FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Indexação de Documentos Clínicos

João de Sá Balão Calisto Correia

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Gabriel David (Prof. Associado, FEUP)

Proponente: Francisco Correia (Engº, Glintt - Healthcare Solutions)

27 de Junho de 2016

Indexação de Documentos Clínicos

João de Sá Balão Calisto Correia

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente:

Vogal Externo:

Orientador:

_____

27 de Junho de 2016

Resumo

Hoje em dia o acesso a informação estruturada e bem organizada é cada vez mais essencial no planeamento, desenvolvimento e desempenho de uma empresa. No meio clínico o acesso a este tipo de informação influencia os serviços prestados pelas entidades médicas, uma vez que a informação encontra-se demasiado dispersa por diversas fontes. Torna-se assim claro que inovar o acesso à informação nestes serviços é essencial para melhorar a eficiência e qualidade dos mesmos.

Esta dissertação foi realizada em ambiente empresarial, na empresa de software da Saúde Glintt, e usou como caso de estudo a aplicação EResults, uma das aplicações de processo clínico eletrónico desenvolvidas na empresa, que contém documentos clínicos de diversas naturezas.

A solução apresentada tem três passos. O primeiro consiste numa migração dos metadados do EResults para o sistema CaboLabs, que representa a informação segundo a norma *openEHR*. O segundo passo assenta numa indexação, usando o software *Apache Solr*, tanto dos metadados em EHR como dos documentos anexos. O terceiro é a construção de uma interface de pesquisa especializada para o contexto clínico. Os formatos em causa podem ser PDF, Word, XML, JSON entre outros. Relativamente ao conteúdo da informação temos por exemplo: resultados analíticos laboratoriais, relatórios clínicos, diagnósticos codificados em ICD-9, notas clínicas dos médicos, requisições de exames, prescrições de medicamentos e informação demográfica de pacientes. As principais vantagens desta solução são a construção de um sistema de pesquisa integrado, para o qual é possível fazer convergir outras aplicações de processo clínico, e que evita sobrecarregar a base de dados principal.

Este projeto destina-se a profissionais de saúde e, por isso, disponibiliza uma forma centralizada e pesquisável de obter todos os dados clínicos de um doente/paciente assim como a informação documental correspondente.

Dito isto, a solução pretende causar um forte impacto no dia-a-dia dos profissionais de saúde assim como nos seus pacientes, tornando o acesso à informação clínica mais rápido e simples.

Abstract

Nowadays access to structured and well-organized information is increasingly essential in the planning, development and performance of a company. In a clinical environment, the access to this type of information influences the services provided by the medical entities, since the information is too dispersed by various sources. Thus it becomes clear that innovation access to information on these services is essential to improve efficiency and quality.

This work was carried out in a business environment, in the health software company Glintt, and used as a case study EResults, an implementation of the electronic health record applications developed in the company, containing clinical documents of various kinds.

The solution presented has three steps. The first is a migration of EResults metadata to CaboLabs system, that represents the information by *openEHR* standard. The second step is based on an index, using *Apache Solr* software, both the metadata in EHR as the accompanying documents. The third is the construction of a specialized search interface for the clinical context. The formats in question can be PDF, Word, XML, JSON and more. Regarding the content of the information we have for example: laboratory analytical results, clinical reports, diagnoses coded in ICD-9, clinical notes from doctors, requisitions, drug prescriptions and demographic information of patients. The main advantages of this solution is the construction of an integrated search system, to which it is possible to converge other clinical procedure applications, and which avoids overloading the master database.

This project is intended to health professionals and therefore provides a centralized, searchable way to get all the clinical data of a patient / patient and the corresponding documentary information.

That said, the solution aims to have a strong impact on the day-to-day health professionals as well as their patients, making access easier and faster clinical information.

Agradecimentos

Gostaria de agradecer ao Professor Gabriel David, por me ter aceite como seu orientando e por toda a disponibilidade e suporte que me deu ao longo deste percurso.

À Professora Carla Teixeira gostaria também de agradecer pela disponibilização do dicionário de termos médicos.

À Glintt, pela oportunidade dada de realizar o projeto em ambiente empresarial e, principalmente, aos Engenheiros Francisco Correia e Rui Pereira. O Eng.º Francisco pelo suporte e pelas diversas dicas que me fizeram olhar para o projeto de outro ângulo. O Eng.º Rui pelo apoio no estudo da base de dados da Glintt e do serviço de visualização de documentos.

Por fim, e não menos importante, gostava de agradecer ao desenvolvedor do *EHRServer*, Pablo Pazos, pelo apoio no estudo do EHR, na construção dos arquétipos, *templates* e xml necessários para o desenvolvimento do projeto e pela sua disponibilidade na resposta às minhas frequentes perguntas.

Como não podia deixar de ser, tenho que agradecer também à minha família pelo apoio emocional constante que foi fundamental para concluir este projeto.

Obrigado,
João Correia

"*Só fazemos melhor aquilo que repetidamente insistimos em melhorar. A busca da excelência não deve ser um objetivo e sim um hábito*."

— Aristóteles

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	Application Programming Interface
BD	Base de Dados
EER	Extended Entity Relationship
EHR	Electronic Health Record
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
ICD-9-CM	International Classification of Diseases, 9th Revision, Clinical Modification
JSON	JavaScript Object Notation
MeSH	Medical Subject Headings
MySQL	My Structured Query Language
PDF	Portable Document Format
REST	Representational State Transfer
RI	Recuperação de Informação
SNOMED CT	Systematized Nomenclature of Medicine – Clinical Terms
UMLS	Unified Medical Language System
XML	eXtensible Markup Language

Capítulo 1

Introdução

Neste capítulo introdutório apresenta-se o contexto da dissertação, a motivação e os objetivos mostrando também as razões pelas quais o tema em causa é importante e que influência tem no dia-a-dia dos seus utilizadores. É também apresentada uma visão global sobre a estrutura da mesma.

1.1 Contexto/Enquadramento

Com o passar dos anos, constata-se que a sociedade em que vivemos é baseada cada vez mais na informação. A quantidade de informação aumenta a um ritmo crescente, o que torna a sua gestão bastante difícil. Assim, para que a sociedade se desenvolva, torna-se progressivamente necessária a existência de soluções e mecanismos inovadores que possibilitem o acesso à informação de forma rápida e eficaz. Para fazer face a estes factos, surgem assim métodos de recuperação e de indexação que tornam possível o acesso à informação com as características referidas anteriormente.

Assim sendo, é possível assumir que esta dissertação se encontra nas áreas das ciências da informação e das tecnologias de bases de dados, na medida em que aborda a importância do acesso e recuperação da informação em meios clínicos.

Esta dissertação foi desenvolvida no âmbito empresarial da Glintt - Healthcare Solutions, a qual é focada no ramo da saúde e onde, a nível nacional, é líder destacada neste segmento do mercado. A sua sede está situada na cidade do Porto e é constituída por cerca de 300 colaboradores. A Glintt trabalha com uma grande quantidade de informação clínica em formatos diversos, desde documentos PDF, Word e JSON a informação distribuída pelas tabelas das bases de dados, sendo por isso necessário uma boa organização e tratamento da mesma.

1.2 Motivação e Objetivos

No meio clínico a quantidade de informação é elevada, o que torna difícil o acesso à mesma pelas entidades médicas. Além disto, outros fatores alarmantes são que a mesma quantidade de informação encontra-se dispersa por várias fontes e diferentes formatos e, por vezes, não se encontra estruturada o que faz com que as entidades médicas percam demasiado tempo na pesquisa dos dados clínicos de um doente. Relativamente aos diferentes formatos e fontes em que a informação se encontra, tanto se pode falar de documentos como de campos de bases de dados em texto livre, sendo exemplos destes: resultados analíticos laboratoriais, relatórios clínicos, diagnósticos codificados em ICD-9, notas clínicas dos médicos, requisições de exames, prescrições de medicamentos e informação demográfica de pacientes.

Devido a esta diversidade na origem e tipos dos dados, há um atraso no acesso à informação por parte dos médicos, há falta de informação e, por vezes, esta nem chega corretamente às mãos das entidades clínicas. Assim sendo, está clara a necessidade de organizar e tratar esta informação. Porém fazê-lo não é tão trivial como possa parecer pois, apesar da crescente atualização dos documentos clínicos para suporte digitais, estes muitas vezes resultam apenas numa digitalização do documento original e, por isso, surgem diversos problemas que levantam algumas questões: Como indexar imagens? Como pesquisar algo sobre elas?

Outro aspeto é também a forma de apresentar os resultados da pesquisa, atendendo a que as fontes podem estar em documentos autónomos ou como valores em campos de bases de dados. Estes últimos pontos realçam a grande variedade de dados e, com isso, alguns dos problemas que é necessário resolver nesta área.

O método escolhido foi o de construir um sistema de pesquisa, baseado em EHR, para o qual é possível converter os metadados dos vários sistemas com informação clínica. Esses dados e os respetivos documentos anexos são em seguida indexados. O conjunto é pesquisável a partir de uma interface de pesquisa por palavras, com algumas extensões adequadas ao contexto clínico. Em suma, o acesso à informação torna-se mais fácil e de qualidade, uma vez que toda a informação se encontra num só ponto.

1.3 Estrutura do Documento

O presente documento está dividido em seis capítulos:

- **Capítulo 1 – Introdução**: é um capítulo onde é apresentado o contexto e motivação do tema em causa;
- **Capítulo 2 – Revisão Bibliográfica**: é onde se descreve o estudo do estado da arte e está dividido em quatro subsecções: Recuperação de Informação, Indexação, Classificação e EHR;
- **Capítulo 3 – Solução**: neste capítulo é possível observar a arquitetura da solução com uma breve explicação de cada um dos passos;

- **Capítulo 4 – Implementação**: é onde se descreve toda a etapa de implementação da solução, relatando todas as dificuldades encontradas e as decisões tomadas ao longo do processo;
- **Capítulo 5 – Simulação e Resultados**: é o capítulo no qual se descrevem os testes realizados e os seus resultados;
- **Capítulo 6 – Conclusões e Trabalhos futuros**: é onde se apresentam conclusões relativas a este projeto e se avalia a satisfação dos objetivos. Remete também para possíveis implementações/etapas futuras da solução em causa.

Capítulo 2

Revisão Bibliográfica

As secções seguintes descrevem o estado da arte relativa aos processos de recuperação de informação (RI) e indexação com uma breve descrição das tecnologias mais recentes e mais utilizadas na área. Após este tópico é apresentada uma secção sobre a classificação, a qual é seguida de algumas normas mais relacionadas com a área médica, que podem ajudar no processo de indexação e pesquisa de informação clínica, o *Unified Medical Language System* (UMLS) e o *Electronic Health Record* (EHR).

2.1 Introdução

Desde há muito tempo atrás que a comunidade tem vindo a armazenar grandes quantidades de informação para que esta possa ser consultada por diversas pessoas de geração em geração. Face a esta acumulação de informação recolher/extrair informação com significado/utilidade começou a tornar-se bastante complicado. Entre 1948 e 1950, Calvin Mooers [7] cria o termo "Recuperação de Informação" pressionado pela necessidade de gerir a explosão da informação na literatura científica na segunda metade do século XX.

Nos anos seguintes foram feitas várias experiências na área e durante os anos 70 os sistemas de recuperação de informação começaram a tornar-se reais essencialmente devido ao aparecimento de processadores de texto [8].

Em 1980, com a diminuição constante do preço do espaço em disco a informação disponível em suporte digital crescia rapidamente. Nesta década começaram a estar disponíveis textos completos e não apenas resumos e índices.

Com o passar dos anos e na sequência do aparecimento de diversos dispositivos para armazenamento de informação digital, a informação continuou a crescer, forçando os sistemas de

recuperação de informação a desenvolverem-se da mesma forma, com o objetivo de acompanhar o aumento de informação.

Hoje em dia, o processo de recuperação de informação está bastante disseminado na sociedade, sendo que o desenvolvimento de ferramentas de pesquisa veio ajudar neste processo. O conceito de recuperação de informação surgiu então da forte necessidade de extrair informação com valor das diversas fontes, organizando e facilitando o acesso aos conteúdos informacionais através dos processos de indexação e de descrição.

2.2 Recuperação de Informação

A RI pode ser definida como a atividade de obtenção de informação, devido a uma necessidade, a partir de uma coleção de recursos de informação. Segundo Christopher D. Manning a RI é descrita da seguinte forma:

"*Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)*.", [9]

De uma forma geral, os sistemas RI devem revelar ao utilizador resultados interessantes e com valor para o mesmo, recorrendo, para isso, a um processo de ordenação da informação encontrada após a pesquisa. Este processo é realizado tendo em conta a pesquisa feita pelo utilizador, ou seja, resultados mais próximos dos termos pesquisados deverão aparecer nas primeiras posições, enquanto que resultados mais distantes deverão aparecer nas últimas posições da lista de resultados ordenada.

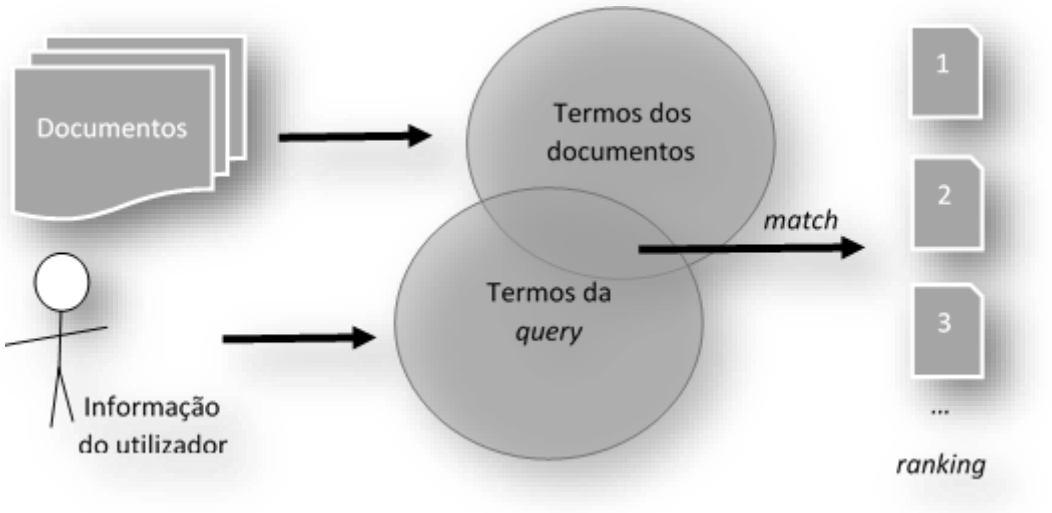


Figura 1 - Sistema RI

Na imagem anterior é ilustrado o funcionamento de um sistema RI. Todavia, a estratégia apresentada na **figura 1** não é tão trivial como possa parecer, uma vez que um sistema RI enfrenta vários problemas, entre os quais se destaca o facto de não se saber quais os documentos relevantes para determinada pergunta do utilizador. Esta tarefa é normalmente realizada com base em alguma heurística, que decide quais os documentos com maior relevância a serem recuperados e ordena-os a partir de critérios estabelecidos [10].

Para a resolução deste problema foram propostos vários modelos para recuperação de informação. Estes estão divididos em modelos clássicos e modelos estruturados. Os modelos clássicos [7] são baseados em termos de indexação (palavras-chave), ou seja, cada documento é representado por um conjunto desses termos. Já os modelos estruturados podem especificar alguma informação sobre a estrutura do texto, por exemplo, proximidade de palavras. A descrição de cada um dos modelos referidos encontra-se nos tópicos seguintes.

Em relação aos sistemas RI apenas resta dizer que têm um papel muito importante ou até fundamental na satisfação das necessidades do utilizador e por isso, adaptar um sistema deste tipo num meio clínico tem como objetivo ajudar, neste caso, as entidades médicas, de modo a que estas melhorem o seu desempenho no dia-a-dia.

2.2.1 Modelos Clássicos

2.2.1.1 Modelo Booleano

É um modelo clássico baseado na lógica booleana, isto é, os termos pretendidos são combinados com os operadores AND, OR, e NOT de forma a fornecer ao utilizador informação mais específica, por exemplo: “presidente AND lincoln”. Oferece algumas vantagens como o

facto de os resultados serem previsíveis e fáceis de explicar e, como é um conceito intuitivo, os utilizadores sentem-se no controlo do sistema. No entanto são também visíveis algumas desvantagens dentro das quais se destaca o facto de não providenciar uma ordenação dos documentos encontrados em resposta à pergunta do utilizador [11].

2.2.1.2 Modelo Vetorial

No modelo vetorial os documentos e as *queries* são representados por vetores de termos que têm um peso associado, por exemplo: (palavra1, peso1), (palavra2, peso2), … (palavra n, peso n). Este peso não é binário de forma a não limitar os resultados da pesquisa (como é feito no modelo booleano) e é obtido através da quantidade de vezes que um termo aparece no próprio documento e nos outros documentos da coleção. Os pesos da *querie* e do documento são calculados através do peso dos respetivos vetores. Uma vez efetuado este cálculo parte-se para o grau de similaridade que pode ser obtido com base no ângulo entre os dois vetores referidos (*query* e documento) [12].

