**Universidade de Aveiro** Departamento de Eletrónica

**2017** Telecomunicações e Informática

**Udo Filipe Branco Alves**

# Sistema de Informação Mãos Livres para Manipulação de Estudos de Imagem Médica

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Carlos Manuel Azevedo Costa, professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

À minha namorada pelo suporte incondicional, à minha mãe, irmã e família por serem o meu apoio e aos meus amigos pelas palavras de motivação.

**o júri**

presidente

Prof. Doutor Augusto Marques Ferreira Silva
Professor auxiliar da Universidade de Aveiro

vogais

Prof. Doutor José Paulo Ferreira Lousado
Professor adjunto, Dep. de Informática, Comunicações e Ciências Fundamentias, da Escola Superior de Tecnologia e Gestão de Lamego do Instituto Politécnico de Viseu

Prof. Doutor Carlos Manuel Azevedo Costa
Professor auxiliar do Departamento de Eletrónica, Telecomunições e Informática da Universidade de Aveiro (orientador)

**palavras-chave** Mãos livres, cirurgia, PACS, RIS, DICOM, MVC.

**resumo** A utilização das tecnologias de informação e comunicação na área da saúde está sendo explorada nos mais diversos cenários de diagnóstico e tratamento de pacientes. O acesso aos meios complementares de diagnóstico e tratamento no bloco operatório é um requisito fundamental. No entanto, as restrições de manuseamento dos dispositivos de IO (rato, teclado, touch-screen), por parte do cirurgião, limita grandemente a sua utilização.

Esta dissertação teve como objetivo desenvolver duas componentes principais de uma aplicação "HandsFree Operating Theatre" para imagem médica, nomeadamente a camada da lógica de negócios e de persistência de dados. Entre outros, a solução contempla a criação de um arquivo de imagem médica em formato DICOM e um conjunto de serviços que permitam suportar os requisitos da camada de representação gráfica. Os serviços estão expostos através de Webservices.

.

**keywords** HandsFree, operation theatre, PACS, RIS, DICOM, MVC.

**abstract** Nowadays healthcare institutions are integrating more and more information and communication technologies into diagnosis and treatment of illness. This medical complementary diagnosis and treatment records are valuable during surgeries but the need of IO devices to use and manipulate them imposes a major limitation for surgeons during the procedure.
This thesis had his focus on developing two core components for a larger application "HandsFree Operating Theatre". Among others, the logic layer and data persistence components aims the creation of an archive for medical images using DICOM format and developing services, later exposed as WebServices, to fulfill the requirements of the graphic display layer.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Acrónimos

AJAX - Asynchronous Javascript and XML
AOP - Aspect Oriented Programming
API - Application Programming Interface
DAO - Data Access Object
CDTT - Complementary Diagnostic Tests and Therapy
CRUD - Create, Read, Update, Delete
DICOM - Digital Imaging and Communication in Medicine
EE - Enterprise Edition
EJB - Enterprise Java Beans
Gb - Gigabyte
HTTP - Hypertext Transfer Protocol
IoC - Inversion of Control
JB - Java Beans
JPA - Java Persistence API
JPG - Joint Photographic Experts Group
JDBC - Java Database Connectivity
JMS - Java Message Service
JSP - Java Server Pages
JVM - Java Virtual Machine
Mb - Megabyte
MCDT - Método Complementar de Tratamento e Diagnóstico
MRI - Magnetic Resonance Imaging
MVC - Model View Controller
NoSQL - Not Only SQL
OOP - Object Oriented Programming
ORM - Object Relational Mapping
POJO - Plain Old Java Object
POM - Project Object Model
RAM - Random Access Memory
REST - Representational State Transfer
SPOF - Single Point of Failure
SQL - Structured Query Language
URL - Uniform Resource Location
TDD - Test Driven Development
WSDL - Web Services Description Language
XML - eXtensible Markup Language

# 1. Introdução

## 1.1 Enquadramento

Vivemos num mundo cada vez mais rodeado de tecnologia, termo usado como uma forma genérica de designar tudo o que dispõe de alguma capacidade eletrônica. Na verdade, se pesquisarmos por "o que é tecnologia?", muitas das vezes encontramos apenas publicações ou artigos que retratam sistemas eletrônicos. É por esta relação estreita entre eletrônica e desenvolvimento de tecnologias que é frequente associar tecnologia a tudo o que tenha, de alguma forma, algum sistema eletrônico, com ou sem um sistema computorizado. É importante clarificar que, por definição, tecnologia é a aplicação de conhecimento científico em fins práticos [1]. Desta forma, quando temos em consideração este real significado da palavra é possível expandir a nossa perceção do que é a tecnologia e considerar esta, como tudo o que é uma solução para um problema que existe. A eletrónica, por seu lado, como sistemas computorizados que surgem como um meio para atingir este fim, com todas as vantagens associadas a estes. Este nível da abstração face a este termo consegue assim levar a que seja possível expandir cada vez mais a perceção humana do que é ou não exequível e assim surgem novas áreas de investigação e desenvolvimento, bastando apenas ser-se capaz de pensar "fora da caixa" e olhar para uma invenção tecnológica e fazer a pergunta "o que aconteceria se esta invenção fosse utilizada num contexto totalmente diferente do que foi desenhada para?"

Com este ponto de partida, olhando para algumas das limitações que existem atualmente nos blocos operatórios e equipamentos mãos livres no mercado que foram desenvolvidos para lazer, surgiu o tema de como utilizar estes equipamentos como forma de mitigar algumas dessas limitações.

## 1.2 Contexto

Os inúmeros avanços que se fizeram sentir nas últimas décadas ao nível da eletrónica e sistemas computorizados, tiveram um grande impacto em muitas vertentes do quotidiano alterando a própria conceção da realidade dessas vertentes, sendo uma delas e no contexto desta tese a

Medicina. Da união da medicina a estes avanços permitiu o aparecimento de novas formas de diagnóstico e tratamento de fatalidades, que até então eram de difícil tratamento ou inclusive mortais. A título de exemplo temos a Ressonância Eletromagnética, em inglês MRI (*"Magnetic Resonance Imaging"*), que é visto como uma das principais formas de diagnóstico médico imagiológico, uma vez que permite obter inúmeros detalhes e pormenores face a outras modalidades mais clássicas como, por exemplo, o Raio X. Embora a Ressonância Eletromagnética tenha sido considerada como "*one of the great medical breakthroughs of the 20th century*" [2], a sua invenção data dos anos 70. Os avanços tecnológicos que têm vindo a acontecer aliados à necessidades da medicina permitiram o aparecimento de novos métodos de diagnóstico complementar (MCDT), em inglês "*CDTT"* (*"complementary diagnostic tests and therapy"*), que contribuem para que muitas das doenças até então de difícil diagnóstico pudessem passar a ser detetadas precocemente, com resultados diretos através na diminuição da taxa de mortalidade associada a algumas doenças, o desenvolvimento de técnicas menos invasivas de tratamentos e, por último, o melhoramento dos procedimentos durante as cirurgias. Com vista a dar um contributo nesta área, a Kcs-It, empresa de consultoria que oferece não só serviços na área de consultoria, mas também disponibiliza serviços tecnológicos bem como formação, lançou como desafio do seu *"innovation lab"* pegar em equipamentos que existem no mercado que permitem a interação sem toque e ser capaz de ousar usá-los para auxiliar em ambiente cirúrgico. Surge então o *"HandsFree Operation Theatre".*

## 1.3 Motivação

O ambiente cirúrgico pode-se dividir em três áreas distintas: a área de proteção, a área limpa e a área esterilizada. A primeira, a área de proteção, corresponde ao balneário onde os profissionais de saúde se vestem adequadamente para realizar a cirurgia. A segunda, a área limpa, é todo o espaço compreendido entre a área de proteção e a área esterilizada e onde se encontram os materiais necessário à cirurgia. Por último a área esterilizada corresponde à zona onde decorre a cirurgia em si, sendo assim o espaço que rodeia o paciente. Nos materiais que são necessários à cirurgia enquadra-se, atualmente, a necessidade de um computador que o cirurgião responsável pela intervenção necessita para verificar informações contidas nos *MCDT* durante o procedimento cirúrgico. Uma vez que este não pode ser submetido às mesmas formas de esterilização que os restantes equipamentos médicos, na generalidade dos blocos operatórios surge uma limitação para o profissional face à questão de como interagir com o computador enquanto consegue garantir a assepsia da área esterilizada.

Para complicar ainda mais a situação, atualmente existe uma limitação ao nível da área dos movimentos do cirurgião, por forma a garantir que as mãos do mesmo se mantêm dentro de perímetro considerado seguro para a cirurgia. Assim, caso o cirurgião pretenda interagir com os MCDT, nomeadamente com as imagens médicas em formato DICOM ("*Digital Imaging and Communication in Medicine*")[3], tem de recorrer normalmente a um assistente que manipula as imagens ao seu comando. Esta dependência de terceiros torna-se frustrante para os profissionais uma vez que a manipulação indireta induz problemas como a má interpretação do pretendido, além de toda a insatisfação que a profissional experiência de não poder interagir livremente com as imagens. Em alternativa a esta dependência, o médico pode descalçar as luvas, mas terá de se submeter de novo a todo o processo de esterilização antes de poder calçar novas luvas e prosseguir a cirurgia [4]. Para resolver esta problemática, surgiu a proposta da Kcs-It para o desenvolvimento de um software inovador, que introduzisse o conceito de mãos-livres à manipulação das imagens médicas e pudesse, assim, ser resposta a este problema.

## 1.4 Objetivos

O objetivo principal desta tese é desenhar e implementar uma solução de *"backend"* que suporte as funcionalidades principais necessárias à possibilidade de sobre ele se desenvolver uma interface "web" para a interação com as imagens médicas, utilizando para tal tecnologias e ferramentas *"state of the art"*, recorrendo a *"frameworks"* e a padrões de desenho atualizados que permitam a integração de novos componentes e/ou funcionalidades. Os requisitos básicos que o *"backend"* deverá dar resposta são:

- Ser modular de forma a que possam ser acrescentadas novas funcionalidades ou integrar com as existentes via conectores;
- Permitir a inserção de pacientes no sistema e modificar dados dos mesmos;
- Permitir a manipulação de DICOM;
- Suportar a conversão entre o formato DICOM para JPEG;

## 1.5 Estrutura do Documento

Este documento é constituído por 7 capítulos

- Introdução -Contexto e motivação do projeto.
- State of the art - Uma análise das tecnologias existentes
- Levantamento de requisitos – Levantamento de requisitos da aplicação

- Proposta de Arquitetura – Apresentação da arquitetura e do planeamento para a realização do trabalho
- Implementação, Resultados e Validação – Serviços implementados, estrutura da aplicação, validação dos serviços implementados e testes de carga à base de dados
- Continuidade do projeto
- Conclusão

# 2. Estado da arte

Este capítulo serve para fazer o levantamento e caracterização de todas as tecnologias que existem atualmente no mercado e que poderiam ser parte integrante da solução, com a respetiva analisando as vantagens e desvantagens associadas, por forma a chegar a uma decisão final sobre o que será mais vantajoso à realização do trabalho.

.

## 2.1 Imagiologia na Saúde

Muitos progressos têm sido feitos a nível das imagens médicas que são utilizadas para diagnóstico e tratamento de problemas de saúde. Num passado não muito distante era normal os médicos usarem dispositivos analógicos com filme que era analisando à contraluz como forma de avaliação do estado clínico da pessoa. Com o avanço dos sistemas informáticos, foi possível desenvolver ferramentas digitais que permitiam um melhor armazenamento e visualização, como foi o caso da introdução dos CD/DVD. Não obstante, continuava em falta a resolução da questão do acesso e distribuição dos estudos imagiológicos, que continuava dependente de meios físicos. Os progressos tecnológicos continuaram a evoluir e apareceram novas realidades como a centralização da informação, partilha de conteúdos entre máquinas e sistemas. Atendendo a estes progressos e ao facto de que os exames médicos já se encontrarem num formato digital muito antes de serem impressos ou armazenados em um DVD, encorajou o próximo passo evolutivo de simplesmente disponibilizar diretamente esses registos digitais, sem necessitar de passos intermédios como armazenar num DVD.

Para que esta realidade seja possível, surgiram novos sistemas associadas à imagiologia médica, denominando-se PACS (*"Picture Archiving and Communication System"*), RIS (*"Radiology Information System"*) e CIS (*"Clinical Information System"*).

### 2.1.1 PACS

Dependendo da complexidade do exame, bem como do equipamento utilizado na aquisição, é possível que no decorrer do mesmo sejam gerados imensos recursos, quer imagens quer vídeos, que necessitam ser armazenados. Uma forma inicial de armazenar estes registos seria utilizar uma solução baseada em registos, ou seja, para cada exame existir um registo, seja ele digital ou em papel, que guardasse a informação sobre como localizar os recursos dos exames. Embora seja uma solução perfeitamente válida, tal revela-se ineficiente, uma vez que introduz a possibilidade de erros, associado por exemplo à má catalogação dos registos, à possibilidade de se perderem registos, além de aumentar o tempo que demora a obter o registo pois o processo de localização das copias pode ser moroso. Para dar resposta surgiram os PACS[5]. Estes sistemas visam agilizar toda a gestão dos recursos durante o ciclo de vida do exame. As imagens e/ou vídeos produzidos são armazenados utilizando um arquivo PACS e, posteriormente, o profissional de saúde, que tenha necessidade de os aceder, pode fazer uma pesquisa no sistema e visualizar os recursos pretendidos.

É graças aos PACS que é possível, hoje em dia, solicitar um exame durante uma consulta, já que, ao armazenar os resultados no PACS, estes ficam disponíveis para visualização quase de imediato. Graças aos PACS é também possível que os exames produzidos para uma especialidade médica possam ser rapidamente partilhados com outras especialidades, para um maior benefício do paciente.

### 2.1.2 RIS

A gestão de uma instituição médica pode ser analisada a partir de um ponto de vista empresarial e tem, portanto, necessidades de gerir processos internos que permitem o funcionamento estrutural da mesma. Duas terminologias bastante interligadas com o ciclo de vida de uma empresa são o BPM *(* de *"Business Process Management")* e o *"Workflow"* (fluxo de trabalho). Estes sistemas gerem não só o *"workflow"*, toda a lógica de negócio (ou *"business"*) mas disponibilizam também conteúdo, como é o caso dos relatórios clínicos. A gestão do *"workflow"* está inerente ao ciclo de vida de um paciente, como por exemplo o agendamento de consultas e exames e manter um registo do historial clínico, desde a sua primeira admissão, até às consultas agendadas. Em termos de BPM evidencia-se na medida em que permite manter, por exemplo, registos das prescrições de exames que foram passadas pelos médicos ou as faturações das consultas e material médico gasto. Estes registos podem ser posteriormente utilizados para gerar relatórios clínicos[6].

A nível dos registos e armazenamento de recursos de exames médicos, os RIS suportam essa funcionalidade, mas, como os PACS têm sido cada vez mais aceites na comunidade médica e por

todas as vantagens apresentadas anteriormente, tem existido a tendência de conjugar os PACS com RIS num único sistema.

### 2.1.3 DICOM

Globalmente, a transição dos registos do formato papel (ou físico) para o formato digital é um fenómeno que afeta todas as indústrias pelo que, inevitavelmente, a área da saúde também sofre pressões para acompanhar esta tendência. Uma forma de fazer esta transição em sistemas que exigem uma grande interoperabilidade e intercomunicação, mas cujos intervenientes provêm de realidades totalmente diferentes, não só em termos de nacionalidade e linguísticos, mas também culturais é através da definição de *"standards"*. Surge assim o propósito do DICOM. Como *"standard"* procura garantir uma melhor colaboração entre profissionais de saúde, visto que a comunicação entre estes fica mais fácil quando se segue um conjunto de regras bem definidas, mas também é responsável pela integração entre os diversos sistemas, desde os próprios equipamentos aos RIS/PACS. O DICOM permite assim definir diretrizes para os estudos imagiológicos, nomeadamente a aquisição, armazenamento distribuição impressão e validação das imagens médicas e para a forma como as imagens médicas com as respetivas anotações são transmitidas. Perspetiva-se que no futuro se possa continuar a estender o raio de ação do DICOM por forma a abranger todas as especialidades além da radiologia que necessitam de imagens médicas para complementar os diagnósticos e tratamentos. A título de exemplo, algumas das especialidades que já começam a beneficiar dos DICOM são a cardiologia, oftalmologia e gastroenterologia.

Um registo DICOM é constituído por três partes; um cabeçalho, ou *"header"*, seguido de um "*data set"* e dentro deste estão 1 até N *"data elements"*. A Figura 1 ilustra a estrutura detalhada de um registo DICOM.

