

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CÂMPUS CAMPINAS

RENATO DOS SANTOS RIBEIRO

PLATAFORMA DE BUSCA E INDEXAÇÃO EM NUVEM PRIVADA

CAMPINAS

2017

RENATO DOS SANTOS RIBEIRO

PLATAFORMA DE BUSCA E INDEXAÇÃO EM NUVEM PRIVADA

Trabalho de Conclusão de Curso apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia Câmpus Campinas.

Orientador: Prof. Dr. Andreiuid Sheffer Correa.

CAMPINAS

2017

R484p Ribeiro, Renato dos Santos
 Plataforma de busca e indexação em nuvem privada / Renato dos Santos
 Ribeiro. – Campinas, 2017.
 39f. : il.

 Orientador: Andreiwid Sheffer Correa.

 Monografia (Graduação) – Instituto Federal de São Paulo – Câmpus
Campinas. Curso de Tecnologia em Análise e Desenvolvimento de Sistemas,
2017.

 1. Indexação. 2. Sistemas web. 3. OCR. 4. Busca em conteúdo de arquivo. I.
Instituto Federal de São Paulo - Câmpus Campinas. Curso de Tecnologia em
Análise e Desenvolvimento de Sistemas. II. Título.

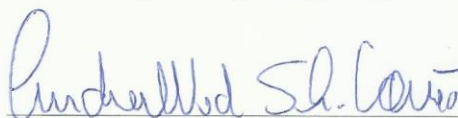
Renato dos Santos Ribeiro

PLATAFORMA DE BUSCA E INDEXAÇÃO EM NUVEM PRIVADA

Trabalho de Conclusão de Curso apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo Câmpus Campinas.

Aprovado pela banca examinadora em: 12 de dezembro de 2017.

BANCA EXAMINADORA



Prof. Dr. Andreiuid Sheffer Corrêa (orientador)

IFSP Câmpus Campinas

Prof. Dr. Tiago José de Carvalho

IFSP Câmpus Campinas

Dr. Rodrigo Bonacin

Centro de Tecnologia da Informação Renato Archer

*Dedico este trabalho aos meus familiares,
colegas de classe, professores e servidores do Instituto
que colaboraram em minha jornada formativa.*

AGRADECIMENTOS

Agradeço primeiramente a Deus, pelo dom da vida e pela oportunidade de concluir mais uma etapa de minha experiência acadêmica.

Agradeço a todos os professores e servidores do IFSP
Câmpus Campinas, que contribuíram direta e
indiretamente para a conclusão desse trabalho.

Agradeço também à minha família, que deu todo o apoio
necessário para que eu chegasse até aqui.

Agradeço ao meu orientador, Andreiuid Sheffer Correa, que me auxiliou a solucionar as
dificuldades encontradas no caminho.

*"O que prevemos raramente ocorre;
o que menos esperamos
geralmente acontece".
Benjamin Disraeli*

RESUMO

A busca por arquivos é algo comum no dia a dia, e o próprio sistema operacional nos dá suporte a uma interface básica de busca. Entretanto, essa ferramenta é limitada e não é adequada ao meio empresarial, onde há pastas de rede com milhares e até milhões de arquivos. Isso traz um grande desafio quanto ao modo de disponibilizar estes arquivos, pois o custo computacional para a indexação é alto, e sua não indexação faz a busca ser muito lenta. Adiciona-se o complicador de que muitas vezes o documento está digitalizado com seu conteúdo como imagem, o que já não o torna acessível à maioria dos sistemas de busca que vem por padrão no computador. Deste modo, o objetivo deste trabalho é o desenvolvimento de uma plataforma web que possa servir para armazenar arquivos e busca de documentos por conteúdo. O projeto foi desenvolvido utilizando a linguagem Groovy, com o *front end* utilizando as seguintes tecnologias: Html5, CSS, e Javascript, com React. O armazenamento dos dados indexados foi feito com o Elasticsearch. O sistema atingiu seus objetivos, tendo uma busca com cobertura 40% superior ao Google Drive, que é uma aplicação semelhante. Entretanto a precisão do Google Drive foi 16% mais eficiente e a medida F, teve uma vantagem de 3% em relação ao sistema desenvolvido.

