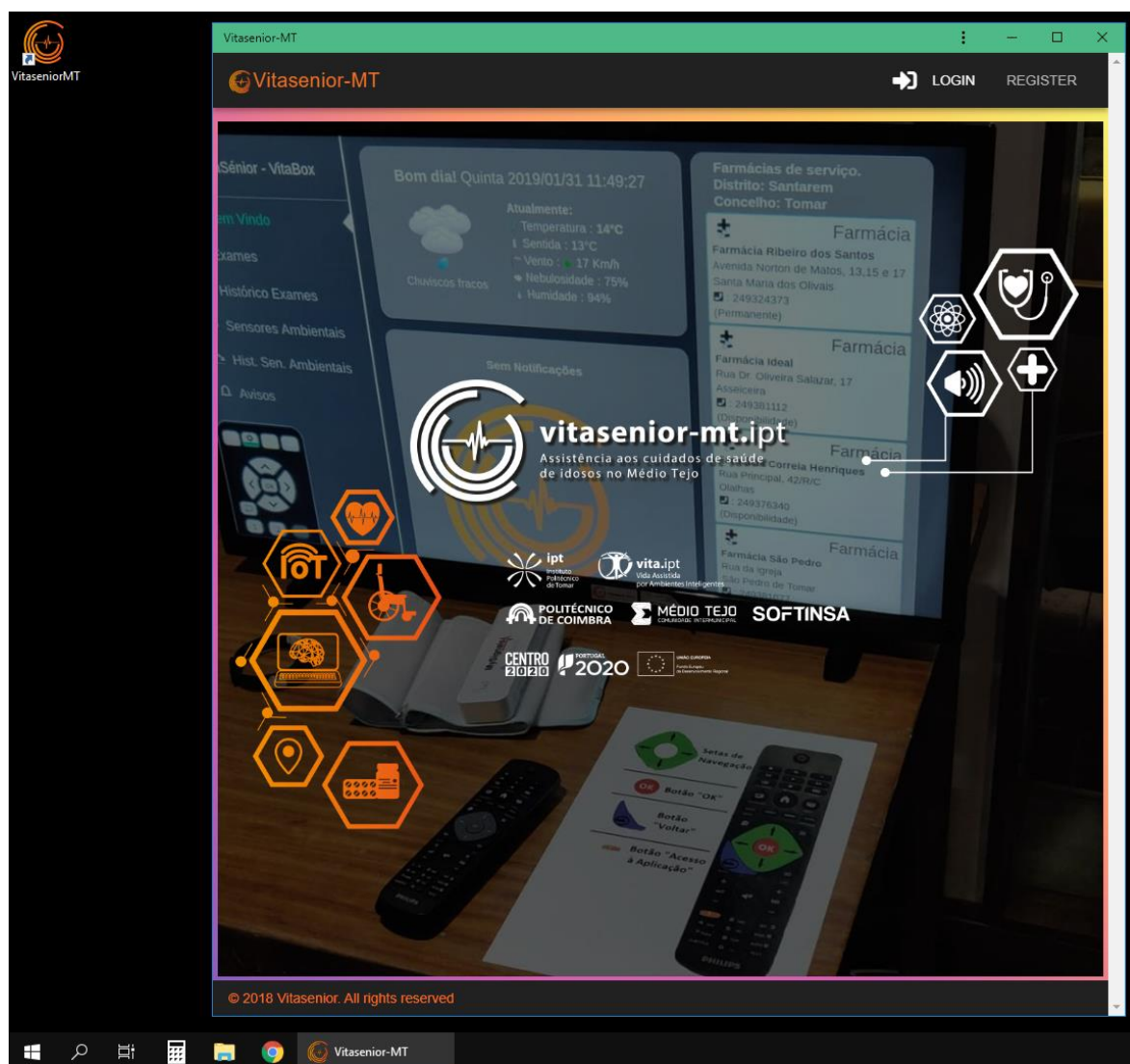


Fig. 27 - Execução da PWA.

Esta funcionalidade facilita em muito a utilização do sistema, em especial para os cuidadores e médicos, que em muitos casos não estão tão familiarizados com novas tecnologias, tornando o processo de abertura e utilização semelhante a uma aplicação de *desktop*. Esta funcionalidade aliada às bases de dados locais, embutidas nos *browsers*, e à facilidade de integração com APIs externas, possibilita o desenvolvimento de muitas funcionalidades *offline*¹². Ou seja, é possível transferir previamente um conjunto de configurações para o cliente e caso não seja possível à *Cloud* satisfazer os pedidos de imediato, é guardado um registo dos pedidos para que possam ser mais tarde executados, e aplicando as devidas alterações localmente na aplicação, transparecendo ao utilizador o funcionamento total.