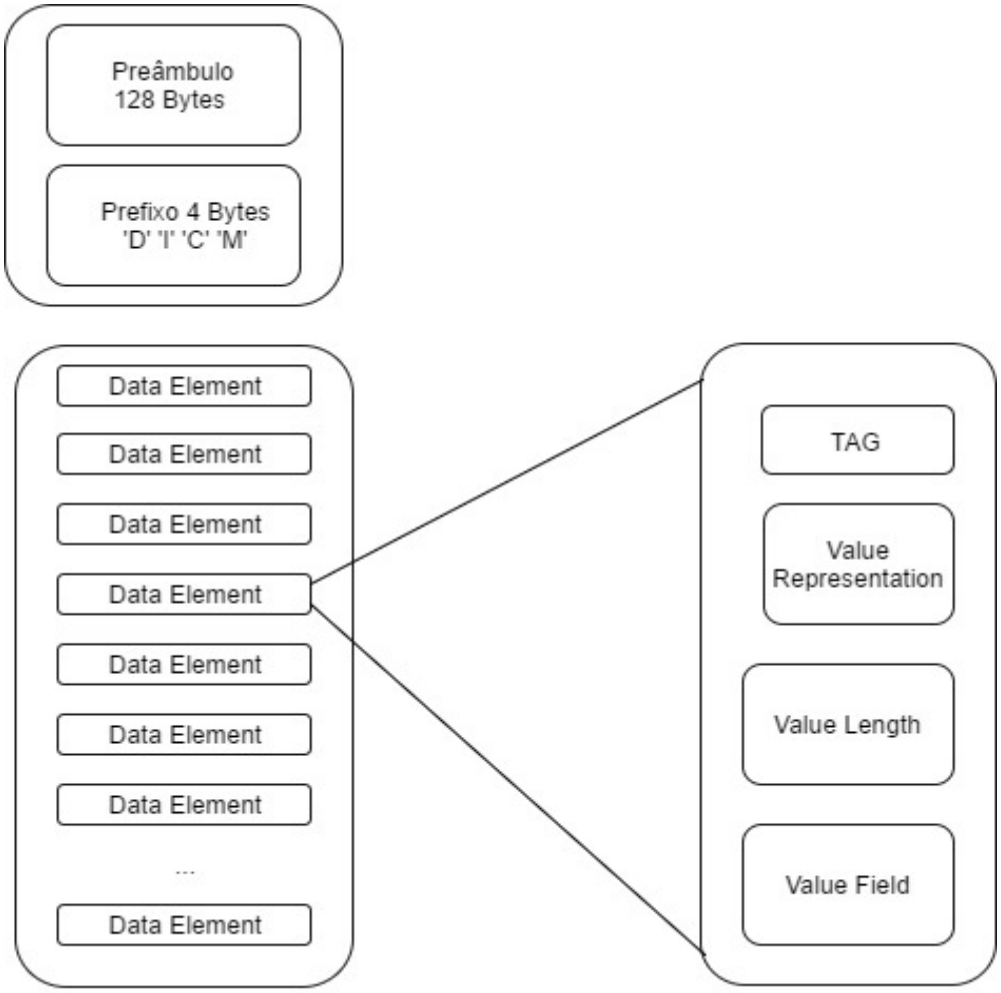


*Figura 1 Estrutura DICOM[7]*

O *"header"* é constituído por um preâmbulo de 128 *bytes* seguido de um prefixo de 4 *bytes*. Não havendo uma regra estrutural estabelecida para o preâmbulo, apenas se estabelece que quando este não esteja a ser utilizada, deverá ter todos os *bytes* escritos com o valor "00H" sinalizando que o mesmo não está a ser utilizado.

O *"data set"* é uma representação de um objeto de informação médica armazenado nos *"data elements".* Se o último elemento armazenado tiver o valor (FFFC, FFFC) então está a ser utilizado *"padding"*.

Os *"data elements"* são identificados pela sua TAG e, portanto, esta tem de ser única. A TAG é constituída por quatro "*bytes"*, sendo que dois são para identificar o ID do grupo e dois para o ID do elemento segundo a estrutura (gggg, eeee) além de também definir uma ordenação crescente destes elementos. Existe também um campo opcional de dois bytes que permite identificar o tipo de dados que estão contidos no elemento designado por *"value representation".* Outro campo presente é o tamanho do elemento e finalmente os dados encontram-se armazenado no campo *"value"*.

### 2.1.4 CIS

Um sistema desenhado com o propósito de recolher, armazenar, manipular e disponibilizar informação clínica é chamado de sistema de informação clínica ou CIS em inglês. Estes sistemas podem ter um âmbito genérico e abranger um vasto leque de áreas clínicas ou podem estar confinadas a apenas áreas especificas como radiologia ou cardiologia. É da competência do CIS a disponibilização de um repositório onde se armazene informação relativa à informação clínica do paciente, desde os episódios médicos até aos procedimentos que foram administrados em cada um deles. Tudo com o propósito de poder garantir que o profissional de saúde dispõe de toda a informação necessária para poder oferecer o melhor cuidado de saúde possível. A Figura 2 pretende demonstrar a estrutura básica de um sistema CIS[8].

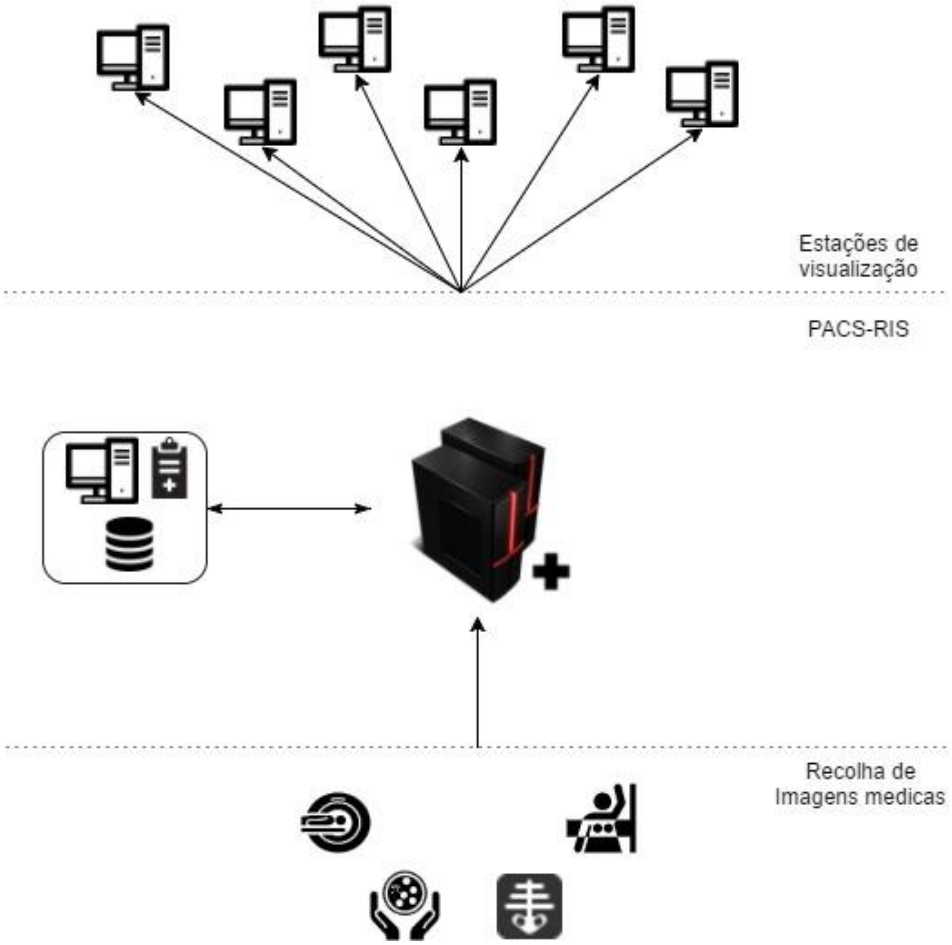


*Figura 2 Esquema de um sistema CIS*

### 2.1.5 Resumo

Em resumo, a ausência de PACS acarreta em prejuízo direto o aumento do tempo necessário à obtenção dos resultados dos exames, aumento do tempo necessário à organização dos registos e a ocorrência de erros, como má indexação dos registos. DICOM é importante quer a nível humano, pois permite quebrar barreiras culturais, quer a nível tecnológico, pois sem este não haveria garantias de compatibilidades entre os diferentes módulos de exames, ou que a compra de novos equipamentos não provoque conflitos nem com o sistema nem com as imagens médicas do passado. Os CIS estão a demonstrar-se como uma mais valia no tratamento de vidas humanas e por isso são sistemas que vieram para ficar. O facto de oferecerem uma forma estruturada de armazenar a informação, melhorar o acesso ao historial clínico dos utentes e permitirem um melhor controlo na prescrição das receitas médicas, são vantagens práticas e que realmente beneficiam os cuidados médicos prestados. No

entanto ainda têm alguns pontos de melhoria. O investimento inicial poder ser elevado, a integração com os sistemas em uso e pode não ser fácil a segurança e proteção de dados dos pacientes.

## 2.2 Equipamentos mãos livres

Um dos principais objetivos do estudo do estado da arte foi identificar o equipamento a utilizar para a manipulação mãos livres da aplicação, uma vez que só tendo esta decisão tomada seria possível escolher a linguagem de programação, na medida em que o equipamento iria ditar quais as opções com base nas *APIs*, do inglês ("*application programing interface"*) existentes. Os equipamentos em análise foram a pulseira Myo, o Kinect v2 e o Leap Motion.

O Kinect V2 surgiu como um periférico para a consola de jogos Xbox One, com o objetivo de controlar a consola por gestos, fazendo com que o utilizador fosse o comando. Foi desenvolvido pela Microsoft ™.

O *Software Development Kit* (SDK) oficial do Kinect V2 inclui suporte para a C#, C++ com XAML, C++ com DirectX e JavaScript com HTML/CSS. Existe ainda, uma biblioteca Java desenvolvida pela Universidade da Flórida, o J4K, para fazer a tradução de código Java para o SDK oficial.

Dispõe de um sistema de "*tracking"* corporal baseado na incorporação de uma câmara de profundidade para visualização 3D e melhoramentos de *software*. Desde que foi lançada no mercado até ao presente já permite identificar 6 utilizadores e aumentou de 20 para 25 o número de marcações por pessoa, o que tornou o sistema mais preciso. Este equipamento tem uma câmara de alta definição para garantir um melhor *input* e um melhoramento do sistema de infravermelhos, de forma a poderem gerar vistas totalmente independentes de fontes de luz, para uma melhor visão em ambiente escuro.

A Myo *"Gesture Control Armband"* é uma bracelete que se insere na categoria dos "*wearable gear".* Neste caso, trata-se de um equipamento de controlo de gestos que permite controlar diversos equipamentos sem necessidade de contacto direto com os mesmos. Foi desenvolvida por Thalmic Labs ™.

A Myo Gesture Control Armband é uma bracelete de 19 a 34 cm para ser utilizada no antebraço. Tem um peso de 93 gramas e existe nas cores: branco ou preto. É compatível com Windows 7/8/10, OS X 10.8, IOS 7.0+ e Android 4.4+ desde que os equipamentos disponham de Bluetooth 4.0 LE (*BLE*). É composta por secções independentes chamadas de *"pods"* equipadas com sensores de eletromiografia (*EMG*), giroscópio, acelerômetro e magnetómetro, dispondo de bateria incorporada.

Os SDK oficiais disponibilizados pelo fabricante incluem suporte para Windows C++, IOS, Android e Unity. Existe um fórum de discussão oficial no qual se apresentam outras propostas de linguagens, como ferramentas para C#, Java e JavaScript.

O Leap Motion Inc é uma empresa cuja missão é remover as barreiras entre pessoas e tecnologias. Para tal, eles desenvolveram um produto onde as mãos das pessoas podem ser uma porta de comunicação entre os utilizadores e a tecnologias. Atualmente, este produto é usado para ambientes de realidade virtual ou realidade aumentada. Este produto pode ser usado de uma forma estacionária (normalmente em cima de uma mesa) ou "*docked"* com óculos de realidade aumentada ou virtual. Esta tecnologia foi a primeira a ser descartado uma vez que a área de reconhecimento do sensor é reduzida, o que obrigaria a ter o equipamento muito perto do paciente, representando assim um risco para a assepsia do ambiente.

Após análise cuidada concluí que o Kinect V2 apresenta algumas limitações no contexto do presente trabalho; mais precisamente devido à restrição dos movimentos que o cirurgião pode executar quando está dentro da zona cirúrgica, em virtude das mãos deste encontrarem-se condicionadas à área formada entre os seus ombros e a cintura, por forma a assegurar assepsia dessa zona. Este *hardware* tem ainda a limitação no que respeita à variação da luminosidade dentro do bloco operatório, nomeadamente certas configurações das luzes na sala podem trazer ruído para a câmara de infravermelhos.

Em relação ao Myo Gesture Control Armband identificam-se algumas limitações aquando o seu uso no bloco operatório. Primeiramente, existe a necessidade de um equipamento por utilizador, já que a permuta do equipamento por usuários durante o procedimento não é possível sem o risco de comprometer o ambiente esterilizado. Em segundo lugar, é importante garantir que o ativar e desativar do equipamento não interfere com a cirurgia. Apesar destes contras, a Myo Gesture Control Armband acabou por demonstrar ser a melhor escolha.

## 2.3 Linguagens de Programação

Tendo sido escolhido o equipamento a utilizar, a próxima decisão a tomar passou por escolher a linguagem de programação a utilizar no desenvolvimento da plataforma.

Das linguagens suportadas pelo equipamento isolei as duas com as quais me sentia mais confortável nomeadamente C# e Java para candidatos a linguagem de programação.

As linguagens apresentam algumas semelhanças em pontos fortes, como serem ambas *"type-safe"*, utilizam objetos como referências, sendo por isso designadas de *"object-oriented"*, embora

com implementações diferentes suportam um *"garbage collector"* bem como sincronização, tratamento de exceções e *"multi-threading"*. Contudo C# é mais complexa, havendo a necessidade de ter cuidado com acessos à memória. Uma das forças mais alusivas ao Java é a sua independência do sistema operativo, uma vez que corre através da JVM ("*Java Virtual Machine"*) que pode ser instalada em Windows, Linux e respetivas distribuições e Mac. Em virtude do meu percurso académico ter sido predominante em Java e associado ao facto de ser independente do sistema operativo, decidi utilizar esta linguagem de programação[9].

Após a decisão que a linguagem de programação a utilizar seria Java, foi necessário escolher qual das versões, que à data da realização do trabalho estavam em produção, iria optar. As versões disponíveis são o Java 7 e Java 8, sendo esta decisão alvo de uma análise cuidada.

As grandes modificações de versão Java 7 para Java 8 passaram, não só pela correção de bugs, mas também pelo acrescento de novas funcionalidades como expressões lambdas, alteração da API até então usada para datas/hora para uma nova API mais sólida e robusta e algumas melhorias ao nível da segurança em geral. Pela correção dos *"bugs",* bem como a possibilidade de expressões lambda, que permitem melhorar o desempenho do código, a opção tomada recaiu sobre a versão 8.

## 2.4 "Standalone" ou "web based"

A decisão seguinte passou por escolher se seguir uma abordagem *"standalone"* ou *"web-based"*. Uma abordagem *"standalone"* consiste em desenvolver um programa sem qualquer dependência de *"software"* de terceiros, existindo a necessidade de ser instalado em cada máquina cliente. Estas são mais fáceis de implementar pois requerem substancialmente menos esforço para controlar um nó que uma rede inteira. Por outro lado, fica mais fácil de controlar eventuais danos, uma vez que, em caso de falha, apenas um nó é afetado. Contudo, esta também é a maior desvantagem desta abordagem pois introduz um ponto único de falha, em inglês SPOF ("*single point of failure"*), o que obriga a um investimento superior em infraestruturas uma vez que o servidor e aplicação encontram-se na mesma máquina, obrigando a que esta tenha de ter requisitos mínimos de *"hardware"*, para poder dar resposta ao "*software"*, ao próprio sistema operativo e outros "*softwares"* que estejam instalados.

Abordagens "*web-based"* permitem uma abstração das especificações necessárias pela máquina utilizada pelo cliente, dado que esta só tem de fazer a interpretação dos conteúdos e eventuais modificações dos mesmos. A própria modularidade destes sistemas acaba por contribuir para a escalabilidade dos mesmos, já que pode ser feitas melhorias nos servidores como replicação sem que o cliente tenha consciência de tal, bem como alta disponibilidade, na medida em que caso

um nó falhe este pode ser rapidamente substituído, sendo estas duas propriedades um bom efeito secundário.

Um dos requisitos principais da aplicação é a capacidade de partilha de informação. Não só pelas diferentes especialidades médicas dentro da mesma unidade hospitalar como também com profissionais médicos que estejam remotamente a auxiliar a cirurgia.

Atendendo a que o investimento necessário para uma solução *"web based"* por parte do cliente é menor, uma vez que só precisa de investir em máquinas que processam as informações, podendo utilizar as que já dispõe para a interpretação dos mesmos, e atendendo à necessidade de partilha de informações optei por seguir uma abordagem *"web based".*

### 2.4.1 Padrão MVC

Ao seguir uma abordagem *"web-based"*, um padrão de desenho frequentemente associado é o MVC (*"model-view-controller"*). Este é um padrão de desenho para o desenvolvimento de software no qual uma aplicação é dividida em três partes, evitando assim problemáticas como *"spaghetti code"*. Estas três partes são o modelo, a vista e os controladores, como o próprio nome indica, sendo cada um definido por[10]:

- Modelo: representa um objeto que contém dado, as regras e lógica de negócio. Em Java esses objetos são vulgarmente designados por POJO ("*Plain Old Java Object"*).
- Vistas: corresponde à visualização dos dados.
- Controladores: atuam como corretor, distribuindo os pedidos externos da aplicação aos respetivos serviços que os processam.

Como mencionado anteriormente, o meu objetivo é desenhar um *"backend"* operacional para a aplicação, pelo que este padrão de desenho surge como escolha óbvia, uma vez que me permite desenvolver uma solução modular, focando-me apenas no desenvolvimento dos modelos e dos controladores, já que são as componentes *"backend"*, delegando as vistas para integração ao *"frontend"*.

