LISBOA SCHOOL OF ECONOMICS & MANAGEMENT

MESTRADO

DECISÃO ECONÓMICA E EMPRESARIAL

TRABALHO FINAL DE MESTRADO

RELATÓRIO DE ESTÁGIO

UM ALGORITMO PARA MELHORAR A GESTÃO E O ACESSO DOCUMENTAL UTILIZANDO MEDIDAS DE SIMILARIDADE

DIOGO MANUEL DA SILVA MAGALHÃES

SETEMBRO - 2013

LISBOA SCHOOL OF ECONOMICS & MANAGEMENT

MESTRADO EM

DECISÃO ECONÓMICA E EMPRESARIAL

TRABALHO FINAL DE MESTRADO

RELATÓRIO DE ESTÁGIO

UM ALGORITMO PARA MELHORAR A GESTÃO E O ACESSO DOCUMENTAL UTILIZANDO MEDIDAS DE SIMILARIDADE

DIOGO MANUEL DA SILVA MAGALHÃES

ORIENTAÇÃO:

PROFESSOR DOUTOR JOÃO PAULO VICENTE JANELA
ENGENHEIRO RODRIGO SERAFIM

SETEMBRO - 2013

AGRADECIMENTOS

Quero agradecer desde já à coordenação do Mestrado, em especial, à Prof.ª Margarida Vaz Pato que me ajudou na escolha do estágio e sempre se mostrou disponível para quaisquer esclarecimentos.

Um agradecimento também para a Prof.ª Nicoletta Rosati que numa primeira fase me orientou, não tendo sido mais tarde possível por motivos de saúde, e ao Prof. João Janela, pela sua disponibilidade em me orientar já em fase mais avançada do estágio.

Um agradecimento a todos os colaboradores da Quidgest, S.A., em especial ao Prof. João Paulo, responsável da Quidgest, S.A., pela oportunidade de realização do estágio e ao departamento de I&D pela sua ajuda, esclarecimentos e boa disposição durante a realização do mesmo.

Por fim, quero agradecer à minha família, namorada, amigos e colegas pelo apoio constante que me deram ao longo destes anos.

A todos, um muito obrigado.

RESUMO

Este trabalho sintetiza essencialmente o grande foco do estágio curricular que realizei na Quidgest, S.A., no departamento de Investigação e Desenvolvimento.

O estágio proporcionou um grande contacto com *software* de escrita de código, sendo que, a minha principal função, foi a programação de funcionalidades para a página do Qsearch.

Ao longo do estágio, foi também avaliado o desempenho da página sempre que novas funcionalidades eram adicionadas.

Neste trabalho, encontra-se uma pequena definição do estágio logo na Introdução (Capítulo 1).

De seguida, na Contextualização do Problema (Capítulo 2), temos um breve enquadramento do problema no contexto real e também um enquadramento teórico.

O grosso do trabalho, que é o Desenvolvimento e Análise da Solução (Capítulo 3), contém a recolha de dados (amostra), uma breve introdução às plataformas utilizadas para programação, a metodologia utilizada e uma descrição mais pormenorizada do trabalho realizado na empresa, bem como da página do Qsearch e da sua performance.

A parte final, termina com as habituais conclusões (Capítulo 4).

ABSTRACT

This report synthesizes essentially the major focus of the curricular traineeship that I realized in Quidgest, SA, in the department of Research and Development.

The traineeship provided a great contact with writing software code, and my main function was the programming features for the page of QSearch.

Along the traineeship, it was also rated the performance of the page whenever new features were added.

In this report you will find a small description of the traineeship right at Introduction (Chapter 1).

Then, in the Contextualization of the Problem (Chapter 2), there is a brief framing of the problem in a real context and also a theoretical framing.

The most important part of the paper, which is the Development and Analysis of the Solution (Chapter 3), contains the collection of data (sample), a brief introduction to the platforms used for programming, the methodology used and a more detailed description of the work performed in the company as well as the QSearch page and its performance.

The final part contains the usual conclusions (Chapter 4).

ÍNDICE

INTRODUÇÃO

O estágio que realizei na Quidgest, S.A. teve como objectivo definir medidas de semelhança entre documentos, propostas, currículos e outros documentos e aferir a semelhança entre estes em função da utilização de termos comuns, objectivos, gostos, competências, nomes próprios, entre outros. O estágio teve a duração de 3 meses, entre 15 de Janeiro e 14 de Abril do corrente ano. De modo a realizar o que me foi proposto, fui integrado no projecto Qsearch juntamente com alguns elementos da área de Investigação e Desenvolvimento. Para que fique mais claro, o projecto Qsearch é um projecto coordenado pela Quidgest em parceria com a FCT/UNL, com o objectivo de melhorar a gestão e o acesso documental através de 3 tecnologias fundamentais:

1. Algoritmos para análise de documentos de textos, a fim de extrair os seus meta-dados
2. Algoritmos avançados de pesquisa e ordenação
3. Pesquisa Multifacetada com interface de utilizador.

O caminho que foi seguido inicialmente, de modo a alcançar esse objectivo, foi a detecção de documentos duplicados, caso em que todos os campos de um documento seriam exactamente iguais aos de outro. Mais tarde, após a conclusão desta tarefa, seguiu-se a detecção de documentos similares, através da análise dos seus termos mais frequentes. De modo a que o utilizador tivesse acesso a essa informação, foi feito, conjuntamente com essas tarefas, o melhoramento do estilo da página do

projecto Qsearch, página essa que servia para o utilizador fazer pesquisas de documentos na intranet.

CONTEXTUALIZAÇÃO DO PROBLEMA

ENQUADRAMENTO DO PROBLEMA NUM CONTEXTO REAL

A melhoria de produtividade dos trabalhadores numa empresa é sempre um objectivo que se pretende atingir e hoje em dia com a enorme quantidade de documentos que existem nas bases de dados torna-se cada vez mais complicado encontrar o que pretendemos. De modo a conseguirmos facilitar a vida dos trabalhadores e a melhorar o seu rendimento, utilizámos a página do projecto Qsearch para levar a cabo pesquisas de documentos, fornecendo ao utilizador um ambiente muito mais amigável no qual a procura desses documentos se pudesse tornar mais rápida e intuitiva.

Às vezes sem darmos conta, outras vezes propositadamente, criamos documentos duplicados ou documentos em que grande parte do conteúdo se repete. Saber onde se encontram esses documentos, e quantos documentos duplicados ou similares esses documentos têm, são informações que também consideramos ser úteis aos utilizadores, tendo sido esse o grande foco do meu estágio.

ENQUADRAMENTO TEÓRICO/LITERATURA

A informação é cada vez mais registada directamente em meios digitais. A publicação e criação de conteúdos tornam-se mais fáceis e, consequentemente, informações

irrelevantes, de baixa qualidade e mesmo de baixa confiabilidade dão origem a um "lixo informacional" crescente.

Um dos principais campos de estudo da Ciência da Informação compreende o tratamento e organização da informação de forma a possibilitar resultados de pesquisa satisfatórios, atendendo à procura do utilizador, sem a interferência do "lixo informacional".