¹² Sem que haja ligação constante à internet

Capítulo 5

Testes e resultados

Ao longo do desenvolvimento da componente de *Cloud* do VITASENIOR-MT o código-fonte foi sendo constantemente validado, de forma a garantir a coerência e funcionalidade do sistema. Numa fase primária de teste de cada nova funcionalidade foi usado o Postman [63], um ambiente de desenvolvimento integrado (IDE) de APIs que na sua funcionalidade mais básica permite submeter pedidos a serviços *web* e obter a sua resposta, tornando possível perceber se a rota está a desempenhar devidamente a sua função.

Após os testes iniciais e com o crescimento do sistema foi necessário automatizar o processo de execução de testes, pois a partir de certo ponto as alterações de determinadas funcionalidades poderiam influenciar os resultados de outras e era fácil para o programador perder a noção de todas as implicações de cada alteração. Foi então utilizada a *framework* de testes Mocha [64], que permite simular um conjunto de pedidos à API e validar se o resultado obtido é o esperado (Fig. 28).

```
✓ GET /sensor -> must refuse user:test2@ipt.pt
✓ GET /sensor -> must accept user:admin@a.aa and must receive an array (48ms)
✓ PUT /sensor/:id -> must refuse user:test2@ipt.pt
✓ PUT /sensor/:id -> must accept user:admin@a.aa
✓ DELETE /sensor/:id -> must refuse user:test2@ipt.pt
✓ DELETE /sensor/:id -> must accept user:admin@a.aa
✓ POST /sensor -> Recreate the sensor to future tests
```

Fig. 28 - Testes automatizados com Mocha.

A *framework* Mocha, além de automatizar os testes, permite ainda contabilizar o tempo de execução de cada rota, permitindo ao programador captar as que estão a demorar demasiado tempo a responder, de forma a otimizar o código da respetiva rota. Foi através desta funcionalidade que foi possível perceber o atraso que a rota de receção de valores de sensores estava a causar.

Partindo desse ponto foram realizados então testes de carga em rotas específicas, sendo os testes mais exaustivos precisamente na rota de recepção de valores de sensores. A ferramenta utilizada para os testes de carga foi o Vegeta [65], através de um script na linguagem Go que simula o comportamento de um ataque *brute force* a um URL. Assim foram realizados vários testes às diferentes abordagens pensadas para a API (descritas em 4.4.Escalabilidade) de forma a encontrar a melhor implementação para o sistema. Desses testes destacam-se os realizados durante a fase de arquitetura monolítica e após o desenvolvimento da arquitetura em microsserviços.

Nos testes realizados foi definido como valor a testar o envio de 50 valores de sensores por pedido, em 50 pedidos por segundo durante 80 segundos. No teste realizado à arquitetura monolítica (Fig. 29) foi possível verificar que para os parâmetros definidos o tempo de resposta da API rapidamente aumenta para os 30 segundos, começando depois a deixar de responder, sendo emitidas apenas mensagens de erro, possivelmente um “504 Gateway Timeout” após os 20 segundos de execução.