### 2.4.2 Web Services

Uma implementação *"web-based"* tem, naturalmente, serviços *"web",* vulgo *"web services"* associados. Quando falamos em *"web services"*, estes podem ser de dois tipos, SOAP

(“*Simple Access Object Protocol”*) ou REST (“*REpresentational State Transfer”*). Um erro frequente é a comparação destes dois como se uma fosse oposta à outra, contudo essa comparação não deve ser feita nesses termos, pois SOAP é, por definição um protocolo, enquanto que REST é um estilo de arquitetura.

#### 2.4.2.1 SOAP

SOAP apresenta-se como um protocolo que tem uma forte dependência com o formato *“XML”* (“*extensible markup language”*) para definir uma “*framework”* de mensagem fortemente tipificada. É uma excelente escolha quando existe uma necessidade de um contrato formal entre API e consumidores, uma vez que permite a utilização de WSDL (“*web services description language”*), o que obriga a que qualquer alteração que seja feita, quer ao nível do produtor, quer ao nível do consumidor, tem de ser comunicada ao respetivo através da modificação do WSDL, sobe risco de perda de comunicação entre ambos. Permite ainda definir operações *“stateful”*, isto é a aplicação guarda informação de todo o contexto desde o início da mesma, nomeadamente o que foi alterado durante a execução e dispõe de mecanismos de segurança para garantir o funcionamento na eventualidade de algum caso de perda de sincronismo.

#### 2.4.2.2 REST

REST é um estilo muito simples de aprender, uma vez que os seus pilares assentam na ideia de pedidos HTTP (“*Hypertext Transfer Protocol”*) associados a operações CRUD (“*Create, Read, Update, Delete”*). Sendo menos detalhada que SOAP, requer menos largura de banda e, como não segue uma abordagem *“stateful”,* não retendo em memória informações de transações passadas pelo que cada transação é sempre vista como nova e, portanto, pode ser colocada na cache do sistema o que melhora o desempenho e escalabilidade.

Optou-se por seguir uma abordagem REST uma vez que, por um lado, seguindo SOAP iria aumentar desnecessariamente a complexidade do sistema e, por outro, ao herdar as operações HTTP de *“get”, “put”, “post”* e *“delete”* facilita o desenvolvimento e a compatibilidade com o posterior com as vistas do MVC.

## 2.5 Frameworks

Relativamente a "*frameworks"* de desenvolvimento, foram analisadas três das muitas existentes no mercado por forma a escolher a que mais se adequa às necessidades do projeto, passando de seguida a explicar cada uma delas.

### 2.5.1 JPA & Hibernate

Codificar pesquisas em SQL ("S*tructured Query Language"*) num ambiente Java representa um grande esforço por parte do programador, além de ser uma tarefa que consome bastante tempo. Para aumentar a dificuldade, cada motor de base de dados tem as suas pequenas especificidades que fazem com que se torne difícil mudar pragmaticamente entre fabricantes. Existe ainda o problema de que Java é uma linguagem de programação orientada a objetos, como anteriormente mencionado, o que significa que os dados em Java são representados por um objeto, constituído pelos seus atributos e métodos, que pode herdar outras propriedades de outras classes enquanto que o modelo de mapeamento relacional representa os seus dados como tabelas, contendo uma chave primária e opcionalmente chaves estrangeiras para outras tabelas, além de conceitos como por exemplo índices secundários e procedimentos. Esta distância entre estes dois mundos é de difícil gestão e, como forma de resposta, começaram a aparecer as primeiras frameworks ORM (*"Object-Relational Mapping"*) [11].

JPA ("*Java Persistence API"*) é uma coleção de classes e métodos que permitem armazenar dados numa base de dados de uma força persistente, isto é, os dados permanecem muito depois do aplicativo que os gerou ter terminado o seu ciclo de vida. Normalmente JPA aparece associado a outros produtos de forma a expandir as potencialidades desses mesmo produtos.

O Hibernate, enquanto *"framework"* de ORM foi desenvolvido sem recurso ao JPA. No entanto, a combinação que se deu destes dois produtos tornou-se algo tão forte que não se consegue desassociar o uso de Hibernate sem as especificações do JPA. Por um lado, o Hibernate cria uma abstração do programador face à complexidade associada às operações sobre a base de dados, o que significa que para o programador é indiferente às peculiaridades do fabricante bem como a uma eventual troca do mesmo, uma vez que esta framework gera a camada JDBC ("*Java Database Connectivity"*) durante a execução para o motor de base de dados utilizado[12]. Por outro lado, ao conciliar com o uso do JPA, é introduzida mais uma camada de abstração que permite que o programador não tenha de saber todos os detalhes sobre o ORM que está por debaixo desta, sendo assim possível inclusive mudar a qualquer altura o ORM em uso desde que este utilize JPA. Portanto

o programador só tem de seguir os princípios estipulados pela API, que esta abstrai a complexidade do ORM em uso que, por sua vez, abstrai da complexidade da gestão do motor de base de dados.

Algumas vantagens inerentes ao uso do Hibernate são:

- Facilidade de escrever operações CRUD;
- Independente da base de dados;
- Suporta cache, o que permite otimizar as pesquisas;
- Os resultados da base de dados são devolvidos como objetos Java, pelo que não é preciso gerir manualmente a persistência dos dados.

Como desvantagens apresentam-se:

- Eventuais períodos de latência visto que tudo é gerido durante a execução da aplicação
- Aumento do tempo de início da aplicação uma vez que as informações das configurações das tabelas são feitas no arranque.
- Quebras de desempenho associadas ao aumento do volume de dados com o tempo e à complexidade das *"queries"*.

### 2.5.2 Structs2

Struts2 é uma *"framework"* para aplicações empresariais "*web"* em Java. Segue o padrão de desenho MVC analisado previamente na seção 2.4.1 com a nuance de o controlador ser designado por *"ActionController"*. Esta framework foi desenvolvida com vista a ser utilizada em todos os ciclos de desenvolvimento de uma aplicação, a começar pelo *"build"* e acabando no *"deploy"* passando pela manutenção do mesmo. Possui também uma linguagem própria para o desenvolvimento de vistas, pelo que é possível desenhar, desenvolver e implementar toda uma aplicação completa dentro do ambiente desta *"framework"*. Os componentes MVC são mantidos num ficheiro XML, o que permite que seja fácil fazer alterações em qualquer momento. Entrando em detalhe nos *"ActionController"* estes são na verdade POJOs o que exclui a necessidade de heranças e interfaces. Também suporta Hibernate e AJAX ("*Asynchronous Javascript and XML"*) sendo o último uma forma de construir aplicações que conseguem processar pedidos do utilizador quase de forma imediata, evitando que seja necessário um refrescamento total da página, mas apenas os conteúdos que necessitam de ser atualizados, contrariamente aos pedidos HTTP que forçam sempre ao refrescamento total da página [13]. Atualmente AJAX tem sido utilizado com bastante ênfase nas situações em que existe uma necessidade de exibir conteúdo dinâmico ou quando é necessário a validação de campos.

### 2.5.3 Spring

Spring é uma *"framework"* que personaliza a filosofia "dividir para conquistar". Permite o desenvolvimento de aplicações "*web"* complexas que assentam em módulos simples. Na verdade, a própria *"framework"* apresenta-se como um vasto ecossistema de módulos que podem ser utilizados no desenvolvimento da aplicação, em que o programador tem total liberdade de escolher quais usa, sendo estes módulos unidos por um módulo central. Spring MVC é um desses módulos que já existe e que permite utilizar esta ferramenta diretamente sobre o padrão de desenho de software MVC. Baseia-se num *"dispatcherServlet"* que é responsável por enviar cada pedido ao seu respetivo recetor para tratamento, através de um mapeamento estruturado dessas ligações.

Embora suporte uma implementação em que se use um ficheiro XML com as configurações do projeto, existe também a possibilidade de recorrer a anotações, o que faz com que se possa seguir uma abordagem baseada em classes ao invés de estarmos forçosamente dependentes de um XML como o Struts2. Essas anotações também trazem vantagens como permitir a injeção de dependências o que simplifica enormemente a forma como se gera as dependências cruzadas entre as classes, introduzindo assim um novo paradigma de programação, o AOP ("A*spect Oriented Programming"*) que pode ser usado para complementar o paradigma mais conhecido associado ao Java, o OOP ("O*bject Oriented Programming"*)[14].

Enquanto que no paradigma OOP tudo assenta no conceito de objetos, em AOP existe o conceito de aspetos e são estes que permitem a modularização da aplicação. Outra vantagem do Spring MVC é que o objeto utilizado nas operações não sofre duplicação, sendo as ações executadas diretamente no original. O Spring MVC não utiliza POJOS na definição do modelo, mas sim "*Spring Beans"*, que é um "*standard"* para o encapsulamento de classes, sendo as vistas frequentemente interpretadas com recurso a JSP ("*JavaServer Pages"*). No entanto, não existe a obrigação de uso de JSP, já essa escolha pode ser adequada caso a caso. Poder-se-ia apresentar a falta de suporte nativo de AJAX como uma desvantagem. Contudo, existem imensas bibliotecas disponíveis para tal, o que torna esta desvantagem numa espécie de vantagem já que permite liberdade na escolha da biblioteca a usar.

Das possibilidades analisadas, acabei por resumir as escolhas ao Struts2 e Spring. Ambas as *"frameworks"* dispões de pontos bastante fortes e apresentam-se como soluções mais do que adequadas ao tema. Contudo, e apesar da curva de aprendizagem para o Struts2 ser melhor, escolheu-se Spring devido essencialmente a dois aspetos: flexibilidade, pois o Struts2 tem uma relação muito forte entre cliente e servidor, o que força a que qualquer alteração em um dos lados tenha de ser

comunicada o mais rapidamente possível ao outro pois, em caso de divergência, o sistema deixa simplesmente de funcionar; a facilidade de implementação dos *"web services"* REST no Spring, já que apenas é preciso escolher as anotações apropriadas, com o benefício de que essas anotações ajudam ainda à identificação e manutenção dos próprios *"web services"*. No caso do Struts2 é preciso realizar um processo a dois níveis: primeiro a criação de um ficheiro para os "*URL"* ("*Uniform Resource Locator"*) e respetivos encaminhamentos e, em segundo, um outro ficheiro para a lógica de mapeamento[15].

Spring baseia o seu conceito de objetos para as aplicações em *"beans"*, sendo este geridos pelo módulo de IoC ("*Inversion of Control"*) da ferramenta. Estes *"beans"*, independentemente de serem do tipo *"Spring beans"* ou *"JavaBeans"* são um exemplo da política que a Sun seguia quando introduziu o Java no mercado sendo esta: *"Write once run anywhere"* - escrever (código) uma vez, correr em qualquer lado - uma vez que uma das aspirações era incentivar fortemente a criação de componentes de software reutilizáveis em qualquer. Existe, portanto, a garantia que o resultado final corresponde a um software reutilizável, onde existem contratos, vulgo interfaces, que garantem a interoperabilidade das várias classes.

### 2.5.4 JavaBeans, Enterprise JavaBeans e SpringBeans

O aparecimento do conceito de *"beans"* surgiu com "*JavaBean",* sendo os primeiros a unidade que compõe os segundos. Por definição são componentes de um software que podem ser reutilizados e que podem ser manipulados visualmente, numa ferramenta de compilação [16].

Esta definição relaciona-se com o facto de que, maioritariamente, são usados num contexto de desenvolvimento de componentes para interfaces gráficas para utilizadores. Contudo, o seu âmbito expandiu-se e agora os *"JavaBeans"* são vistos como uma forma de criar e implementar componentes reutilizáveis ao longo do desenvolvimento da aplicação. Dissecando um pouco mais o que são estes *"JavaBeans"* podemos considerar que são POJO que cumprem uma implementação estritamente definida:

- têm de ser serializáveis, isto é, têm de conter uma implementação da classe *Java.io.Serializable*;
- têm de conter pelo menos um construtor sem argumentos, que não pode ser o construtor omisso do Java. Este tem de ser explicitamente criado e não pode receber argumentos;
- têm de seguir o princípio de encapsulamento, onde todas os atributos só podem ser acedidos utilizando os respetivos *"getters"* e *"setters".*

O primeiro requisito serve para garantir a persistência dos dados na aplicação. Uma classe *"serializable"* garante que o estado de um objeto é devidamente salvaguardado entre as operações de acesso, seja de armazenamento ou de obtenção. O segundo requisito garante uma instanciação mais fácil do *"bean"*. O último surge como uma forma de, não só garantir uma forma mais acessível para atualizar e inspecionar o conteúdo dos *"beans",* mas também para garantir algum controlo na forma como essas duas ações são executadas. Desta forma, os JavaBeans permitem abstrair das problemáticas associadas às bases de dados, como a pesquisa e conexão à mesma. Contudo, são demasiado simplistas para as necessidades que muitas vezes enfrentam. Como forma de colmatar esta simplicidade e como resposta, surgiu uma nova versão destes, conhecida como *"Enterprise JavaBeans"*. Estes permitiram dar resposta a questões emergentes preocupantes para as empresas, nomeadamente a nível do comportamento transicional, que é uma necessidade de aplicações que sejam multitarefas e questões a nível da segurança das aplicações, através da disponibilidade de um ambiente no qual é possível desenvolver componentes distribuídos e até multinível. Estes *"Enterprise JavaBeans"* dividem-se em três grupos: *"session beans"*, que são responsáveis por manter as informações sobre a sessão do utilizador, que são destruídos aquando do término da sessão; *"entity beans"* que são responsáveis pela persistência dos dados na base de dados; por fim os *"message bean"* que são utilizados pelo serviço de mensagens do Java, o JMS ("*Java Message Service"*).

Spring introduziu o seu próprio conceito de *"beans"* que é distinto do *"JavaBeans"* e do *"Enterprise JavaBeans"*. Nas primeiras versões desta ferramenta, pretendia-se recorrer ao uso de *"JavaBeans"*, daí surgir o termo *"beans"*. No entanto, este refere-se a qualquer objeto que seja gerido pela ferramenta. Estes podem ser configurados com recurso a um ficheiro de configuração, tipicamente um XML de configuração, ou utilizando as anotações anteriormente mencionadas, sendo neste caso a anotação *@Bean.* O Spring encarrega-se de injetar o código necessário.

Os *"beans"* em Spring podem ser POJO, podendo, portanto, seguir a abordagem *"JavaBeans"* ou *"Enterprise JavaBeans"*, (classe "*serializable"*, com um construtor sem argumentos e encapsulamento), mas também podem ser definidos sem seguir todas os requisitos dos POJOS.

Estas características permitem que o *"beans"* de Spring sejam mais flexíveis que os "*JavaBeans"* e mesmo até que os *"Enterprise JavaBeans"*, acrescentando ainda mais uma vantagem; como estes são geridos a nível do módulo de Spring IoC, o Spring torna-se independente do servidor em uso, enquanto que os *"Enterprise JavaBeans"* requerem a existência de servidores especializados, como é o caso do *TomEE*, que se trata de uma instância especializada do servidor *Apache* uma vez que a instância vulgarmente distribuída, o *TomCat*, não suporta.

## 2.6 Base de Dados

Os sistemas de informação das instituições de saúde têm a sua própria implementação de base de dados que contêm os diversos registos dos seus pacientes. O acesso à referida base de dados no contexto deste trabalho como forma de teste e validação de resultados, não é possível por razões de privacidade e confidencialidade da informação. Assim, surge a necessidade de implementar uma base de dados para suportar a aplicação a desenvolver. Por outro lado, as bases de dados das instituições contêm mais informação do que é necessário por parte dos utilizadores da nossa aplicação, pelo que dotar o sistema de uma base de dados própria permite a criação de uma *"cache"*. Em ambiente produtivo, esta base de dados iria conter replicações de dados extraídos da base de dados principal, permitindo uma melhoria de performance de resposta do sistema. Nesta secção analiso as ofertas de mercado em termos de bases de dados como resposta à necessidade supra exposta ressalvando que não é do âmbito do projeto que se desenvolva todo uma nova implementação de uma base de dados que substitua as atualmente existentes nos sistemas de informação hospitalares.

À semelhança da análise efetuada para as "*frameworks"* de Java também procedi a uma análise elaborada das várias bases de dados que existem, começando por escolher qual a abordagem da aplicação face os dados que pretendo inserir. Atualmente existem duas grandes filosofias para o armazenamento de dados; com ou sem estrutura definida. A primeira, dados estruturados, corresponde ao grupo mais conhecido de base de dados, fazendo parte destes os fabricantes que seguem a abordagem relacional para o armazenamento e utilizam SQL para pesquisa de dados. A segunda, dados não estruturados, corresponde ao grupo emergente da abordagem NoSQL, sendo esta abordagem muitas vezes incorretamente interpretada pois a mesma defende a ideia de *"Not Only SQL"* - não apenas SQL - sendo muitas vezes confundida com "*No SQL"* - que seria não SQL. Este grupo encontra-se cada vez mais com maior representatividade no mercado, sendo utilizado em várias plataformas de redes sociais. Não obstante de não ter a mesma notoriedade que o *"SQL"*, esta abordagem não é recente, já tendo surgido em 1998.

### 2.6.1 SQL e NoSQL

As bases de dados *"SQL"* seguem a abordagem relacional para o armazenamento de dados das aplicações. Esta abordagem assenta no modelo relacional dos dados introduzido na década de 70 como uma forma matemática de estruturar dados em relações. Os princípios do modelo assentam na álgebra relacional e foi criado pelo Edgar F. Codd quando trabalhava para a IBM.