Um Sistema de Recuperação da Informação (SRI) deve analisar os documentos para saber os itens do seu conteúdo que são relevantes perante uma consulta do utilizador. O objectivo é atender de forma satisfatória as pesquisas do utilizador, fornecendo somente os resultados mais relevantes. Para atingir este objectivo, desenvolvem-se constantemente pesquisas envolvendo técnicas e algoritmos aplicáveis em SRI. Actualmente, com todo o aporte computacional disponível, os diversos programas de computador (software) podem valer-se de um processamento rápido para melhorar ainda mais a satisfação do utilizador no uso destes sistemas.

A análise de texto (*text analysis*) corresponde a uma área que envolve outras subáreas como, por exemplo, a mineração de texto (*text mining)* e a área de Processamento de Linguagem Natural (PLN). A PLN também é uma subárea da inteligência artificial e da linguística que estuda os problemas da geração e tratamento automático de línguas humanas naturais.

A mineração de texto (t*ext mining*) refere-se ao processo de obtenção de informação a partir de texto em línguas naturais. Se praticada em conjunto com a mineração de dados, que consiste em extrair informação de bases de dados estruturadas, a mineração de texto extrai informação de dados não estruturados ou semi-

estruturados. O texto corresponde à principal parte das muitas que podem compor um documento, e o seu tratamento, como um processo de criação dos índices, é explorado pelos SRIs.

O índice tem como objectivo a recuperação rápida da informação. A forma como se constrói, armazena e manipula o índice, muda de acordo com a tecnologia empregue e pela sua consequente evolução. Anteriormente, as CPUs eram lentas e a utilização de técnicas de compactação não seria interessante. Hoje, as CPUs já são mais rápidas, no entanto temos um armazenamento em disco rígido relativamente lento que, para ser contornado, necessita de uma diminuição do espaço de armazenamento ou mesmo da utilização de memórias mais rápidas como a RAM.

Basicamente, a criação do índice significa criar um dicionário de palavras utilizadas em todos os documentos da colecção e criar um índice invertido indicando em qual documento cada palavra aparece.

Com a criação deste índice torna-se mais rápida a busca de informações, quando comparada com o varrimento de todos os textos, palavra por palavra.

A maior parte dos SRI tem como base um modelo clássico ou um modelo estruturado.

Nos modelos clássicos, cada documento é descrito por um conjunto de palavras-chave representativas, também chamadas de termos de indexação, que procuram representar o assunto do documento e sumarizar o seu conteúdo de forma significativa. (BAEZA-YATES; RIBEIRO, NETO,1999).

Nos modelos estruturados, podem-se especificar, além das palavras-chave, algumas informações acerca da estrutura do texto. Estas informações podem ser as secções a ser pesquisadas, fontes de letras, proximidade das palavras, entre outras.

Entre os modelos clássicos podemos destacar o booleano, o vectorial e o probabilístico. O modelo booleano é baseado na teoria dos conjuntos e possui consultas especificadas com termos e expressões booleanas. Nas consultas são utilizados operadores lógicos como E, OU, NÃO para filtragem do resultado.

Apesar de ser um modelo bastante simples e muito utilizado, ele apresenta as seguintes desvantagens, segundo Baeza-Yates e Ribeiro (1999):

- A recuperação é baseada numa decisão binária sem noção de *matching* parcial;
- Não é fornecida nenhuma ordenação de documentos;
- A formulação da informação pretendida pelo utilizador em termos de uma expressão booleana é considerada complicada;
- As consultas booleanas formuladas pelos utilizadores são frequentemente simplistas;
- Consequentemente, o modelo booleano retorna poucos documentos em resposta às consultas;
- O uso de pesos binários é limitador.

Para contornar estas limitações, novos modelos são desenvolvidos tendo como base alguns destes modelos clássicos.

Um dos modelos que permite aferir similaridade entre documentos é o modelo vectorial. O vector é definido através do conjunto de documentos que formam a colecção. Todo o texto dos documentos é extraído e convertido num formato que permita a fácil manipulação. Toda a ordem de palavras é ignorada, o que pode ser interpretado como colocar todas as palavras de cada documento num saco separado (utiliza-se a expressão *bag of words*). Todas as palavras em cada saco são contadas

(processo de indexação) e o número de vezes que cada palavra aparece (forma mais simplista de dar valor ao peso) é armazenado num vector termo-por-documento.

Este é construído de forma que cada linha representa uma palavra (termo) e cada coluna representa um documento. Os valores contêm o peso dos termos para cada documento. Em geral, este tipo de vector é extenso e a maioria dos pesos dos termos é zero.

Sobre o uso de pesos no modelo vectorial, Baeza-Yates e Ribeiro-Neto (1999) apresentam algumas considerações:

- Pesos não binários podem considerar mais adequadamente *matchings* parciais;
- Estes pesos são utilizados para calcular um grau de similaridade entre a consulta e o documento;
- A fórmula com que são calculados os pesos, varia de implementação para implementação;

Cada documento (coluna) pode ser considerado como um vector ou como uma coordenada no espaço do vector multidimensional, em que cada dimensão representa um termo. Organizada a informação, estão disponíveis algumas medidas relevantes para a sua análise. O indicador *term frequency* (TF) corresponde à frequência com que um dado termo aparece no documento, sendo calculado através da fórmula

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}.$$

Na expressão anterior $n_{i,j}$ designa o número de ocorrências do termo *i* no documento *j*, pelo que o denominador corresponde ao número total de ocorrências de todos os termos considerados no docu*mento j.*

Já o inverse document frequency (IDF) é uma medida de grande importância para complementar a equação anterior, avaliando a importância do termo na colecção. É obtida dividindo a quantidade de documentos pelo número de documentos contendo o termo e, finalmente, calculando o logaritmo do resultado, i.e.

$$IDF_i = \log \frac{|D|}{\left|\{d_j : t_i \in d_j\}\right|}$$

Sendo:

|D| : total de documentos na colecção

$\left|\{d_j : t_i \in d_j\}\right|$: número de documentos onde o termo t_i aparece.

Na prática, para precaver a possibilidade de em dado momento não existir nenhum documento contendo o termo a pesquisar, caso em que existiria uma divisão por zero, a fórmula anterior é modificada para

$$IDF_i = \log \left[\frac{|D|}{\max\{1, \left|\{d_j : t_i \in d_j\}\right|\}}\right]$$

Combinando estas duas medidas obtém-se uma medida designada por *term frequency–inverse document frequency* (TF-IDF), calculada através da expressão

$$TFIDF_{i,j} = TF_{i,j} . IDF_i$$

A medida TF-IDF corresponde a uma medida estatística tipicamente utilizada para avaliar o quanto uma palavra é importante para um documento, relativamente a uma colecção. Esta importância aumenta proporcionalmente com o número de vezes que a palavra aparece e diminui de acordo com a frequência da palavra na colecção.