Palavras-chave: Indexação. Sistemas Web. OCR. Busca em conteúdo de arquivo.

ABSTRACT

The files search is commonplace day by day, and the operating system itself supports a basic search interface. However, these tools are limited and not suited to business environment, where there are network folders with thousands and even millions of files. This brings a great challenge to the way of making files available, since the computational cost for indexing is high, and its non-indexing makes the search very slow. It adds the complication that the document is often scanned, which is no longer the most accessible in most search systems by default in OS. In this way, the objective of this work is the development of a web platform that can be used to store files and search for documents by content. The project was developed, use a Groovy language, with the front-end, use as following technologies: Html5, CSS, and Javascript, with React. The indexed data was stored using elastic search. The system has achieved its goals, having a search coverage 40% higher than Google Drive, which is an similar application. However, Google Drive was 16% more efficient and the F-measure had 3% advantage over the developed system.

Keywords: Index. Web systems. OCR. Files search.

LISTA DE FIGURAS

Figura 1 – Visão geral do sistema e responsabilidade dos módulos.....	22
Figura 2 – Arquitetura do sistema.....	25
Figura 3 – Estrutura do Documento.....	26
Figura 4 – Algoritmo de indexação.....	27
Figura 5 – Extração de documento digitalizado.....	34
Figura 6 – Página inicial.....	35
Figura 7 – Exibição de conteúdo filtrado.....	36
Figura 8 – Download do arquivo.....	37
Figura 9 – Exibição de conteúdo filtrado.....	38

LISTA DE SIGLAS

API.	Application Programming Interface
ARTESP	Agência Reguladora de Serviços Públicos Delegados de Transporte do Estado
CRUD	Create Retrieve Update Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
ECMA.	Standardization, Information and Communication Technology
NFS.	Network File System
OCR.	Optical Character Recognition
RAM	Random Access Memory
REST	Representational State Transfer

SUMÁRIO

1	INTRODUÇÃO	12
2	OBJETIVOS	13
2.1	Objetivo geral.....	13
2.2	Objetivos específicos.....	13
3	JUSTIFICATIVA	14
4	FUNDAMENTAÇÃO TEÓRICA.....	15
4.1	Indexação de arquivos.....	15
4.1.1	<i>Lucene</i>	15
4.1.2	<i>Elasticsearch</i>	16
4.1.3	<i>Tika</i>.....	17
4.1.4	<i>Tesseract</i>.....	18
4.2	Spring.....	18
4.3	Groovy	19
4.4	React	19
4.5	Avaliação do sistema de busca.....	20
5	MÉTODO	21
5.1	Arquitetura do sistema.....	22
5.2	Persistência dos dados	24
5.3	Implementação do lado servidor	24
5.4	Implementação do <i>front end</i>	26
5.5	Configuração do ambiente	27
6	RESULTADOS OBTIDOS.....	29
7	CONCLUSÃO.....	37
	REFERENCIAS	38

1 INTRODUÇÃO

A rápida expansão da Internet e o aumento da banda tem transformado a forma com que a utilizamos. A cada dia temos mais serviços prestados a partir dela, e a cada dia nos tornamos mais dependentes de conteúdo digital.

Tal convergência fez com que cada vez mais os documentos e informações importantes estivessem armazenados em computadores. Além disso, cada vez mais as empresas fazem a guarda de seus documentos em sistemas na nuvem ou em intranets.

Williams (2012, p.2) descreve a computação na nuvem como uma evolução da computação que já conhecemos, porém com um enfoque na rápida resposta ao mercado e redução de custos.

O armazenamento dos dados na nuvem permite reduzir os custos na medida que desobriga a empresa de software manter toda a infraestrutura para dar suporte à aplicação na empresa. Além disso, permite a adição e remoção de servidores de forma muito mais rápida e a um custo de manutenção mais baixo.