Por definição de álgebra, é necessário que existam operadores e operandos. Desta forma, para que se possa falar de álgebra relacional, é preciso definir estes dois pilares da álgebra e tal assim o é, uma vez que:

- os operandos são as relações ou variáveis que representam uma relação;
- as operações são definidas de forma a poderem representar as operações expectáveis de se fazer com essas relações, produzindo novas relações como resultados, passíveis de se conjugarem e criarem expressões mais complexas.

O conjunto de operações permitidas para este tipo de álgebra são[16]:

- seleção de valores de uma relação sendo este localizados na linha;
- projeção de atributos da relação sendo estes localizados nas colunas;
- união, diferença, produto cartesiano e junção;
- interseção e divisão;
- renomeação de relações e colunas.

Por estas características, pode-se considerar a álgebra relacional como uma linguagem teórica de pesquisa.

Outra parte do modelo relacional introduzido por F. Codd é o cálculo de tuplos ou cálculo relacional de tuplos. Estes baseiam-se em predicados, ao invés de álgebra pelo que o seu propósito é descrever a pesquisa que permite obter um dado resultado e não na forma como se obtém esse resultado. A título de exemplo, um caso de cálculo de tuplos seria "obter todos os detalhes de todos os engenheiros numa organização", enquanto que este caso será descrito como "*SELECT the worker's tuples from ORGANIZATION relation with EMPLOYEES = 'ENGINEER'"* em SQL.

Algumas das vantagens das bases de dados SQL são:

- Desempenho, pois, as pesquisas podem obter grandes volumes de dados de uma forma rápida em virtude dos *"standards";*
- SQL é um *"standard"* com bastante tempo de mercado, o que o torna maduro e com o suporte de uma grande comunidade, pelo que não apresenta resistência a qualquer adaptação a novas realidades;
- Geralmente, as pesquisas em SQL são simples e com uma baixa curva de aprendizagem, necessitando assim de muito pouco trabalho de programação.

As bases de dados *"Not Only SQL"* são cada vez mais utilizadas devido a dar resposta a um problema que representa o endereçamento de dados não estruturados, semiestruturados ou com estrutura dinâmica durante o ciclo de vida da base de dados, especialmente quando estes se apresentam em grandes volumes. Estes dados não requerem a definição de um modelo de dados como nas bases de dados relacionais. Dados não estruturados significam que os dados introduzidos possuem diferenças na sua estrutura, podendo ser desde ligeiramente, diferindo por exemplo no número de campos, a totalmente diferentes, como por exemplo não existirem campos comuns entre os dados. Esta realidade implica que as soluções que seguem esta abordagem são focadas e, portanto, específicas, para o problema que pretende dar resposta. Estas bases de dados foram desenhadas com vista a suportar ambientes distribuídos (partição de dados, replicação, processamento distribuído), como é o caso desta aplicação, permitindo escalar horizontalmente.

Posso assim comparar estas duas abordagens e estabelecer que em termos de fiabilidade, as bases de dados SQL apresentam-se mais vantajosas nos casos em que existam transações envolvidas nos processos, já que as NoSQL não dispõem de garantias ACID, do inglês *"Atomicity, Consistency, Isolation, Durability".* Em termos de pesquisas, todas as bases de dados SQL estendem, até um determinado nível os princípios inerentes ao modelo relacional, sendo que depois cada fabricante segue a sua metodologia, enquanto que as NoSQL têm um sistema de pesquisas desenhado especificamente para o problema que endereçam. Ambas são escaláveis, sobretudo a nível vertical, contudo as NoSQL permitem escalar facilmente de forma horizontal[18].

Em resumo, com o modelo SQL beneficia-se da integridade dos dados, possibilidade de transações, não acontece duplicação dos dados, um maior suporte em termos de comunidade e facilidade de escalonamento vertical, caso exista orçamento para investir em equipamentos e "*hardware"*. Porém fica-se limitado em termos estrutura dos dados, uma vez, que esta tem de ser bem definida como consequência da própria linguagem de pesquisa ser estruturada. Com o modelo NoSQL temos uma implementação mais simples e rápida, com boa performance no acesso aos dados, pesquisas rápidas quando estas sejam simples, facilidade de escalonamento horizontal através da adição de mais servidores. No entanto, pode ser preciso mais pedidos ao servidor na obtenção dos dados e se a implementação, por um lado, é mais simples, por outro é mais complicada de configurar como consequência da falta de estrutura.

Existe ainda quatro grandes tipos de bases de dados NoSQL pelo que primeiro é preciso analisar qual o tipo adequado ao caso antes de poder analisar fabricantes. Os tipos existentes são: *"document", "key-value", "column based"* e grafos. Seguindo a analogia à realidade hospitalar, um processo clínico é uma pasta que contém documentos, sendo alguns desses documentos imagens médicas. Por outro lado, esses documentos são editados ao longo do tempo, pelos vários especialistas

que os analisam, acrescentando ou removendo informações. Atendendo às razões apresentadas, o tipo escolhido foi *"document"*.

De entre as várias possibilidades dentro do modelo de SQL decidi considerar o MySQL e no modelo NoSQL do tipo *"document"* considerei o MongoDB pelas suas grandes presenças no mercado, como pela existência de uma forte comunidade e serem independentes do sistema operativo em que estão instaladas.

### 2.6.3 Sistemas de bases de dados

O MySQL começou por ser uma plataforma aberta que surgiu por volta de 1995. Na altura não seguia uma abordagem inteiramente relacional, mas acabou por seguir este rumo cedo por forma a poder incorporar mais funcionalidades. Esta decisão fez com que este motor fosse visto como menos maduro que os rivais diretos, nomeadamente PostgreSQL. Em 2009, foi adquirida pela "Oracle" e desde então têm sido feitos bastantes desenvolvimentos. Esta aquisição fez também com que o MySQL passasse a ter uma equipa dedica a melhorias, ao contrário do que sucedia no passado onde as melhorias vinham de programadores espalhados pelo planeta, permitindo um aumento da modularidade do sistema. Esta modularidade serve o propósito de se continuar a disponibilizar uma versão comunitária, gratuita, e robusta, em paralelo com oferta de módulos proprietários, só passíveis de utilização na compra de licenças. Assim, este motor continua a manter-se uma opção viável, mesmo quando surgem novos concorrentes, como MariaDB, e que aposta nas novas tendências da atualidade, como é o caso do investimento no produto. MySQL *Applier* com capacidade de integração com a *"framework Hadoop"* e assim poder prestar serviço em ambientes de "*big data".*

Atualmente, o MySQL apresenta-se como um motor de base de dados seguro em termos de proteção dos dados armazenados, sendo por isso utilizado em aplicações "web" bastante conhecidas como Facebook ou WordPress, com uma boa performance, uma vez que:

- suporta vários motores de armazenamento o que possibilita uma configuração mais pormenorizada às necessidades;
- suporta variadas soluções para alta disponibilidade, afirmando-se como sendo capaz de fornecer serviço 24x7;
- possibilita transações complexas, uma vez que possui características como atomicidade das operações, isolamento e consistência;
- tem a maioria da flexibilidade que as plataformas abertas tem, uma vez que é passível de utilização com uma licença gratuita que compreende a maioria das funcionalidades necessárias na generalidade dos casos.

Sendo MongoDB uma base de dados documental, importa começar por referir que o seu modelo assenta no modelo par de atributos, no qual um documento não é mais do que um conjunto de atributos e respetivos valores. É assim possível coexistir documentos com pares de chaves diferentes, seja por terem mais ou menos chaves ou inclusive atributos de um outro domínio, isto é, atributos que para um documento faça parte das suas características, mas para outro não. Uma vantagem deste tipo de bases de dados está associada ao facto de que costumam utilizar árvores hierárquicas, com recurso ao a ficheiros do tipo *"XML"* ou *"JSON",* o que faz com que a estrutura seja auto descritiva, independentemente da complexidade dos dados. Este motor apresenta-se sempre como sendo fácil de aprender, com uma curva de aprendizagem reduzida, para o caso de uma instalação simples é possível que em poucas horas se tenha algo pronto a utilizar e com o modelo de dados flexível já que não impõe nenhum esquema pré-definido, podendo cada documento ter o seu próprio conjunto de atributos tal como mencionado anteriormente. Outras vantagens associadas a esta base de dados são a sua performance, uma vez que uma boa parte dos dados são mantido em RAM (*"Random Access Memory"*), enquanto que o disco rígido serve principalmente para persistência de dados, pelo que o acesso aos dados é bastante mais rápido, a existência de bastante documentação e a possibilidade de desenvolver "*scripts"* do lado do servidor. Como aspetos menos positivos, apresenta-se o facto de MongoDB não suportar transações, pelo que não pode ser utilizada em cenários onde tal seja necessário; não suporta gatilhos (*"Triggers"*) pelo que é preciso desenvolver *"scripts"* para propagar entre documentos eventuais alterações, na eventualidade de algum documento ter de ser eliminado, essa alteração só irá libertar automaticamente espaço em disco quando a aplicação reinicie ou alternativamente esse processo terá de ser manual; a utilização de junções (*"joins"*) pode tornar o código mais difícil de interpretar já que a melhor forma de atingir este objetivo passa por realizar múltiplas pesquisas e por último a gestão de dados duplicados, quer seja propositado por redundância ou intencional; gerir dados duplicados pode ser complicado uma vez que não garante os princípios de atomicidade, consistência, isolamento e durabilidade. Por último, o suporte de indexação é uma mais valia quando feito corretamente, no entanto uma má escolha de índices pode fazer com que a base de dados se torne bastante lenta[19].

Existe ainda uma característica única associada ao MongoDB que está relacionada com o tamanho máximo dos documentos. Os documentos são podem exceder os 16 Mb de tamanho. Esta restrição existe pelo facto de evitar o consumo desmesurado de RAM [208]. Esta limitação pode ser facilmente superada com recurso ao GridFS, uma especificação criada para tratar todos os documentos que tenham de ultrapassar este limite de 16Mb. Para o conseguir, o GridFS primeiro divide o documento original em sub-blocos de 16Mb, armazena esses sub-blocos e guarda os identificadores de cada um deles num documento. Finalmente, é devolvido à aplicação o identificador desse documento que contêm os identificadores para cada sub-bloco.

As tabelas 1 e 2 pretendem resumir as palavras-chave, forças e fraquezas tanto do MySQL como o MongoDB. Existem sistemas inclusive que optam por uma abordagem híbrida, na qual o cliente *"NoSQL"* se encarrega da parte da apresentação dos conteúdos, nos casos em que se pretende apresentar itens com várias propriedades eventualmente diferentes entre si, e a nível do *"core"* deixam à responsabilidade de um cliente *"SQL"* a gestão de todas as complexas transações que possam ser necessárias.

| MySQL | MongoDB |
|---|---|
| Filosofia RDBM | Orientado ao documento |
| Grande foco na estrutura dos dados | Independente da estrutura dos dados |
| - Pesquisa *"full-text"* | Pesquisas específicas para cada caso |
| Transações e *"triggers"* | Processamento mais rápido com recurso à RAM |

*Tabela 1 Principais características do MySQL e MongoDB*

| MySQL | MongoDB |
|---|---|
| Maturidade (+) | Integração com vários motores de armazenamento (+) |
| *"Joins"* e transações (+) | Validação dos documentos e tempos de resposta curta entre falha e recuperação (+) |
| Difícil escalonamento (-) | Ausência de transações (-) |
| Necessidade de licenças para módulos proprietários da Oracle (-) | Solução recente (-) |

*Tabela 2 Forças e fraquezas do MySQL e MongoDB*

Como referido no início da secção, o âmbito do projeto não passar por apresentar uma base de dados que substitua ou concorra com a utilizada presentemente pelos hospitais. Por esta razão, considero que, embora uma abordagem SQL pudesse ser mais apropriada, iria exigir demasiado tempo, nomeadamente em termos da definição cuidada das entidades, e introduziria complexidade, como a gestão de chaves primárias e validação cuidada dos valores de cada campo, desnecessária ao

problema. Assim, sabendo que a necessidade basilar de uma base de dados é apenas o suporte à validação da aplicação e, eventualmente, servir como uma espécie de *"cache"* para a apresentação dos conteúdos restringidos ao paciente e ao procedimento em curso, melhorando a *"performance"* do sistema, decidi escolher uma abordagem NoSQL com recurso ao MongoDB.

## 2.7 Plataforma Servidor

Antes de entrar em detalhe nos servidores, seja aplicacional ou "web" importa primeiro indicar a razão da sua necessidade no contexto do trabalho. Uma vez que este trabalho é parte de um sistema maior, que será integrado com uma parte de *"frontend"*, existe a necessidade disponibilizar o código Java.

Um servidor aplicacional disponibiliza um enorme leque de ferramentas que podem ser utilizadas como por exemplo contentores "*web"*, contentores de EJB, acessos via JMS, Java Mail, ligação JDBC entre outras que juntas formam o ambiente EE (*"Enterprise Edition"*). Como o objetivo destes servidores é poderem disponibilizar um ambiente EE robusto para o desenvolvimento de aplicações, indiretamente estes oferecem ainda a vantagem de os seus módulos podem inclusive ser utilizados fora deles. Assim, um servidor aplicacional é capaz de fornecer lógica de negócio à aplicação em que esteja inserido. Por sua vez, um servidor "web" trata apenas da gestão de pedidos HTTP. Quando um pedido é feito ao servidor "web" este, por um lado, delega ao programa adequado ao seu tratamento e, por outro, disponibiliza um ambiente simples para que a parte *"server-side"* do programa possa executar e realizar as operações internas necessárias para produzir uma resposta.

A nível de servidor aplicacional decidi focar-me no GlassFish e para servidor "*web*" no TomCat, tendo em conta a sua representatividade no mercado.

GlassFish é um servidor aplicacional *"open source"* disponibilizado pela Oracle e é visto como uma referência para aplicações Java EE. As suas últimas versões têm tentado resolver questões de performance, desde os tempos de arranque à própria complexidade das operações, o que permite uma melhor utilização já desde a fase de desenvolvimento da aplicação.

TomCat é um servidor *"web open source"* pertencente à fundação Apache. Disponibiliza uma implementação para Java Servlet e páginas JavaServer cujas especificações são parte integrante da *"Java Community Process".* Este servidor é também visto como uma implementação leve de um ambiente EE uma vez que também suporta algumas possibilidades deste ambiente. Regra geral aplicações que consigam executar no TomCat conseguem também executar no GlassFish sem praticamente ser necessário alterações.

Tanto o GlassFish como o TomCat são programas feitos em Java com objetivo de disponibilizar uma forma de implementação de aplicações Java num ambiente *"web"* e suportados pela maioria dos IDEs.

Em reuniões com a Kcs-It ficou estabelecido que o resultado final deveria ser integrado no servidor TomCat da mesma.

Uma vez que o desenvolvimento da aplicação acaba por não ficar condicionada à escolha do servidor, decidi utilizar GlassFish durante o desenvolvimento, pela comodidade de poder delegar no IDE toda a temática da configuração, instalação e *"deploy"*, sabendo que o produto final irá ser executável em qualquer um dos dois, bastante apenas alterar o local onde é feito o *"deploy"* do ficheiro WAR.

## 2.8 Maven

Não sendo um servidor, aproveito ainda este capítulo para referir o Maven. Esta ferramenta permite que a compilação de um projeto em Java seja mais automatizada O projeto é descrito num ficheiro chamado POM (*"Project Object Model"*) no qual constam todas as dependências dos recursos na aplicação. Este ficheiro também permite definir a ordem com que o projeto é compilado. Por considerar esta ferramenta uma mais valia na gestão de dependências, nomeadamente na forma como as mesmas se obtém, que neste caso é a partir de um repositório central, além de controlar a versão das mesmas, foi decidido utilizá-la para o desenvolvimento da aplicação.

## 2.9 Design Pattern

Os padrões de desenho de software são essenciais no desenvolvimento de uma aplicação pois garantem a que a mesma pode ter módulos reaproveitáveis, quer internamente no projeto quer externamente para outros projetos, além ajudarem a garantir qualidade no produto desenvolvido já que fazem parte e contribuem para a utilização de boas práticas de programação. São metodologias de desenvolvimento de software testadas e reconhecidas para a resolução de problemas típicos.

### 2.9.1 IoC

O padrão de desenho Inversão de Controlo é conhecido por Injeção de Dependência (DI). Uma aplicação Java tradicionalmente é composta por mais que uma classe e estas têm de cooperar entre si para realizar uma qualquer operação lógica no âmbito do negócio da aplicação. Se por um

lado, uma forma de desenvolvimento passa por uma classe deter as suas próprias referências para objetos das demais classes, por outro lado uma outra forma de desenvolvimento passa por cada classe disponibilizar as suas dependências a uma outra entidade responsável por gerir e coordenar os objetos sendo esta última designada por injeção de dependência [13].

Em termos práticos, durante o ciclo de vida do programa, as classes que o compõe vão inicializando as suas dependências *"on demand"*. Com o recurso ao IoC, existe uma entidade responsável pela gestão das dependências e, desta forma, quando no decorrer da execução uma classe é instanciada, esta entidade disponibiliza de imediato as dependências necessárias invertendo assim a lógica tradicional e justificando a designação de inversão de controlo. Também por esta razão muitas vezes encontra-se este padrão de desenho associada à máxima de Hollywood: *"Don't call us! We'll call you!"*.

Este padrão pode ser complicado de implementar em aplicações com elevado número de classes. Como tal, existem diversas "frameworks" que permitem facilitar este tema, através de recurso a ficheiros XML por exemplo para gerir as dependências. Por exemplo o *"framework"* Spring, que será utilizado neste projeto, já disponibiliza meios para utilizar este padrão de desenho, pelo que faz todo o sentido tirar partido de tal funcionalidade.