A análise semântica latente é uma técnica da PLN, relacionada com a manipulação de vectores do índice, para analisar a relação entre termos e documentos e decompor o vector de índice. O processo matemático utilizado é o SVD (*Simple Value*

Decomposition). Alguns autores e pesquisas também a chamam de *Latent Semantic Indexing* (LSI).

A LSI trabalha com vários vectores, criando desta forma uma matriz, onde nas linhas estão representados os termos indexados de cada documento e nas colunas o documento. Desta forma é criada uma relação descrita por uma matriz termo-documento. Explicando melhor esta relação, t_i seria a linha, d_j a coluna e O_{ij} o elemento da matriz que representaria o número de vezes que o termo i aparece no documento j.

Após ser criada esta matriz termo-documento, é aplicado o *Simple Value Decomposition* – SVD. Essa decomposição divide a matriz termo-documento em três matrizes: a matriz U que contém os termos, a matriz S que contêm os valores mais representativos da matriz termo-documento (os valores singulares) e a matriz V que contém os documentos. Depois de criadas estas três matrizes, é escolhido um tamanho (nível k) para trabalhar com as mesmas. Escolhido este valor, são criadas três matrizes (que serão chamadas U', S' e V') de nível k, a estas três novas matrizes é multiplicado o vector Q, que representa uma consulta. O resultado desta multiplicação será um vector cujo conteúdo é uma lista dos documentos mais relevantes para a consulta fornecida.

Os algoritmos que retornam similaridade entre documentos trabalham com métricas que retornam o quanto um documento é similar a outro. Existem diversos algoritmos e métricas utilizadas em fins diversos.

Na maioria dos estudos estatísticos deste tipo de problemas utilizam-se duas medidas básicas de similaridade: correlação e distância angular, baseada no cosseno. A

correlação de Pearson (ou só correlação) entre dois vectores retorna um valor entre -1 e 1. Se for 1, eles estão fortemente correlacionados, isto é, os valores de um vector podem predizer os valores do outro. Se for 0 não existe correlação. E se for -1 existe uma correlação inversamente proporcional. A distância angular baseada no cosseno é similar à correlação, retornando valores entre -1 e 1. Esta mede o ângulo entre dois vectores num espaço vectorial. Quanto mais próximo de 1 for o valor, mais similares são os dois vectores. Para apurar esta última medida de similaridade entre dois documentos num SRI utilizando VSM (Vector Space Model), calcula-se o cosseno do ângulo formado no vector termo-por-documento. No VSM padrão, quanto menor o ângulo, mais próximo de 1 será o cosseno e mais similar será o documento em relação àquele termo. Concretamente, define-se

$$sim(\overrightarrow{d_1}, \overrightarrow{d_2}) = \cos\left(\measuredangle(\overrightarrow{d_1}, \overrightarrow{d_2})\right) = \frac{\overrightarrow{d_1}.\overrightarrow{d_2}}{|\overrightarrow{d_1}|.|\overrightarrow{d_2}|} = \frac{\sum_i w_{i,1}.w_{i,2}}{\sqrt{\sum_i w_{i,1}^2}.\sqrt{\sum_i w_{i,2}^2}}$$

onde $w_{i,j}$ designa o peso do termo t_i no documento d_j.

Baeza-Yates e Ribeiro-Neto (1999) apresentam algumas outras observações sobre este modelo como um todo:

- É devolvido um conjunto ordenado de documentos, fornecendo uma melhor resposta à consulta.
- Documentos que têm mais termos em comum com a consulta tendem a ter maior similaridade;
- Aqueles termos com maiores pesos contribuem mais para o aumento de similaridade do que os que têm menores pesos;
- Documentos maiores são favorecidos;

- A similaridade calculada não tem um limite superior definido.

O uso de um SRI e de um algoritmo de *clustering* para agrupar documentos envolve o cálculo da distância entre estes documentos na matriz. Existem, além do cosseno e da correlação, outras medidas de similaridade, sendo que a distância euclidiana é também muito utilizada. A distância euclidiana entre dois documentos d_1 e d_2 é definida por:

$$dist(\overrightarrow{d_1}, \overrightarrow{d_2}) = \sqrt{\sum_i (w_{i,1} - w_{i,2})^2}$$

Sendo, $w_{i,j}$ o peso do termo t_i no documento d_j.

Para que a medida assim definida possa efetivamente ser designada por distância, gozando das propriedades matemáticas associadas a este tipo de aplicação, é necessário que, para quaisquer vectores x, y, z, sejam verificadas as seguintes propriedades:

1. $dist(x, y) \geq 0$ [Positividade]
2. $dist(x, y) = 0 \Leftrightarrow x = y$
3. $dist(x, y) = dist(y, x)$ [Simetria]
4. $dist(x, z) \leq dist(x, y) + dist(y, z)$ [Desigualdade triangular]

Mais uma vez o tamanho do documento tem grande influência quando se utiliza a distância euclidiana.

DESENVOLVIMENTO E ANÁLISE DA SOLUÇÃO

RECOLHA DE DADOS (AMOSTRA)

Os dados utilizados durante o estágio foram documentos que se encontravam disponíveis num servidor de ficheiros designado por *Fileserver*. No leque de documentos aí disponíveis encontram-se propostas, apresentações, currículos, contratos, manuais, entre outros. São documentos em diversos formatos, como por exemplo PDF, DOC, PPT, CSV, XML, TXT. O número total de documentos disponíveis nesse servidor é da ordem de grandeza dos 100.000. Este número é naturalmente variável pois são diariamente adicionados/eliminados documentos.

Em fases iniciais e de testes foi utilizado um menor número de documentos, de modo a encurtar o tempo necessário para o processo de indexação que, executado para a totalidade dos ficheiros disponíveis no *Fileserver*, pode demorar cerca de 6 horas. Por este motivo, a indexação foi frequentemente agendada para o período nocturno, permitindo prosseguir os testes e verificações no dia seguinte.

PLATAFORMA UTILIZADA PARA INDEXAÇÃO E PESQUISA: SOLR

O SOLR é uma plataforma de pesquisa empresarial de código aberto do projecto Apache Lucene. As suas principais características incluem a pesquisa de texto completo, destaque do(s) termo(s) pesquisados, pesquisa facetada, agrupamento dinâmico, integração de bases de dados e manipulação de documentos ricos (por

exemplo: Word, PDF). O SOLR é altamente confiável, escalável e tolerante a falhas, fornecendo a indexação distribuída, replicação e o balanceamento da carga da consulta. SOLR potencia a pesquisa e os recursos de navegação de muitos dos maiores sites de internet do mundo. É escrito em Java e é executado como um servidor de pesquisa de texto completo independente. Usa a biblioteca de busca Lucene Java no seu núcleo para a indexação de texto completo e pesquisa e tem a REST como HTTP/XML e JSON APIS que o tornam fácil de usar, a partir de qualquer linguagem de programação. De facto, o SOLR potencia uma configuração externa poderosa o que lhe permite ser adaptado a quase qualquer tipo de aplicativo sem codificação Java.