Como consequência do crescimento do uso pelas empresas, o volume de arquivos se tornou imenso, o que pode tornar a busca por um determinado conteúdo custosa. E um ponto a se considerar, é que o arquivo nem sempre está em um formato que permita a busca sem um tratamento prévio, como nos casos em que o conteúdo está digitalizado em imagem.

Assim, ferramentas de indexação que conseguem lidar com um grande leque de tipos de arquivos são bastante valorizadas no mercado, pois tão importante quanto ter a informação é conseguir localizá-la e utilizá-la.

Desse modo, o objetivo deste trabalho é o desenvolvimento de um sistema web que possa servir de diretório virtual indexado. Esse diretório fará a indexação de forma automática dos conteúdos inseridos, suportando a maiorias dos arquivos de formatos populares como .PDF, .DOC, etc. Além dos formatos citados, ele também fará a indexação de conteúdo de textos em imagens.

2 OBJETIVOS

No tópico 2.1 é abordado o objetivo geral do trabalho. No tópico 2.2 são definidos os objetivos específicos.

2.1 Objetivo geral

O objetivo deste trabalho é o desenvolvimento de uma plataforma web que possa servir para guarda, indexação e busca de documentos por conteúdo.

2.2 Objetivos específicos

- a) Realizar o controle de acesso por usuário.
- b) Indexar o conteúdo em formatos legíveis por máquina, como:
 - HTML
 - XML e formatos derivados, tais como XHTML, OOXML e ODF.
 - Documentos do pacote Office: DOC, XLS, PPT, RTF e documentos do Open Office.
 - PDF.
 - Imagens da extensão JPG, GIF , PNG, BMP e TIFF.
- c) Indexar o conteúdo de texto encontrado nas imagens, em arquivos do tipo PDF.
- d) Realizar a busca de arquivos por conteúdo.
- e) Disponibilizar um *front-end* para utilização do sistema.

3 JUSTIFICATIVA

As ferramentas de busca, tais como o Google, Bing, entre outros, são bastante populares, por serem eficazes em localizar conteúdo hospedado em sites. Tais ferramentas são capazes de encontrar conteúdo em típicos documentos de textos, como PDF, DOC, PPT, etc. Assim, apesar do vasto conteúdo da Internet, as informações são acessíveis e podem ser consumidas por usuários sem grande esforço.

Entretanto, quando é analisado o conteúdo produzido por entidades privadas, é comum encontrar casos, onde os dados não estão facilmente acessíveis. Isto ocorre porque as ferramentas de indexação utilizadas na Web não podem ser utilizadas pelas empresas, uma vez que os dados indexados por elas são disponibilizados ao público. Além disso, as ferramentas de busca do sistema operacional não são adequadas para o volume de dados que há nas empresas.

Neste cenário, as ferramentas de indexação possuem grande valia, pois economizam um tempo precioso, em que o usuário deixa de fazer o que agregaria valor e gasta até mesmo horas abrindo arquivos para encontrar o que deseja.

Além da problemática apresentada, muitas vezes, apesar de o usuário usar um indexador eficiente, o conteúdo se encontra digitalizado, o que pode ser um obstáculo para obter conteúdo existente no documento.

Nas repartições públicas, por exemplo, é comum muitos documentos estarem digitalizados em imagem. Caso algum usuário precise encontrar seu nome em um documento, por exemplo, ele teria de ler o documento, ou então aplicar uma ferramenta OCR na imagem para convertê-la em texto, para assim poder usar as ferramentas de busca dos editores de texto. Se o usuário tem um volume de dados muito grande isso se torna inviável sem o uso de ferramentas que façam esta tarefa de forma automatizada.

Assim, a ferramenta que foi desenvolvida, possui relevância ao permitir uma busca mais eficiente, ajudando o usuário a encontrar os documentos que necessita em um tempo menor e sem grande esforço.