### 2.9.2 DAO

Este é um dos padrões de desenho mais divulgado e aceite entre os programadores de Java uma vez que permite uma total abstração da persistência dos dados numa aplicação. Esta camada de abstração permite que qualquer alteração a nível da persistência de dados não implique mudanças em toda a aplicação, mas sim nesta camada já que esta é responsável por disponibilizar os dados à aplicação.

Atendendo ao uso do modelo MVC, esta camada de abstração servirá também como forma de providenciar os pontos de acesso aos controladores para a inserção e extração de dados.

### 2.9.3 Nested Builders

Os *"builders"* são classes criadas com o propósito de serem responsáveis pela criação das instâncias de uma classe. Tradicionalmente, as instâncias de uma classe podem-se obter através da invocação dos métodos construtores, omisso ou explícito de uma classe. Contudo, este padrão de desenho permite garantir um maior controlo na forma como são criados os objetos, uma vez que não trabalha diretamente sobre o objeto, mas sim uma referência do objeto, isto é, não fazem as

inicializações dos valores dos atributos da classe diretamente sobre o próprio objeto criado, como os construtores, mas sim retorna uma referência para um novo objeto criado com os valores pretendidos. Desta forma é possível garantir que os objetos criados são os expectáveis, sem correr o risco de ter problemas de referências erradas e consequentemente acessos indevidos, e permitem um maior encapsulamento por parte de quem fornece o *"builder"*. Os *"nested builders"* surgem como um meio termo entre uma classe "builder" e um construtor. Em primeiro lugar é preciso recorrer a outra das funcionalidades do Java, o *"method overload"*. Esta funcionalidade permite que uma classe tenha mais que um método com o mesmo nome desde que difiram nos argumentos, o que permite que uma classe possa definir tantos construtores quantos sejam precisos. Tendo a classe todos os construtores definidos e declarados como privados, o passo seguinte é criar um ou mais métodos estáticos que instanciam um objeto da classe com os valores desejados, usando os construtores e depois retornam a referência para esse objeto. Cada um destes métodos estáticos é designado como *"nested builder"*. Desta forma conseguimos ter as vantagens de uma classe "builder" sem a necessidade de aumentar o número de classes da aplicação.

# 3. Levantamento de requisitos

Um *"software"* aplicacional tem em vista a resolução de um problema do mundo real. Por forma a conseguir garantir um produto de qualidade e focado na solução, é preciso elaborar um caderno de requisitos a cumprir. Esses requisitos dividem-se na categoria de funcionais, responsáveis pela definição das capacidades do sistema, e os não funcionais, responsáveis pelo comportamento esperado. Neste capítulo apresenta-se a análise de requisitos realizada e que foi validade pela empresa onde se desenvolveu a dissertação.

## 3.1 Requisitos Funcionais

### 3.1.1 Gestão de utentes

É necessário registar um novo paciente ou modificar um que já exista. Em termos de modificar, admite-se que um paciente possa modificar dados pessoais ou introduzir um novo processo clínico.

### 3.1.2 Gestão de processos

É necessário criar um novo processo clínico para atribuir a um paciente. Também é requerido que o mesmo se possa modificar para introduzir novos registo de MCDTS, sempre que estes sejam solicitados, ou acrescentar anotações médicas por parte do profissional que esteja a analisar o processo.

### 3.1.3 Gestão de MCDTs

Em primeiro lugar é preciso criar um MCDT que seja associado a um processo clínico. Depois de se criar o registo, é então necessário associar-lhe todos os recursos visuais, isto é, imagens DICOM, bem como anotações que o profissional encarregue do exame considere necessárias.

## 3.2 Requisitos não funcionais

Em termos de requisitos não funcionais, os mesmos foram divididos em 3 categorias.

### 3.2.1 Requisitos de recursos

A distributividade da aplicação é uma mais valia, uma vez que permite tirar o máximo partido das infraestruturas existentes. Com vista a conseguir uma solução multiplataforma, é necessário um servidor para poder fazer o *"deploy"* da aplicação que irá correr nas máquinas cliente, bem como uma base de dados para armazenar as imagens médicas de suporte à validação.

### 3.2.2 Requisitos de interface

É necessário que existam contratos que especifiquem as operações suportadas pelo *"backend"* e como as mesmas se utilizam.

### 3.2.3 Requisitos de usabilidade

É necessário criar acessos à aplicação que permitam uma extração concisa dos dados. Os mesmos passam por implementar serviços que permitam cumprir os requisitos funcionais.

## 3.3 Casos de uso

Os casos de uso servem para ilustrar comportamentos esperado, da aplicação, contando uma história que retrate de uma forma generalista uma Acão na aplicação.

Após reunião com responsáveis da empresa definiram-se os seguintes casos de uso.

### 3.3.1 Adição de novo paciente

Uma pessoa que necessita de cuidados médicos dá entrada na unidade hospitalar. Essa pessoa nunca esteve na presente instituição de saúde, sendo necessário proceder ao registo da sua ficha médica. Um funcionário recolhe os seus dados pessoais e criar um registo do paciente no sistema *(ver Figura 3)*.

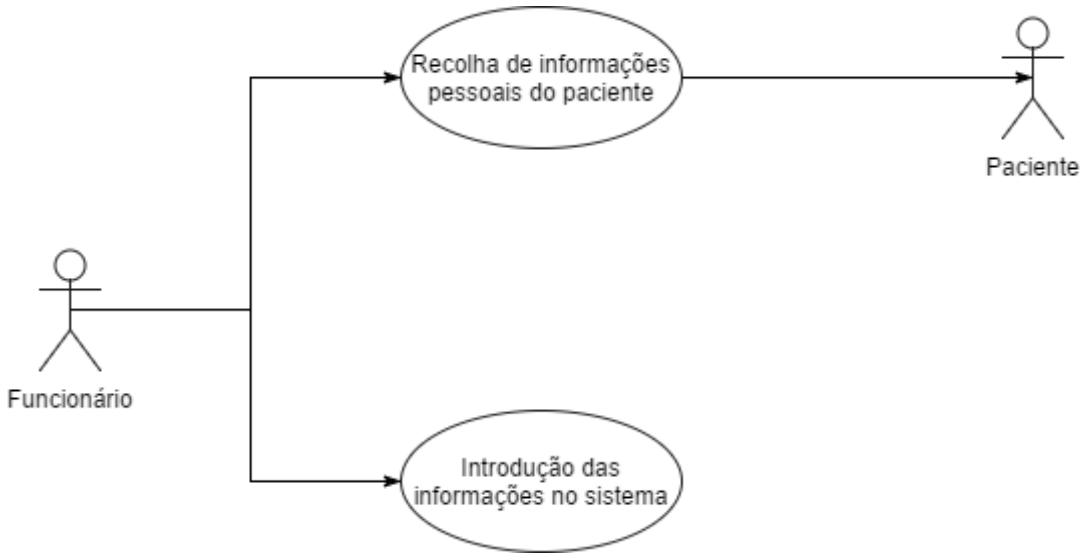


*Figura 3 Caso de uso novo paciente*

### 3.3.2 Adição de novo processo com MCDT

Um médico examina um paciente, iniciando um episódio clinico. O paciente após a consulta médica recebe uma requisição para a realização de MCDTS imagiológicos. Após a realização do(s) exame(s), o funcionário que os receciona tem de atualizar o processo com a introdução das imagens médicas *(ver Figura 4)*.

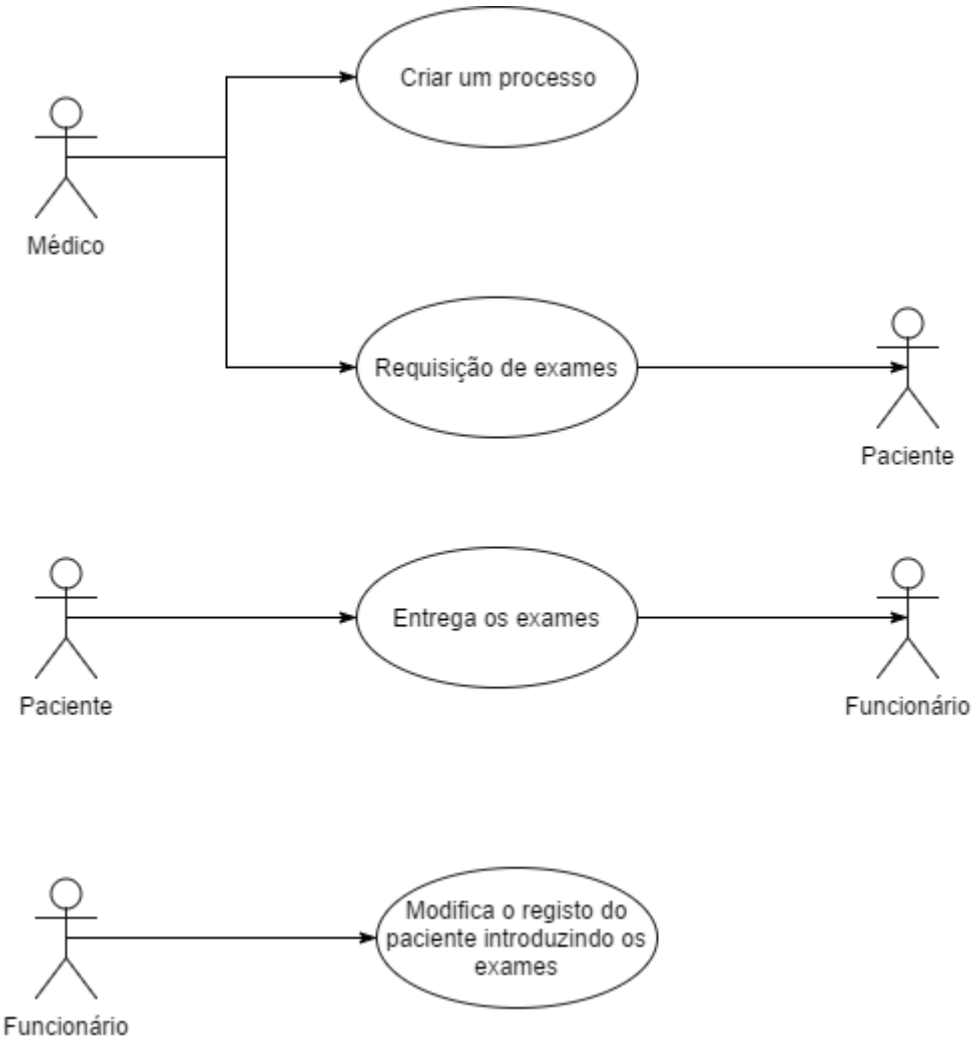


*Figura 4 Caso de uso novo processo com requisição de exames*

### 3.3.3 Ato cirúrgico

Durante a cirurgia, o médico tem necessidade de consultar imagens médicas armazenadas no sistema, bem como eventuais anotações que tenha introduzido. Para tal, primeiro tem à sua disposição as categorias de exames disponíveis, contendo uma *"thumbnail"* alusiva ao tipo de exame. Escolhida a categoria do exame também pode visualizar *"thumbnails"* dos exames que foram carregados nessa categoria bem como as anotações que tenham sido feitas. Por último seleciona a imagem pretendida. São feitos pedidos ao sistema por *"thumbnails"* das imagens médicas, anotações e as próprias imagens *(ver Figura 5).*

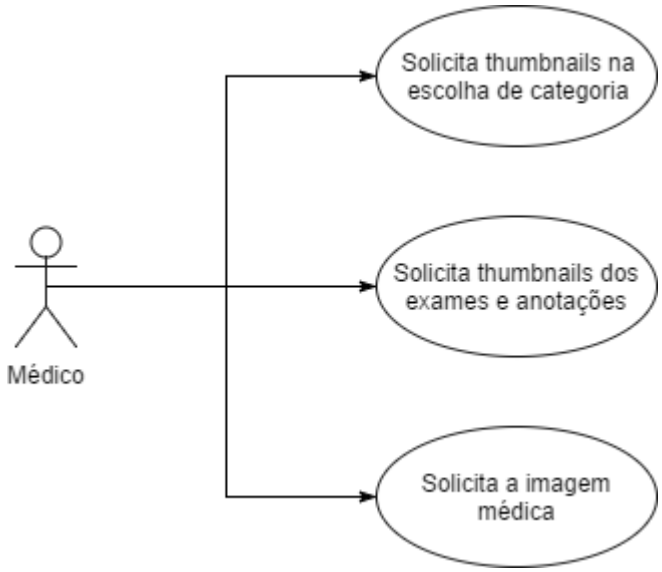


*Figura 5 Caso de uso solicitar imagens durante cirurgia*

# 4. Proposta de Arquitetura e Implementação

No presente capítulo pretende-se expor a proposta de arquitetura, a proposta dos documentos constituintes do modelo de dados para o desenvolvimento da aplicação, precedido pela implementação do mesmo.

## 4.1 Entidades intervenientes

Ao nível de entidades intervenientes no sistema, considero que existem duas externas e uma interna. Como entidades externas existem a base de dados e os médicos que interagem com o sistema, sendo a gestão da aplicação responsabilidade da atividade interna designada por *"core"*.

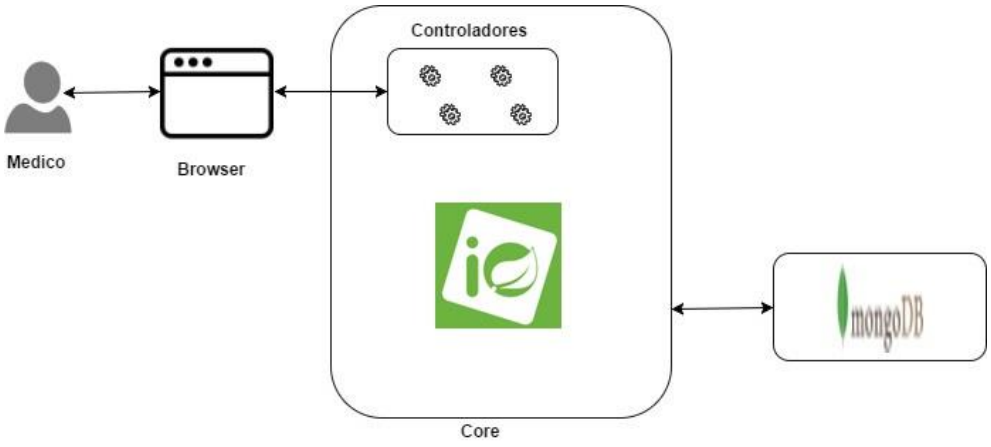


*Figura 6 Entidades do sistema*

Aprofundando um pouco mais o grau de intervenção de cada entidade, posso considerar o(s) médico(s) como entidades ativas no sistema, isto é, responsáveis por gerar uma resposta por parte do mesmo. Interagindo com o mesma via "*browser"* onde está a parte *"web"* da aplicação, são criados pedidos REST que são entregues aos controladores presente no *"core"* do sistema. Estes controladores dispões de objetos de acesso a dados, geridos pelo Spring e despoletam uma resposta ao nível da base de dados.

## 4.2 Arquitetura

Das entidades intervenientes detalhadas na subsecção anterior, duas são do âmbito da presente tese, sendo estas o *"core"* e a base de dados.

A nível do *"core"*, o cerne do *"backend"* aplicacional proposto está em Spring. Esta *"framework"* será responsável pela gestão dos pedidos à aplicação, tanto para o envio como para a receção de dados. Esta entidade será dotada de controladores que, tirando proveito das vantagens do Spring como a injeção de dependências, ficarão responsáveis tanto por enviar dados a armazenar na base de dados como por fornecer dados ao *"frontend"* para visualização. Os controladores são acedidos via *"endpoints"* e encontram-se divididos de acordo com a natureza dos pedidos que tratam, o que implica que os controladores se dividam em *"gets"* (obter) e *"sets"* (inserir), devidamente mapeados para os serviços suportados pela aplicação. Estas ações de inserção e obtenção de dados utilizam *"Spring Beans"* via interfaces, designadas também por contractos (do inglês *"contracts"*) e que definem o conjunto de operações permitidas. Será utilizado GlassFish como servidor aplicacional durante a fase de desenvolvimento. No entanto, o produto final usará o TomCat da empresa.

Como se pode ver pela Figura 7, o fluxo das comunicações passa por um pedido REST ao controlador, sempre que o utilizador intervém no sistema, seja para obter uma informação ou introduzir um dado novo, que depois se propaga dentro do sistema pela injeção de dependências, até ser feita uma consulta à base de dados para obtenção da informação solicitada ou uma informação de sucesso/erro na introdução de novos dados.