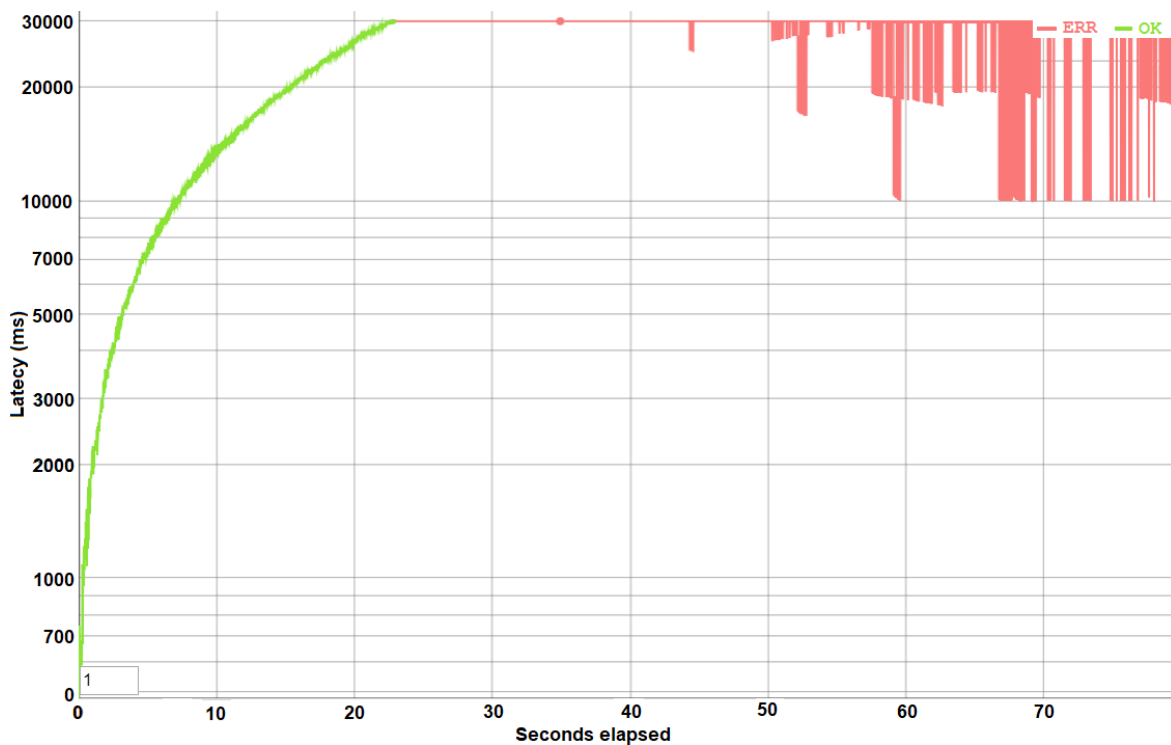


Fig. 29 - Teste de carga à API (envio de dados de sensores) com abordagem monolítica

Após a implementação da arquitetura em microsserviços, com apenas um nó de API e um nó de Worker, utilizando os mesmos parâmetros de teste (Fig. 30), foi possível constatar que a API raramente ultrapassava os 30 milissegundos a responder e o serviço nunca ficava indisponível, o que representava uma melhoria tremenda.