O fato dos dados estarem na nuvem, de forma centralizada e indexada, também pode trazer outras vantagens para as empresas, uma vez que dá ferramentas que permitem à empresa consumir os dados de maneira mais eficaz.

4 FUNDAMENTAÇÃO TEÓRICA

Nesta sessão será explicado as tecnologias utilizadas no projeto. Na sessão 4.1 é explicado as tecnologias relacionadas à indexação utilizadas no projeto. A sessão 4.2 trata do *framework* utilizado no *back end* do sistema, o Spring. Já a sessão 4.3 fala sobre a linguagem de programação utilizada para o desenvolvimento do *back end*. Por fim, o módulo 4.4, traz informações sobre o React, uma biblioteca utilizada para desenvolvimento do *front end*.

4.1 Indexação de arquivos

A indexação de arquivos é útil para agilizar a busca. Ela pode ser feita tanto usando metadados (nome do arquivo, extensão, data de modificação, etc), como usando o próprio conteúdo do arquivo. Os sistemas operacionais da Microsoft, por exemplo, possuem a opção de indexar o conteúdo de texto de alguns tipos de arquivos (MICROSOFT, 2017).

4.1.1 Lucene

Segundo a Fundação Apache (2017), Lucene é uma ferramenta de busca com alta performance escrita em Java. Lucene usa licença Apache versão 2 e traz diversas funcionalidades e vantagens que são interessantes para o projeto, tais como:

- Exige pouca RAM (apenas 1MB).
- Ocupa pouco espaço: em média entre 20% e 30% do tamanho do texto.
- É escalável.
- Possui recursos de rankeamento dos resultados da busca.
- Suporta *highlight*, *joins* e agrupamento de resultados.
- Além disso, conforme apontado por Michael McCandless (2010), Lucene não possui dependências e possui um tamanho de aproximadamente 1 MB.

A biblioteca Lucene faz a indexação e permite criar diversos índices diferentes. Cada um desses índices pode se tornar um arquivo. Isto é uma funcionalidade interessante, pois permite a criação de estruturas que podem tornar as buscas mais eficientes no caso do uso de

filtros, ao mesmo tempo que permite a junção de índices quando as buscas são mais abrangentes. Isto também permite um melhor controle sobre o acesso aos dados.

Suponhamos, por exemplo, que fosse feito um índice para controlar os arquivos dos Institutos Federais de São Paulo: poderia ser feito o controle de qual documento pertence a qual Câmpus por uma estrutura de pastas, sem a necessidade de colocar esta informação no documento. O uso desta estratégia permitiria reduzir a complexidade do sistema e aumentaria a velocidade de suas buscas, pois não seria necessário realizar esta verificação em todos os arquivos.

4.1.2 Elasticsearch

De acordo com Gormley e Tong (2015), o Elasticsearch é uma ferramenta código aberta, com sua construção feita com base no Lucene. Sua licença é Apache (versão 2). Segundo o autor, o Elasticsearch possui uma complexidade de uso menor que o Lucene, e possui recursos que o Lucene, por ser uma biblioteca não possui. Dentre estes recursos está a capacidade de criar um *cluster* distribuído capaz de comunicar com a aplicação por meio de uma API RESTful.

Os registros indexados pelo Elasticsearch são chamados documentos, e assim, todos os campos deste documento são indexados e buscáveis. É possível definir um campo como *not_analyzed*, e neste caso ele só será encontrado caso a expressão buscada seja idêntica à que está associada ao documento.

A forma mais comum de fazer buscas no Elasticsearch são realizando *queries* escritas em uma linguagem de busca específica do Elasticsearch. Essa linguagem é bastante rica e permite fazer buscas bastante personalizadas. Ela é escrita em JSON e é chamada de *Query dsl*.