PLATAFORMAS UTILIZADAS PARA PROGRAMAÇÃO

VISUAL STUDIO

O Microsoft Visual Studio é um pacote de programas da Microsoft para desenvolvimento de software especialmente dedicado ao .Net Framework e às linguagens Visual Basic (VB), C, C++, C# (C Sharp) e J# (J Sharp). Também é um grande produto de desenvolvimento na área web, usando a plataforma do ASP.NET. As linguagens com maior frequência nessa plataforma são: VB.NET (Visual Basic.Net) e o C#.

ECLIPSE

Eclipse é um IDE desenvolvido em Java, seguindo o modelo *open source* de desenvolvimento de software. O projecto Eclipse foi iniciado na IBM que desenvolveu

a primeira versão do produto e doou-o como software livre para a comunidade. O gasto inicial da IBM no produto foi de mais de 40 milhões de dólares. Hoje, o Eclipse é o IDE Java mais utilizado no mundo. Possui como características marcantes o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em *plugins* e o amplo suporte ao programador, com centenas de *plugins* que procuram atender as diferentes necessidades identificadas pela comunidade de desenvolvimento de software.

Com o uso de *plugins*, pode ser usado não só para desenvolvimento em Java, mas também em C/C#, PHP, ColdFusion e Python.

ESCOLHA DA METODOLOGIA

Após a leitura de alguma documentação e de tutoriais sobre a arquitectura do SOLR e sobre como detectar documentos duplicados e similares, a metodologia que me pareceu mais adequada utilizar seria adicionar à pesquisa o método *MoreLikeThis*, disponível no SOLR.

O método *MoreLikeThis* constrói uma pesquisa baseada nos termos detectados no documento. A utilização do método *MoreLikeThis* é feita através da escolha dos seguintes parâmetros:

Parâmetro	**Descrição**
Mlt.fl	Campos a ser utilizados na análise de similaridade
Mlt.mintf	Frequência mínima do termo

Mlt.mindf	Frequência mínima de documento
Mlt.minwl	Comprimento mínimo do termo
Mlt.maxwl	Comprimento máximo do termo
Mlt.maxntp	Número máximo de termos a analisar em cada campo do documento
Mlt.qt	Número máximo de termos de consulta
Mlt.boost	[verdadeiro/falso] determina se a consulta será impulsionada pela relevância do termo de interesse
Mlt.qf	Campos de consulta
Mlt.match.include	Deve a resposta incluir o documento encontrado? Se falso, a resposta virá exactamente como uma resposta normal
Mlt.match.offset	Por definição, a pesquisa *MoreLikeThis* opera sobre o primeiro resultado de 'q'
Mlt.interestingTerms	Um de: "list", "details", "none" – isto irá mostrar quais são os termos interessantes usados na pesquisa *MoreLikeThis*. Estes são os termos tf/idf de topo. Nota: se seleccionar "details", este mostra-lhe o termo e o impulso usado em cada termo. A menos que mlt.boost=true, todos os termos terão impulso = 1

Uma parte substancial do trabalho correspondeu à utilização criteriosa do *MoreLikeThis (Mlt)* como ferramenta de pesquisa de documentos invocada a partir das *queries* contidas nos diversos formulários disponíveis na aplicação desenvolvida. A

eficiência da pesquisa e a relevância dos resultados apresentados dependem de forma muito sensível da escolha adequada destes parâmetros. Apesar de nos termos limitado a uma análise casuística das diversas escolhas à nossa disposição, a sua escolha optimal pode ser alvo de futuros estudos.

DESCRIÇÃO E ANÁLISE DE RESULTADOS

Numa primeira fase procedeu-se à identificação de documentos duplicados, o que foi implementado através da criação de um *HashTransformer* (cujo código se encontra mais abaixo). Este *HashTransformer* calcula o *Hash* de cada documento, o que permite a transformação de uma grande quantidade de dados numa pequena quantidade de informação, geralmente uma sequência de bits em base hexadecimal. Essa sequência identifica um documento unicamente, pelo que, caso existam documentos com o mesmo *Hash*, eles são necessariamente duplicados.

Depois de aplicado o *HashTransformer* ao conjunto dos documentos, estes passam a ter preenchido um campo designado por *Hash*, após o que uma pesquisa efectuada com o MoreLikeThis relativamente a este campo permite obter os documentos duplicados (com o mesmo *Hash*). Apresenta-se de seguida o código do *HashTransformer* desenvolvido:

Código HashTransformer:

```
package quidgest.solr.dataimport;

import java.security.MessageDigest;
```

```
import java.security.NoSuchAlgorithmException;
import java.util.Map;
import org.apache.solr.handler.dataimport.Context;
import org.apache.solr.handler.dataimport.Transformer;
public class HashTransformer extends Transformer {
	@Override
	public Object transformRow(Map<String, Object> row, Context arg1) {
		String body = ((String) row.get("text"));
		String res=calcularHash(body);
		row.put("hash", res);
		return row;
	}
	private static String calcularHash(String body) {
		 try {
			MessageDigest md = MessageDigest.getInstance("SHA");
			byte[] hash = new byte[0];
			if(body!=null) {
				hash = md.digest(body.getBytes());
			}
			StringBuilder sb = new StringBuilder();
		  for (byte b : hash) {
		     sb.append(String.format("%02X", b));
		  }
```

```
                return sb.toString();
            } catch (NoSuchAlgorithmException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return "";
        }
}
```

A título de exemplo, apresenta-se uma pesquisa na qual são procurados documentos com o hash = DB951BAA7D338511A080970B022AAF5A9C33A7C6, sendo que foram encontrados 4 documentos, ou seja, esses 4 documentos são iguais. A pesquisa utilizada para esta verificação foi a seguinte:

```
http://localhost:11001/solr/FILESERVERNB/select?q=hash:DB951BAA7D338511A0809
70B022AAF5A9C33A7C6&wt=xml&indent=true&start=0&rows=20&fl=id,filename,hash
&mlt=true&mlt.fl=hash
```

O resultado da pesquisa é devolvido em formato XML, tal como pode ser visualizado na figura 1. Este código XML pode ser posteriormente utilizado para uma visualização mais *user friendly* dos resultados.

Figura 1: Exemplo de código XML gerado por uma pesquisa

Seguidamente, na pesquisa por documentos similares optou-se inicialmente por um processo semelhante mas utilizando o campo *body* em vez do campo *hash,* para efeitos da análise de similaridade. Após os primeiros testes em pesquisas por documentos similares verificou-se que os resultados não eram satisfatórios, encontrando similaridades inadequadas. Tentou-se melhorar os resultados adicionando mais parâmetros do *MoreLikeThis* de modo a limitar a pesquisa. No entanto, as melhorias não foram significativas. O ajuste dos parâmetros no *MoreLikeThis* faz com que varie o número de resultados fornecidos pelo *MoreLikeThis*. Por outro lado, se esses parâmetros não forem ajustados ou forem demasiado refinados podemos obter como resultado da pesquisa quase todos os documentos (no primeiro caso) ou nenhum documento (no segundo caso). Por exemplo, ao pesquisar por documentos similares com a palavra anexo, com o código

```
http://localhost:11001/solr/FILESERVERNB/select?q=anexo&wt=xml&indent=true&start=0&rows=20&fl=id,filename,hash&mlt=true&mlt.minwl=4&mlt.mindf=1&mlt.mintf=1&mlt.maxqt=25&mlt.fl=body&mlt.boost=true&mlt.interestingTerms=details&mlt.match.include=true
```

os resultados devolveram um número de ficheiros similares demasiado elevado. Neste exemplo, os documentos que continham a palavra anexo eram 220 e cada um desses documentos teria cerca de 900 a 1000 documentos similares.