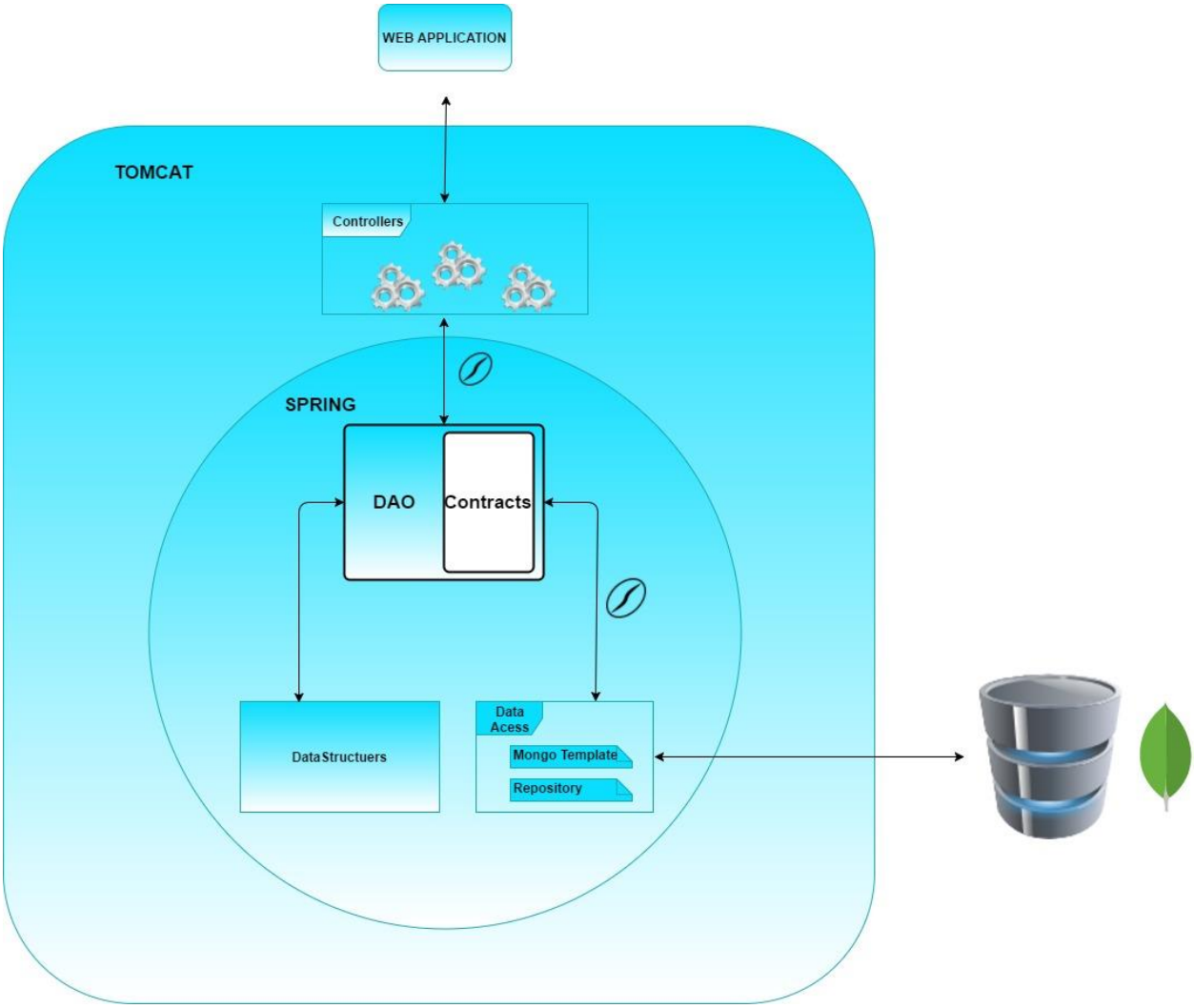


*Figura 7 Ilustração da arquitetura do sistema*

O motor de base de dados em uso é MongoDB. Caso os documentos a arquivar sejam de tamanho inferior a 16Mb, como por exemplo as informações pessoais dos pacientes, estes são armazenados diretamente no MongoDB como se observa na Figura 8.

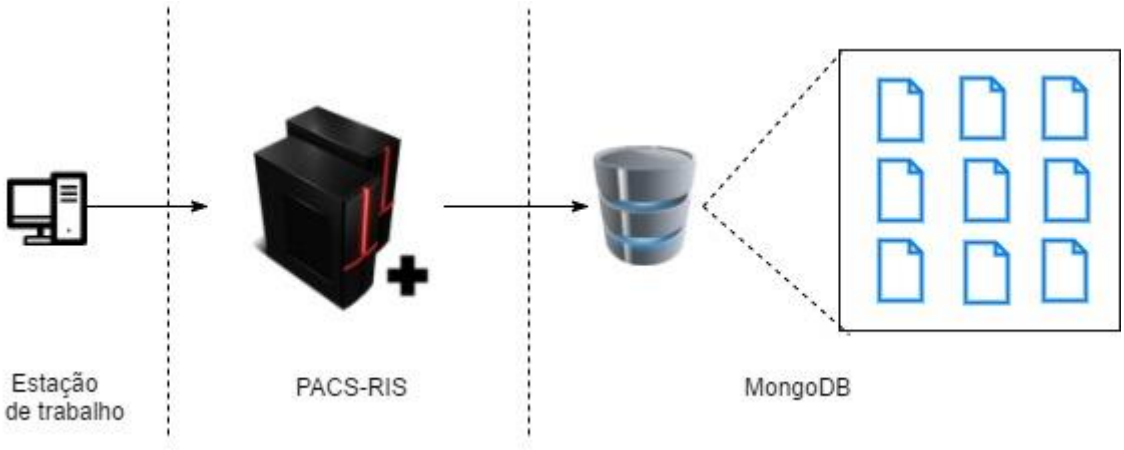


*Figura 8 Armazenamento de Documentos com menos de 16Mb*

Para os registos digitais médicos, no formato DICOM, que facilmente ultrapassam esta limitação de 16Mb por documento, o armazenamento é feito com recurso ao GridFS. Nesta situação o MongoDB armazena um documento que contêm os identificadores de cada um dos blocos de 16Mb em que as imagens médicas são divididas conforme ilustrado na Figura 9.

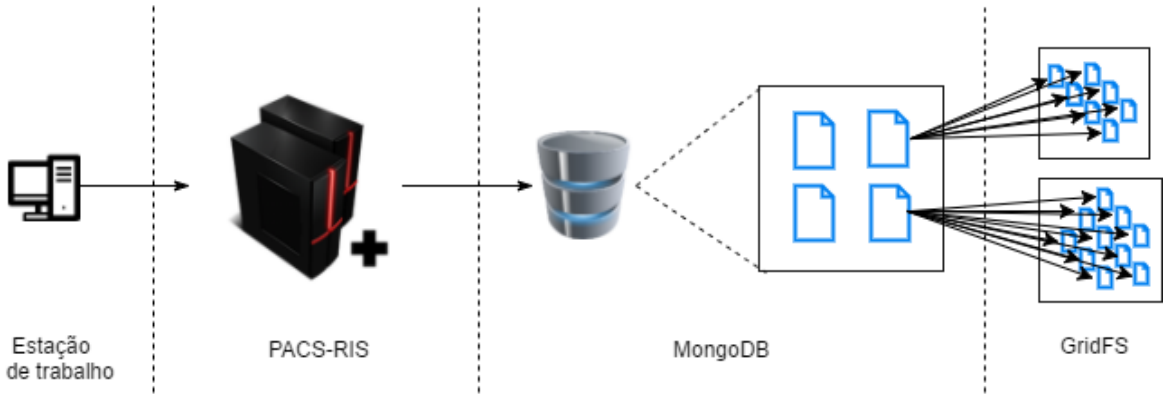


*Figura 9 Armazenamento de um documento utilizando GridFS*

## 4.3 Documentos

A nível da base de dados e em conformidade com a minha decisão de seguir uma abordagem NoSQL, através de uma implementação do MongoDB, não existe uma definição de um modelo de dados relacional, contudo foi preciso definir os tipos de documentos que fazem parte da aplicação. Uma abordagem poderia ser um único documento por cada paciente contendo toda a informação relativamente aos MCDTs do mesmo. Contudo gerir um documento com tal complexidade seria um trabalho complexo e com bastante propensão ao erro pelo que optou-se por diferentes tipologias de documentos, por forma a reduzir a complexidade, o risco de erros e também a retratar a realidade hospitalar.

Um paciente tem sempre associado pelo menos um processo clínico e esse processo pode conter ou não MCDTs. Os documentos são registos no formato JSON e, seguindo a linha de pensamento anteriormente exposta, os tipos de documento e respetiva definição são os seguintes:

- Paciente
    - Um paciente é alguém que se dirige a uma instituição hospitalar e se caracteriza no mínimo pelas seguintes características: nome, número identificativo (cartão de cidadão ou identificação fiscal), sexo, morada, telefone, e-mail. Pode ainda estar no estado arquivado, isto é ter pedido para ser removido da instituição e pode conter 0 ou mais processos. A figura 8 ilustra um documento "paciente" na base de dados.

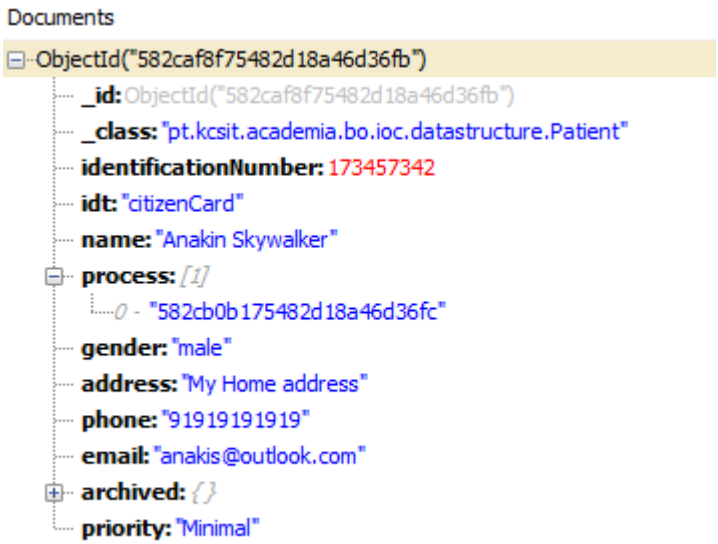


*Figura 10 Exemplo de um documento Paciente*

- Processo:
    - Sempre que um paciente necessita de alguma intervenção médica é aberto um processo. Um processo contém a informação do paciente ao qual se refere, uma descrição, uma data de criação, uma lista que pode conter 0 ou mais MCDTs mediante a necessidade de tal e por último uma lista de cirurgias caso este processo surja como necessidade de uma intervenção cirúrgica. A figura 9 ilustra um documento "processo" na base de dados

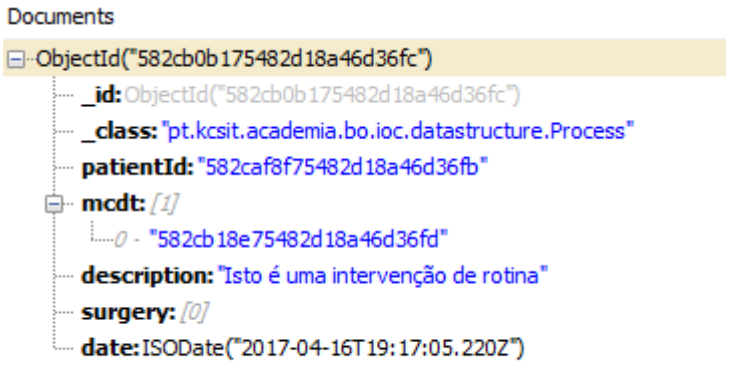


*Figura 11 Exemplo de um documento Processo*

- MCDT
    - Os MCDT são os dados mais importantes da aplicação. Têm associados a si um paciente e um processo. Contêm uma lista de recursos, isto é 1 ou mais ficheiros DICOM com respetivas anotações por parte de quem executou o exame e uma tipologia (ex. ECG, RaioX), data de realização, nome da imagem e por fim um campo próprio de anotações que o cirurgião pode utilizar para o auxiliar durante a cirurgia. A figura 10 ilustra um documento "MCDT" na base de dados

w

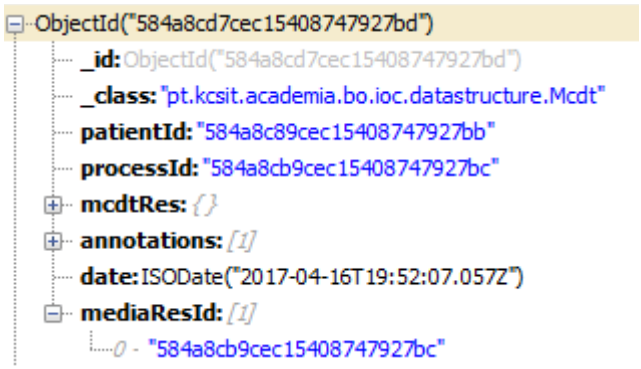


*Figura 12 Exemplo de um documento MCDT*

Com vista a desenvolvimentos futuros, contemplo ainda a existência dos seguintes tipo de documentos que se encontram por definir: cirurgia, bloco operatório, equipa médica e profissional de saúde.

## 4.4 Metodologias

O modelo de desenvolvimento de software escolhido foi o "Waterfall". Este modelo data de 1970 no qual o desenvolvimento encontra-se dividido em várias etapas, sendo que o término de uma implica o início da seguinte, seguindo assim a analogia de cascata. As atividades foram implementadas utilizando metodologias ágeis, nomeadamente o Scrum. Para um controlo de versões foi utilizado o repositório Apache Subversion (SVN) da empresa. Foi assim possível agendar as várias atividades necessárias ao desenvolvimento do produto, havendo um controlo do tempo gasto por tarefa e por conjunto de tarefas; tendo ainda sido possível garantir o funcionamento de cada unidade planeada de forma correta com recurso ao modelo TDD (*"test driven development"*).

O sistema utiliza Spring como *"framework"*, REST como protocolo de comunicação, MongoDB associado ao GridFS como sistema de armazenamento de informação e esta apto para se utilizado na instância TomCat da empresa.

## 4.5 Serviços implementados

Assim como os serviços servem o propósito de poder cumprir o estipulado no levantamento de requisitos, os controladores são os responsáveis por garantir os serviços em uso pela aplicação. Com vista a garantir uma transparência da implementação dos mesmos e com o objetivo de facilitar futuros desenvolvimentos, decidi criar controladores específicos consoante o seu propósito.

Em caso de erro em algum dos serviços, existe um controlador dedicado para disponibilizar um código de erro e respetivo significado.

Os três principais controladores implementados são o controlador dos processos, o dos pacientes e o dos MCDTs.

### 4.5.1 Controlador processos

Este controlador é responsável pela gestão dos processos clínicos.

Os *"endpoints"* para estes controladores começam por: {server}/api/process

#### 4.5.1.1 Novo processo

Este serviço permite criar um processo no sistema. *Ver tabela 3.*

- Endpoint: /save/newProcess/
- Descrição: cria e armazena um novo processo

| Método | Parâmetros | Resposta |
|---|---|---|
| Post | Sem parâmetros | 200 “Ok” |
| | | 400 “Bad Request. Internal error” |

*Tabela 3 Serviço para criar um processo*

#### 4.5.1.2 Listar processos

Este serviço permite listar os processos associados a um paciente. *Ver tabela 4.*

- Endpoint: /{patientId}/all/
- Descrição: obter a lista de processos de um paciente

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Identificador do paciente | Uma lista contendo os identificadores de todos os processos do paciente. |
| | | 400 Bad Request. Patient not found |

*Tabela 4 Serviço listar os processos de um paciente*

#### 4.5.1.3 Detalhes processo

Este serviço permite obter as informações referentes a um processo. *Ver tabela 5.*

- Endpoint: /{processId}
- Descrição: listar detalhes do processo

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo | Retorna os detalhes do processo em causa |
| | | 400 Bad Request. Process not found. |

*Tabela 5 Serviço para obter os detalhes de um processo*

#### 4.5.1.4 Obter o número total de MCDTs do processo

Este serviço permite saber quantos MCDTs de cada tipo estão associados ao processo. *Ver tabela 6.*

- Endpoint: /{processId}/mcdt/
- Descrição: obter o número de MCDTs por tipo.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo | Número de MCDTs por tipo |
| | | 400 Bad Request. Process not found |

*Tabela 6 Serviço para criar um processo*

#### 4.5.1.5 Obter MCDTs de um dado tipo

Este serviço permite uma lista de MCDTs de um dado tipo que estejam no processo. *Ver tabela 7.*

- Endpoint: /{processId}/mcdt/{type}
- Descrição: obter os MCDTs por tipo

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo<br>Tipologia MCDT | Lista de MCDTs do tipo solicitado |
| | | 400 Bad Request. MCDT type not found |

*Tabela 7 Serviço para listar MCDTs por tipo*

### 4.5.2 Controlador Pacientes

Este é controlador é responsável pela gestão de pacientes.