A similaridade permite a saída de resultados ordenados, mas com esta abordagem um documento considerado relevante pode não conter os termos pesquisados pelo utilizador. Isto acontece essencialmente devido ao facto de o modelo assumir que os termos são independentes. Assim sendo, pode-se dizer que o modelo encontra apenas resultados parciais face a uma pesquisa.

2.2.1.3 Modelo Probabilístico

O modelo probabilístico foi proposto por S.E. Robertson e K. Sparck Jones [13], e, de uma forma geral, consiste em estimar a probabilidade de um documento d_i ser relevante para uma *query* q. Este modelo assume que a probabilidade de relevância depende apenas da *query* e do documento. O resultado ideal de uma pesquisa seria um conjunto que contivesse todos e apenas os documentos relevantes para o utilizador. Este modelo ordena os documentos na ordem decrescente da probabilidade de relevância da informação.

Existem algumas limitações, entre as quais se destaca a ausência de atribuição de pesos a cada termo. Além disto, é necessário fazer uma estimativa inicial de forma a refinar o conjunto da resposta ideal sendo que, para cada termo, não é considerada a quantidade de vezes que este aparece num documento.

2.2.2 Modelos Estruturados

2.2.2.1 Modelo da Proximidade de Nós

O modelo da proximidade de nós visa a recuperação de documentos através de uma estrutura hierárquica de índices e, por isso, permite que várias estruturas possam ser definidas no mesmo texto, cada uma sendo uma hierarquia restrita, mas permitindo a sobreposição entre áreas delimitadas por hierarquias diferentes. Uma *query* pode relacionar diferentes hierarquias, mas retorna um subconjunto de nós de apenas uma delas. Cada nó tem associado um segmento que corresponde à área de texto a que este está associado. O segmento de um nó inclui os dos seus descendentes.

De uma forma geral, o modelo em causa possibilita a definição de estruturas de indexação hierárquicas e independentes sobre um mesmo documento [14].

2.2.2.2 Modelo de Listas Não Sobrepostas

O modelo de listas não sobrepostas foi proposto por Burkowski [10] em 1992 e consiste na divisão de todo o texto de um documento em regiões não sobrepostas, que se juntam numa lista. Existem diversas formas de dividir o texto em regiões não sobrepostas que levam à geração de listas múltiplas:

- Lista para capítulos;
- Lista para secções;
- Lista para subsecções.

Regiões de texto de listas distintas podem sobrepor-se.

Relativamente à implementação, o modelo utiliza um ficheiro invertido que combina as palavras-chave e as regiões de texto. Para cada entrada neste ficheiro invertido é associada uma lista de regiões de texto. Listas de regiões de texto podem ser juntadas com listas de palavras-chave.

2.3 Indexação

Como referido no capítulo anterior, os sistemas RI organizam e viabilizam o acesso à informação através da descrição de documentos e da operação do tratamento documental, a indexação. Mas afinal, em que consiste a indexação?

A indexação tenta descrever e caracterizar um documento com o auxílio de representações dos conceitos contidos nesses documentos, ou seja, em transcrever para linguagens de indexação

os conceitos depois de terem sido extraídos dos documentos por meio de uma análise dos mesmos. Este processo de indexar segue assim três etapas:

1. Compreensão do documento;
2. Seleção dos conceitos para a pesquisa;
3. Tradução destes conceitos nas linguagens de indexação.

Na compreensão do documento deve ser feita uma análise global do mesmo, isto é, ter em consideração as partes mais úteis como fonte de informação do tema/assunto. As partes referidas podem ser: título, resumo, índice, prefácio, introdução, entre outros. Nesta etapa não é, por isso, necessário ler o documento na íntegra.

Relativamente à segunda etapa, a seleção dos conceitos para a pesquisa, deve ser feita de forma exaustiva e específica, ou seja, todos os assuntos de que trata o documento devem ser registados e nunca um conceito deve ser traduzido.

No terceiro passo, é feita a tradução destes conceitos nas linguagens de indexação, isto é, transformar os conceitos selecionados em termos ou símbolos autorizados para representá-los no sistema. Após a execução destas etapas dá-se por terminado o processo de indexação.

Neste processo de indexar é ainda importante realçar dois conceitos: *stemming* e *stopwords*.

2.3.1.1 Stemming

No processo de indexar, na maioria das vezes, existem palavras que, apesar de pertencerem à mesma família, são indexadas separadamente. Como é possível observar, nestes casos estão a ser criados vários índices que possivelmente representarão o mesmo conteúdo. Face a este e outros casos semelhantes surge o *stemming* [15] que consiste em reduzir as palavras à sua raiz morfológica. A raiz morfológica de uma palavra é o conjunto de caracteres que está presente em todas as suas derivações.

Pode parecer um processo simples, mas, se o fosse, apenas existiria uma única implementação dele, o que não é o caso. O *stemming* por vezes falha em reduzir as palavras com o mesmo significado para a mesma raiz (*understemming*) e por outras vezes falha em manter palavras com significados diferentes separadas (*overstemming*). Apesar disto, um algoritmo de *stemming* bem implementado é capaz de melhorar significativamente a qualidade da solução.

2.3.1.2 Stopwords

Em todos os textos/documentos existem palavras que, por aparecerem em muitos documentos, não acrescentam valor discriminatório relativo ao conteúdo de um documento como o “e”, “de”, etc. Além disso, estas palavras são raramente pesquisadas pelo utilizador uma vez que não têm valor para o documento. As palavras referidas são conhecidas como *stopwords*, pois

devido à sua falta de valor contextual são eliminadas do processo de indexação. Este processo de eliminação referido permite um aumento de qualidade nos resultados.

Nos tópicos seguintes são apresentadas algumas das ferramentas mais usadas nos dias de hoje para indexar e pesquisar conteúdos de uma forma mais rápida e fácil. Essas ferramentas são: *Apache Lucene*, *Apache Solr*, *ElasticSearch* e *SphinxSearch*.

2.3.2 Apache Lucene

O *Apache Lucene* foi desenvolvido em 1997-8 por *Doug Cutting*. Inicialmente denominado apenas de *Lucene*, foi disponibilizado gratuitamente no *SourceForge* em 2000. Mais tarde, em 2001, juntou-se à *Apache Software Foundation's* onde passou a chamar-se *Apache Lucene*. Em 2005, já era um projeto de alto nível na *Apache Software Foundation's*. Segundo os autores Hatcher e McCandless o *Lucene* é descrito da seguinte forma:

"*Lucene is a high-performance, scalable information retrieval (IR) library. IR refers to the process of searching for documents, information within documents, or metadata about documents. Lucene lets you add searching capabilities to your applications. It's a mature, free, open source project implemented in Java, and a project in the Apache Software Foundation, licensed under the liberal Apache Software License. As such, Lucene is currently, and has been for quite a few years, the most popular free IR library.*", McCandless, et al. [16]

Por outras palavras, *Lucene* é uma ferramenta que permite indexar e realizar pesquisa sobre os elementos indexados. Um índice é constituído por uma lista de documentos (ou outra fonte de informação), os quais são compostos por uma sequência de campos e em que cada um destes consiste numa sequência nomeada de termos, ou seja, por nome do campo e termo, por exemplo: ("título", "introduction to Lucene"). O termo pode conter uma ou várias palavras com valor no contexto do documento a indexar. O *Lucene* segue um sistema de indexação designado por *inverted índex*, ou em português, índice invertido, pois em vez de, para cada documento, guardar os termos que nele existem, guarda, para cada termo, os documentos onde estes se encontram.

Para determinar a relevância de um documento perante a pergunta do utilizador, o *Apache Lucene* utiliza uma combinação entre o modelo booleano e o modelo vetorial.

A **figura 2** apresenta a arquitetura do *Apache Lucene* em 2004, apesar de continuar bastante atual.

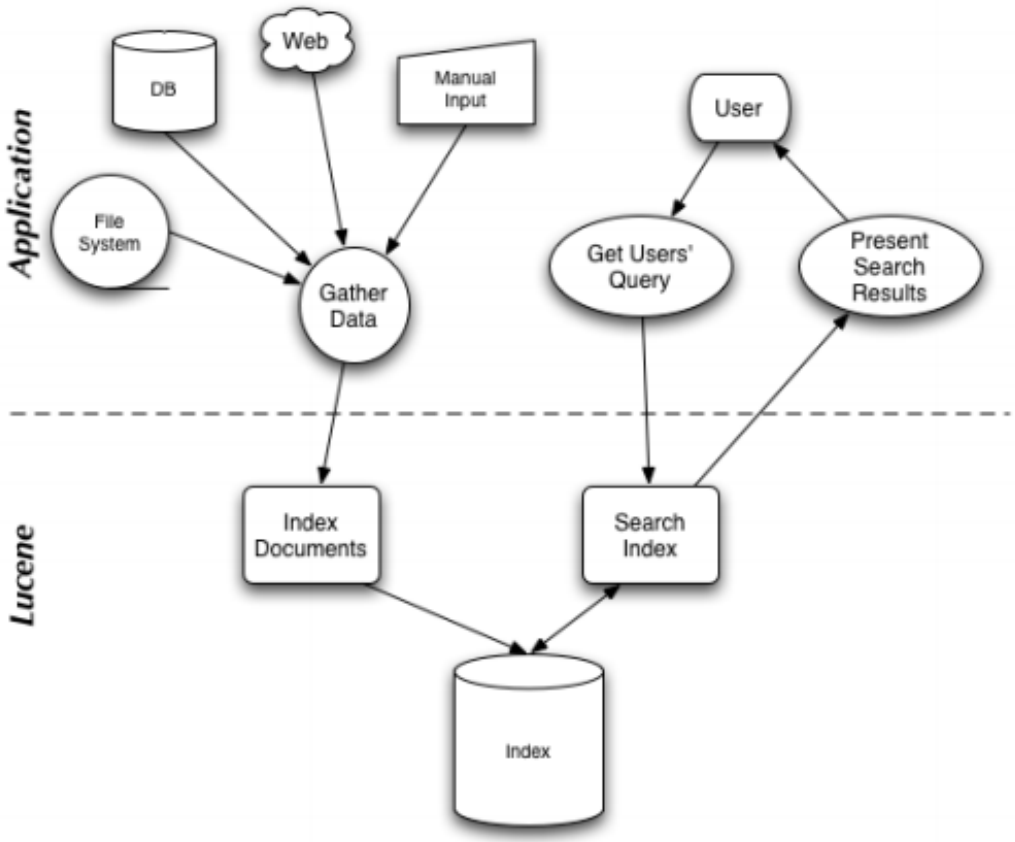


Figura 2 - Arquitetura do Apache Lucene [4]

- **Gather Data:** nesta etapa são recolhidos os conteúdos dos diferentes documentos.
- **Index Documents:** é feita uma análise do documento e, após esta, inicia-se o processo de indexação.
- **Index:** base de dados de índices.
- **Get Users' Query:** uma vez feito o pedido de pesquisa do utilizador, a aplicação constrói a *query* com base no texto de pesquisa a fim de ser utilizada na interrogação às bases de dados de índices.
- **Search Index:** é a etapa onde se realiza a pesquisa por índices de acordo com a *query* construída a partir dos dados de pesquisa introduzidos pelo utilizador.
- **Present Search Results:** apresenta o resultado da pesquisa ao utilizador.

Com isto, termina a análise desta ferramenta afirmando que é uma ferramenta poderosa de indexação e pesquisa, que permite rapidamente e eficazmente criar índices e realizar pesquisa sobre os mesmos.

2.3.3 Apache Solr

O *Apache Solr* surgiu em 2006 e consiste numa ferramenta construída a partir do *Apache Lucene*. Devido à sua origem a partir do *Apache Lucene*, implementa muitas das suas funcionalidades e acrescenta muitas outras. O *Apache Solr* apresenta assim as seguintes características [17]:

- Altamente escalável e tolerante a falhas;
- Otimizado para alto volume de tráfego;
- Fornece APIs baseadas em REST o que lhe permite ser integrado em praticamente qualquer linguagem de programação (XML, HTTP e JSON APIs);
- Fácil extensão através de *plugins*;
- Quase em tempo real, isto é, os documentos estão disponíveis para pesquisa quase imediatamente a seguir à sua indexação;
- Integra o *Apache Tika* [18], por isso, aceita para indexação documentos com diversas extensões como PDF, Word, XML, etc.

Além destas características pode ser ainda comentado o facto de este possuir uma boa documentação [19] que, juntamente com o excelente suporte que recebe, o torna uma ferramenta bastante utilizada hoje em dia.

2.3.4 ElasticSearch

Em semelhança ao *Apache Solr*, o *ElasticSearch* foi desenvolvido com base no *Apache Lucene* ao qual foram adicionadas algumas melhorias. Foi desenvolvido por Shay Banon e teve o primeiro lançamento em 2010.

O *ElasticSearch* permite facilmente aceder às funcionalidades do *Lucene* para indexação e pesquisa [20]. No entanto, proporciona um novo nível de abstração de análise em tempo real. Outro nível de abstração é a forma como podem ser organizados os documentos: múltiplos índices podem ser pesquisados em simultâneo ou separadamente, e é possível colocar diferentes tipos de documentos dentro de cada índice.

Por fim o *ElasticSearch*, como o próprio nome sugere, é elástico, ou seja, é possível acrescentar gradualmente servidores de forma a aumentar a capacidade e tolerância a falhas. De forma semelhante, é possível remover estes gradualmente de forma a reduzir os custos.

2.3.5 Sphinx Search

O *Sphinx Search* começou a ser desenvolvido em 2001 e consiste num servidor de pesquisa *full-text*, escrito em C++ que funciona com diversos sistemas operativos. Esta ferramenta permite indexar e pesquisar informação guardada em bases de dados SQL e NoSQL, ou apenas em ficheiros.

Na figura seguinte é apresentada a arquitetura da ferramenta.

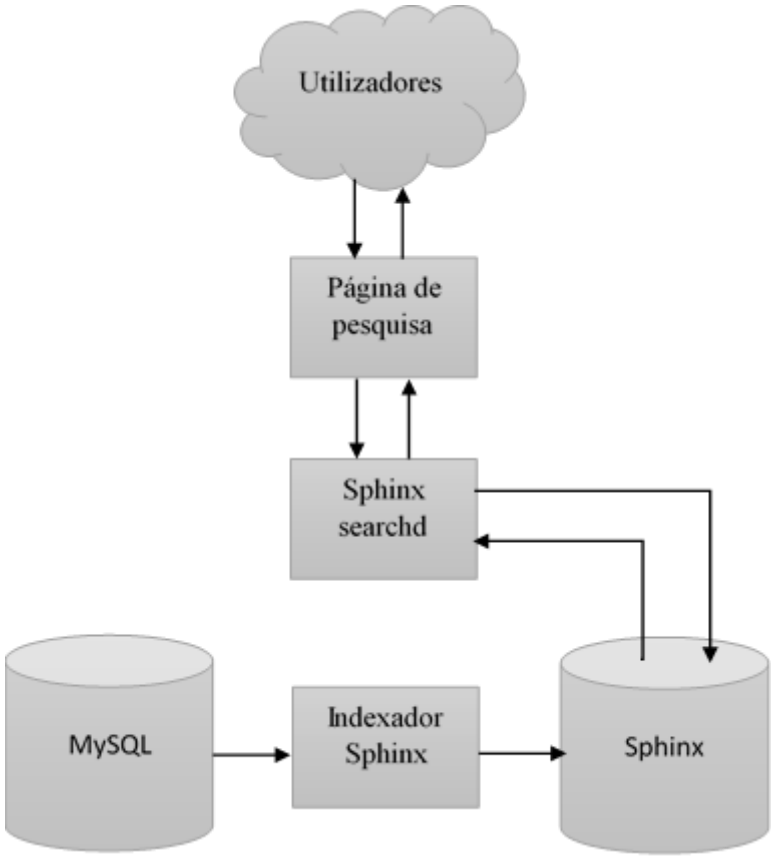


Figura 3 - Arquitetura do *Sphinx Search* [2]

Descrevendo com mais pormenor o que se observa na imagem, o **searchd** é responsável por receber pedidos do cliente e executar pesquisa comparando com os índices no **Sphinx**. Já o **Indexador Sphinx** é responsável por recolher a informação do **MySQL** (ou outra fonte suportada) e indexá-la.

Perante esta análise é ainda possível realçar algumas características [2] desta ferramenta:

- Indexação em tempo real;

- Indexação de base de dados SQL (MySQL, PostgreSQL, Oracle, SQLite, etc);
- Indexação de base de dados NoSQL;
- Vem com o *SphinxAPI* que é uma biblioteca nativa disponível para Java, PHP, Python, Perl, C entre outras linguagens;
- Suporta pesquisas sintaticamente complexas;
- Analisa proximidade entre palavras e ordena em rankings.

2.3.6 Análise Tecnológica

Nos tópicos anteriores foram descritas algumas das ferramentas usadas hoje em dia para efeitos de indexação e recuperação de informação. Não esquecendo que estamos perante uma indexação de documentos clínicos, é importante ter em conta quais das tecnologias melhor respondem às necessidades em causa. Para tal, e relembrando o que está em causa, é necessário recolher e extrair informação de diferentes fontes e com diferentes formatos. As fontes possíveis são: *webservices*, sistemas de ficheiros, bases de dados relacionais e *document-oriented*. Relativamente aos formatos podem variar desde ficheiros de texto simples a PDF, Word, JSON, XML entre outros. Face a estes dados é possível partir para uma análise comparativa entre as ferramentas em estudo.