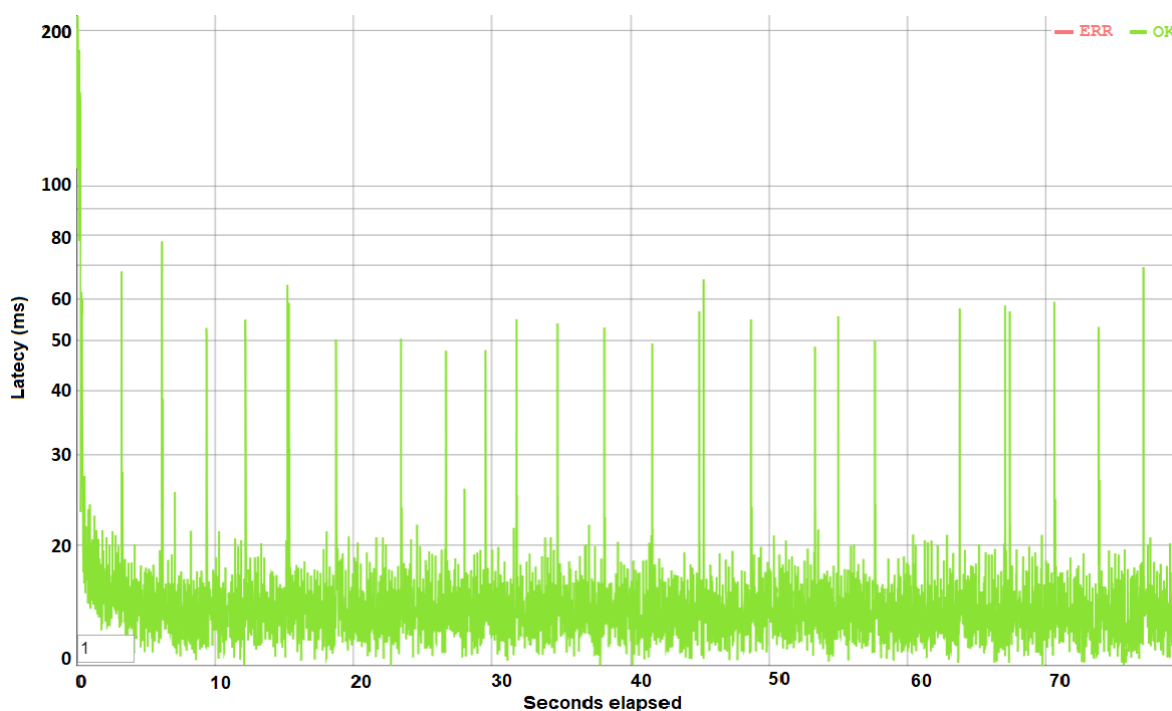


Fig. 30 - Teste de carga em microsserviços

Por fim, foram também realizados alguns testes de usabilidade junto de utilizadores com o perfil do consumidor final da solução do VITASENIOR-MT. O sistema foi instalado no Lar da Santa Casa da Misericórdia de Tomar [66], onde foram seleccionados alguns utentes para a utilização contínua do sistema, com os enfermeiros e o médico do lar associados (Fig. 31). Outros sistemas de teste foram também dispersos por outras localizações, sendo que um foi instalado no laboratório na Escola Superior de Tecnologia de Tomar [67], servindo para testes de usabilidade de apenas uma utilização por utilizadores de diferentes perfis e outro na Escola Superior de Tecnologia da Saúde de Coimbra [68], para testes ao equipamento

utilizado pelo sistema e à interface gráfica apresentada no *frontend* para o utilizador de perfil médico. Durante esta fase foi possível perceber o nível de aceitação da solução por parte de todos os intervenientes, o grau de complexidade que estava associado à utilização das suas interfaces (*frontend* e Vitabox) e a capacidade de resposta da *Cloud* num ambiente real. Com essa informação foi possível a constante melhoria e adaptação do sistema às necessidades dos potenciais interessados.

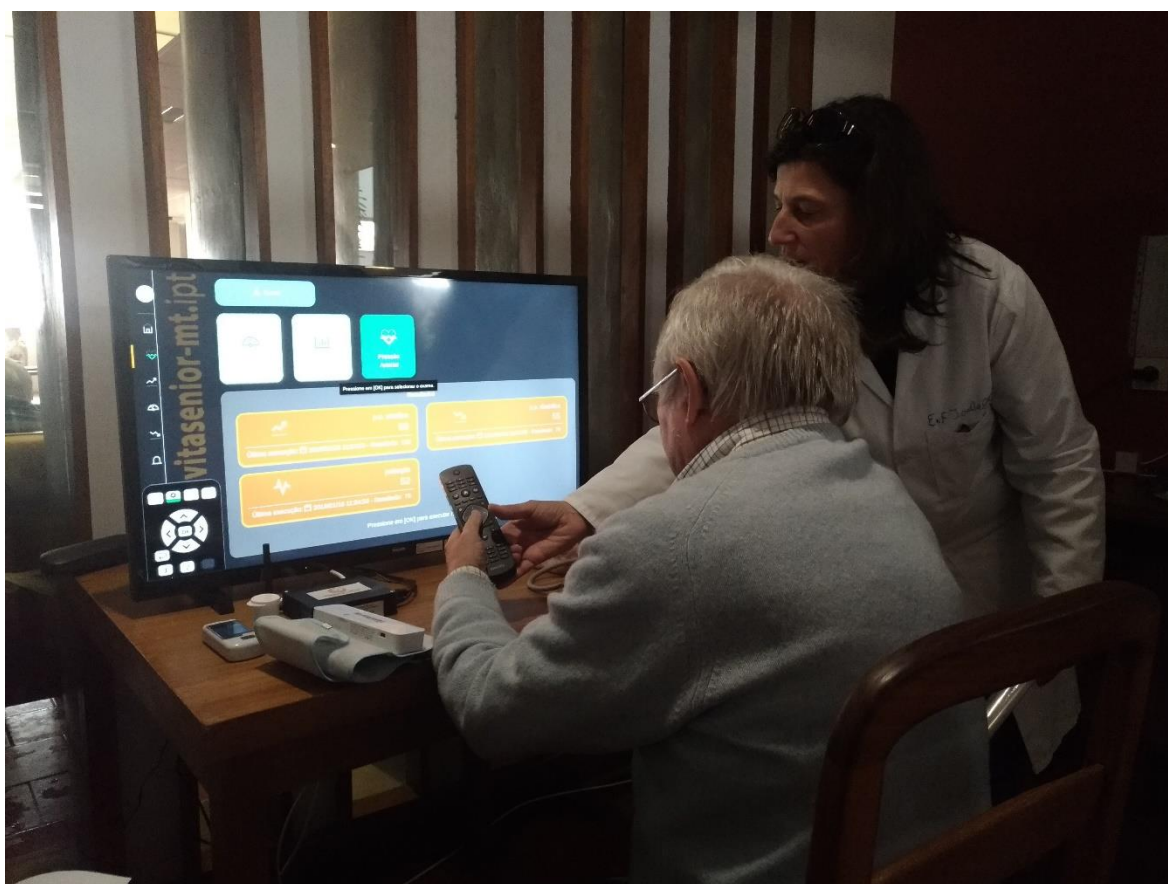


Fig. 31 - Teste de usabilidade por utente com auxílio de enfermeira.