Um conceito interessante aplicado nas buscas é o uso de *analyzers*, que definem como as *strings* serão tratadas. Um dos itens mais importantes que compõem o *analyzer*, é o *tokenizer*, o qual Gormley e Tong denominam as suas funções como as seguintes:

- *Character filters*: normaliza o conteúdo da string antes de passar para os próximos passos.
- *Tokenizer*: quebra a *string*, transformando-a em termos individuais. Essa quebra, por padrão, é feita utilizando espaço em branco e pontuações.

- *Token filters*: última etapa, nela é aplicado algumas ações, como por exemplo transformar todos caracteres para minúsculo, remover termos (tais como a, e, o, que são usados apenas para ligação) e adição de termos (adição de sinônimos, por exemplo).

O Elasticsearch permite que você crie seu próprio *analyzer*. E segundo a Elastic (2017), desenvolvedora do Elasticsearch, ele possui um analyzer adequado para o português brasileiro.

Segundo Gormley e Tong o Elasticsearch pode calcular uma pontuação, para o quanto o documento dá *match* com o termo buscado. E para calcular esta pontuação ele leva em consideração o número de ocorrências do termo, qual a frequência em que ele é encontrado (termos que aparecem mais frequentemente possuem pontuação **menor**) e qual o tamanho do termo (termos maiores possuem uma pontuação **maior**).

Um outro recurso importante que os autores definem e o Elasticsearch implementa é o “*fuzziness*”, que eles definem como o fato de considerar como a mesma palavra quando elas são muito semelhantes. O uso deste recurso permite que pequenos erros de escrita possam ser desconsiderados na busca. Por exemplo, se quiséssemos buscar o termo “vestibulinho”, mas ao invés disso buscássemos “vestibolinho”, o Elasticsearch também poderia trazer os casos que usam o termo “vestibulinho” uma vez que eles são semelhantes.

4.1.3 Tika

O Tika é uma ferramenta, sobre licença Apache (versão 2), que atua na identificação do tipo do arquivo e na recuperação de seus dados de texto e metadados (MATTMANN e ZITTING, 2012).

O Tika consegue identificar o tipo do arquivo analisando seus metadados. Assim, mesmo que o usuário tenha renomeado o arquivo, trocando sua extensão, ele ainda conseguirá identificar o tipo do arquivo.

O Tika consegue recuperar o conteúdo de texto das seguintes extensões:

- HTML
- XML e formatos derivados
- Documentos do Microsoft Office
- Formato OpenDocument
- PDF (utilizando a biblioteca Apache PDFBox)
- RTF

- E outros que não serão mencionados, por não serem relevantes para o projeto.

O Tika também suporta o uso de extratores personalizados, o que permite por exemplo passar um extrator personalizado que analisa um tipo de dado que não é extraído pelo extrator de fábrica. Por exemplo, no projeto é utilizado um extrator de PDF personalizado, que permitiu a aplicação do OCR no documento para extração de conteúdos digitalizados e texto em imagens.

4.1.4 Tesseract

De acordo a documentação oficial do projeto (2017?), o Tesseract é uma biblioteca para C e C++, sobre licença Apache (versão 2), que atua na leitura OCR de imagens. Apesar de não ser escrito em Java, ele pode ser usado a partir de bibliotecas que o encapsulam, como o Tess4J, por exemplo.

A leitura OCR consiste na recuperação de conteúdo de texto em imagens. Assim, ele será utilizado junto ao Tika para recuperação do conteúdo de texto.

O Tesseract suporta as seguintes extensões: JPG, GIF , PNG, BMP e TIFF.

Ademais, o Tesseract suporta a configuração para que o resultado seja mais eficiente para determinados casos (para a língua portuguesa, por exemplo) e possui suporte para receber treinamentos, sendo assim capaz de se especializar.

4.2 Spring

O Spring é um importante *framework* Java que tem como uma de suas principais funcionalidades a injeção de dependência. O Spring possui diversos módulos que reduzem consideravelmente o esforço para o desenvolvimento e manutenção. Seu uso também garante compatibilidade e facilita a integração entre módulos importantes da aplicação. Entre os módulos mais relevantes para este projeto estão:

- *Spring Security*: Adiciona segurança nos recursos protegidos
- *Spring Data Elasticsearch*: Permite a integração com o Elasticsearch, sem a necessidade de fazer a requisição manualmente. Ele também traz os métodos CRUD por padrão, sem a necessidade de o desenvolvedor implementar. Além disso, gera

o índice automaticamente de acordo a modelagem feita na classe que representa a entidade na aplicação.