Relativamente ao *Apache Lucene*, é possível afirmar que não trará grandes vantagens na sua utilização direta no âmbito em questão, pois os seus "sucessores", *Solr* e *ElasticSearch* trazem uma abrangência muito maior relativamente, não só, às linguagens que disponibilizam, mas também aos diferentes formatos nos quais pode ser realizada a indexação e extraída a informação. Perante esta situação, na tabela seguinte é feita uma análise comparativa das ferramentas *Apache Solr*, *ElasticSearch* e *Sphinx Search*.

	Solr	**ElasticSearch**	**Sphinx Search**
API			
Bibliotecas	Ruby, Rails, PHP, Java, Python, Perl, C#, .Net, JavaScript	Java, Groovy, PHP, Ruby, Perl, Python, .NET, JavaScript	C++, Java, Perl, PHP, Python, Ruby
Indexação			
Importação de dados	JDBC, CSV, XML, Tika [18], URL, Flat File	Amazon SQS, CouchDB, Dropbox, DynamoDB, FileSystem, Git, GitHub, Hazelcast, JDBC, JMS, Kafka, LDAP, MongoDB, neo4j, OAI, RabbitMQ, Redis, RSS, Sofa, Solr, St9, Subversion, Twitter, Wikipedia	MySQL, PostgreSQL, MSSQL, ODBC source, XML pipe
Tempo Real	✓	✓	✓
Licença			
	Apache License 2.0	Apache License 2.0	GPLv2 e comercial

Tabela 1 - Comparação entre Solr, ElasticSearch e Sphinx Search

Como é possível observar na tabela anterior, e começando por analisar a licença, tanto o *Solr* como o *Elasticsearch* têm a licença da *Apache*. Já o *Sphinx Search* tem a licença GPLv2 que, para fins comerciais, é necessário comprar. Face a esta informação temos aqui dois pontos de vista, ferramentas *opensource* e com licença comercial. Todavia ter licença comercial não é sinónimo de maior utilidade, isto é, no caso da importação de dados é possível observar que o *Sphinx Search* é muito mais orientado para as bases de dados e, por isso, carece de leitura de ficheiros PDF, Word entre outros que foram referidos anteriormente como uma necessidade para a solução em causa. Por outro lado, o *Solr* e o *Elasticsearch* implementam o *Apache Tika* [18], que consiste numa biblioteca que suporta a extração de texto de ficheiros binários, como por exemplo: PDF, Word, XML, formatos de áudio, etc. Mesmo relativamente às bibliotecas, o *Sphinx Search* apresenta menos variedade que as outras duas ferramentas. A figura seguinte apresenta um *ranking* de popularidade de diversas ferramentas de pesquisa e de bases de dados entre as quais se encontram as ferramentas estudadas neste projeto.

Rank			DBMS	Database Model	Score		
Feb 2016	Jan 2016	Feb 2015			Feb 2016	Jan 2016	Feb 2015
12.	12.	↑ 16.	Elasticsearch	Search engine	77.84	+0.63	+25.01
13.	↑ 14.	13.	Teradata	Relational DBMS	73.38	-1.57	+3.93
14.	↓ 13.	↓ 12.	Solr	Search engine	72.27	-3.12	-9.21
15.	15.	↑ 17.	Hive	Relational DBMS	52.77	-0.81	+16.21
16.	16.	↓ 14.	HBase	Wide column store	52.02	-1.34	-5.12

Figura 4 - Ranking de popularidade de ferramentas de pesquisa e bases de dados [3]

Observando a figura anterior chegamos à conclusão que o *Solr* e *Elasticsearch* se encontram nas primeiras posições no que diz respeito a ferramentas de pesquisa e indexação. O *Sphinx Search* encontra-se na 35ª posição com uma popularidade de 9.08, bem inferior à do *Solr* e *Elasticsearch*. Perante esta informação e pelas características analisadas anteriormente segue-se a análise apenas com o *Solr* e *Elasticsearch*.

Decidir qual das duas ferramentas se adapta melhor ao problema em causa torna-se complicado, uma vez que as suas características são muito semelhantes. Segundo alguns autores tanto uma como outra merecem atenção:

"*Both search engines provide similar functionality, and features evolve quickly with each new version … the functionality you need is covered by both, and, as is often the case with competitors, choosing between them becomes a matter of taste.*", Gheorghe, et al. [20]

Tendo em consideração a opinião dos autores, apenas resta apelar à documentação. No *ElasticSearch*, vê-se que há uma tentativa de produzir boa documentação, mas como ainda é uma

ferramenta relativamente recente esta carece de algum conteúdo. Já a do *Solr* é bastante boa e a comunidade é bastante ativa o que leva a tender para uso desta ferramenta.

2.4 Classificação

Com base no que foi dito até agora, é possível assumir que os resultados de uma pesquisa de elementos indexados seguem uma ordenação, ou não, por relevância, de acordo com os modelos clássicos e estruturados referidos anteriormente no tópico de recuperação de informação. Perante esta situação parece que o estudo levará a uma indexação simples de documentos sem uma perspetiva clínica associada, o que não é o caso. Pretende-se que a solução tenha teor clínico, isto é, que esteja ligada a uma classificação de termos, com um ponto de vista médico, de forma a apresentar ao utilizador resultados com mais significado e até mesmo acelerar o processo de pesquisa. A classificação referida consiste na ordenação sistemática de todos os conceitos numa área científica, neste caso, na área da medicina.

2.4.1 Relações Hierárquicas

Segundo Rothwell, et al. [21], a relação mais importante usada para a classificação é uma relação hierárquica. Este tipo de classificação está dividido em:

- Classificação hierárquica simples: um conceito só pode ter um pai na hierarquia;
- Classificação hierárquica múltipla: um conceito pode ter vários pais na hierarquia.

Os autores referem que uma hierarquia simples não é suficiente para aplicações complexas. No entanto, as hierarquias múltiplas tendem a ser demasiado volumosas e redundantes não sendo, por isso, muitas vezes utilizadas. Para resolver tal problema os autores chegam à conclusão que a hierarquia múltipla pode ser substituída por:

"*Several disjoint monohierarchic classifications connected by a semantic model*", Rothwell, et al. [21]

Exemplo desta solução é o *Systematized Nomenclature of Medicine – Clinical Terms* (SNOMED CT), o qual se baseia num sistema de nomenclatura de termos médicos que providencia códigos, termos, sinónimos e definições usadas na documentação clínica.

2.4.2 Classificações Clínicas

O SNOMED CT é uma nomenclatura que faz parte das ferramentas do *Unified Medical Language System* (UMLS) que consiste num conjunto de ficheiros e *software* que contemplam diversos vocabulários de saúde e biomedicina. O UMLS é composto por três ferramentas (fontes de conhecimento):

- ***Metathesaurus*** que consiste em termos e códigos de diversos vocabulários, incluindo CPT, ICD-10-CM, LOINC, MeSH, RxNorm, e SNOMED CT;
- ***Semantic Network*** que consiste em relações semânticas entre uma vasta gama de categorias;
- ***SPECIALIST Lexicon and Lexical Tools*** que consiste no processamento de linguagem natural.

As duas últimas ferramentas são utilizadas para construir o *Metathesaurus*. A construção deste envolve o processamento de termos e códigos usando as ferramentas lexicais, agrupando termos sinónimos em conceitos, categorizando conceitos semanticamente e incorporando relações e atributos providenciados pelos vocabulários. Nos tópicos seguintes são apresentadas as nomenclaturas SNOMED CT, MeSH e ICD-9.

2.4.2.1 SNOMED CT

Como já foi referido, o SNOMED CT consiste num conjunto de termos organizados hierarquicamente e que têm relações entre si. A estrutura do SNOMED CT encontra-se na figura seguinte.

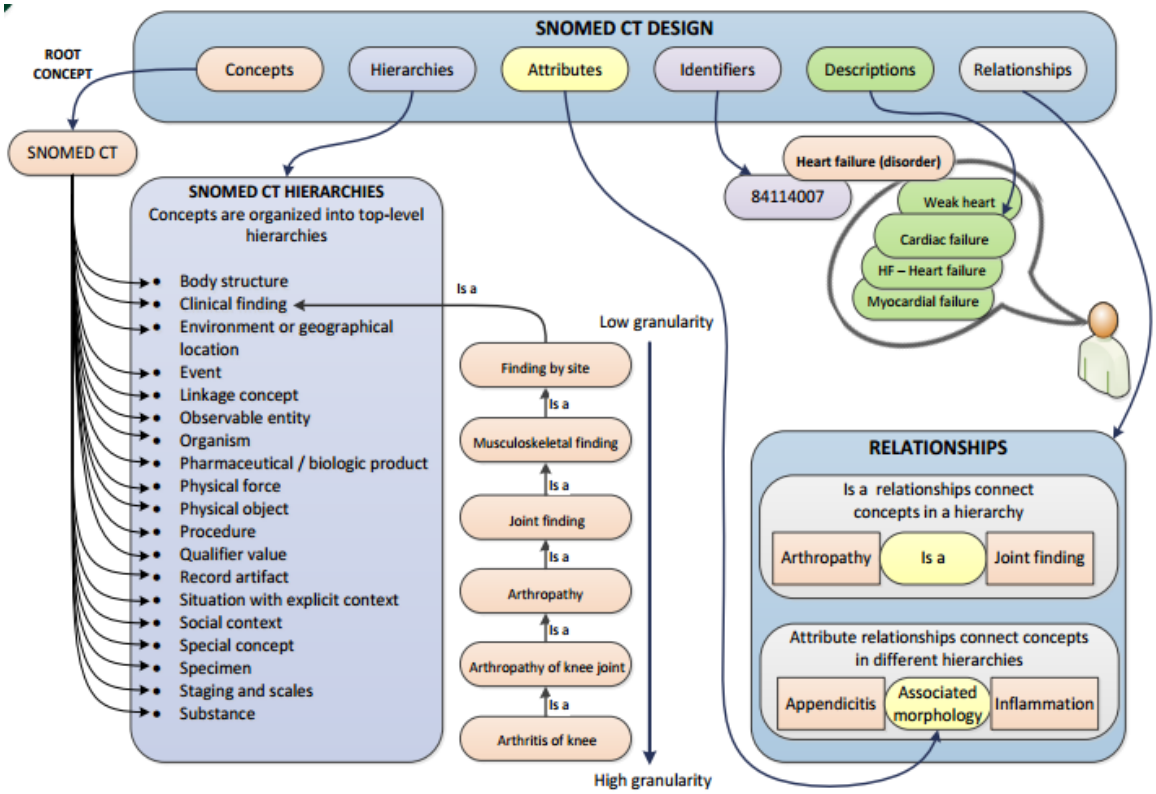


Figura 5 - Estrutura hierárquica e relações dos termos do SNOMED CT [5]

2.4.2.2 ICD-9

O ICD-9 consiste num conjunto de códigos de diagnósticos e de procedimentos utilizados para classificação e codificação da informação de doenças e intervenções cirúrgicas, como é possível observar na tabela seguinte.

Código	Descrição
0010	Cholera due to vibrio cholerae
0011	Cholera due to vibrio cholerae el tor
0019	Cholera, unspecified
0020	Typhoid fever

Tabela 2 - Exemplo de ICD-9-CM [22]

Esta nomenclatura fornece códigos relativos à classificação de doenças e de uma grande variedade de sinais, sintomas, aspetos anormais, queixas, circunstâncias sociais e causas externas para ferimentos ou doenças. A cada estado de saúde é atribuída uma categoria única à qual corresponde um código, que contém até 6 caracteres. Tais categorias podem incluir um conjunto de doenças semelhantes. Esta nomenclatura começou a ser desenvolvida do século XIX e é mantida pela Organização Mundial de Saúde. A revisão mais conhecida é o ICD-9 que surgiu em 1975 [23]. Em Portugal a sua utilização na codificação sistemática dos episódios de internamento iniciou-se em 1989.

Dentro do conjunto dos códigos aceites num determinado momento, são feitas validações em relação à idade, ao sexo, e à posição de cada código específico dentro da hierarquia da categoria a que pertence, de modo que não sejam recolhidos códigos que tenham subcategorias e se obtenha o máximo de especificação possível.

2.4.2.3 MeSH

MeSH é um vocabulário controlado de termos organizado hierarquicamente em 16 árvores. A cada árvore é atribuída uma letra de forma a identificar a mesma.

A	Anatomy
B	Organisms
C	Diseases
D	Chemicals and Drugs
E	Analytical, Diagnostic and Therapeutic Techniques and Equipment
F	Psychiatry and Psychology
G	Phenomena and Processes
H	Disciplines and Occupations
I	Anthropology, Education, Sociology and Social Phenomena
J	Technology and Food and Beverages
K	Humanities
L	Information Science
M	Persons
N	Health Care
V	Publication Characteristics
Z	Geographic Locations

Figura 6 - Estrutura hierárquica do MeSH [1]

Cada termo está localizado numa ou várias árvores simulando, desta forma, uma relação entre diferentes hierarquias, como o SNOMED. A estrutura da árvore tende de conceitos gerais para conceitos mais específicos.

Com a ajuda destas hierarquias pretende-se que o resultado de uma pesquisa por parte do utilizador venha associado a um maior valor no contexto clínico.

2.5 Eletronic Health Record

O *Electronic Health Record* (EHR), como o próprio nome indica consiste no registo digital de documentos clínicos. Surgiu em meados dos anos 60 e 70, essencialmente devido a estudos realizados pelos centros médicos académicos e organizações clínicas governamentais. A Universidade de *Utah* juntamente com a Corporação *3M* desenvolveram um dos primeiros EHR (*Health Evaluation through Logical Processing - HELP*). Ao longo dos anos, foram desenvolvidas várias outras ferramentas nesta área como, por exemplo, o *EHRServer*. O EHR surge da necessidade de resolver vários problemas como: a constante necessidade de adaptação das bases de dados, a dificuldade em manter compatibilidade com dados anteriores, a dificuldade em documentar alterações, a heterogeneidade na forma de introdução de dados, as dúvidas na interpretação na leitura, o facto de os conceitos evoluírem e a falta de avaliação na qualidade dos dados. Além destes problemas, a escolha das estruturas de dados (e a respetiva qualidade) dependem muito do objetivo em causa (prestação de cuidados, investigação clinica, gestão/financeira, ensino, legal).

Perante os problemas referidos e, uma vez que o EHR pretende corrigi-los, surgem diversas normas que tentam resolver esses problemas. Uma delas é o *openEHR* que permite a representação de conceitos clínicos complexos, de uma forma flexível, mas documentada.

A norma *openEHR* está estruturada da seguinte forma [24]:

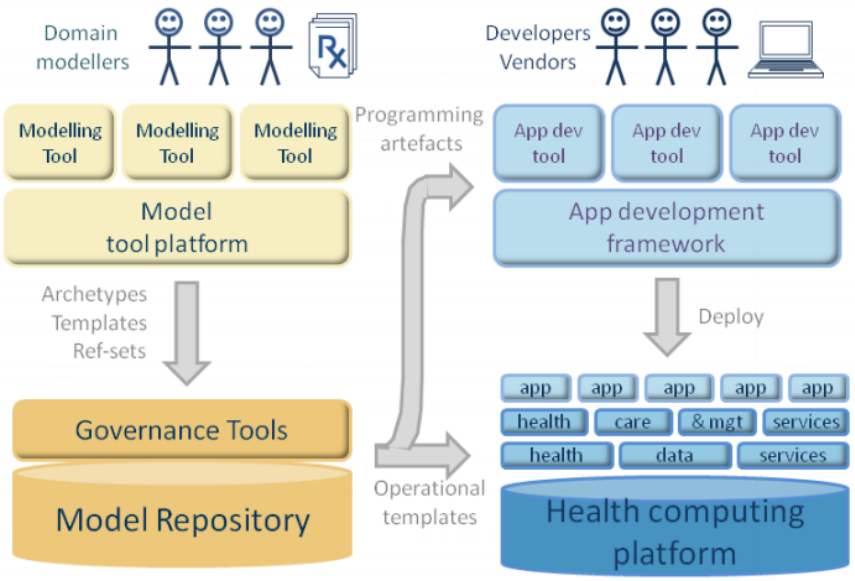
- **Arquétipo**: conceito mais elementar nesta estrutura (equivalente a uma peça de LEGO), consiste em dados clínicos, como por exemplo, altura, peso, sumário de gravidez e ecocardiograma;
- **Template**: consiste num conjunto de peças LEGO, isto é, num conjunto de arquétipos.

Perante estes conceitos é possível construir um esquema que define a estrutura de um documento podendo, assim, adaptar a informação presente nos documentos existentes de forma a homogeneizar os seus dados.