Capítulo 6

Conclusão

De uma forma geral, o objetivo principal do projeto VITASENIOR-MT foi alcançado, que era implementar uma solução de teleassistência que permitisse monitorizar o meio envolvente e os parâmetros biométricos de um (ou mais) idoso(s), com possibilidade de adaptar o sistema às suas necessidades. Em termos de adoção e usabilidade, a solução apresentada provou ser bastante acessível e *user friendly*, com uma compreensão relativamente rápida das suas funcionalidades por parte dos utilizadores finais quando comparado ao que era esperado inicialmente. O portal foi avaliado positivamente pelos utilizadores de teste, uma vez que apesar de permitir efetuar inúmeras tarefas por diferentes tipos de utilizadores, a sua complexidade de utilização foi abstraída com sucesso, apresentando uma interface simples e acessível.

Em relação à infraestrutura de *Cloud*, o objetivo principal era criar um sistema robusto, escalável, onde a segurança dos dados fosse garantida e os tempos de resposta reduzidos, sem penalizar toda a funcionalidade do sistema e a acessibilidade aos diferentes clientes. Estas metas foram atingidas, podendo, como referido anteriormente, ainda ser melhorado.

Como indicador do sucesso deste projeto, foram já publicados dois artigos científicos em conferências de revisão entre pares, sendo o segundo inteiramente focado no trabalho apresentado neste relatório:

1. “VITASENIOR-MT: a telehealth solution for the elderly focused on the interaction with TV” [69]
2. “VITASENIOR-MT: A distributed and scalable cloud-based telehealth solution” [70].

Está ainda em período escrita um artigo para uma revista científica.

6.1. Trabalho futuro

Apesar de a solução desenvolvida apresentar robustez, o sistema poderá ser otimizado e até simplificado a vários níveis no futuro.

Em primeira instância, o mecanismo de validação de *tokens* nos 3 microsserviços de Peer, WebSocket e API deveria ser reestruturado, pois para a atual validação da autenticidade do utilizador é necessária a partilha de uma chave pública e privada entre esses serviços, além da replicação de código, e perda de performance. Para colmatar essa questão deveria ser desenvolvido um serviço apenas para a autenticação, responsável pela validação do cliente e pela emissão de JWT. Após a autenticação bem-sucedida este serviço alocaria num servidor de base de dados chave-valor em memória, como o caso do Redis [71], os dados do cliente com o *token* como chave e impondo um tempo de validade igual à validade do *token*. Desta forma todos os serviços que necessitem de autenticar o utilizador requeriam ao servidor de chaves a informação do utilizador, caso não houvesse retorno, significava simplesmente que o cliente não estava autenticado.

Outra alteração possível seria a alteração da organização do sistema em diferentes servidores de base de dados persistentes, pois a realidade da tecnologia hoje não é a mesma que no início do projeto e hoje seria possível optar apenas por um provedor em vez de dois servidores diferentes. Poderia até ser considerada a possibilidade de utilização de uma base de dados orientada a grafos, que tira partido do melhor dos dois mundos: a segurança transacional e relações entre entidades bem definidas, presente nas bases de dados relacionais, e o tratamento de grandes volumes de dados, típico das bases de dados não relacionais, uma vez que a complexidade de pesquisa é sempre linear.