Os *"endpoints"* para estes controladores começam por: {server} /api/patient/

#### 4.5.2.1 Novo paciente

Este serviço permite criar um paciente no sistema. *Ver tabela 8.*

- Endpoint: /save/
- Descrição: cria e armazena um novo paciente

| Método | Parâmetros | Resposta |
|---|---|---|
| Post | Sem parâmetros | 200 Ok |
| | | 400 Bad Request |

*Tabela 8 Serviço para criar um paciente*

#### 4.5.2.2 Listar todos

Este serviço permite listar todos os pacientes. *Ver tabela 9.*

- Endpoint: /all/
- Descrição: listar todos os pacientes

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Sem parâmetros | Lista de todos os pacientes. |
| | | 400 Bad Request. Internal error |

*Tabela 9 Serviço para listar pacientes*

#### 4.5.2.3 Pesquisar paciente

Este serviço permite pesquisar um paciente no sistema. *Ver tabela 10.*

- Endpoint: /id/{patientId}/
- Descrição: pesquisar um paciente

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Detalhes do paciente |
| | | 400 Bad Request. Patient not found |

*Tabela 10 Serviço para pesquisar um paciente*

#### 4.5.2.4 Obter número de identificação paciente

Este serviço permite obter o número de identificação (cartão de cidadão, carta de condução, etc…) do paciente aquando o seu registo no hospital. *Ver tabela 11.*

- Endpoint: /id/{patientId}/identification/number/
- Descrição: obter número identificação.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Número de identificação |
| | | 400 Bad Request. Patient not found |

*Tabela 11 Serviço para obter o número de identificação do paciente*

#### 4.5.2.5 Obter tipo de identificação paciente

Este serviço permite obter o tipo de documento de identificação do paciente. *Ver tabela 12.*

- Endpoint: /id/{patientId}/identification/type/
- Descrição: obter o tipo de identificação.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Tipo de identificação |
| | | 400 Bad Request. Patient not found |

*Tabela 12 Serviço para obter o tipo de identificação do paciente*

#### 4.5.2.6 Obter o nome do paciente

Este serviço permite obter o nome do paciente. *Ver tabela 13.*

- Endpoint: /id/{patientId}/name/
- Descrição: obter número identificação.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Nome de paciente |
| | | 400 Bad Request. Patient not found |

*Tabela 13 Serviço para obter o nome do paciente*

#### 4.5.2.7 Obter os MCDTs do paciente

Este serviço permite obter a lista de MCDTs associada ao paciente. *Ver tabela 14.*

- Endpoint: /id/{patientId}/mcdt/get/
- Descrição: obter MCDTs do paciente.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Lista de MCDTs |
| | | 400 Bad Request. Patient not found |

*Tabela 14 Serviço para obter o número de identificação do paciente*

#### 4.5.2.8 Obter o género do paciente

Este serviço permite obter o género do paciente. *Ver tabela 15.*

- Endpoint: /id/{patientId}/gender
- Descrição: obter género.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Género do paciente |
| | | 400 Bad Request. Patient not found |

*Tabela 15 Serviço para obter o género do paciente*

#### 4.5.2.9 Obter a morada do paciente

Este serviço permite obter a morada do paciente. *Ver tabela 16.*

- Endpoint: /id/{patientId}/address/
- Descrição: obter morada.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Morada |
| | | 400 Bad Request. Patient not found |

*Tabela 16 Serviço para obter a morada do paciente*

#### 4.5.2.10 Obter número de telefone do paciente

Este serviço permite obter o número de telefone do paciente. *Ver tabela 17.*

- Endpoint: /id/{patientId}/phone/
- Descrição: obter número telefónico.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Número de telefone |
| | | 400 Bad Request. Patient not found |

*Tabela 17 Serviço para obter o número de telefone do paciente*

#### 4.5.2.11 Obter e-mail paciente

Este serviço permite obter o e-mail do paciente. *Ver tabela 18.*

- Endpoint: /id/{patientId}/email/
- Descrição: obter e-mail.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | E-mail do paciente |
| | | 400 Bad Request. Patient not found |

*Tabela 18 Serviço para obter o e-mail do paciente*

#### 4.5.2.12 Obter estado arquivado

Este serviço permite saber se um paciente solicitou o arquivamento dos seus dados pessoais. *Ver tabela 19.*

- Endpoint: /id/{patientId}/archived/status/
- Descrição: obter a informação sobre se os dados do paciente foram arquivados

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Estado do paciente face ao arquivamento |
| | | 400 Bad Request. Patient not found |

*Tabela 19 Serviço para obter o estado de arquivamento do paciente*

#### 4.5.2.13 Obter razão de arquivamento

Este serviço permite criar obter a razão pela qual um paciente se encontra no estado arquivado. *Ver tabela 20.*

- Endpoint: /id/{patientId}/archived/reason/
- Descrição: obter razão do arquivamento.

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id paciente | Motivo de arquivamento |
| | | 400 Bad Request. Patient not found |

*Tabela 20 Serviço para obter o motivo de um paciente estar arquivado*

#### 4.5.2.14 Inserir um processo ao registo do paciente

Este serviço permite adicionar um processo ao registo do paciente. *Ver tabela 21.*

- Endpoint: /id/{patientId}/process/set/{processId}
- Descrição: armazena um processo no registo do paciente

| Método | Parâmetros | Resposta |
|---|---|---|
| Post | Id do paciente<br>Id do processo | 200 Ok |
| | | 400 Bad Request |

*Tabela 21 Serviço para criar um processo*

#### 4.5.2.15 Modificar registo do paciente

Este serviço permite modificar dados do paciente. *Ver tabela 22.*

- Endpoint: /id/{patientId}/modify/
- Descrição: modifica os dados do paciente

| Método | Parâmetros | Resposta |
|---|---|---|
| Post | Id do paciente<br>JSON com os novos dados do paciente | 200 Ok |
| | | 400 Bad Request |

*Tabela 22 Serviço para modificar o registo do paciente*

### 4.5.3 Controlador MCDT

Este é controlador o responsável pela gestão dos MCDTs no sistema. Visto que no contexto clínico um MCDT não existe sem um processo, para retratar esta necessidade os *"endpoints"* para estes controladores começam por: {server} /api/process/{processId}/mcdt/

#### 4.5.3.1 Novo MCDT

Este serviço permite criar um MCDT no sistema. *Ver tabela 23.*

- Endpoint: /save/
- Descrição: cria e armazena um novo MCDT

| Método | Parâmetros | Resposta |
|---|---|---|
| Post | Id do processo | 200 Ok |
| | | 400 Bad Request |

*Tabela 23 Serviço para criar um MCDT*

#### 4.5.3.2 Listar todos os MCDTs do processo

Este serviço permite listar todos os MCDTs existentes no processo. *Ver tabela 24.*

- Endpoint: /all/
- Descrição: listar MCDTs do processo

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo | Lista de MCDTs |
| | | 400 Bad Request |

*Tabela 24 Serviço para listar MCDTs de um processo*

#### 4.5.3.3 Obter todos os detalhes de um MCDT

Este serviço permite obter todos os detalhes de um processo no sistema. *Ver tabela 25.*

Nota: Este serviço não permite obter as imagens médicas. É listado apenas o Id das mesmas em base de dados, Id esse que é utilizado por um outro serviço para a obtenção das mesmas.

- Endpoint: /{type}/{mcdtId}
- Descrição: obter os detalhes MCDT

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo<br>Tipo de MCDT | Detalhes do MCDT |
| | | 400 Bad Request |

*Tabela 25 Serviço para obter os detalhes de um MCDT*

#### 4.5.3.4 Inserir uma imagem médica.

Este serviço permite inserir uma imagem médica de sistema. *Ver tabela 26.*

- Endpoint: /{mcdtId}/GridFS/set/{mcdtType}/{filename:.+}
- Descrição: cria e armazena um novo processo

| Método | Parâmetros | Resposta |
|---|---|---|
| Post | Id do processo<br>Id do MCDT<br>Tipo de MCDT<br>Nome do ficheiro com extensão | 200 Ok. |
| | | 400 Bad Request. File not |
| | | 415 Unsupported media File |
| | | 425 Bad Request. MCDT not found |

*Tabela 26 Serviço para introduzir um MCDT no sistema*

#### 4.5.3.5 Obter imagem médica

Este serviço permite obter uma imagem médica do sistema. *Ver tabela 27.*

- Endpoint: /GridFS/get/{gridId}
- Descrição: obter imagem médica,

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo<br>Id no GridFS | Imagem médica |
| | | 400 Bad Request. No GridFS entry |

*Tabela 27 Serviço para obter uma imagem médica*

#### 4.5.3.6 Obter uma "thumbnail" da imagem médica.

Este serviço permite obter uma imagem JPEG da imagem original DICOM. Desta forma é possível oferecer uma pré-visualização da imagem sem comprometer o desempenho da aplicação. *Ver tabela 28.*

- Endpoint: /GridFS/get/{gridId}/thumbnail/
- Descrição: obter uma pré-visualização da imagem DICOM

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo<br>Id no Grid | Imagem médica |
| | | 400 Bad Request |

*Tabela 28 Serviço para obter uma pré-visualização imagem médica*

#### 4.5.3.7 Obter imagem para o disco para validação

Este serviço foi implementado com o propósito de validação da correta extração das imagens da base de dados e poder fazer testes ao serviço de conversão de formatos de imagem. Permite obter uma imagem que tenha sido carregada da base de dados para o disco do pc. *Ver tabela 29.*

- Endpoint: /image/{gridId}
- Descrição: obter uma imagem do disco

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo<br>Id no GridFS | Imagem |
| | | 400 Bad Request. Image not found |

*Tabela 29 Serviço para obter uma imagem armazenada no disco*

#### 4.5.3.8 Converter uma imagem DICOM em JPEG

Este serviço permite converter uma imagem armazenada no formato DICOM no formato JPEG para ser utilizada, por exemplo, para gerar a *"thumbnail"*. *Ver tabela 30.*

- Endpoint: /GridFS/convert/{gridId}
- Descrição: converte o formato DICOM em JPEG

| Método | Parâmetros | Resposta |
|---|---|---|
| Get | Id do processo<br>Id no GridFS | Imagem em JPEG |
| | | 400 Bad Request |

*Tabela 30 Serviço para converter uma imagem médica no formato JPEG*

### 4.5.4 Controlador das exceções

Este serviço permite que, na eventualidade de existir um erro na aplicação, como por exemplo, um pedido contendo um identificador inválido, seja enviado um código de erro seguido de uma mensagem personalizada.

## 4.6 Árvore de diretórios da plataforma

Nesta secção explico a árvore de diretórios da Figura 13, que representa a forma como foi divido o projeto na sua árvore de diretórios.

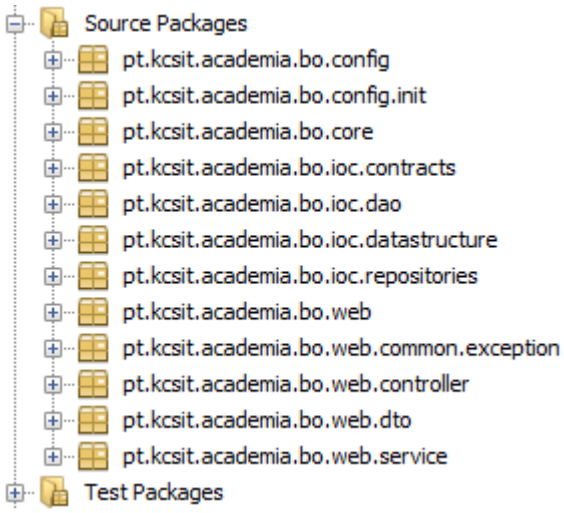


*Figura 13 Árvore de diretórios*

### 4.6.1 Package Config

Configura o Spring, o Mongo, o Repositório, o IoC e o GridFS.

### 4.6.2 Package Init

Configura o início do Spring. O Spring necessita de saber a rota, o *"server"* e como está mapeado.

### 4.6.3 Package Core

Package que agrupa código utilizado em diversos "*packages"*. Contém, entre outras, a classe responsável por converter imagens DICOM em JPEG, a classe que torna os url de pedidos insensíveis à capitalização de letras de modo a garantir que o URL não seja sensitivo, uma vez que serão disponibilizados para uso de terceiros, sendo o objetivo mitigar hipóteses de erro de integração e a classe que agrega várias configurações globais necessárias, bem como métodos transversais a várias classes.

### 4.6.4 Package IOC

Configura toda a lógica do "*backend"*. Contém as interfaces respeitantes ao padrão de desenho IoC referido na arquitetura.

#### 4.6.4.1 DAO

Concretização dos contratos.

#### 4.6.4.2 dataStructure

POJOs e enumerados.

#### 4.6.4.3 Repositories

Interface de acesso ao MongoDB com recurso a repositórios.

### 4.6.5 Package Web

Configura a forma como o backend expõe os seus dados.

#### 4.6.5.1 Package common exception

Atribui definições internas aos erros HTTP.

#### 4.6.5.2 Package controller

Endpoints onde o frontend se liga ao backend.

#### 4.6.5.3 Package dto

Padrão de desenho para transferir dados.

### 4.6.6 Package Test Packages

Teste às classes do sistema.

### 4.6.7 Package Dependencies

Dependências geradas pelo Maven.

### 4.6.8 Package DICOM

#### 4.6.8.1 GridFS

Configuração do GridFS.

#### 4.6.8.2 Format and File extension conversion

Utilitários para uma melhor performance na integração final do projecto.

# 5. Resultado e validação

Neste capítulo apresento resultados finais da aplicação como forma de validação da mesma.

## 5.1 Validação de resultados

Apresento agora exemplos de utilização dos serviços implementados.

Na Figura 14 podemos verificar a utilização de dois controladores. Está em uso o controlador dos processos que permite obter o número de MCDT's por tipologia *(ver 5.2.1.5)* visivel na label com a tipologia seguida do número de imagens que se encontra entre parenteses (ex: PET(2)), e o controlador de MCDT's que permite obter uma "*thumbnail" (ver 5.2.3.6).* A navegação neste carrossel de imagens é feita mediante os gestos da Myo de *"wave in"*, para deslocamento no sentido dos ponteiros do relógio, *"wave out"* para deslocamento no sentido contrário dos ponteiros do relógio e *"spread fingers"* para selecionar a categoria

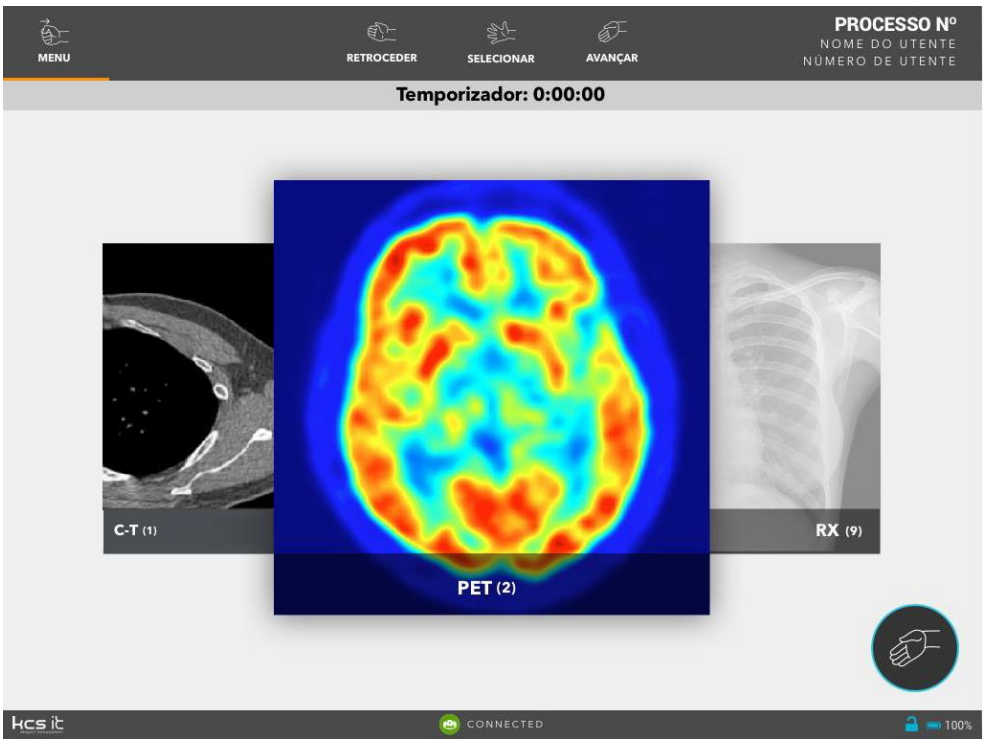


*Figura 14 Vista carrossel de MCDT*

Na Figura 15 observamos novamente o recurso ao controlador dos processos que permite obter o número de MCDTs por tipologia *(ver 5.2.1.5)* visível abaixo do nome da imagem com o indicador de posição face ao total de imagens (Ex: 1/9) , e o controlador de MCDTs que permite obter uma "*thumbnail" (ver 5.2.3.6)* de cada imagem e o controlador de MCDTs que permite obter todos os detalhes do mesmo, sendo que neste caso os detalhes incluem o nome da imagem, data do exame, as anotações de quem realizou o exame e o relatório do médico que solicitou o mesmo *(ver 5.2.3.3)*. A navegação neste painel é feita com os gestos combinados de mão fechada e para cima ou para baixo para se deslocar entre os *"thumbnails"*, o gesto combinado de palma da mão aberta e para cima ou para baixo para se deslocar entre as anotações, *"wave in"* para retroceder e por fim *"spread fingers"* para ampliar a imagem.