EHR	Registo clínico eletrónico por pessoa
Folders	Organização de alto nível do EHR (ex.: por episódio, por especialidade clínica)
Compositions	Conjunto de entradas submetidas numa determinada data (ex.: relatório)
Sections	Cabeçalhos clínicos que refletem o fluxo de trabalho
Entries	Declarações clínicas sobre observações, avaliações e instruções
Clusters	Entradas compostas
Elements	Entradas elementares
Data values	Termos codificados do conjunto de termos

Tabela 3 - Modelo de referência, [24]

Uma das principais vantagens na utilização desta norma é o facto de os utilizadores poderem alterar os arquétipos e *templates* sem alterar o *software*, ou seja, estamos perante uma ferramenta bastante geral, que aceita qualquer tipo de documento (desde que seja elaborado um bom *template* para receber os seus dados). Assim sendo, na imagem seguinte estão presentes os atores envolvidos no *openEHR*:



A partir desta norma foi desenvolvida a ferramenta *EHRServer* [25], que consiste num repositório de informação clínica gratuito, que segue uma arquitetura orientada a serviços (REST API) e é compatível com a norma *openEHR*. Com a REST API, o *EHRServer* possibilita o armazenamento e a pesquisa de dados clínicos de muitas formas, suportando diversos formatos como XML e JSON. É ainda possível criar *queries* de forma simples através da interface do utilizador, com o *EHRServer Query Builder*. Além disto, como é de código aberto é possível adaptá-lo às diferentes necessidades.

Em suma, o *EHRServer* pode ter um papel importante na solução em causa, pois, ao seguir a norma *openEHR*, possibilita uma organização e estruturação dos dados numa perspetiva clínica.

2.6 Resumo

Em suma, este capítulo permitiu realizar uma análise da recuperação, indexação e classificação de informação de um modo geral e direcionada para o meio clínico.

Relativamente à indexação, é possível afirmar que, durante os últimos anos, têm vindo a ser desenvolvidas diversas ferramentas que permitem a realização da mesma de forma eficaz. Exemplos disso são o *Apache Lucene*, *Apache Solr*, *Elasticsearch* e *Sphinx Search*. Com estas, torna-se possível a indexação de documentos provenientes de diversas fontes e com diferentes formatos e, como são implementadas com a ajuda de alguns dos modelos referidos neste capítulo, os modelos clássicos e estruturados, permitem que a informação apresentada ao utilizador seja a mais relevante perante a pesquisa feita pelo mesmo. No entanto, os modelos referidos apenas dizem respeito a uma indexação geral, isto é, a uma indexação sem conteúdo clínico e, por isso, com este tipo de abordagem podem ser rejeitados resultados com bastante importância. Face a este problema alguns autores sugerem a utilização de métodos de classificação como forma de apresentar resultados com maior relevância para o utilizador. No âmbito dos métodos de classificação surgem assim nomenclaturas que, de uma forma hierárquica, permitem relacionar termos clínicos, como por exemplo: SNOMED CT, ICD-9 e MeSH.

Acrescentando ao referido, surge ainda a norma *openEHR* que ajuda no processo de indexação na medida em que junta num só local toda a informação documental que inicialmente se encontrava dispersa pelas diversas fontes. Apesar de ser possível fazê-lo sem o *openEHR*, este último garante a integridade de toda a informação clínica assim como a sua uniformização e estruturação.

Em suma, com o *openEHR*, com as ferramentas de indexação e com as nomenclaturas em causa, é possível criar uma solução com resultados mais úteis e mais direcionados para as necessidades do utilizador (médicos), tendo em conta que a relevância dos resultados e a apresentação dos mesmos ao utilizador são essenciais para esse objetivo.

Capítulo 3

Solução

Como analisado no **capítulo 1**, é possível concluir que vários problemas advêm da grande quantidade de informação clínica. Entre estes podem referir-se a grande variedade de dados, os diferentes formatos desses dados, as diferenças na sua estrutura, etc. A partir disto e uma vez escolhidos como ferramenta de indexação, o *Sorl*, e como solução para uniformizar a informação, o *EHRServer*, a arquitetura apresentada a seguir foi desenhada a pensar não só nestes dois fatores, mas também nas necessidades exigidas pela aplicação de forma a apresentar ao utilizador resultados interessantes do ponto de vista clínico.

Sendo assim, neste capítulo pretende-se apresentar a arquitetura da aplicação e evidenciar como é que esta contribui para a resolução do problema em causa.

3.1 Arquitetura

A arquitetura elaborada é, em termos abstratos, constituída por dois componentes: uma aplicação cliente-servidor, através da qual todos os clientes podem conectar-se ao servidor e realizar ações, e uma aplicação de *back-end* destinada a concentrar no servidor de pesquisa a informação existente nos servidores operacionais.

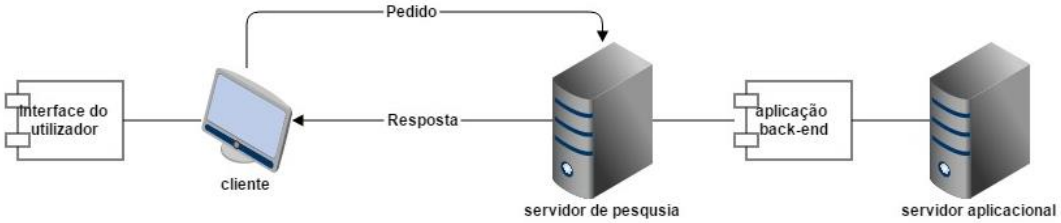


Figura 8 - Arquitetura geral do sistema

O lado do cliente é meramente composto pela interface, a qual é usada pelo utilizador para realizar ações e ver resultados. As componentes principais do sistema encontram-se do lado do servidor. Todavia o lado do cliente não deve ser negligenciado, pois sem este, não existem *queries* para o lado do servidor funcionar de acordo com a sua implementação.

Na **figura 9** é apresentada, em maior detalhe, a arquitetura da aplicação, que é seguida pela explicação de cada um dos passos.

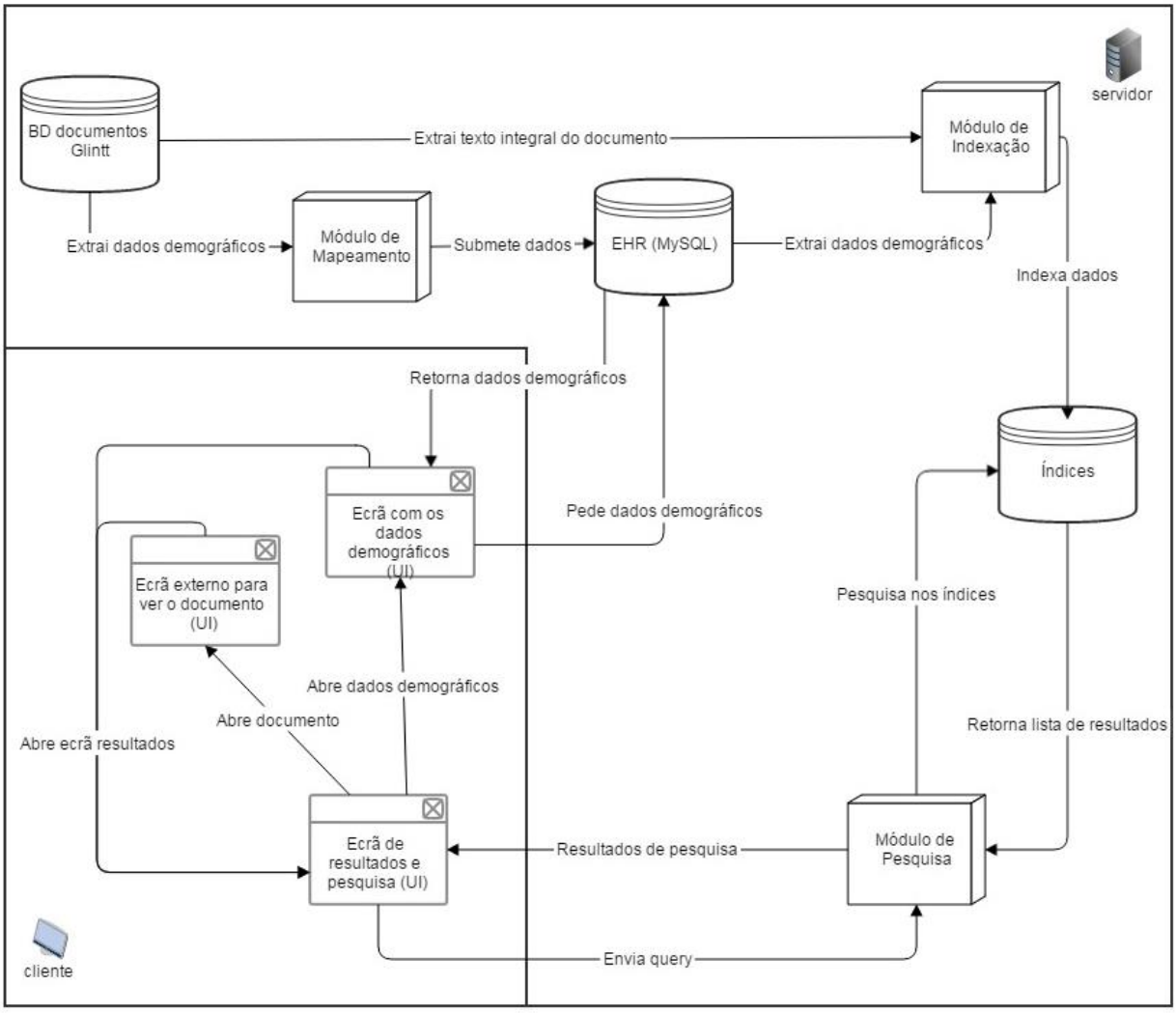


Figura 9 - Arquitetura da solução

3.1.1 BD documentos Glintt

A base de dados de documentos da Glintt contém todos os dados que se querem ver tratados. Aqui estão armazenados todos os documentos, desde JSON, XML, PDF, PNG, entre outros. A cada um destes está associada a respetiva informação demográfica do paciente.

No processo de recolha de informação da base de dados em causa é retornada uma linha por documento, ou seja, para cada documento vêm associados os dados demográficos do doente correspondente.

3.1.2 Módulo de Mapeamento

Este módulo é responsável pelo mapeamento entre os dados da base de dados da Glintt com as tabelas do EHR. Usa os arquétipos e *templates* para mapear corretamente os dados.

3.1.3 Servidor EHR

Como já referido na revisão bibliográfica, *EHRServer* é um repositório de informação clínica que segue a norma *openEHR*. A base de dados EHR é responsável por armazenar os dados relativos ao *EHRServer*, o qual tem um papel fundamental no que diz respeito à uniformização dos dados, sendo, por isso, um dos focos principais desta solução.

A quantidade de dados no EHR varia de acordo com as atualizações e inserções feitas no sistema operacional da Glintt.

3.1.4 Módulo de Indexação

Previamente, antes de se iniciar a indexação, são definidos num esquema (*schema*), os parâmetros de indexação e de recolha da informação a ser indexada. Uma vez concluída a construção deste esquema, com a ajuda do *Solr*, procede-se à indexação propriamente dita. Aqui são indexados tanto os dados do sistema operacional da Glintt, como do *EHRServer*. Nesta fase, do sistema operacional da Glintt é apenas indexado o texto integral dos documentos. Já do *EHRServer* são indexados os dados demográficos dos pacientes, previamente convertidos do sistema operacional para EHR, e os identificadores (ids) dos documentos, para no futuro ser possível mostrar os mesmos ao utilizador, sem ter que os duplicar.

3.1.5 Índices

Os índices serão o alvo de todas as *queries* feitas pelo utilizador e esta base de dados é composta por diversos índices. O número destes varia de acordo com as atualizações que vão sendo feitas ao longo do tempo tanto na base de dados da Glintt como no EHR.

3.1.6 Módulo de Pesquisa

Este módulo é responsável por receber *queries* do utilizador na interface do lado do cliente da aplicação e procurar as fontes previamente indexadas. Após encontrar uma resposta retorna-a para o lado do cliente novamente.

3.1.7 Interface

A solução em causa apresenta dois tipos de interface: interna, pertence à aplicação, e externa, vinda de um serviço externo à aplicação. As interfaces são as seguintes:

- **Ecrã de resultados e pesquisa**: interface interna responsável pela pesquisa e apresentação de resultados. Permite filtrar os resultados de pesquisa por data de nascimento e por data de criação/alteração de um documento;
- **Ecrã com os dados demográficos**: interface interna responsável por mostrar os dados demográficos de um paciente;
- **Ecrã externo para ver documento**: interface externa que permite a visualização dos documentos indexados.

3.2 Diagrama de casos de uso

Um diagrama de casos de uso descreve o que um utilizador pode fazer num determinado ambiente, neste caso, na aplicação em causa. Neste sentido existe apenas um utilizador, que é o utilizador comum (médicos). Estes podem realizar diversas ações, como é possível observar pelo seguinte diagrama.

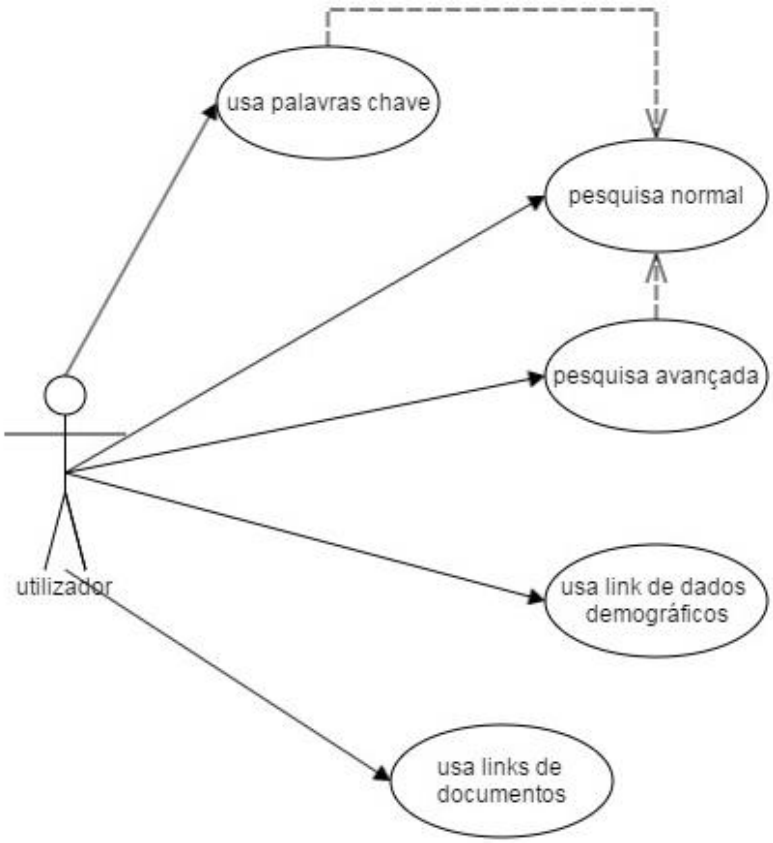


Figura 10 - Diagrama de casos de uso do utilizador

Os utilizadores podem realizar pesquisa sobre os índices previamente criados pelo indexador. Se ainda não existirem índices então o utilizador, apesar de poder efetuar pesquisa, não receberá nenhum resultado.

3.3 Conclusões

Após a análise da arquitetura da aplicação é possível assumir que com esta os problemas de acesso e outros no que diz respeito à informação podem ser resolvidos. Mas como? De acordo com a figura da arquitetura, é fácil reparar na existência de três módulos. Estes são responsáveis pela resolução dos problemas em causa.

No que diz respeito a documentos com diferentes formatos e estruturas, o módulo do mapeamento resolve isto mapeando os dados para o EHR. Desta forma não interessa quais os dados que estão guardados pois seguem todos a mesma estrutura, a estrutura do *template*.

Relativamente ao facto de nem toda a informação ser necessária ou importante de indexar, é um problema possível de resolver pelo módulo de indexação. Neste, e como já referido, é definido um esquema com os campos a serem indexados assim como os parâmetros para a indexação. Definindo os campos que são indexados, estão-se a restringir os dados que vão ocupar

lugar na base de dados dos índices. Além disto é também possível definir campos que, não sendo indexados, e por isso não são pesquisáveis, podem ser acedidos, ou seja, mesmo se for necessário manter alguns campos, como por exemplo ids, estes não precisam de ser indexados, são apenas armazenados para futuras consultas.

O restante módulo, o de pesquisa, é responsável por solucionar a apresentação dos documentos com diferentes formatos, isto é, retorna todos os dados necessários para visualizar os documentos, os dados demográficos e cada linha de resultados. No caso da visualização dos documentos, para poder mostrar ao utilizador documentos com diferentes formatos, recorre-se a um serviço da Glintt que, após receber alguns campos (enviados pelo módulo de pesquisa) pode facilmente mostrar o documento.

Além destes problemas existem outros casos mais específicos que, para serem solucionados, recorrem a mais do que um módulo. Exemplo destes são documentos com apenas imagens. Nestes casos não existe informação textual indexável, sendo necessário associar alguns dados que possam fornecer informação suficiente para que o documento seja pesquisável. Para tal recorre-se aos módulos de mapeamento e indexação, pois ao mapear os dados demográficos de um paciente, uma vez que estão associados ao documento, ao indexá-los é possível realizar uma pesquisa, neste caso por informação demográfica, que retornará não só essa informação como também a informação documental associada.

Em suma, e segundo a análise anterior, é possível afirmar que a arquitetura em causa é viável para que se possam atingir os objetivos desta dissertação e, com isto, resolver os problemas associados à indexação e pesquisa de documentos clínicos.