Bibliografia

- [1] D. T. Rowland, “Global Population Aging: History and Prospects,” in *International Handbook of Population Aging*, Springer, Dordrecht, 2009-2018, pp. 37-65.
- [2] European Commission, “Population structure and ageing,” May 2018. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php/Population_structure_and_ageing. [Accessed 20 Jan 2018].
- [3] Serviço Nacional de Saúde, “História do SNS,” 19 Oct 2018. [Online]. Available: <https://www.sns.gov.pt/sns/servico-nacional-de-saude/historia-do-sns/>. [Accessed 20 Jan 2019].
- [4] L. Borges, “Mais de metade dos idosos sinalizados pela GNR vivem sozinhos,” PÚBLICO Comunicação Social SA, 4 Apr 2017. [Online]. Available: <https://www.publico.pt/2017/04/04/sociedade/noticia/mais-de-metade-dos-idosos-em-portugal-vivem-sozinhos-1767648#gs.R1AlScAI>. [Accessed 20 Jan 2019].
- [5] E. Wilkins, L. Wilson, K. Wickramasinghe, P. Bhatnagar, M. Rayner, N. Townsend, J. Leal, R. Luengo-Fernandez and R. Burns, “European Cardiovascular Disease Statistics,” European Heart Network, Brussels, 2017.
- [6] European Commission, “2018 Ageing Report: Policy challenges for ageing societies,” 25 May 2018. [Online]. Available: https://ec.europa.eu/info/news/economy-finance/policy-implications-ageing-examined-new-report-2018-may-25_en. [Accessed 20 Jan 2019].
- [7] Instituto Politécnico de Tomar, “VITA.IPT - Vida Assistida por Ambientes Inteligentes,” [Online]. Available: http://portal2.ipt.pt/pt/ipt/unidades_de_i_d_tecnologico_e_artistico/vita/vitasenior_mt_assistencia_aos_cuidados_de_saude_de_idosos_no_medio_tejo/. [Accessed 20 Jan 2019].
- [8] IBM, “IBM Cloud is the cloud for smarter business,” [Online]. Available: <https://www.ibm.com/cloud/>. [Accessed 23 Jan 2019].
- [9] A. Botta, W. d. Donato, V. Persico and A. Pescapé, “Integration of Cloud Computing,” *Future Generation Computer Systems*, vol. 56, 2015.

- [10] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin and D. Aharon, “Unlocking the potential of the Internet of Things,” McKinsey & Company, Jun 2015. [Online]. Available: <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world>. [Accessed 28 Jan 2019].
- [11] K. L. Lueth, “State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating,” IoT Analytics, 8 Aug 2018. [Online]. Available: <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>. [Accessed 27 Jan 2019].
- [12] TelefonicaIoT, “Barriers to IoT adoption,” Telefónica S.A., 17 Oct 2018. [Online]. Available: <https://iot.telefonica.com/blog/barriers-to-iot-adoption>. [Accessed 28 Jan 2019].
- [13] J. Zhou, Z. Cao, X. Dong and A. V. Vasilakos, “Security and Privacy for Cloud-Based IoT: Challenges, Countermeasures, and Future Directions,” *IEEE Communications Magazine*, no. 0163-6804/17, pp. 26-32, 2017.
- [14] M. Díaz, C. Martín and B. Rubio, “State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing,” *Journal of Network and Computer Applications*, vol. 67, no. 1084-8045, pp. 99-117, 2016.
- [15] A. Mostafa, “MQTT Protocol – How it Works,” 1Sheeld, 4 Jul 2018. [Online]. Available: <https://1sheeld.com/mqtt-protocol/>. [Accessed 29 Jan 2019].
- [16] knolleary, “MQTT used by Facebook Messenger,” MQTT.org, 12 Aug 2011. [Online]. Available: <https://mqtt.org/2011/08/mqtt-used-by-facebook-messenger>. [Accessed 29 Jan 2019].
- [17] R. K. Naha, D. Georgakopoulos, S. Garg, P. P. Jayaraman, R. Ranjan, L. Gao and Y. Xiang, “Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions,” *IEEE access*, vol. 6, no. 10.1109/ACCESS.2018.2866491, pp. 47980-48009, 2018.
- [18] M. Quwaider and Y. Jararweh, “Cloudlet-based Efficient Data Collection in Wireless Body Area Networks,” *Simulation Modelling Practice and Theory*, vol. 50, pp. 57-71, 2015.