Como alternativa, fez-se uma pesquisa inicial onde eram devolvidos os documentos que continham a palavra pesquisada, juntamente com os 25 termos mais utilizados em cada um desses documentos, pesquisando-se de seguida os documentos que tivessem esses 25 termos, sendo deste modo todos os resultados obtidos considerados documentos similares. Aqui surgiram alguns problemas, entre os quais, o facto de a instrução *select* não devolver os *interestingTerms*. Esta dificuldade foi ultrapassada com a utilização do comando *mlt*, que é equivalente ao *select*, sendo que este já devolvia os *interestingTerms*. Os códigos das instruções podem ser vistos no ANEXO 1.

A pesquisa utilizada para saber os *interestingTerms* passou a ser seguinte:

```
http://localhost:11001/solr/FILESERVERNB/mlt?q=hash:DB951BAA7D338511A080970B022AAF5A9C33A7C6&wt=xml&indent=true&start=0&rows=10&fl=filename&mlt.fl=body
```

Na figura 2 mostram-se os *interestingTerms* devolvidos pela pesquisa anterior.

Figura 2: Exemplo de código XML

Determinados os *interestingTerms* procede-se a uma nova pesquisa devolvendo os documentos similares, ou seja, os documentos que continham os mesmos *interestingTerms*. A pesquisa utilizada foi a seguinte:

```
http://localhost:11001/solr/FILESERVERNB/select?q=body:objetiv%20AND%20body:estrategic%20AND%20body:balanced%20AND%20body:organizacion%20AND%20body:scorecard%20AND%20body:implantacion%20AND%20body:model%20AND%20body:indicador%20AND%20body:pued%20AND%20body:estrategi%20AND%20body:iniciativ%20AND%20body:clav%20AND%20body:persa%20AND%20body:perspectiv%20AND%20body:gestion%20AND%20body:important%20AND%20body:ejempl%20AND%20body:t
```

ambien%20AND%20body:proyect%20AND%20body:element%20AND%20body:mapa%20AND%20body:iese%20AND%20body:debe%20AND%20body:cion%20AND%20body:pueden&wt=xml&indent=true&start=0&rows=10&fl=filename,id

Neste caso particular, os resultados da pesquisa consistiram em 4 documentos similares, coincidentes com os documentos duplicados. Apesar de pelo menos estes serem necessariamente incluídos nos resultados, poderiam ter sido encontrados mais documentos.

Figura 3: Exemplo de código XML resultante de uma pesquisa

Outra hipótese de desenvolvimento considerada foi a utilização de *Fuzzy Searches*, ou seja, em vez de se procurar apenas por palavras exactas, procurar-se também por palavras parecidas. As alterações necessárias são bastante reduzidas, bastando

adicionar à frente da palavra por exemplo ~0.9, onde 0.9 é o coeficiente de semelhança entre as palavras. Esta situação ao início parecia ser um bom rumo a tomar mas, mais tarde, viria a causar problemas em pesquisas que só se diferenciavam por um acento e que, apesar de deverem devolver os mesmos resultados, tal não acontecia. As *Fuzzy Searches* foram por isso abandonadas.

Outro dos problemas com que nos deparámos foi o aumento do tempo que as pesquisas sofreram, passando de poucos milésimos de segundo para alguns segundos, devido à quantidade de pesquisas que tinham de ser feitas (1 pesquisa de documentos pela palavra pretendida, 1 pesquisa com o *hash* do documento pretendido para encontrar os documentos duplicados, 1 pesquisa que devolvesse os *interestingTerms* e, por fim, mais 1 pesquisa para encontrar os documentos similares). Como é óbvio, não queremos aumentar desnecessariamente o tempo de espera do utilizador pois a página do Qsearch quer-se rápida.

Conseguimos simplificar um pouco o código, de modo a numa só secção fazer as várias pesquisas, consoante os parâmetros que eram passados. Apesar de se terem registado algumas melhorias, não foram ainda suficientes.

```
private static SolrMoreLikeThisHandlerResults<LooseDocument>
GetMoreLikeThis(string hash, string area, int start, int rows, string field)
    {
        var solr = SolrSearchQueries.Instance.Resolve(area);
        AbstractSolrQuery q = new SolrQuery(hash);
```

```
        MoreLikeThisHandlerQueryOptions options = new
MoreLikeThisHandlerQueryOptions(new MoreLikeThisHandlerParameters(new[] { field
})
        {
                MatchInclude= (field=="hash") ? true : false,
                ShowTerms = (field == "hash") ? InterestingTerms.None :
InterestingTerms.Details
        });
        options.Rows = rows;
        options.Start = start;
        SolrMoreLikeThisHandlerResults<LooseDocument> results =
solr.MoreLikeThis(SolrMLTQuery.FromQuery(q), options);
        return results;
    }
```

A solução final encontrada para baixar o tempo de pesquisa foi transferir a *query* para a fase de indexação. A procura de similaridade e de duplicação passou assim a ser feita durante a indexação, sendo guardados os documentos duplicados de cada documento em *duplicates* e os documentos similares em *similars*. Assim, voltámos a ter pesquisas com a demora de poucos milésimos de segundo, apesar de a fase de indexação se ter tornado mais morosa. Apesar disso, como a fase de indexação não é muito relevante

para o tempo de espera do utilizador, pareceu-nos um bom compromisso. O código utilizado agora é o seguinte:

```
private static SolrQueryResults<LooseDocument> GetResultsMVC(IDictionary<string, string> filters, SolrFacet[] facetFields, string query, string qf, string area, int start, int rows)
{
        var solr = SolrSearchQueries.Instance.Resolve(area);
        AbstractSolrQuery q = new SolrQuery(query);
        QueryOptions options = new QueryOptions {
         Start = start,
           Rows = rows,
           ExtraParams = new Dictionary<string, string> {
               { "defType", "edismax" },
               { "tie", "0.01" },
               { "mm", "70%" },
               { "qf", qf }
           }
        };
        if(facetFields == null || filters == null) {
           options.Fields = new[] { "id" };
        } else {
           SolrFacetFieldQuery[] facets = new SolrFacetFieldQuery[facetFields.Length];
           for (int i = 0; i < facetFields.Length; i++)
```

```
            facets[i] = new SolrFacetFieldQuery(facetFields[i].field) { MinCount = 1,
Limit = facetFields[i].limit };
            options.FilterQueries = BuildFilterQueries(filters);
            options.Fields = new[] {
"id","filename","last_modified","author","tags","similars","duplicates" };
            options.Highlight = new HighlightingParameters {
                Fields = new[] { "text","body" },
                Snippets=3,
            };
            options.SpellCheck = new SpellCheckingParameters {
               Collate = true
            };
            options.Facet = new FacetParameters {
               Queries = facets
            };
        }
        SolrQueryResults<LooseDocument> results = solr.Query(q, options);
        return results;
    }
```

Antes de se ter chegado à solução anterior, foram realizados 2 testes para verificar se o tempo das pesquisas melhorava. Um correspondia à utilização de *shards* nas pesquisas e o outro à criação de 1 *core* para o *MoreLikeThis*.