Capítulo 4

Implementação

Após definida a arquitetura é importante referir como foi desenvolvida a aplicação e quais as decisões tomadas e o porquê das mesmas ao longo deste processo de desenvolvimento. Com isto, neste capítulo pretende-se especificar cada um dos pontos referidos e descrever detalhadamente todos os processos em causa.

4.1 Protótipo

O protótipo que se pretende construir pretende garantir, a partir das suas funcionalidades, um melhor acesso e pesquisa de informação clínica. Para tal, o mesmo deve ser capaz de recolher, indexar e pesquisar informação que está dispersa nas bases de dados da Glintt.

A base de dados utilizada é composta por documentos com diversos formatos. Sendo assim, a aplicação terá de ser capaz de tratar as várias extensões de forma a que os requisitos sejam todos cumpridos.

4.2 Bases de dados da aplicação

4.2.1 EHR

O EHR, como já foi referido ao longo deste documento, é a base de dados onde são armazenados os dados do serviço *EHRServer*. Compreender a estrutura deste foi fundamental para o poder aplicar na solução em causa. Após feita a análise do *EHRServer* concluiu-se assim quais as etapas que deveriam ser percorridas para submeter os dados de um doente no EHR. Para tal utilizou-se a REST API disponibilizada pelo *EHRServer*.

4.2.2 *EHRServer*

A API do *EHRServer* expõe vários serviços que ajudam a preencher os dados na base de dados EHR. Estes são:

- **Login**: pode não parecer importante para a aplicação em causa, mas sem este pedido era impossível realizar qualquer outro, uma vez que este devolve um *token* de autorização necessário para a execução dos restantes pedidos;
- **CreatePerson**: responsável por adicionar os dados de um doente ao EHR. Apenas adiciona os dados essenciais como o nome e data de nascimento. Este passo ainda não adiciona informação consoante um *template*;
- **Commit**: adiciona dados, consoante um *template*, ao EHR. Foi um dos passos onde existiram mais problemas, pois o *EHRServer*, sendo uma ferramenta recente e em desenvolvimento, ainda não estava preparado para receber dados demográficos dos pacientes. O problema surgia precisamente na criação de um xml necessário para a submissão do pedido;
- **EhrForSubject**: retorna o id do **ehr** de um determinado paciente. **ehr** é uma tabela da base de dados EHR.

Para se perceber melhor como funcionam alguns destes pedidos, num dos tópicos seguintes é apresentado o modelo de dados desta base de dados.

4.2.3 *EHRServer* interface

O *EHRServer* além do lado do servidor, tem o lado do cliente, o qual ajudou bastante em diversos testes que foram sendo realizados ao longo do desenvolvimento da aplicação. Sendo assim, seguem-se algumas figuras representativas do lado do cliente do *EHRServer*. As imagens em causa apenas mostram os ecrãs responsáveis pelas contribuições, uma vez que estas são o foco da aplicação. Uma contribuição é um registo clínico de um paciente. Na secção **4.2.4** este conceito é descrito com mais pormenor.

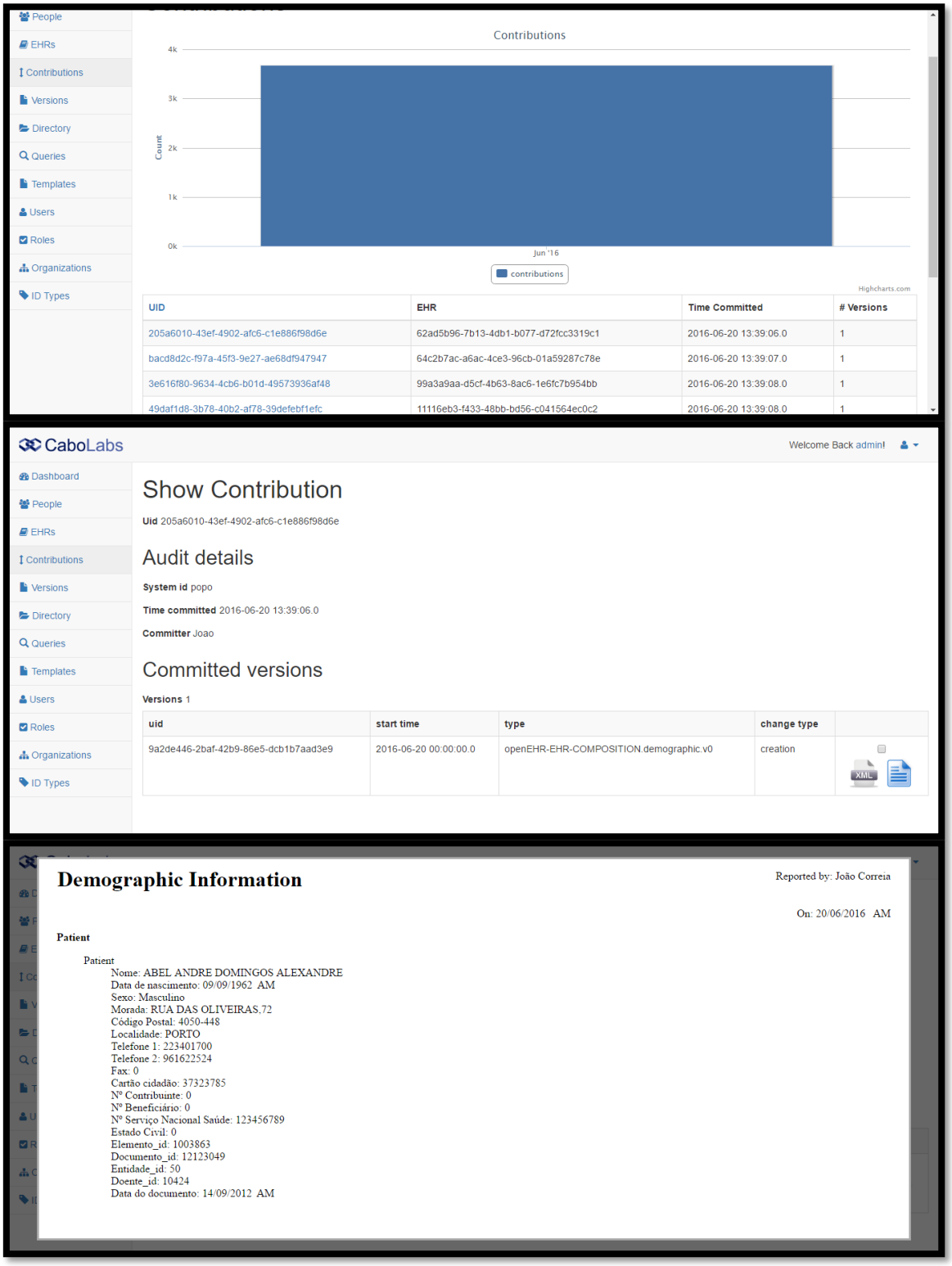


Figura 11 - Ecrãs de uma contribuição no *EHRServer*

4.2.4 Modelo de dados

No **anexo A** é apresentado o modelo EER da base de dados do *EHRServer*.

Como pode ser observado e analisando apenas as tabelas diretamente relacionadas com o EHR, cada pessoa pode ter vários **ehr**'s e para cada um destes existem várias **contribution**, isto é, cada **ehr** tem vários registos/contribuições clínicas de um doente. Para cada uma destas contribuições é fácil entender que a tabela **version** faz o registo de versões de cada registo clínico. Ao lado da tabela **version** tem a tabela **audit_details** que é responsável pelo armazenamento de dados como data em que foi submetido um registo, médico que o submeteu, etc. Seguindo em direção a **composition_index**, é possível observar que várias versões seguem apenas uma composição (**composition_index**), ou seja, apesar das várias versões nas quais pode ser diferente o conteúdo de um registo, a sua composição/estrutura são sempre as mesmas (seguem a estrutura do *template*). Sendo assim cada composição pode conter vários valores, os quais são registados em **data_value_index**. Para cada um destes é feita a separação pelas tabelas respetivas de acordo com o tipo de dados que representam.

Com isto, e como já referido anteriormente, torna-se claro que o *EHRServer* permite uma uniformização dos registos clínicos, aproveitando-se da estrutura em *templates* e arquétipos para guardar dados com diferentes estruturas.

4.3 Mapeamento

Este é sem dúvida um dos aspetos fundamentais do projeto e também o que levou mais tempo a concluir. Aqui é realizado o mapeamento entre os dados da base de dados da Glintt e o EHR.

É possível dividir este tópico em dois pontos: a base de dados da Glintt e o *EHRServer*. Iniciando pelo segundo, o *EHRServer* é uma aplicação desenvolvida em grails (*groovy and grails*) e que segue a norma *openEHR*. Logo no início surgiram diversas dificuldades associadas a este, desde problemas a executar o servidor, problemas com o funcionamento da base de dados entre outros. No entanto, a dificuldade que mais se evidenciou foi o não saber como é que se iriam passar os dados para o EHR seguindo os *templates*. Após algum estudo e pesquisa sobre o assunto concluiu-se que a forma mais eficaz de o fazer era utilizando a REST API que o *EHRServer* disponibilizava. Todavia este problema ainda nem perto estava de ser resolvido, pois para submeter estes dados era necessário criar, não só arquétipos e *templates*, como também um xml com uma estrutura bem específica que não estava descrita na documentação do *EHRServer*. Para que tal se tornasse exequível foi necessário entrar em contacto com o desenvolvedor do *EHRServer*, Pablo Pazos. Após várias discussões foi possível concluir como se iriam criar os arquétipos, *templates* e xml, e como submetê-los no *EHRServer*.

Assim sendo, para construir os arquétipos recorreu-se ao uso do *Archetype Editor* e, para os *templates*, o *Template Designer*. Já o xml resultou de uma discussão constante com Pablo Pazos. Para testar os pedidos *EHRServer* utilizou-se uma extensão do Google Chrome designada por Insomnia. Com isto finalizado e pronto para testar com vários dados começou-se a pensar quais os dados que seria importante ver indexados e, desses, quais os que necessariamente deveriam estar presentes no EHR. Rapidamente se concluiu que o texto integral dos documentos não deveria ser passado para as tabelas do EHR, pois além do processo ser extremamente demorado, teriam que ser criados diversos *templates* para cada estrutura diferente. Além disto, a ideia desta solução é, no futuro, ser aplicada não só na gestão documental como também noutras bases de dados em que o texto se encontra distribuído pelas colunas das diferentes tabelas. Sendo assim ainda menos sentido faz a passagem do texto integral dos documentos para o EHR. No meio deste processo de escolhas e decisões chegou-se a uma questão fundamental no projeto: será que é mesmo importante e necessária a utilização do EHR? A resposta é sim, pois além de uniformizar a informação permite que esta esteja distribuída por doentes que é o que se pretende encontrar após uma pesquisa realizada a partir desta aplicação. A acrescentar a este argumento tem-se o facto referido em cima de que a aplicação é, no futuro, para ser aplicada a outras bases de dados que poderão mostrar-se muito mais recetivas ao uso do EHR. Assim sendo, e seguindo com o EHR, chega-se ao segundo ponto, a base de dados da Glintt.

Relativamente à base de dados da Glintt, foi gerado o modelo EER para se perceber melhor como é que esta estava estruturada. Após a análise da mesma voltou-se novamente à questão no que diz respeito aos dados que devem ser indexados. Chegou-se rapidamente à conclusão que o texto integral dos documentos e os dados demográficos dos pacientes eram informação suficiente para ser indexada. Uma vez já concluído que o texto integral dos documentos não deveria ser mapeado para o EHR, rapidamente se decidiu que os únicos dados com interesse de serem mapeados para o EHR seria a informação demográfica dos pacientes, sendo que os restantes dados seriam indexados diretamente da base de dados da Glintt. Quanto à relação entre paciente e documento essa seria mantida através dos respetivos ids que seriam indexados juntamente com a informação demográfica.

Em suma, a aplicação funciona com dois tipos de bases de dados, o EHR, que armazena os dados demográficos dos pacientes recorrendo a arquétipos, *templates* e xml, e a base de dados da Glintt, que armazena o texto integral dos documentos.

Na imagem seguinte encontra-se o *template* usado na aplicação.

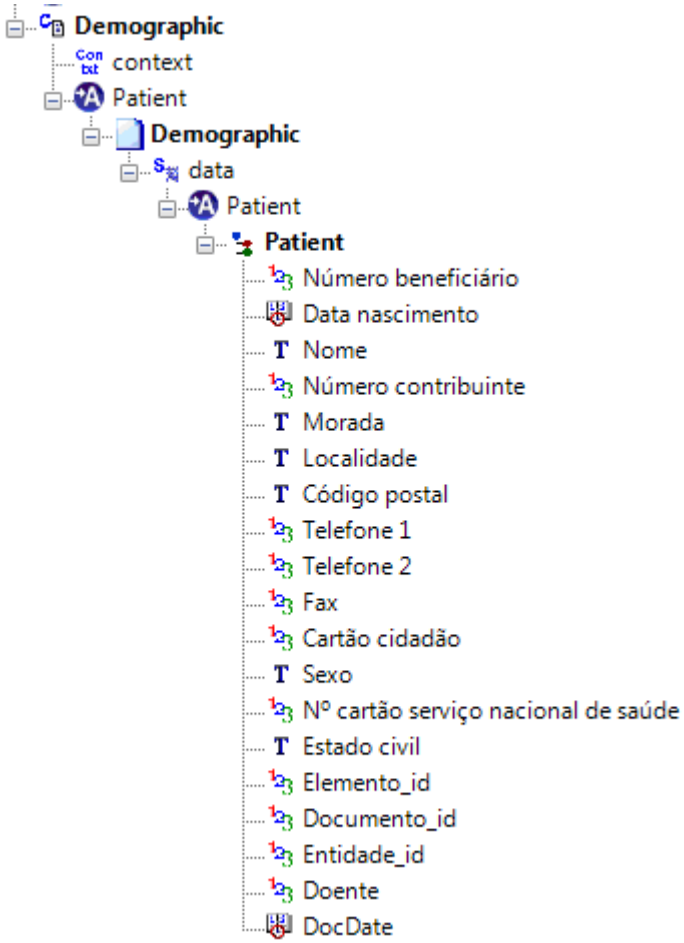


Figura 12 - *Template* (dados demográficos)

Como se pode observar pela figura anterior, por ordem hierárquica, este *template* é formado por três arquétipos:

1. ***Demographic*** – é uma *composition*;
2. ***Demographic*** – é uma *entry*;
3. ***Patient*** – é um *cluster* (o arquétipo inicia-se no *Patient* com as cores azul, vermelho e verde).

Para mais pormenores rever **tabela 3**.

Com isto, e analisando apenas o terceiro arquétipo, é fácil perceber que tipo de dados demográficos estão a ser armazenados no EHR.

4.4 Indexação

Nesta etapa existiram muito menos dificuldades, essencialmente devido à boa documentação. Para a indexação recorreu-se ao uso do *Solr*, o qual permite através da edição de

alguns ficheiros xml definir todos os parâmetros e variáveis necessários para a indexação dos dados em causa.

Existem três ficheiros de configuração importantes no *Solr*:

- **data-config.xml** – responsável pelas *queries* essenciais para a recolha de informação das diferentes fontes de dados (neste caso a base de dados da Glintt e o EHR);
- **solr-config.xml** – responsável pelas configurações do *Solr*. É neste ficheiro que estão definidos todos os parâmetros ligados à pesquisa, como por exemplo, em que campo ou campos é que o *Solr* vai realizar pesquisa, qual o tamanho do texto que vai aparecer no ecrã de resultados, etc;
- **schema.xml** – responsável pela estrutura dos dados que vão ser indexados, armazenados, ou não, no *Solr*. É o ficheiro de configuração mais importante, pois é aqui que também se define como será efetuado o processo da indexação.

Inicialmente começou-se por testar estes ficheiros xml com as configurações predefinidas. Seguidamente começaram-se a fazer testes com o **data-config.xml**, o qual, após algum tempo já estava preparado para receber dados do EHR. Após este ponto, surgiram alguns problemas no que diz respeito à quantidade de pedidos e velocidade de indexação. Demorava entre 8 e 12 segundos para indexar 8 documentos. Além disto, eram feitos cerca de 467 pedidos à base de dados para a mesma quantidade de documentos (nesta altura ainda só se estavam a indexar dados do EHR). Com isto, veio associada uma grande preocupação em reduzir a quantidade de pedidos e aumentar a velocidade de indexação. Para atingir este objetivo, além de haver uma correção nas *queries*, foi realizada uma nova análise dos tipos de dados que havia interesse em indexar, ficando decidido que inicialmente apenas se indexaria texto (mais tarde foram inseridas datas para uma filtragem dos resultados). Após isto a velocidade de execução tornou-se muito maior, cerca de 2 segundos para os mesmos 8 documentos. Já para não falar na quantidade de pedidos que foi reduzida para aproximadamente 86. Mais tarde foram realizadas novas alterações às *queries* com resultados ainda melhores. Estes podem ser observados no **capítulo 5**.