- [19] Cisco, “Fog Computing and the Internet of Things: Extend,” 2015. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf. [Accessed 29 Jan 2019].
- [20] A.-M. Rahmani, N. K. Thanigaivelan, T. N. Gia, J. Granados, B. Negash, P. Liljeberg and H. Tenhunen, “Smart e-Health Gateway: Bringing Intelligence to,” *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, no. 10.1109/CCNC.2015.7158084, 2015.
- [21] Y. Saleem, F. S. Shaikh and M. H. Rehmani, “Resource Management in Mobile Sink Based Wireless Sensor Networks through Cloud Computing,” *Springer-Verlag Handbook*, vol. 3, no. 10.1007/978-3-319-06704-9-20, pp. 439-459, 2014.
- [22] OpenIoT Consortium, “OpenIoT - The Open Source Internet of Things,” 23 Nov 2015. [Online]. Available: <https://github.com/OpenIoTOrg/openiot>. [Accessed 7 Feb 2019].
- [23] MDSLAb, “Stack4things,” 25 Jan 2018. [Online]. Available: <https://github.com/MDSLAb/stack4things>. [Accessed 7 Feb 2019].
- [24] CloudPlugs Inc., “Edge to Cloud Intelligence and Automation for Industrial IoT,” 2019. [Online]. Available: <https://cloudplugs.com/>. [Accessed 8 Feb 2019].
- [25] Particle, “MEET THE ONLY ALL-IN-ONE IOT PLATFORM ON THE MARKET,” 2019. [Online]. Available: <https://www.particle.io/what-is-particle>. [Accessed 8 Feb 2019].
- [26] Particle, “Particle Firmware for the Electron, P1, Photon and Core.,” 4 Feb 2019. [Online]. Available: <https://github.com/particle-iot/device-os>. [Accessed 8 Feb 2019].
- [27] IBM, “Internet of Things – IoT,” [Online]. Available: <https://www.ibm.com/cloud/internet-of-things>. [Accessed 10 Feb 2019].
- [28] JS Foundation, “Node-RED : Flow-based programming for the Internet of Things,” [Online]. Available: <https://nodered.org/about/>. [Accessed 10 Feb 2019].
- [29] Google, “Google Cloud IoT,” [Online]. Available: <https://cloud.google.com/solutions/iot/>. [Accessed 9 Feb 2019].
- [30] Microsoft, “Azure IoT Hub,” 2019. [Online]. Available: <https://azure.microsoft.com/en-us/services/iot-hub/>. [Accessed 8 Feb 2019].

- [31] Amazon Web Services, Inc, “AWS IoT,” 2019. [Online]. Available: <https://aws.amazon.com/iot/>. [Accessed 9 Feb 2019].
- [32] ThingSpeak, “Understand Your Things: The open IoT platform with MATLAB analytics,” [Online]. Available: <https://thingspeak.com/>. [Accessed 10 Feb 2019].
- [33] S. S. Kazi, G. Bajantri and T. Thite, “Remote Heart Rate Monitoring System Using IoT,” *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, no. 2395-0056, pp. 2956-2963, 2018.
- [34] ThingSpeak, “Public Channels,” ThingSpeak, [Online]. Available: <https://thingspeak.com/channels/public>. [Accessed 10 Feb 2019].
- [35] S. Babu, J. G. Udayasankaran, B. Krishnan, A. S. R. Tamanampudi, S. P. Shaji, A. Vishwanatham, P. Raja and S. S. S. Sanagapati, “Smart telemetry kit for proactive health monitoring in rural India: The journey so far and the road ahead,” *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, no. 10.1109/HealthCom.2018.8531188, 2018.
- [36] I. Villanueva-Miranda, H. Nazeran and R. Martinek, “CardiaQloud: A Remote ECG Monitoring System Using Cloud Services for eHealth and mHealth Applications,” *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, no. 10.1109/HealthCom.2018.8531164, 2018.
- [37] Libelium Comunicaciones Distribuidas S.L., “MySignals changes the future of medical and eHealth applications,” [Online]. Available: <http://www.my-signals.com/>. [Accessed Jul 2019].
- [38] Apple Inc., “A bold way to look at your health.,” [Online]. Available: <https://www.apple.com/ios/health/>. [Accessed Jul 2018].
- [39] E. AbuKhoua, N. Mohamed and J. Al-Jaroodi, “e-Health Cloud: Opportunities and Challenges,” *Future Internet*, no. 10.3390/fi4030621, 2012.
- [40] Node.js Foundation, “About Node.js®,” [Online]. Available: <https://nodejs.org/en/about/>. [Accessed 6 Jul 2019].
- [41] T. Laurens, “How the V8 engine works?,” 29 Apr 2013. [Online]. Available: <http://thibaultlaurens.github.io/javascript/2013/04/29/how-the-v8-engine-works/>. [Accessed 6 Jul 2019].