Os *shards* são partições do índice do Lucene, que permitem que a informação seja dividida ou replicada por vários sítios: podem estar todos na mesma máquina ou em máquinas diferentes. Como se trata de uma partição, uma pesquisa feita num *shard* não obtém o mesmo resultado se for efectuada noutro *shard*, a não ser que esta seja uma réplica do primeiro. Já a criação de um *core* é diferente: O *core* representa uma instância do índice do SOLR, enquanto os *shards* são subconjuntos do *core*.

Foram realizadas 15 pesquisas semelhantes para cada teste (sem *shards*, com *shards* e com um *core*). Guardaram-se os tempos totais das pesquisas, os tempos para cada *query* (1º select - pesquisa de documentos pela palavra pretendida, mlts dup - pesquisa com o *hash* do documento pretendido para encontrar os documentos duplicados, mlts sim - pesquisa que devolvesse os interestingTerms e 2º select - pesquisa para encontrar os documentos similares) e fez-se uma média dessas 15 pesquisas, comparando-se os resultados (podem ser vistos no ANEXO 2).

A partir destes resultados concluímos que os tempos totais da pesquisa sem *shards* são os melhores, a pesquisa é mais rápida, demorando em média 3747 ms, ao passo que a pesquisa com *shards* demora, em média, 4431 ms. A pesquisa com o *core,* a pior das três opções relativamente ao tempo, demora 5992 ms em média.

Voltando problema inicial, o facto de as 4 *queries* sequenciais tornarem a pesquisa mais lenta, a análise mais pormenorizada de cada *query* permite verificar que o 1º select é mais rápido com a utilização do *core* (4 ms contra 7,7 ms nos *shards* e 16 ms sem *shards*), os mlts dup são mais rápidos sem *shards* (149 ms contra 193,2 ms com o *core* e 252 ms com *shards*), os mlts sim são mais rápidos com *shards* (3174 ms contra

3410 ms sem *shards* e 4508 ms com o *core*) e que o 2º select é mais rápido com *shards* (14 ms contra 24,8 ms sem *shards* e 42,6 ms com o *core*).

Através desta análise, não ficámos ainda elucidados sobre qual seria a melhor opção, mas optámos por continuar a testar a pesquisa com a utilização de *shards*, devido a ter sido favorável em 2 *queries*.

Acabámos por não terminar de implementar a utilização de *shards* pois mais tarde, quando já fazíamos as *queries* na indexação, deparámo-nos com alguns erros gerados pelo SOLR que não puderam ser resolvidos até ao final do estágio.

NOTAS SOBRE O INTERFACE

Para além de serem adicionadas as informações dos documentos duplicados e similares, tal como foi descrito na secção anterior, parte do trabalho desenvolvido passou pelo melhoramento do interface com o utilizador, nomeadamente através da página do QSearch, de modo que o utilizador pudesse trabalhar num ambiente mais amigável. Basicamente, tornámos a página do Qsearch num motor de busca com uma barra no topo onde o utilizador insere as palavras pelas quais deseja procurar nos documentos. Ao mesmo tempo que se vai inserindo as letras das palavras, o motor de busca vai sugerindo palavras que considera enquadrarem-se nas sequências de letras inseridas. No caso do utilizador se enganar a inserir, ou seja escrever com erro a palavra, o motor de busca indica o seguinte: “Será que quis dizer: [a palavra correctamente escrita] ”. Após a inserção da palavra correcta, irá fazer a pesquisa e indicar os documentos que tenham essa palavra ou expressão associada a eles, e

destacará a negrito essa palavra num pedaço de texto do documento que é devolvido. Os resultados serão apresentados mostrando um ícone do formato do ficheiro, o nome do ficheiro, a localização do ficheiro, o número de ficheiros duplicados e/ou similares que existem, a última modificação que teve e as categorias a que está associado. Após a apresentação dos resultados o utilizador pode fazer *download* do ficheiro pretendido com um clique sobre o nome do ficheiro e pode também ver a localização dos ficheiros duplicados e similares clicando sobre a âncora "Ver mais".

Na coluna mais à esquerda da página do Qsearch, encontram-se os filtros da pesquisa, nomeadamente um filtro por categoria, um filtro por autor e um filtro pela extensão do documento.

Na parte inferior encontra-se a paginação, onde o utilizador pode alterar a página de modo a poder visualizar os restantes resultados da pesquisa.

Algumas destas funcionalidades podem ser visualizadas no ANEXOS 3 e 4.

CONCLUSÕES

Quanto ao contributo do estágio não tenho qualquer dúvida de que foi uma experiência positiva e valiosa a nível profissional e pessoal. Possibilitou um primeiro contacto com o mercado de trabalho e o aprofundamento de alguns conceitos teóricos adquiridos durante o percurso académico, essencialmente na área da programação.

No que diz respeito aos objectivos do estágio, penso que foram cumpridos, tendo sido criadas novas funcionalidades para a página do *Qsearch*. Foram criadas pesquisas para a detecção de documentos duplicados e similares, sucessivamente aperfeiçoadas relativamente ao desempenho, assim como uma diversidade de outras soluções secundárias. A página do QSearch ficou com um ambiente bastante mais amigável para o utilizador e as pesquisas são bastante rápidas. De modo geral, conseguiu-se que os utilizadores da página obtivessem o conhecimento de que existem documentos duplicados e documentos similares bem como a sua localização, facilitando-lhes a forma como agora podem aceder aos mesmos e pesquisá-los.