À medida que iam sendo feitas alterações no **data-config.xml**, os outros dois ficheiros iam também sendo modificados de acordo com as necessidades. Assim, e como o **schema.xml** é o mais importante, segue-se um pequeno excerto do mesmo, mostrando como os dados no *Solr* estão estruturados.

```
<field name="uid" type="string" indexed="true" stored="true"/>
<field name="version_uid" type="string" indexed="false"
stored="true"/>
<field name="value" type="text_general" indexed="true" stored="true"
multiValued="true"/>
<field name="dates" type="date" indexed="true" stored="true"
multiValued="true"/>

<field name="first_name" type="string" indexed="false" stored="true"/>
<field name="last_name" type="string" indexed="false" stored="true"/>
<field name="dob" type="date" indexed="false" stored="true"/>
<field name="elemento_id" type="int" indexed="false" stored="true" />
<field name="documento_id" type="int" indexed="false" stored="true" />
<field name="content" type="text_general" indexed="true"
stored="true"/>
<field name="text" type="text general" indexed="true" stored="false"
multiValued="true" />
```

Figura 13 - Estrutura de dados no schema.xml

Como é possível observar nem todos os campos têm a opção *indexed* e *stored* como verdadeira, ou seja, nem para todos os campos é necessário armazenar e indexar. No caso dos campos que não são indexados, não podem ser pesquisados, no caso dos campos que não são armazenados, não podem ser vistos no resultado de pesquisa. Face a isto, podem-se tirar diversas vantagens na utilização destes dois campos, por exemplo, os campos **elemento_id** e **documento_id** não estão indexados, mas estão armazenados, ou seja, não tem interesse indexar ids, no entanto, manter a informação destes é extremamente importante para identificar, neste caso, o documento. Relativamente a estes dois campos serão referidos no tópico seguinte com mais detalhe, assim como os campos **first_name**, **last_name** e **dob**, por não serem indexados.

Os campos relativos ao texto integral do documento e à informação demográfica são **content** e **value** respetivamente. Estes são copiados, noutra parte do **schema.xml**, para o campo **text**, o qual não fica armazenado, pois apenas é utilizado para fins de pesquisa e também não interessa armazenar duas vezes a mesma informação. Este último é definido como campo predefinido para a pesquisa.

Relativamente ao campo **uid** é indexado, pois funciona como chave primária de uma contribuição (documento) e também do **schema.xml**. Neste xml as chaves primárias têm que obrigatoriamente ser indexadas.

Quanto ao campo **version_uid**, é responsável por armazenar um identificador para cada versão, cada documento não tendo, por isso, interesse indexá-lo.

Por fim o campo **dates** também é indexado, pois serve para uma pesquisa avançada por intervalo de datas, com o objetivo de restringir os resultados encontrados.

Falando agora do campo **text**, é possível observar que o seu tipo é **text_general**, ou seja, corresponde a um campo de texto no qual, na sua definição, estão presentes todos os dados relativos aos parâmetros de indexação e pesquisa. Na seguinte tabela faz-se uma análise mais cuidada de cada um desses parâmetros.

`<tokenizer class="solr.StandardTokenizerFactory"/>`
Divide o texto em *tokens*, ou seja, em palavras, tratando os espaços em branco e a pontuação como delimitadores. Esses delimitadores são apagados.
Exemplo [26]: *In*: "Please, email john.doe@foo.com by 03-09, re: m37-xq." *Out*: "Please", "email", "john.doe", "foo.com", "by", "03", "09", "re", "m37", "xq"
`<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />`
Como o próprio nome indica, é o parâmetro responsável pela remoção das *stopwords*, as quais estão definidas no ficheiro stopwords.txt (proveniente do *Solr*).
`<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>`
Encontra *tokens* e modifica-os com outros, ou seja, permite que uma pesquisa, por exemplo, por "dor de barriga" encontre também resultados com "dor abdominal".
`<filter class="solr.LowerCaseFilterFactory"/>`
Coloca as letras de cada *token* em minúscula.
`<filter class="solr.RemoveDuplicatesTokenFilterFactory" />`
Remove duplicados para não sobrecarregar o processo de indexação com informação repetida.
`<filter class="solr.SnowballPorterFilterFactory" language="Portuguese"/>`
Coloca as palavras na sua raiz morfológica (*stemming*).

Tabela 4 - Parâmetros de indexação do *Solr*

Ainda na análise dos parâmetros, é possível referir que nem todos estão presentes no processo de pesquisa e indexação, exemplo disto é o parâmetro relativo aos sinónimos que só está definido para a pesquisa e não para a indexação, uma vez que não faz sentido indexar mais palavras que retornarão o mesmo resultado.

Uma vez concluída a definição destes parâmetros, e com o fim de criar uma aplicação cliente-servidor, recorreu-se ao .NET, que consiste numa *framework* em c#. Com esta escolha foi possível utilizar o *Solrnet* que consiste num cliente do *Solr* que permite utilizar as funcionalidades do mesmo através de funções em c#. Sendo assim, a utilização do ficheiro **data-config.xml** tornou-se inútil, uma vez que as *queries* podiam ser feitas através do c#.

Em suma, o processo de indexação é bastante complexo, no entanto a utilização de ferramentas como o *Solrnet* facilitam a sua implementação e melhoram a gestão deste processo.

4.5 Pesquisa

Nesta etapa são utilizadas funcionalidades do *Solrnet* para realizar pesquisa sobre os diversos índices.

Para se realizar este processo com sucesso, o *Solrnet* dispõe de uma opção para executar *queries*. Com esta é possível definir quais os campos onde a pesquisa vai ser realizada, o número de páginas que aparecerão no resultado, o texto que será mostrado ao utilizador em cada linha dos resultados, etc. Uma vez definidos estes campos, tem-se toda a informação necessária para mostrar ao utilizador resultados com interesse.

Remetendo agora para os campos referidos no tópico da indexação, relativamente ao **elemento_id** e ao **documento_id** são apenas falados aqui pois o seu papel como dados com importância reside essencialmente nesta secção. Estes dois campos são retornados em cada linha de resultados como um *link*. Exemplo: "www…?elemId=**elemento_id**&docId=**documento_id**". Com este acede-se ao serviço da Glintt o qual mostra o documento em causa. Através desta implementação apenas se gasta memória para guardar os identificadores, em vez de armazenar novamente toda a informação de um ficheiro. Assim sendo, tem-se aqui uma otimização do espaço necessário para indexar, o que acelera o processo de pesquisa, uma vez que este não terá que retornar no resultado todo o documento.

Relativamente aos campos **first_name**, **last_name** e **dob**, estes não são indexados, pois resultam em informação repetida do paciente. É repetida, pois estes dados já se encontram no campo **value** e, por isso, não há necessidade em indexar a mesma informação duas vezes. Aqui pode suscitar a dúvida do porquê de ter campos repetidos. A resposta é simples. Para que a pesquisa dependa apenas do *Solr* é necessário que todos os campos que aparecerão nos resultados de uma pesquisa estejam presentes no **schema.xml**. É verdade que o campo **value** guarda estes dados e outros, mas não há forma de saber em que posição da lista estes são armazenados (o campo **value** é multivalor), tornando necessária esta repetição.

O processo de pesquisa foi feito a pensar no utilizador e, por isso, para facilitar esta operação, foi desenvolvida também uma opção para pesquisa avançada que é descrita no tópico seguinte.

Após a pesquisa avançada são descritos ao pormenor cada um dos passos de pesquisa.

4.5.1 Pesquisa Avançada

A pesquisa avançada foi desenvolvida com o objetivo de disponibilizar mais opções de pesquisa, mais filtros que permitam ao utilizador restringir os resultados e, com isto, obter resultados mais interessantes.

Inicialmente pensou-se em desenvolver três tipos de pesquisa avançada: pesquisa por hierarquia de doenças, pesquisa por região e pesquisa por datas de nascimento dos pacientes e por

datas de criação/atualização dos documentos. Apenas a pesquisa por datas foi implementada, no entanto, a implementação de outros tipos de pesquisa é possível e pode ter interesse no futuro para melhorar ainda mais a experiência de pesquisa por parte do utilizador.

Assim sendo, o utilizador dispõe de dois campos para realizar uma pesquisa por datas: a data inicial e a data final. Após a introdução dos dados nestes campos o utilizador pode pesquisar pela informação que procura. A pesquisa avançada não funciona apenas como filtro e, por isso, o utilizador pode apenas pesquisar pelo intervalo de datas, não necessitando de introduzir qualquer informação textual para a pesquisa. No entanto, se o fizer os resultados serão bem mais restritos e bem mais direcionados aos interesses do mesmo.

Por fim, no que diz respeito à sua implementação, a pesquisa avançada utiliza uma classe do *Solrnet*, *SolrQueryByRange*, que ao receber duas datas, retorna todos os resultados nesse intervalo.

4.5.2 Construção de *queries*

Para a construção das *queries*, o processo de pesquisa precisa de receber os termos pesquisados pelo utilizador e as datas (estes dados são concatenados numa só *string*). Além destes são necessários também a página inicial e o número de resultados que se querem ver em cada página. O valor predefinido da página inicial é igual a 1 e o de número de resultados por página é igual a 5 (valores configuráveis no ficheiro **Web.config**, ver **anexo B**). Após receber estes dados faz-se o tratamento da *string* para obter cada um dos termos de pesquisa. Com esta implementação é possível acrescentar muitos mais filtros (além da filtragem por datas), pois recebendo apenas uma *string* permite que todos os dados pesquisados estejam presentes nela.

Relativamente à página inicial e quantidade de resultados a serem mostrados, apenas servem para reduzir o tamanho da *querie*, ou seja, a *querie* feita pelo *Solr* apenas vai retornar os resultados que se encaixam nesse limite de resultados, tornando assim claro, que a complexidade da *querie* vai ser menor. Além disto, estes parâmetros servem também para controlo da paginação.

Após a construção e execução da *querie*, procura-se nos índices criados quais os dados que correspondem à informação pesquisada.

4.5.3 Ordenação dos resultados

Cada vez que o processo de pesquisa é efetuado e são encontrados resultados, estes são devolvidos numa certa ordem. A ordem referida não é aleatória e, como já foi estudado e falado na revisão bibliográfica, existem vários métodos que se focam nesta área. Assim sendo, o *Solr* não é diferente, ordenando os resultados do mais relevante para o menos relevante. Esta ordenação é realizada usando uma combinação do modelo vetorial com o modelo booleano [27].

Em suma, e relembrando um pouco o que foi falado na revisão bibliográfica, a ideia por trás do modelo vetorial é qualificar cada termo de acordo com a quantidade de vezes que ele aparece em todos os documentos. Já o modelo booleano foca-se na lógica booleana para classificar os resultados. No entanto a base da classificação no *Solr* assenta principalmente no modelo vetorial.

4.5.4 Construção dos resultados

Nesta etapa são construídos os resultados de acordo com as necessidades de visualização. A maioria destes, excetuando os excertos de texto com os termos pesquisados, não recebem qualquer tratamento e, por isso, são apenas armazenadas num dicionário para futura utilização no lado do cliente para mostrar os dados.

Esta etapa é importante no que diz respeito essencialmente aos excertos de texto. A informação nestes precisa de ser formatada de forma a mostrar apenas os dados necessários. Para tal, recorre-se ao uso do ***HighlightingParameters***, que é uma função disponível no *Solrnet* que permite através da definição de alguns parâmetros presentes nos xml de configuração do *Solr*, colocar umas *tags* à volta dos termos pesquisados e controlar a quantidade de texto que se encontra à volta desses termos. As *tags* referidas neste caso são "termo" que em HTML correspondem ao negrito. Após este passo, o texto final é guardado no mesmo dicionário que os outros resultados.

Assim, com esta etapa, consegue-se retornar todos os resultados necessários para mostrar ao utilizador em cada linha de resultados.

4.6 Atualização de índices

Este ponto nem sempre se verifica. Para manter os dados indexados atualizados pensou-se em implementar também um sistema de atualização incremental, isto é, um sistema que verifica na BD de documentos da Glintt, se existem novos dados ou dados atualizados que necessitem de ser novamente indexados.

Para atingir este objetivo, após a indexação dos dados, é executada uma rotina, que corre em *background*, responsável por fazer um pedido à base de dados que verifica se existem atualizações. Esse pedido é efetuado de n em n milissegundos (o valor predefinido é de 5000 milissegundos, configurável no ficheiro **Web.config**). Caso não existam atualizações nos dados, a *thread* esperará os próximos n milissegundos para fazer novamente o pedido. Caso contrário é iniciado o processo desde a recolha de dados da BD de documentos da Glintt. Os dados recolhidos/extraídos da BD de documentos da Glintt são apenas os dados atualizados e não todos como é feito inicialmente.

Com o fim de detetar os documentos atualizados é executada uma *querie* que verifica a data de criação e de atualização dos documentos. Desta forma apenas os documentos com data superior à última data de verificação serão indexados.

Assim, com esta implementação consegue-se indexar apenas o necessário evitando, por isso, processos demasiadamente longos sempre que houvesse uma atualização.

4.7 Interface

Esta aplicação, como já foi referido no **capítulo 3**, dispõe de três interfaces: resultados e pesquisa, dados demográficos e informação documental. A informação documental é apresentada ao utilizador através de um serviço da Glintt.

A interface de resultados e pesquisa é bastante simplista e tenta apresentar os resultados tipo Google. Por esta razão a interface em causa apresenta apenas os seguintes componentes:

- **Caixa de texto**: onde o utilizador pode colocar todo o texto que pretender pesquisar;
- **Botão de pesquisa**: responsável por executar o processo de pesquisa. Para fazê-lo recolhe o valor da caixa de texto e das datas, caso existam, e executa o método de pesquisa com esses mesmos valores;
- **Área de resultados**: local onde todos os resultados são apresentados em lista. Esta área só é preenchida após executada uma pesquisa. Caso não seja encontrado nenhum resultado, o utilizador receberá uma mensagem apropriada nessa mesma área;
- **Botão de pesquisa avançada**: aqui estão presentes alguns filtros que podem ajudar o utilizador a restringir os resultados. Funcionam também como fator único de pesquisa, isto é, se o utilizador apenas quiser pesquisar, por exemplo, por intervalos de datas, poderá fazê-lo sem qualquer adição de informação na caixa de texto, apenas modificando o campo das datas;
- **Paginação**: este componente apenas aparece caso o número de resultados exceda o limite definido de quantidade máxima de resultados por página (*default*=5). É responsável por mostrar as páginas por onde o utilizador pode navegar.

Apresentam-se a seguir duas imagens da interface em causa:

Figura 14 - Interface de resultados e pesquisa

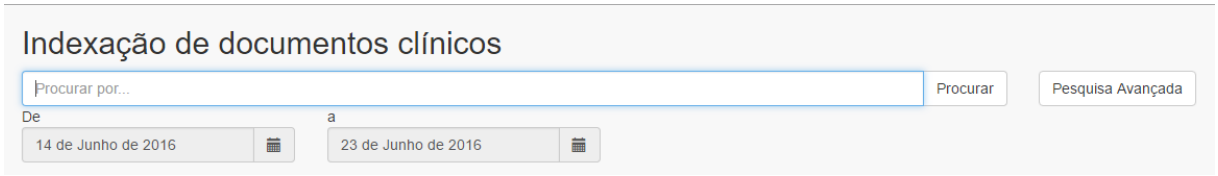


Figura 15 - Pesquisa avançada

Para cada resultado pode ser visto o nome e a data de nascimento do doente seguidos de pequenos excertos de texto onde os resultados da pesquisa se encontram. Nesse texto, os termos usados pelo utilizador na pesquisa aparecem a negrito. Por fim, logo após esse texto, aparecem os *links* responsáveis por redirecionar o utilizador para as interfaces da informação demográfica e dos documentos.

Se o utilizador clicar no *link* de informação demográfica do paciente é redirecionado para a página correspondente. Nesta é apenas apresentada informação demográfica e não possui qualquer tipo de botão ou outra opção semelhante, é apenas para visualização. Caso o utilizador queira voltar para a página de pesquisa, poderá fazê-lo usando a opção de voltar para trás do browser. Após fazê-lo visualizará a última pesquisa efetuada. Uma imagem da interface da informação demográfica é apresentada a seguir.

JOAO DE SA BALÃO CALISTO CORREIA

Figura 16 - Interface de informação demográfica

Por fim, no ecrã de visualização de documentos é usado um serviço da Glintt, que apenas mostra o documento em causa, seja ele PNG, PDF Word, etc.

Esta interface foi pensada para ser intuitiva (tipo Google) e, por isso, é o mais simplista possível de forma a facilitar o trabalho aos utilizadores.

4.8 Ferramentas e Tecnologias

Ao longo deste projeto foram utilizadas diversas ferramentas e tecnologias das quais algumas já foram referidas. Apesar de parte delas não estarem diretamente ligadas à aplicação em causa, ajudaram em testes de algumas partes do projeto. Com isto, neste tópico pretende-se resumir quais as ferramentas e tecnologias utilizadas e quais as suas utilidades para este projeto. Estas encontram-se a seguir:

- **Archetype Editor**: ferramenta utilizada para criar os arquétipos necessários para o desenvolvimento do *template*;
- **Template Designer**: ferramenta utilizada para a criação de *templates*;
- **Insomnia**: extensão do Google Chrome que serve para testar/fazer pedidos REST a um servidor. Utilizado para testar os pedidos ao *EHRServer*;
- **Fiddler**: ferramenta que guarda o tráfego HTTP entre outras funcionalidades. Utilizada juntamente com o Insomnia para testar pedidos ao *EHRServer*;
- **.NET**: *framework* utilizada para desenvolver a aplicação;
- **Visual Studio**: ferramenta onde se desenvolveu a aplicação;

- **SQLNavigator**: usado para gerir e testar *queries* relativas à base de dados da Glintt;
- **MySQL Workbench**: usado para gerir e testar *queries* relativas à base de dados do *EHRServer*, o EHR;
- **Solr**: usado para pesquisa e indexação da informação;
- **Solrnet**: cliente para o *Solr* em c#.