- [42] A. Zlatkov, “How JavaScript works: inside the V8 engine + 5 tips on how to write optimized code,” 21 Aug 2017. [Online]. Available: <https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tips-on-how-to-write-optimized-code-ac089e62b12e>. [Accessed 6 Jul 2019].
- [43] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland and M. Stal, “Layers,” in *PatternOriented Software Architecture*, John Wiley & Sons, 1996, pp. 31-51.
- [44] HashiCorp, “Vagrant - Development Environment Made Easy,” [Online]. Available: <https://www.vagrantup.com/>. [Accessed 7 Jul 2019].
- [45] K. Lee, Y. Kim and C. Yoo, “The Impact of Container Virtualization on Network Performance of IoT Devices,” *Mobile Information Systems*, vol. 2018, no. 9570506, p. 6, 2018.
- [46] M. Raza, “Containers vs Virtual Machines: What’s The Difference?,” 7 Aug 2018. [Online]. Available: <https://www.bmc.com/blogs/containers-vs-virtual-machines/>. [Accessed 7 Jul 2019].
- [47] Docker Inc., “Enterprise Container Platform for High-Velocity Innovation,” [Online]. Available: <https://www.docker.com/>. [Accessed 7 Jul 2019].
- [48] Pivotal, “RabbitMQ is the most widely deployed open source message broker.,” [Online]. Available: <https://www.rabbitmq.com/>. [Accessed 7 Jul 2019].
- [49] MongoDB, Inc., “The database for modern applications,” [Online]. Available: <https://www.mongodb.com/>. [Accessed 8 Jul 2019].
- [50] E. Horowitz, “MongoDB Drops ACID,” 15 Feb 2018. [Online]. Available: <https://www.mongodb.com/blog/post/multi-document-transactions-in-mongodb>. [Accessed 8 Jul 2019].
- [51] Oracle Corporation, “MySQL - The world's most popular open source database,” [Online]. Available: <https://www.mysql.com/>. [Accessed 8 Jul 2019].
- [52] Oracle Corporation, “MySQL Document Store,” [Online]. Available: https://www.mysql.com/products/enterprise/document_store.html. [Accessed 8 Jul 2019].
- [53] União Europeia, “General Data Protection Regulation,” [Online]. Available: <https://gdpr-info.eu/>.

- [54] Linux Foundation, “Let’s Encrypt is a free, automated, and open Certificate Authority.” [Online]. Available: <https://letsencrypt.org/>. [Accessed 9 Jul 2019].
- [55] NGINX Inc., “NGINX - Improve the performance, reliability, and security of your applications,” [Online]. Available: <https://www.nginx.com/>. [Accessed 9 Jul 2019].
- [56] S. Yu, C. Wang, K. Ren and W. Lou, “Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing,” *2010 Proceedings IEEE INFOCOM*, Mar 2010.
- [57] E. You, “The Progressive JavaScript Framework,” [Online]. Available: <https://vuejs.org/>. [Accessed 9 Jul 2019].
- [58] inveris OHG, “APIDOC - Inline Documentation for RESTful web APIs,” [Online]. Available: <http://apidocjs.com/>. [Accessed 10 Jul 2019].
- [59] International Organization for Standardization, “Country Codes - ISO 3166,” [Online]. Available: <https://www.iso.org/iso-3166-country-codes.html>. [Accessed 10 Jul 2019].
- [60] MDN web docs, “WebRTC API,” Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API. [Accessed 9 Jul 2019].
- [61] Postman, Inc., “How Postman Improves API Development,” [Online]. Available: <https://www.getpostman.com/>. [Accessed 10 Jul 2019].
- [62] mochajs.org, “MOCHA - simple, flexible, fun,” [Online]. Available: <https://mochajs.org/>. [Accessed 10 Jul 2019].
- [63] T. Senart, “Vegeta,” [Online]. Available: <https://github.com/tsenart/vegeta>. [Accessed 10 Jul 2019].
- [64] Santa Casa da Misericórdia de Tomar, “Lar,” [Online]. Available: <http://www.scmt.pt/areas-de-intervencao/lar/>. [Accessed 10 Jul 2019].
- [65] Instituto Politécnico de Tomar, “ESTT - Escola Superior de Tecnologia de Tomar,” [Online]. Available: http://portal2.ipt.pt/pt/ipt/estrutura_organica/instituto_politecnico_de_tomar/unidades_organicas/estt_escola_superior_de_tecnologia_de_tomar/. [Accessed 10 Jul 2019].

- [66] Escola Superior de Tecnologia da Saúde de Coimbra , [Online]. Available: <http://www.estescoimbra.pt/>. [Accessed 10 Jul 2019].
- [67] G. Pires, P. Correia, D. Jorge, D. Mendes, N. Gomes, P. Dias, P. Ferreira, A. Lopes, A. Manso, L. Almeida, L. Oliveira, R. Panda, P. Monteiro, C. Grácio and T. Pereira, “VITASENIOR-MT: a telehealth solution for the elderly focused on the interaction with TV,” *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, 2018.
- [68] D. Mendes, D. Jorge, R. Panda, R. António, P. Dias, L. Oliveira and G. Pires, “VITASENIOR-MT: A distributed and scalable cloud-based telehealth solution,” *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019.
- [69] redislab, “Redis,” [Online]. Available: <https://redis.io/>. [Accessed 12 Jul 2019].