BIBLIOGRAFIA

- *Maia, Luiz Cláudio Gomes; Souza, Renato Rocha. Medidas de similaridade em documentos electrónicos*
- *Baeza-Yates, Ricardo; Ribeiro-Neto, Berthier. Modern Information Retrieval. New York: ACM Press, 1999*
- http://lucene.apache.org/solr/index.html
- http://en.wikipedia.org/wiki/Apache_Solr
- http://pt.wikipedia.org/wiki/Visual_Studio
- http://pt.wikipedia.org/wiki/Eclipse_(software)
- http://www.eclipse.org/org/
- http://wiki.apache.org/solr/SolrTerminology
- http://www.asp.net/mvc/tutorials/getting-started-with-aspnet-mvc3/cs/intro-to-aspnet-mvc-3
- http://lucene.apache.org/solr/4_0_0/tutorial.html
- http://wiki.apache.org/solr/DataImportHandler
- http://wiki.apache.org/solr/TikaEntityProcessor
- http://wiki.apache.org/solr/SchemaXml
- http://wiki.apache.org/solr/CommonQueryParameters
- http://wiki.apache.org/solr/MoreLikeThis
- http://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters
- http://docs.oracle.com/javase/6/docs/api/
- http://wiki.apache.org/solr/Deduplication#Document_Duplication_Detection

- http://wiki.apache.org/solr/HighlightingParameters
- http://wiki.apache.org/solr/SpellCheckComponent
- http://www.w3schools.com/tags/ref_byfunc.asp
- http://lucene.apache.org/core/old_versioned_docs/versions/2_9_1/querypars ersyntax.html
- http://pietervogelaar.nl/solr-3-5-stop-words-are-not-removed/#.UUhZvByeNYD
- http://wiki.apache.org/solr/SolrCloud
- https://issues.apache.org/jira/browse/SOLR-4414

ANEXOS

ANEXO 1

Código mlt:

```
<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">
        <lst name="defaults">
                <str name="df">text</str>
                <str name="fl">id</str>
                <str name="qf">hash</str>
                <str name="defType">edismax</str>
                <str name="mlt.fl">body,title,subject,description,text</str>
                <str name="mlt.mintf">1</str>
                <str name="mlt.mindf">2</str>
                <str name="mlt.minwl">4</str>
                <str name="mlt.match.include">false</str>
                <str name="mlt.interestingTerms">details</str>
                <str name="mlt.qf">body,title,subject,description,text</str>
        </lst>
</requestHandler>
```

Código select:

```
<requestHandler name="/select" class="solr.SearchHandler">
        <lst name="defaults">
                <str name="echoParams">explicit</str>
```

```
            <int name="rows">10</int>
            <str name="df">text</str>
        </lst>
        <arr name="components">
            <str>query</str>
            <str>facet</str>
            <str>mlt</str>
            <str>highlight</str>
            <str>suggest</str>
            <str>stats</str>
            <str>debug</str>
            <str>clustering</str>
        </arr>
</requestHandler>
```

ANEXO 2

Análise dos tempos de pesquisa para Proposta no Fileserver (15 pesquisas) - sem a utilização de shards

nº tentativa	tempos totais(ms)	1º select	mlts dup	Média dup	mlts sim	Média Sim	2º select	Média Select		Soma Selects + Mlts
1	4060	28	284	18,93333333	3533	235,5333333	31	2,066666667		3876
2	3650	24	141	9,4	3337	222,4666667	27	1,8		3529
3	4000	19	143	9,533333333	3408	227,2	26	1,733333333		3596
4	3800	19	162	10,8	3468	231,2	25	1,666666667		3674
5	3810	13	137	9,133333333	3440	229,3333333	24	1,6		3614
6	3960	18	144	9,6	3618	241,2	26	1,733333333		3806
7	3670	10	139	9,266666667	3384	225,6	24	1,6		3557
8	3590	15	133	8,866666667	3303	220,2	23	1,533333333		3474
9	3650	15	134	8,933333333	3371	224,7333333	20	1,333333333		3540
10	3750	17	139	9,266666667	3448	229,8666667	27	1,8		3631
11	3590	16	133	8,866666667	3301	220,0666667	20	1,333333333		3470
12	3740	11	141	9,4	3458	230,5333333	25	1,666666667		3635
13	3590	17	149	9,933333333	3302	220,1333333	23	1,533333333		3491
14	3640	9	130	8,666666667	3370	224,6666667	26	1,733333333		3535
15	3710	9	133	8,866666667	3421	228,0666667	25	1,666666667		3588
Média	3747,333333	16	149,4666667	9,964444444	3410,8	227,3866667	24,8	1,653333333		

Análise dos tempos de pesquisa para Proposta no Fileserver (15 pesquisas) - com a utilização de 8 shards - Plano A (shards MLT)

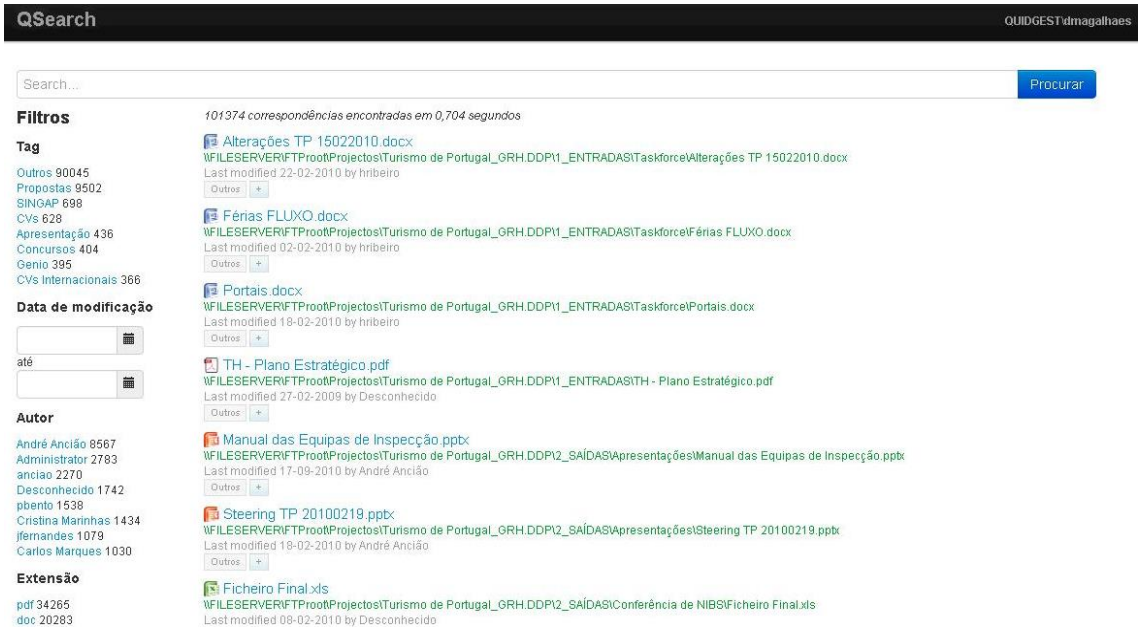
nº tentativa	tempos totais(ms)	1º select	mlts dup	Média dup	mlts sim	Média Sim	2º select	Média Select		Soma Selects + Mlts
1	4170	8	164	10,93333333	2863	190,8666667	13	0,866666667		3048
2	4490	9	263	17,53333333	3226	215,0666667	14	0,933333333		3512
3	4510	7	253	16,86666667	3258	217,2	14	0,933333333		3532
4	4480	5	253	16,86666667	3232	215,4666667	15	1		3505
5	4520	8	355	23,66666667	3144	209,6	14	0,933333333		3521
6	4420	8	244	16,26666667	3156	210,4	15	1		3423
7	4560	8	255	17	3305	220,3333333	13	0,866666667		3581
8	4470	9	246	16,4	3243	216,2	14	0,933333333		3512
9	4460	7	263	17,53333333	3192	212,8	14	0,933333333		3476
10	4360	8	246	16,4	3100	206,6666667	15	1		3369
11	4580	8	259	17,26666667	3335	222,3333333	14	0,933333333		3616
12	4460	10	266	17,73333333	3228	215,2	15	1		3519
13	4590	9	250	16,66666667	3362	224,1333333	15	1		3636
14	4200	7	249	16,6	2980	198,6666667	15	1		3251
15	4200	5	227	15,13333333	2997	199,8	12	0,8		3241
Média	4431,333333	7,733333333	252,8666667	16,85777778	3174,733333	211,6488889	14,13333333	0,942222222		