Como é possível observar a quantidade de ferramentas e tecnologias usadas foi bastante extensa, no entanto cada uma delas teve a sua importância numa determinada etapa do projeto. Uma vez que o estudo e desenvolvimento das partes relacionadas com o EHR levou mais tempo que o previsto, as ferramentas Fiddler e Insomnia foram sem dúvida as mais úteis para se conseguir avaliar da melhor forma os pedidos realizados tanto ao EHR como os da própria aplicação.

4.9 Resumo

Este capítulo teve como objetivo descrever com pormenor todas as etapas ao longo do projeto, as decisões tomadas e as dificuldades encontradas. Assim, com a **figura 17** pretende-se resumir a ordem de execução da aplicação.

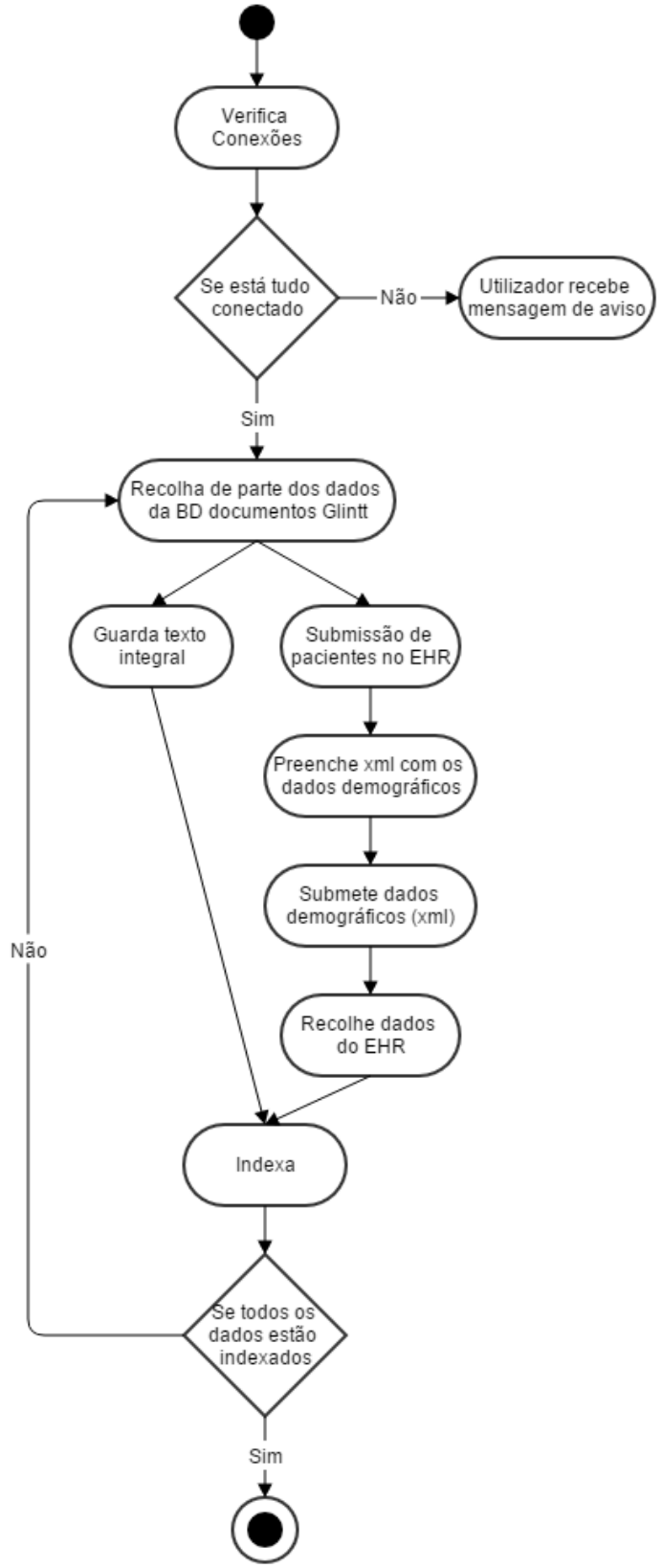


Figura 17 - Diagrama de atividades da aplicação

Inicialmente é feita uma verificação das conexões, ou seja, a aplicação analisa se o *Solr*, o *EHRServer* e a base de dados da Glintt estão em funcionamento antes de se iniciar os processos essenciais. Após esta verificação, caso estejam todos em funcionamento, a aplicação segue sem problemas, caso pelo menos um deles não esteja a funcionar, o utilizador recebe uma mensagem a informar do sucedido.

Uma vez concluídas as verificações, entra-se nos processos essenciais. Como primeiro passo é feita uma *query* à BD de documentos da Glintt para recolha dos dados. O texto integral dos documentos é guardado em memória e os dados demográficos e ids partem para o EHR.

Para os dados essenciais de um doente (primeiro e último nome, data de nascimento, sexo, etc) e, através dos pedidos REST do *EHRServer*, são submetidos os pacientes. É de lembrar que aqui ainda não estão a ser utilizados arquétipos ou *templates*, apenas se está a submeter dados essenciais para o *EHRServer* poder identificar o doente.

Posteriormente, a informação dos pacientes é preenchida num xml necessário para a submissão dos dados através do pedido REST seguinte. Uma vez preenchido o xml é feita a sua submissão no *EHRServer*. Estes dados que vão no xml já seguem o *template*. Após esta submissão o processo de mapeamento encontra-se concluído.

Por fim segue-se o processo de indexação, o qual após recolher os dados submetidos no EHR, indexa-os juntamente com o texto integral do documento armazenado em memória. Para a indexação são editados ficheiros de configuração do *Solr* onde são definidos os parâmetros da mesma.

Como é possível observar há um armazenamento temporário de dados em memória, o que não permite que esta execução seja feita com todos os dados de uma vez. Por esta razão a execução está dividida em partições, ou seja, este processo repete-se várias vezes até indexar todos os dados.

O tamanho das partições é definido pelos utilizadores no ficheiro **Web.config**. Este tamanho representa o número de documentos que são lidos em cada partição, ou seja, se uma partição é de tamanho 100 significa que 100 documentos são lidos da base de dados. Quando nos diagramas se refere a parte dos dados está-se a falar das partições. Por exemplo, recolha de parte dos dados da BD de documentos da Glintt pode significar que apenas 500 documentos estão a ser lidos, enquanto que os restantes serão lidos no próximo ciclo de execução.

Após todos os dados serem indexados, a aplicação está pronta a ser usada pelo utilizador. Neste sentido, após uma pesquisa feita pelo mesmo, segue-se a construção da *query*, a ordenação dos resultados e a respetiva formatação. No fim os resultados são apresentados na interface, a qual permite ao utilizador visualizar tanto dados demográficos como dados documentais.

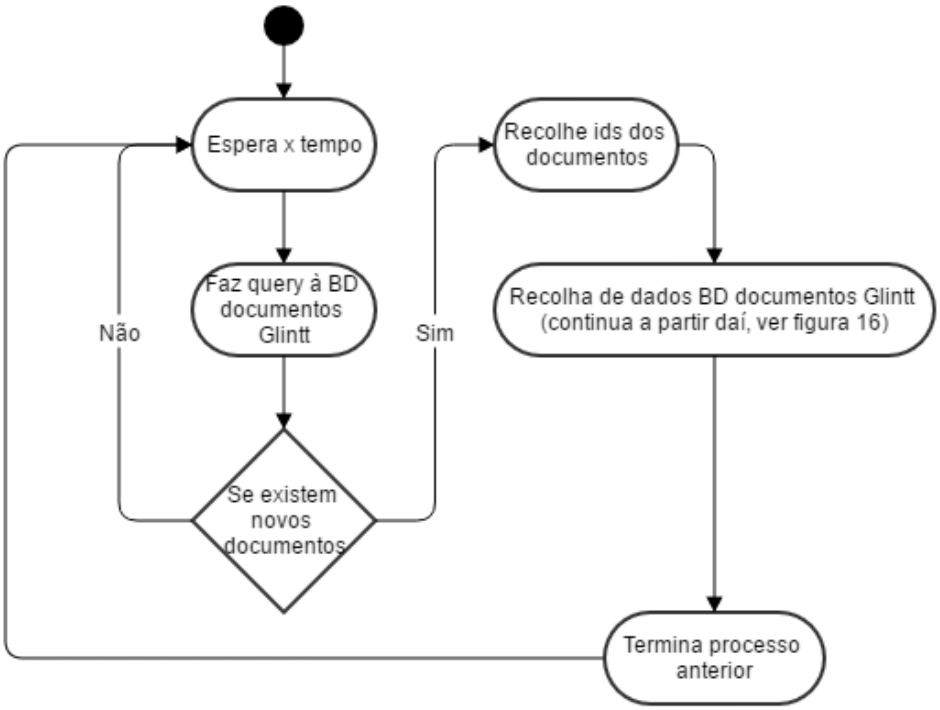


Figura 18 - Diagrama de atividades para a atualização de índices

Ao longo do tempo são feitas verificações para analisar se novos dados necessitam de ser indexados, como pode ser observado pela **figura 18**. Caso existam novos dados estes seguem o percurso desde a recolha de dados da BD documentos da Glintt até ao final do processo. Uma vez no final do processo, os índices ficam atualizados. Após esta etapa a aplicação espera x tempo (definido pelo utilizador no **Web.config**) até nova verificação.

Capítulo 5

Simulação e Resultados

Neste capítulo serão apresentados os resultados experimentais e os respetivos cenários nos quais estes foram aplicados.

5.1 Introdução

A ideia de testar a aplicação e de simular algumas situações da execução da mesma promove a resolução de diversos problemas. Tendo isto em mente, os testes realizados não foram só os que serão apresentados nos tópicos seguintes, mas também outros que foram efetuados ao longo do desenvolvimento e que contribuíram para o melhoramento da aplicação.

Assim sendo, nesta aplicação, os testes realizados serviram principalmente para observar se os tempos de indexação e de mapeamento entre os dados da BD de documentos da Glintt e o EHR se encontravam dentro de valores razoáveis. Rapidamente se chegou à conclusão que o tempo gasto para cada uma destas tarefas iria depender em muito da quantidade e do tamanho dos documentos em causa. À parte destes fatores, ambas as etapas estariam ainda dependentes da máquina na qual seriam executadas.

Tendo em consideração cada um dos pontos referidos anteriormente foram realizados alguns testes para averiguar a eficiência da aplicação.

5.2 Cenário

Para os testes realizados foram utilizados cerca de 3680 documentos da BD de documentos da Glintt. Além disto, com o fim de testar se diferentes máquinas, com diferentes características,

apresentam tempos de execução diferentes, foram realizadas experiências em duas máquinas com características diferentes. A seguir são apresentadas as características de cada uma das máquinas.

	Máquina 1	**Máquina 2**
Processador	Genuine Intel® CPU U7300 @1.30GHz	Intel® Core™ i7-4510U CPU @ 2.00GHz
RAM	4GB	8GB
Sistema Operativo	Windows 7 64bits	Windows 10 64 bits

Tabela 5 - Características das máquinas

Esta ideia de testar com diferentes máquinas surgiu uma vez que, em testes iniciais, o tempo de execução estava a ser demasiado grande e, por isso colocou-se a seguinte dúvida: que processo estará a influenciar mais o tempo de execução? Rapidamente se chegou à conclusão que a demora acontecia no mapeamento dos dados para o EHR. Em conversação com Pablo Pazos obteve-se a seguinte informação:

"…it really depends on the server hardware. How many commits in parallel can be handled by the server would depend on the number of CPUs available, processing speed might depend on CPU speed and available RAM memory, and also on disk speed.", Pazos [28]

Uma vez que depende do *hardware* faz todo o sentido realizar este tipo de testes.

Outro facto que poderá alterar a velocidade de execução é o tamanho de cada partição, ou seja, a quantidade de documentos que serão lidos numa sequência de execução. É necessário realçar que quanto maior for o tamanho das partições maior será a memória ocupada, daí a necessidade de usar partições, para que não haja um consumo excessivo da mesma. Assim sendo, foram realizados testes com partições de diferentes tamanhos: 100, 500 e 1000.

No tópico seguinte são apresentados os resultados experimentais dos testes efetuados.

5.3 Resultados Experimentais

	Máquina 1			**Máquina 2**		
Tamanho da partição	100	500	1000	100	500	1000
Processo de Indexação	1min20seg	33seg	25seg	40seg	16seg	11seg
Processo de Mapeamento	3h50min	2h04min	3h37min	1h17min	56min	1h13min
Todos os processos	3h52min	2h05min	3h38min	1h18min	57min	1h14min

Tabela 6 - Resultados Experimentais

Como pode ser observado pela tabela anterior, os resultados obtidos na **máquina 1** são todos piores dos que os obtidos na **máquina 2**. Assim valida-se a afirmação de Pablo Pazos, que a velocidade do processo de submissão de dados no EHR (mapeamento) depende grandemente do *hardware*. O processo de indexação também é afetado com esta mudança de máquina, no entanto como este é bem mais rápido que o processo de mapeamento, as diferenças não se fazem sentir na execução da aplicação.

Analisando agora o tamanho das partições, e falando apenas do processo de mapeamento, é possível observar que do tamanho 100 para 500 em ambas as máquinas há um aumento da velocidade de execução. Isto acontece, pois, ao aumentar o tamanho das partições, a quantidade de vezes que todos os processos executam é menor. No caso do tamanho 100 a quantidade de vezes que os processos executam é superior, diminuindo com isto a velocidade de execução da aplicação. Sendo assim, então porque não há aumento da velocidade quando o tamanho das partições é 1000?

Ao aumentar em demasia o tamanho das partições cria diversas desvantagens em vários pontos da execução. Por exemplo, na recolha de dados da BD de documentos da Glintt, um aumento demasiado grande nas partições significa que a *query* terá que retornar também mais resultados e, por isso, levará mais tempo para concluir essa tarefa. Outro aspeto é a submissão de dados no *EHRServer*, porque existe um limite da quantidade de dados que podem ser submetidos ao mesmo tempo, ou seja, imaginando que só se podem submeter 10 documentos ao mesmo tempo, aumentar em demasia o tamanho das partições não surtirá efeito, pois no máximo estarão 10 documentos a ser submetidos enquanto os restantes se encontram em espera.

Relativamente ao processo de indexação, pode-se concluir que não tem grande interferência na execução total da aplicação. No entanto, consegue-se verificar na mesma o seu aumento de desempenho com o aumento do tamanho das partições. Isto acontece, pois nesta etapa não há recolhas de informação da BD de documentos da Glintt nem submissão de dados no *EHRServer*, ou seja, a lógica da diminuição do número de processos executados no total com o aumento do tamanho de partições verifica-se e, com menos processos a serem executados, a velocidade aumenta.

Em suma, para um melhor desempenho desta aplicação é necessário ter em consideração dois aspetos fundamentais: a máquina onde é executada e o tamanho das partições usadas. Quanto ao tamanho das partições, o valor ótimo pode diferir dependendo da máquina usada. Por esta razão o tamanho das partições é também configurável no ficheiro **Web.config**. De acordo com estas experiências facilmente se conclui que o resultado ótimo pertence à **máquina 2** com duração de 57 minutos, para partições de tamanho 500.

Capítulo 6

Conclusões e Trabalho Futuro

Neste capítulo serão apresentadas algumas conclusões sobre os objetivos atingidos e também dos não completados de forma a avaliar qual o trabalho futuro que se pensa ser possível desenvolver.

6.1 Conclusão e Satisfação dos Objetivos

Os objetivos desta aplicação procuravam melhorar o acesso à informação clínica e a rapidez nesse acesso. Para conseguir atingir estes pontos, tinha-se então como objetivo criar uma aplicação de indexação e pesquisa de informação clínica que permitisse que vários tipos de documentos provenientes de diferentes fontes fossem pesquisáveis na mesma plataforma.

Com o fim de levar a cabo uma aplicação destas, foi efetuada pesquisa na área para se perceber quais os passos a dar assim como perceber como se poderia melhorar cada um deles. Após isto passou-se para uma análise tecnológica, onde se avaliou as vantagens e desvantagens de diversas ferramentas e as implicações que estas poderiam ter na solução em causa. Chegou-se à conclusão que o *Apache Solr* era a melhor ferramenta a ser utilizada nesta aplicação. Para unificar os dados provenientes de diferentes fontes optou-se pelo uso do EHR.

Posteriormente a este estudo inicial e com o fim de construir uma aplicação bem estruturada, procedeu-se à definição da arquitetura da aplicação. Uma vez que a aplicação deveria ser algo como o Google, direcionada para dados clínicos, decidiu-se dividi-la em duas partes: o lado do cliente e o lado servidor. No lado do cliente há apenas preocupação em mostrar os ecrãs ao utilizador. Já o lado do servidor é responsável por todo o tratamento de dados. Este último ficou dividido em três módulos: mapeamento, indexação e pesquisa. Cada um deles com um papel

importante na conclusão dos objetivos. Uma vez decidida a estrutura da aplicação começou-se a desenvolvê-la.