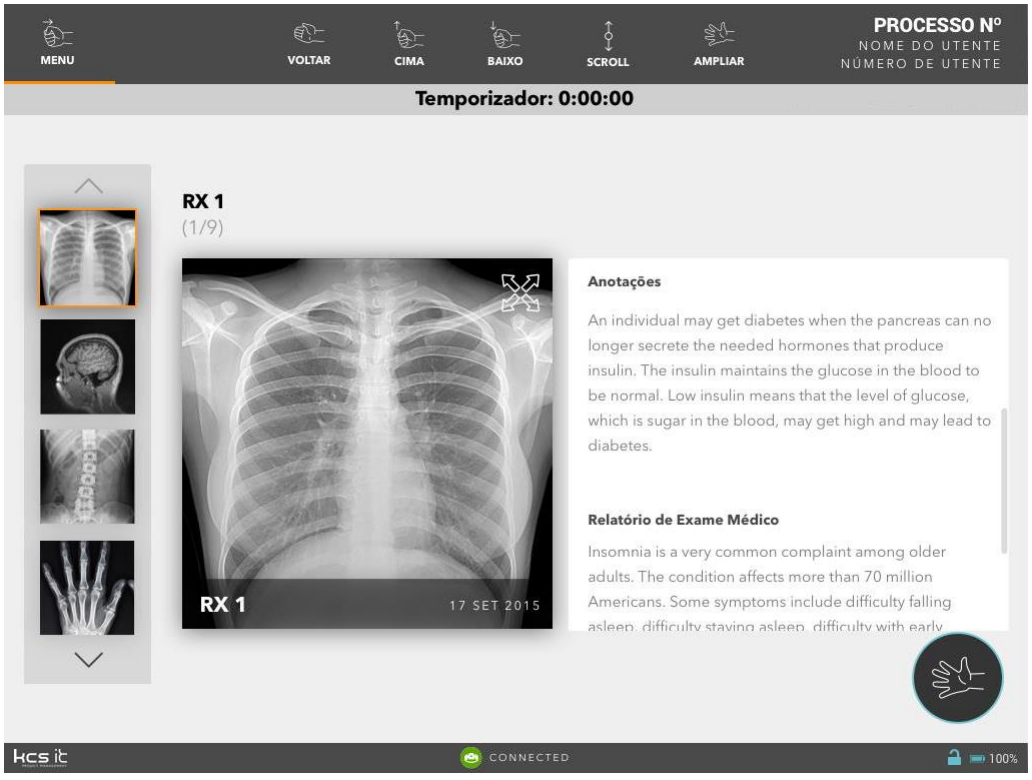


*Figura 15 Vista em detalhe dos MCDTs de uma tipologia*

Na Figura 16 está a ser utilizado o controlador de MCDTs que permite obter uma imagem médica armazenada no disco *(ver 5.2.3.7)*. A navegação neste painel é feita com o estalar dos dedos para alternar entre contraste e brilho, os gestos combinados de punho cerrado e rotação para a esquerda para aumentar o contraste ou o brilho, os gestos combinados punho cerrado e rotação para a direita para reduzir o contraste ou o brilho, os gestos combinados *"spread fingers"* e deslocação para a esquerda para ampliar a imagem, os gestos combinados *"spread fingers"* e deslocação para a direita para reduzir a imagem, os gestos combinados de palma da mão aberta e deslocação livre para movimentar a imagem dentro da caixa de visualização e finalmente *"wave in"* para retroceder.

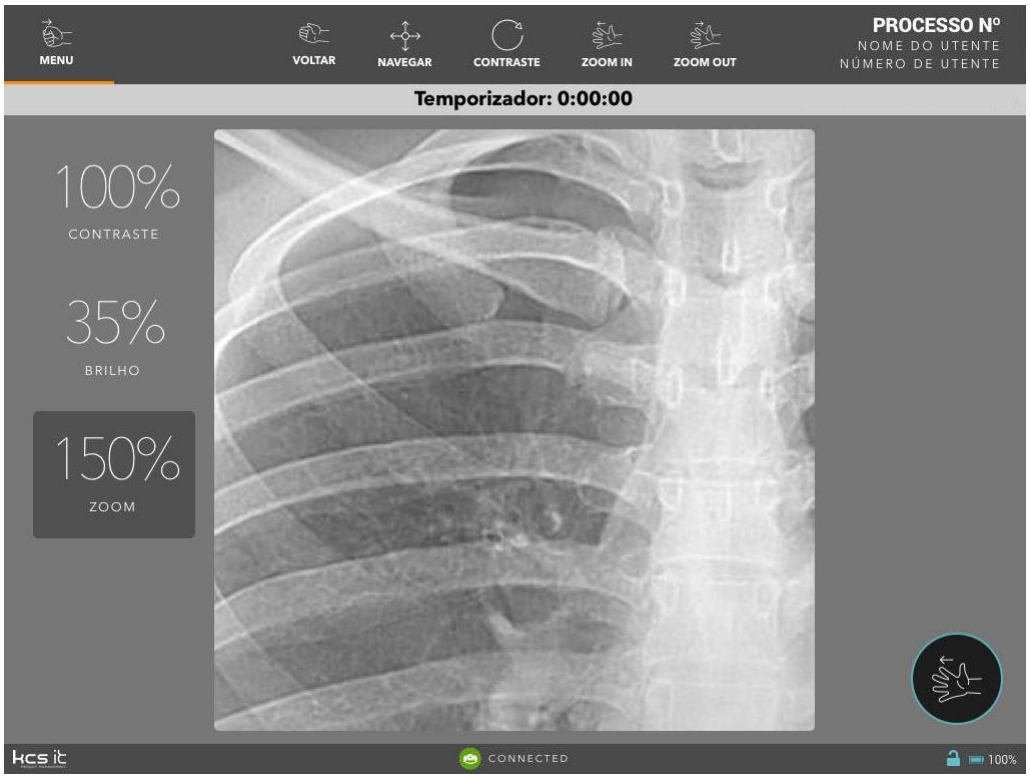


*Figura 16 Visualização de uma imagem MCDT*

## 5.2 Desenvolvimento orientado por teste

Test Driven Development permite que o código seja testado durante o desenvolvimento da aplicação. Esta abordagem adapta-se às metodologias ágeis enquadradas na visão da empresa em questão. Este modelo de testes revelou-se como uma estratégia adequada, pois permite que o código seja testado gradualmente segundo testes de validação previamente definidos.

No modelo TDD, é especificado um teste para cada especificação/requisito de cada funcionalidade, antes do código ser desenvolvido. Pelo facto de o teste ser desenvolvido antes do código, permite que o programador esteja focado nos requisitos em detrimento de estar focado somente nos aspetos de desenvolvimento. No primeiro teste, o código irá falhar já que este ainda não se encontra desenvolvido, só depois de esta condição ser cumprida é que o programador escreve o código para passar no teste previamente desenvolvido. Caso o código não passe no teste, procede-se à correção dos erros existentes. Após passar no teste desenvolvido, o código pode ser reescrito e novos métodos podem ser acrescentados e posteriormente testados. Este ciclo acrescenta valor ao produto, pois o tempo dedicado a testes diminui e a objetividade do código desenvolvido aumenta.

Assim, podem-se distinguir três fases de desenvolvimento da aplicação tendo por base os três momentos importantes de testes. Testes aos POJOs, testes aos DAOs e repositórios, e por último teste aos controladores REST responsáveis pela exposição dos dados.

## 5.3 Testes de carga do MongoDB

Os testes de carga foram desenhados tendo por base as seguintes premissas:

a) Três grupos de imagens, umas pequenas de aproximadamente 10 Mb, umas médias de aproximadamente 17 Mb e umas grandes de aproximadamente 40 Mb.
b) Imposto um limite de 15Gb máximo para uso por parte da base de dados.

Como métricas de avaliação usei o tempo necessário de arquivamento, tempo para obter 50 imagens aleatórias e número máximo de documentos que são possíveis de armazenar nos 15 Gb disponíveis. Os resultados obtidos excluem tempos de operações de suporte, como por exemplo o tempo para obter os identificadores dos documentos para análise da segunda métrica apresentada.

### 5.3.1 Imagens de aproximadamente 10 Mb

Começando com imagens de tamanho pequeno foi possível armazenar aproximadamente 1500 imagens dentro do espaço total de armazenamento. A figura 17 apresenta os resultados das medições dos tempos de escrita e leitura das imagens deste caso.


*Figura 17 Tempos de escrita e leitura para imagens de 10Mb*

Como se pode ver o processo de escrita foi bastante rápido, demorando, no caso de 1000 imagens, aproximadamente 3 minutos para a introdução por volta de 4 segundos para obtenção de uma imagem aleatória. Importa notar que, como as imagens estão dentro do limite das especificações de MongoDB, não existiu recurso ao GridFS, pelo que o espaço de 15Gb foi preenchido praticamente na totalidade com dados.

### 5.3.2 Imagens de aproximadamente 17 Mb

No caso de imagens de tamanho médio foi possível armazenar aproximadamente 625 imagens. A figura 18 apresenta os resultados das medições dos tempos de escrita e leitura das imagens deste caso.

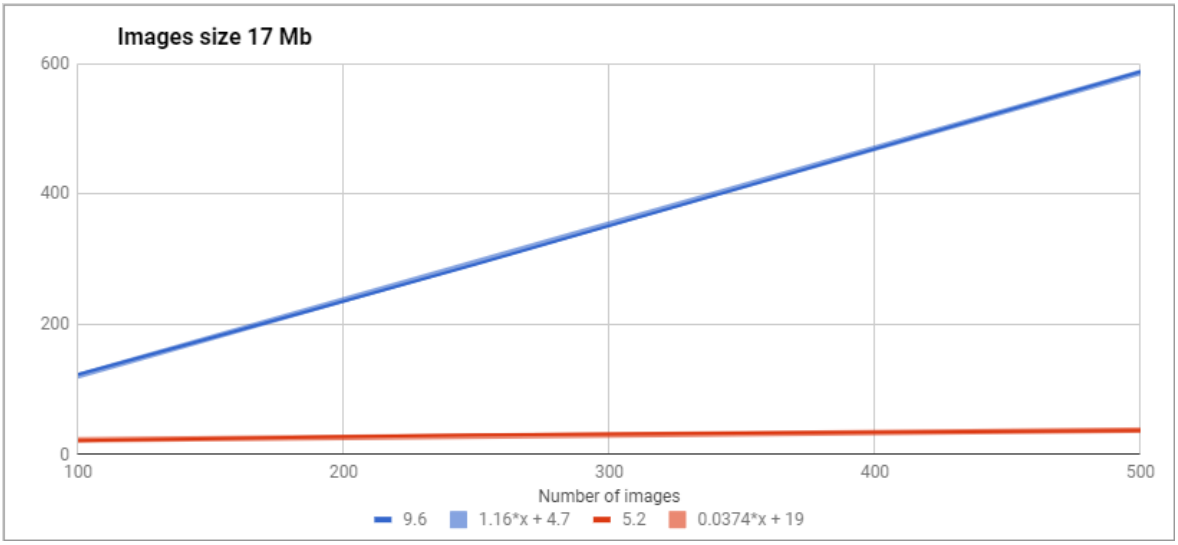


*Figura 18 Tempos de escrita e leitura para imagens de 17Mb*

Como mencionado no capítulo 4, o MongoDB apresenta uma restrição a nível do tamanho máximo dos documentos, sendo este tamanho 16Mb. Assim a escolha deste tamanho foi exatamente para provocar a necessidade da aplicação de utilizar o GridFS auxiliar no armazenamento dos documentos e assim ultrapassar esta limitação. Observa-se que o tempo de armazenamento aumentou ligeiramente, demorando, no caso de 500 imagens, aproximadamente 10 minutos para o seu armazenamento e 5 segundos para obter uma imagem aleatória. Contrariamente ao caso anterior, uma vez que foi necessário recorrer ao GridFS, do espaço total para armazenamento nem todo foi utilizado diretamente pelo MongoDB para o armazenamento de imagens, tendo este que ser partilhado com o GridFS.

### 5.3.3 Imagens de Aproximadamente 40 Mb

Para imagens de grande tamanho foi possível armazenar pelo menos 250 imagens no espaço definido. A figura 19 apresenta os resultados das medições dos tempos de escrita e leitura das imagens deste caso.

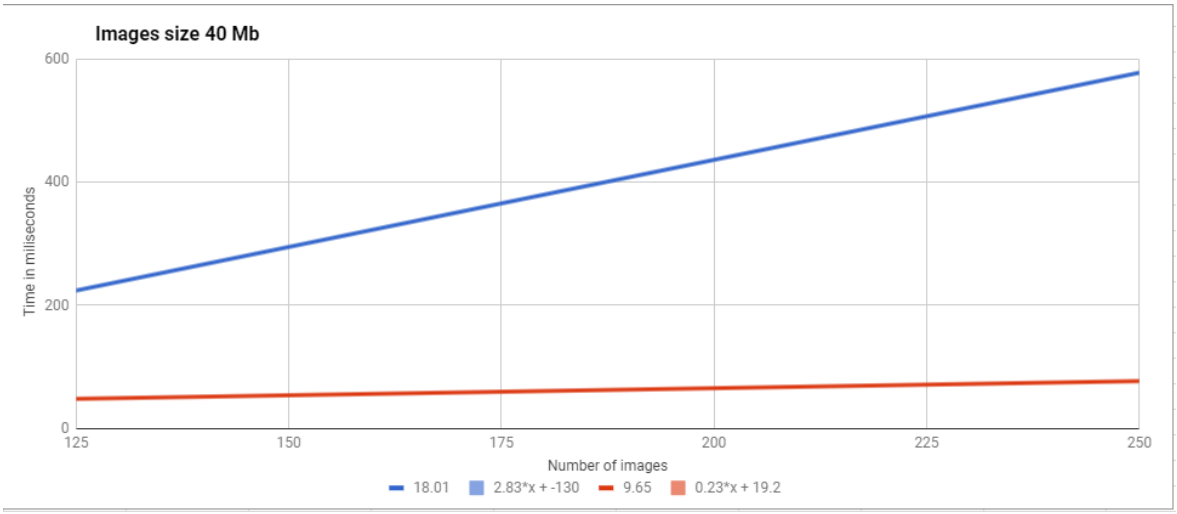


*Figura 19 Tempos de escrita e leitura para imagens de 40Mb*

À semelhança do caso anterior, o sistema teve de recorrer ao GridFS para suplantar o problema do limite de 16Mb por documento. Comparativamente ao caso das imagens de tamanho médio, estas imagens ocupam mais do dobro do espaço de armazenamento pelo que era expectável que isso se refletisse no tempo de armazenamento que duplicou. Como se pode ver no gráfico, no caso de 250 imagens demorou-se quase 10 minutos o armazenamento e 9 segundos a obter cada uma das 50 imagens aleatórias.

Em conclusão, embora o tempo de armazenamento seja significativo à medida que se aumenta o tamanho dos ficheiros, no ambiente cirúrgico esse tempo não é relevante, uma vez que o carregamento das imagens já terá sido feito aquando da preparação do procedimento cirúrgico, importando assim o tempo de resposta do servidor no que respeita à obtenção da imagem. Sendo assim, pegando no tempo de 3 segundos, este é um bom tempo já que se trata de um tempo de pesquisa e obtenção de uma imagem numa amostra de 250 imagens, algo que na realidade não acontece pois estas estariam agrupada por categoria de exame.

# 6. Conclusão

Esta dissertação teve como objetivo desenvolver a camada da lógica de negócios e de persistência de dados de uma aplicação "HandFree Operating Theatre" para imagem médica. O trabalho desenrolou-se em ambiente empresarial e o produto final apresenta-se como uma aplicação estruturada com base em Spring, desenvolvida com recurso a camadas de abstração, o que oferece uma garantia de qualidade inerente à utilização de contratos. A aplicação pode correr no servidor aplicacional GlassFish ou TomCat. Em termos de persistência, optou-se por utilizar uma base de dados NoSQL que propiciou melhores resultados em termos de desempenho nas pesquisas de dados não estruturados. A interoperabilidade com terceiros sistemas e a comunicação com a interface gráfica é suportada por serviços REST.

Em termos pessoais, o trabalho foi aliciante uma vez que me permitiu investigar ferramentas sobre as quais pouco sabia e que têm uma polivalência técnica no mercado atual. Também permitiu consolidar alguns conhecimentos adquiridos durante o percurso acadêmico. A autonomia das decisões que me foi consignada foi sem dúvida um dos fatores do sucesso final, visto que todas as decisões tiveram sempre de ser bem fundamentadas face à empresa.

Com vista a trabalho de futuro, foram criadas as classes necessárias ao desenvolvimento dos restantes serviços, como a gestão de funcionários e gestão de cirurgias.

# Bibliografia

[1]http://www.oxforddictionaries.com/definition/english/technology - acedido em Janeiro 2016
[2]http://www.icr.org/article/raymond-damadian-inventor-mri/ - acedido em Junho 2016
[3]Introdution to the DICOM Standard - https://link.springer.com/article/10.1007/s003300101100 - acedido em Janeiro 2016
[4]https://www.dgs.pt/directrizes-da-dgs/normas-e-circulares-normativas/norma-n-0242013-de-23122013.aspx - acedido em Junho 2016
[5]https://link.springer.com/chapter/10.1007/978-3-642-76566-7_18 - acedido em Janeiro 2016
[6]https://link.springer.com/article/10.1007/BF00999243 - acedido em Janeiro 2016
[7]http://dicom.nema.org/medical/dicom/current/output/pdf/part05.pdf - acedido em Janeiro 2016
[8]http://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC1307152&blobtype=pdf - acedido em Janeiro 2016
[9]https://msdn.microsoft.com/en-us/library/ms836794.aspx - acedido em Janeiro 2016
[10]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.809&rep=rep1&type=pdf - acedido em Janeiro 2016
[11]http://hibernate.org/orm/what-is-an-orm/ - acedido em Janeiro 2016
[12] http://hibernate.org/orm/ - acedido em Janeiro 2016
[13]http://www.seguetech.com/blog/2013/03/12/what-is-ajax-and-where-is-it-used-in-technology - acedido em Janeiro 2016
[14] Spring in Action Paperback – August 26, 2007 by Craig Walls (Author), Ryan Breidenbach (Author) - acedido em Janeiro 2016
[15]https://www.isostech.com/blogs/web-application-development/struts-2-vs-spring-mvc-battle-frameworks/ - acedido em Janeiro 2016
[16]Java SE 8 Programmer I - O guia para a sua certificação OCA - Editora Casa do Código - acedido em Janeiro 2016
[17]https://link.springer.com/chapter/10.1007/3-540-54132-2_48 - acedido em Janeiro 2016
[18] http://www.datastax.com/nosql-databases - acedido em Janeiro 2016
[19] "Top 5 considerations when evaluating NoSQL Database" - A MongoDB white paper - acedido em Janeiro 2016
[20] https://docs.mongodb.com/manual/reference/limits/ - acedido em Janeiro 2016