Análise dos tempos de pesquisa para Proposta no Fileserver (15 pesquisas) - com a utilização de 8 shards - Plano B (1 core MLT)

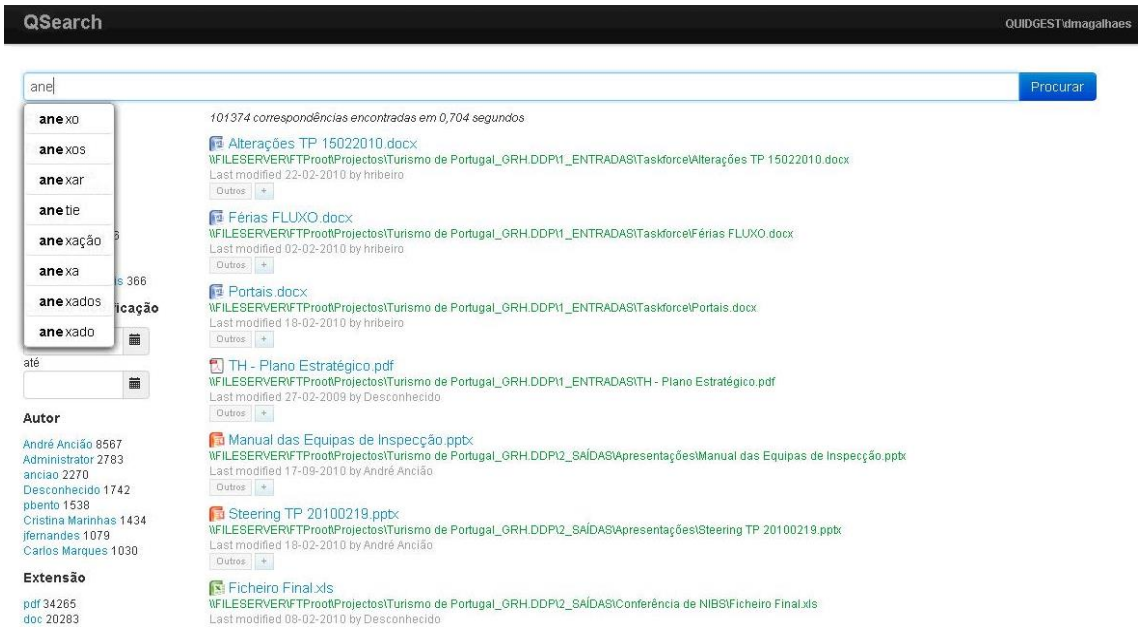
nº tentativa	tempos totais(ms)	1º select	mlts dup	Média dup	mlts sim	Média Sim	2º select	Média Select		Soma Selects + Mlts
1	5680	16	188	12,53333333	4360	290,6666667	48	3,2		4612
2	5950	4	195	13	4407	293,8	47	3,133333333		4653
3	5840	4	176	11,73333333	4419	294,6	43	2,866666667		4642
4	6530	3	210	14	4714	314,2666667	48	3,2		4975
5	6160	6	188	12,53333333	4757	317,1333333	44	2,933333333		4995
6	6320	3	187	12,46666667	4818	321,2	48	3,2		5056
7	6010	3	173	11,53333333	4378	291,8666667	44	2,933333333		4598
8	6110	2	204	13,6	4600	306,6666667	33	2,2		4839
9	6000	4	192	12,8	4667	311,1333333	39	2,6		4902
10	6030	3	198	13,2	4366	291,0666667	36	2,4		4603
11	5960	2	188	12,53333333	4687	312,4666667	41	2,733333333		4918
12	5940	2	260	17,33333333	4391	292,7333333	42	2,8		4695
13	5870	3	191	12,73333333	4611	307,4	49	3,266666667		4854
14	5070	3	146	9,733333333	4014	267,6	34	2,266666667		4197
15	6410	3	202	13,46666667	4444	296,2666667	43	2,866666667		4692
Média	5992	4,066666667	193,2	12,88	4508,866667	300,5911111	42,6	2,84		

ANEXO 3

Início da página do QSEARCH

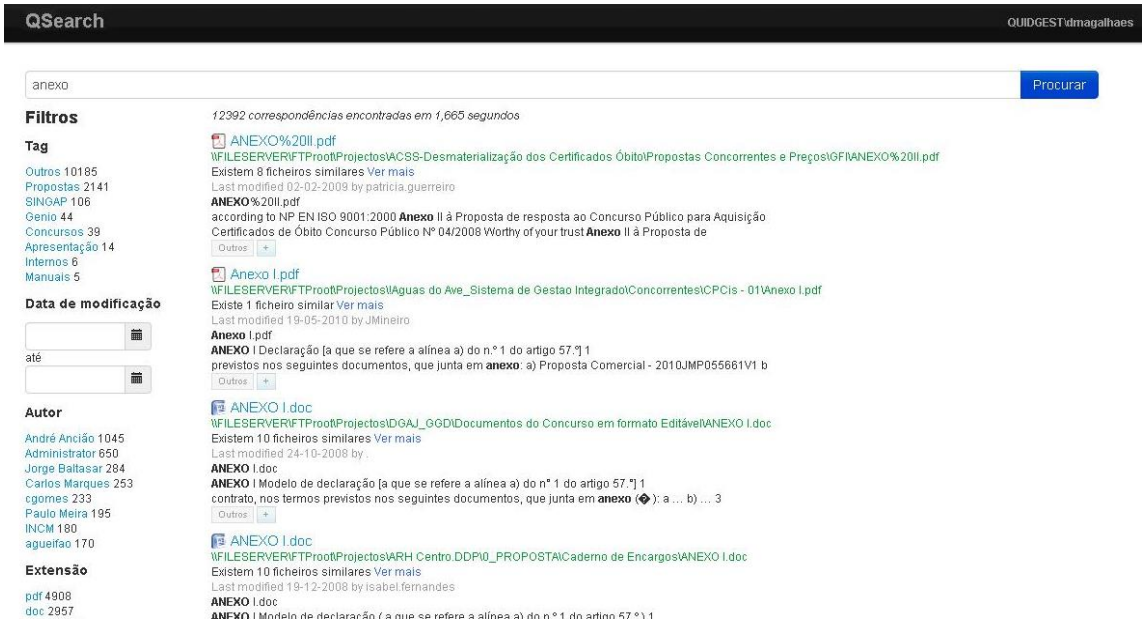


Sugestões de Palavras



ANEXO 4

Pesquisa por anexo



Âncora Ver Mais para Documentos Duplicados e Similares