Ao longo do desenvolvimento foram encontradas diversas dificuldades sendo que, perceber e funcionar com o EHR foi a tarefa que mais problemas gerou e que, por isso, mais esforço exigiu.

Após a conclusão da mesma prosseguiu-se para uma fase de testes onde se realizaram imensas retificações de forma a melhorar a aplicação. Com esta fase concluída, deu-se por completa a aplicação. No entanto ainda há bastante trabalho que pode ser feito no âmbito desta solução, como pode ser verificado na secção seguinte.

A aplicação cumpre com os objetivos inicialmente propostos e, por isso, fornece às entidades médicas um local centralizado onde pode ser realizada pesquisa sobre os diversos documentos. Além disso aceita diversos formatos e em todos eles a pesquisa é possível (por exemplo, no caso especial das imagens) e rápida. No que diz respeito a diferentes fontes de dados, a aplicação só faz uso de uma, mas a adição de outras é possível sem grandes alterações, uma vez que foi feita a pensar nesta situação.

Além disto, e como pode ser observado no **capítulo 5**, a rapidez desta solução depende em muito da máquina na qual será executada, assim como do tamanho das partições. Por esta razão, de forma a atingir um melhor desempenho, é importante encontrar um valor ótimo para o tamanho das partições que depende da máquina em que vai ser executada. Apesar desta restrição da capacidade da máquina e do tamanho das partições, é de levar em conta que este passo apenas executa uma vez no início da aplicação, ou seja, não afeta a velocidade de pesquisa e, por isso, o utilizador pode efetuar pesquisas rápidas sem que estas estejam dependentes do tamanho das partições.

De uma forma mais pormenorizada, a utilização do EHR permite juntar num só local toda a informação e, nesse ponto de vista, o processo de recolha e indexação dos dados torna-se muito mais rápido. Assim sendo, realça-se a importância da sua utilização. O facto de aumentar a rapidez pode não se notar neste protótipo, pois apenas faz uso de uma fonte de dados, no entanto, prevê-se que com o aumento das fontes o propósito do EHR tornar-se-á mais visível, pois diminuirá o tempo que se perderia a fazer pedidos a cada uma das diferentes fontes.

Em suma, todos os objetivos foram atingidos e que por isso este protótipo deve ser aproveitado para o desenvolvimento de uma aplicação que englobe várias fontes e não apenas a BD de documentos da Glintt. Com isto a informação deixa de estar dispersa encontrando-se apenas no EHR, o qual se torna muito mais simples de indexar.

6.2 Trabalho Futuro

Após a conclusão deste projeto é possível observar que, apesar de cumprir com os objetivos, o mesmo pode ser melhorado em diversos pontos.

Começando pela pesquisa, a aplicação atualmente, além da pesquisa por palavras chave e por datas, poderia ainda apresentar outros tipos. Dentro destes poderia ser interessante realizar uma pesquisa por hierarquia de doenças, ou seja, utilizar árvores que relacionam termos clínicos para pesquisar, não só o termo de pesquisa introduzido pelo utilizador, mas também termos relacionados com o esse. Para tal recorrer-se-ia ao uso das nomenclaturas como o SNOMED CT, ICD-9 e MeSH referidas no tópico do estado da arte. Além disto, seria ainda interessante pesquisar por regiões, isto é, uma vez que se indexa informação demográfica, procurar por dados de um paciente apenas pesquisando pelo valor exato da *string*, por exemplo, correspondente ao local de residência, perder-se-ia muito informação. Assim sendo, pretende-se que uma pesquisa, por exemplo, por distrito, retorne resultados também com valores referentes às freguesias que a ele pertencem.

Ainda no âmbito da pesquisa, seria interessante dar uso a dicionários de sinónimos de termos médicos e de *spellcheck*. No caso do dicionário de sinónimos, a sua importância reside no termo de pesquisa, ou seja, pesquisar por um termo deveria retornar não só resultados com esse termo, mas também com sinónimos do mesmo. Por exemplo, pesquisar por "dor de barriga" deveria retornar resultados também com "dor abdominal", assim como outros sinónimos existentes. No caso do *spellcheck*, serviria apenas para evitar pesquisas erradas, com isto, seria interessante ver sugestões de pesquisa como as apresentadas no Google. Para terminar esta parte dos dicionários, é importante referir que o trabalho futuro neste ponto apenas reside, no caso do dicionário de sinónimos, na criação de um dicionário para o efeito, pois a aplicação já lê estes dicionários, só que os mesmos encontram-se vazios. Relativamente ao *spellcheck* deveriam ser feitas pequenas modificações à aplicação para ser possível observar as referidas sugestões.

Outro aspeto, não tão visível para o utilizador, mas que sem dúvida seria mais correto, é a estrutura do *template* usado na aplicação. Este deveria estar dividido em mais arquétipos, por exemplo, um para a informação demográfica, outro para guardar os identificadores do documento e a data do mesmo, e outro para guardar os identificadores do paciente. Desta forma, apesar de não haver grandes alterações na velocidade de execução da aplicação, esta estaria mais preparada para receber outro tipo de ficheiros e de outras fontes, pois os arquétipos poderiam ser reaproveitados para a criação de outros *templates*. Com a implementação atual, poderia ser possível esse reaproveitamento, mas apenas em casos muito específicos.

Por fim, e como última tarefa futura, era interessante conseguir uma avaliação da aplicação, isto é, realizar testes que validem a utilidade/necessidade da solução proposta. Estes testes teriam que ser realizados com utilizadores que atualmente estão em contacto com sistemas de pesquisa médicos para que fosse possível comparar esses sistemas com esta aplicação.

Referências

[1] U. S. N. L. o. Medicine. (2016). *Home - MeSH - NCBI*. Available: http://www.ncbi.nlm.nih.gov/pubmed/

[2] S. T. Inc. (2016). *Sphinx | Open Source Search Engine*. Available: http://sphinxsearch.com/

[3] s. IT. (2016). *DB-Engines Ranking*. Available: http://db-engines.com/en/ranking/search+engine

[4] E. Hatcher and O. Gospodnetic, *Lucene in Action (In Action series)*: Manning Publications Co., 2004.

[5] Ihtsdo. (2016). *SNOMED CT Document Library*. Available: http://ihtsdo.org/fileadmin/user_upload/doc/

[6] T. B. Sam Heard, "O que é openEHR?," 2016.

[7] C. R. L. Coelho, "Acesso e Recuperação de Informação em Catálogos Bibliográficos Online," Faculdade de Engenharia da Universidade do Porto, 2014.

[8] M. Lesk, "The seven ages of information retrieval," ed, 1996.

[9] C. D. Manning, P. Raghavan, H. Sch\, \#252, and tze, *Introduction to Information Retrieval*: Cambridge University Press, 2008.

[10] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*: Addison-Wesley Longman Publishing Co., Inc., 1999.

[11] D. Hiemstra, *Using language models for information retrieval*: Taaluitgeverij Neslia Paniculata, 2001.

[12] D. L. Lee, H. Chuang, and K. Seamons, "Document ranking and the vector-space model," *Software, IEEE,* vol. 14, pp. 67-75, 1997.

[13] S. E. Robertson and K. S. Jones, "Relevance weighting of search terms," *Journal of the American Society for Information science,* vol. 27, pp. 129-146, 1976.

[14] R. Baeza-Yates and G. Navarro, "XQL and proximal nodes," *Journal of the American Society for Information Science and Technology,* vol. 53, pp. 504-514, 2002.

[15] D. A. Hull, "Stemming algorithms: A case study for detailed evaluation," *JASIS,* vol. 47, pp. 70-84, 1996.

[16] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*: Manning Publications Co., 2010.

[17] T. Grainger and T. Potter, *Solr in action*: Manning Publications Co., 2014.

[18] C. Mattmann and J. Zitting, *Tika in Action*: Manning Publications Co., 2011.

[19] T. A. S. Foundation. (2016). *Apache Solr*. Available: http://lucene.apache.org/solr/

[20] R. Gheorghe, M. L. Hinman, and R. Russo, *Elasticsearch in Action*: Manning Publications Co., 2015.

[21] D. Rothwell, F. Wingert, R. Cote, R. Beckett, and J. Palotay, "Indexing Medical Information: The Role of SNOMED," in *Proceedings/the... Annual Symposium on*

Computer Application [sic] in Medical Care. Symposium on Computer Applications in Medical Care, 1989, pp. 534-539.

[22] C. N. C. f. H. Statistics. (2016). *ICD - ICD-9-CM - International Classification of Diseases, Ninth Revision, Clinical Modification*. Available: http://www.cdc.gov/nchs/icd/icd9cm.htm

[23] F. Lopes. (2016). *Classificação Internacional de Doenças (CID) - Portal da Codificação Clínica e dos GDH*. Available: http://portalcodgdh.min-saude.pt/index.php/Classifica%C3%A7%C3%A3o_Internacional_de_Doen%C3%A7as_(CID)

[24] R. Correia, "openEHR," Faculdade de Medicina da Universidade do Porto2015.

[25] pablo, "CaboLabs Healthcare Informatics, Standards and Interoperability," 2016.

[26] S. R. Cassandra Targett. (2016). *Apache Solr Tokenizers*. Available: https://cwiki.apache.org/confluence/display/solr/Tokenizers

[27] L. Imagination. (20-06-2016). *Apache Lucene - Scoring*. Available: https://lucene.apache.org/core/2_9_4/scoring.html#Score

[28] P. Pazos. (2016, 22/06/2016). *Time committing a lot of xml*. Available: https://github.com/ppazos/cabolabs-ehrserver/issues/360#issuecomment-227503058

Anexos

Anexo A

Figura 19 - Modelo EER do EHR (parte 1)

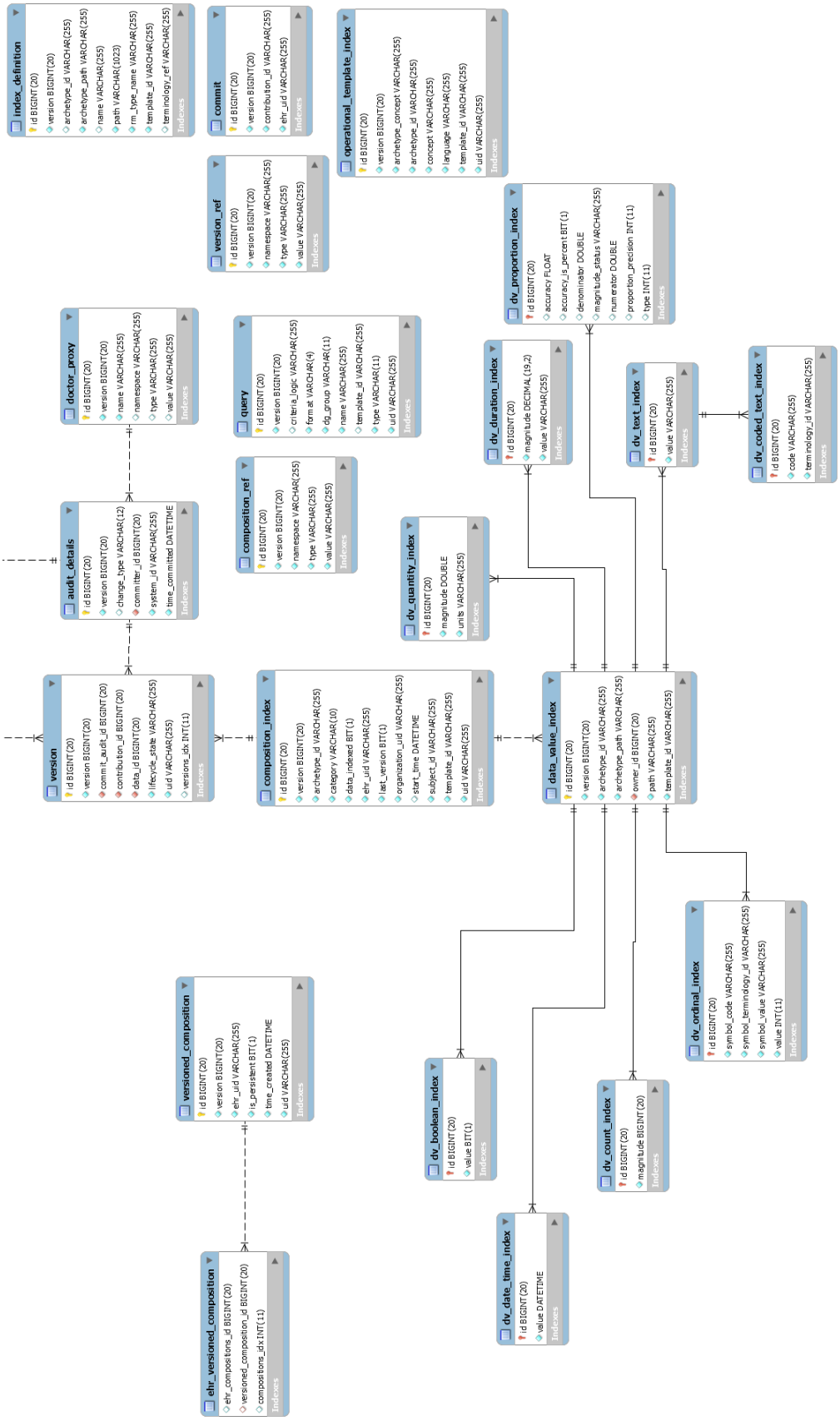


Figura 20 - Modelo EER do EHR (parte 2)

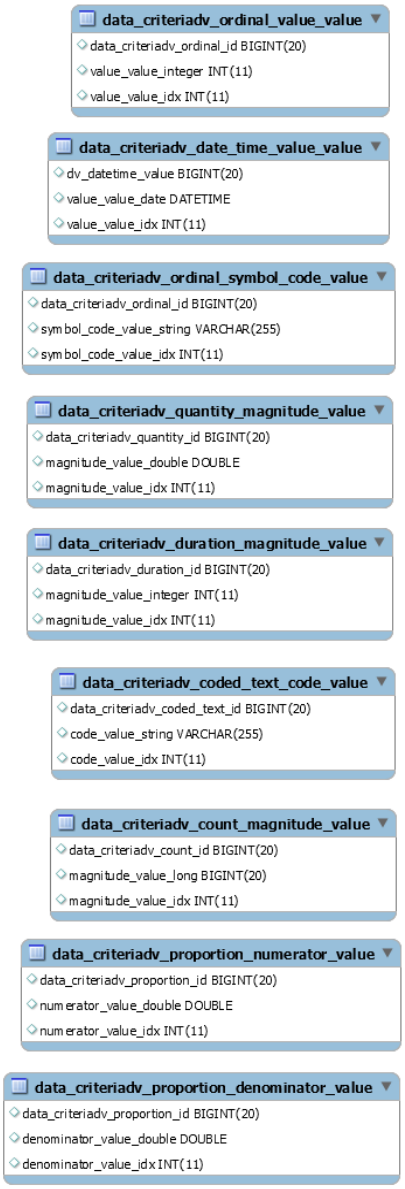


Figura 21 - Modelo EER do EHR (parte 3)

Anexo B

```
<add key="TimeBetweenUpdates" value="5000"/>
<add key="ResultsPerPage" value="5" />
<add key="LimitOfPages" value="5" />
<add key="ChunkSize" value="500" />
<add key="VersionsFolder" value="C:\Users\...\versions" />
<add key="SolrCore" value="http://localhost:8983/solr/ehr" />
<add key="EHR_rest" value="http://localhost:8090/ehr/rest" />
<add key="Eresults_v2_db" value="Data Source=(DESCRIPTION = (ADDRESS =
(PROTOCOL = TCP)(HOST = ****)(PORT = ****))(CONNECT_DATA =
(SERVICE_NAME = ****)));User Id=****;Password=****;" />
<add key="EHR_db"
value="server=****;port=****;database=ehrserver;userid=****;password=*
***;" />
<add key="FileView" value="http://localhost:8080... Viewer.aspx?" />
<add key="demographic_template"
value="C:\Users\...\demographic_patient.xml" />
```

As linhas em cima são um excerto do ficheiro **Web.config**, que é o ficheiro de configurações da aplicação. Os campos que se podem observar são os valores que têm interesse ser editados. A seguir apresenta-se uma breve explicação de cada um deles.

TimeBetweenUpdates	Tempo entre atualizações de índices (em milissegundos)
ResultsPerPage	Quantidade de resultados apresentados em cada página
LimitOfPages	Quantidade de botões da paginação que aparecerão na interface
ChunkSize	Tamanho das partições
VersionsFolder	Diretório onde estão armazenadas todas as versões dos documentos submetidos no EHR
SolrCore	URL onde está a correr o *Solr*
EHR_rest	URL onde está a correr o *EHRServer*
Eresults_v2_db	Dados para abrir uma conexão à BD de documentos da Glintt
EHR_db	Dados para abrir uma conexão à base de dados EHR
FileView	URL responsável por abrir os documentos no serviço da Glintt
demographic_template	Local onde se encontra o xml que segue o *template* do EHR

Tabela 7 - Parâmetros do Web.config