- *Spring boot*: A importação dos módulos Spring por ele, garante a compatibilidade entre os módulos importados (SPRING, 2017). Além disso ele traz a funcionalidade de gerar um *jar* com o servidor web embarcado. Assim é possível distribuir apenas o *jar*, sem se preocupar com a configuração do servidor de aplicação.

4.3 Groovy

Segundo König e King (2015), o Groovy é uma linguagem opcionalmente tipada, desenvolvida para Java e que foi inspirada em linguagens como Python, Ruby e Smalltalk.

Ela é uma linguagem que dá mais flexibilidade que o Java, permitindo o aumento da produtividade, por dispor de algumas funcionalidades que não estão disponíveis no Java, como por exemplo pelo uso de *closures*, que permite simplificar a escrita de códigos, e o os *getters* and *setters* que já possuem uma implementação automática (que pode ser sobrescrita).

Ele pode ser usado junto ao Java, e sua compilação é completamente compatível com a *Java Virtual Machine* (JVM).

4.4 React

O React é um framework javascript criado pelo Facebook para gerenciar telas que envolvem grandes volumes de dados e frequentes mudanças de estado (BYRON, 2013).

Seus módulos são escritos de forma que incentiva a componentização dos itens da tela para o reaproveitamento de código.

Sua escrita geralmente é feita aplicando o ECMAScript5 ou ECMAScript6 (ECMA, 2017), uma padronização utilizada para diversas linguagens, entre elas, o javascript. Os próprios exemplos de uso no site oficial da biblioteca, são feitos usando este padrão.

Entretanto, muitos recursos disponíveis nesses padrões, não estão disponíveis para serem usados em todos os navegadores, e alguns deles sequer tem algum navegador que suporte. Por isso é necessário fazer a transpilação do código. O Mozilla Developer Tools (2017), define a transpilação como algo diferente da compilação, uma vez que o código não é transformado

em linguagem de máquina, mas é transformado em um nível de abstração diferente na mesma linguagem. Por exemplo: o código “ `() => { ... }` ” poderia ser transpilado para “ `function() { ... }` ”. Deste modo, é possível escrever utilizando os padrões ECMAScript5 ou ECMAScript6, e gerar uma versão compatível com os navegadores atuais.

O Mozilla Developer Tools, recomenda o uso compilador **babel** (BABEL, 2017) para fazer a transpilação do código javascript.

Além do babel, é comum utilizar o webpack (WEBPACK, 2017) e alguns *plugins* para automatizar processos de desenvolvimento no *front end* tais como unificar todos os códigos em um único arquivo, gerar versão minificada, transpilar o código usando o babel, etc.

A empresa ThoughtWorks, consultora global em tecnologia de informação com foco no desenvolvimento ágil, recomendou em 2016 a adoção do React (THOUGHTWORKS, 2016), por o mesmo não adotar o “*two-way data binding*”, que é um recurso presente em grande parte dos *frameworks* para *front end* e que está associado a diversos problemas de performance.

4.5 Avaliação do sistema de busca

Segundo Yang (entre 2000?), a precisão, cobertura e medida F são métricas para medir a eficiência de uma busca e podem ser aplicados em buscas ranqueadas ou não.

A precisão pode ser obtida aplicando a equação logo abaixo:

$$Precisão = \frac{\text{documentos relevantes}}{\text{documentos encontrados}}$$

A cobertura pode ser calculada utilizando a seguinte equação:

$$Cobertura = \frac{\text{documentos relevantes}}{\text{documentos relevantes encontrados}}$$

A medida F é calculada aplicando a seguinte equação:

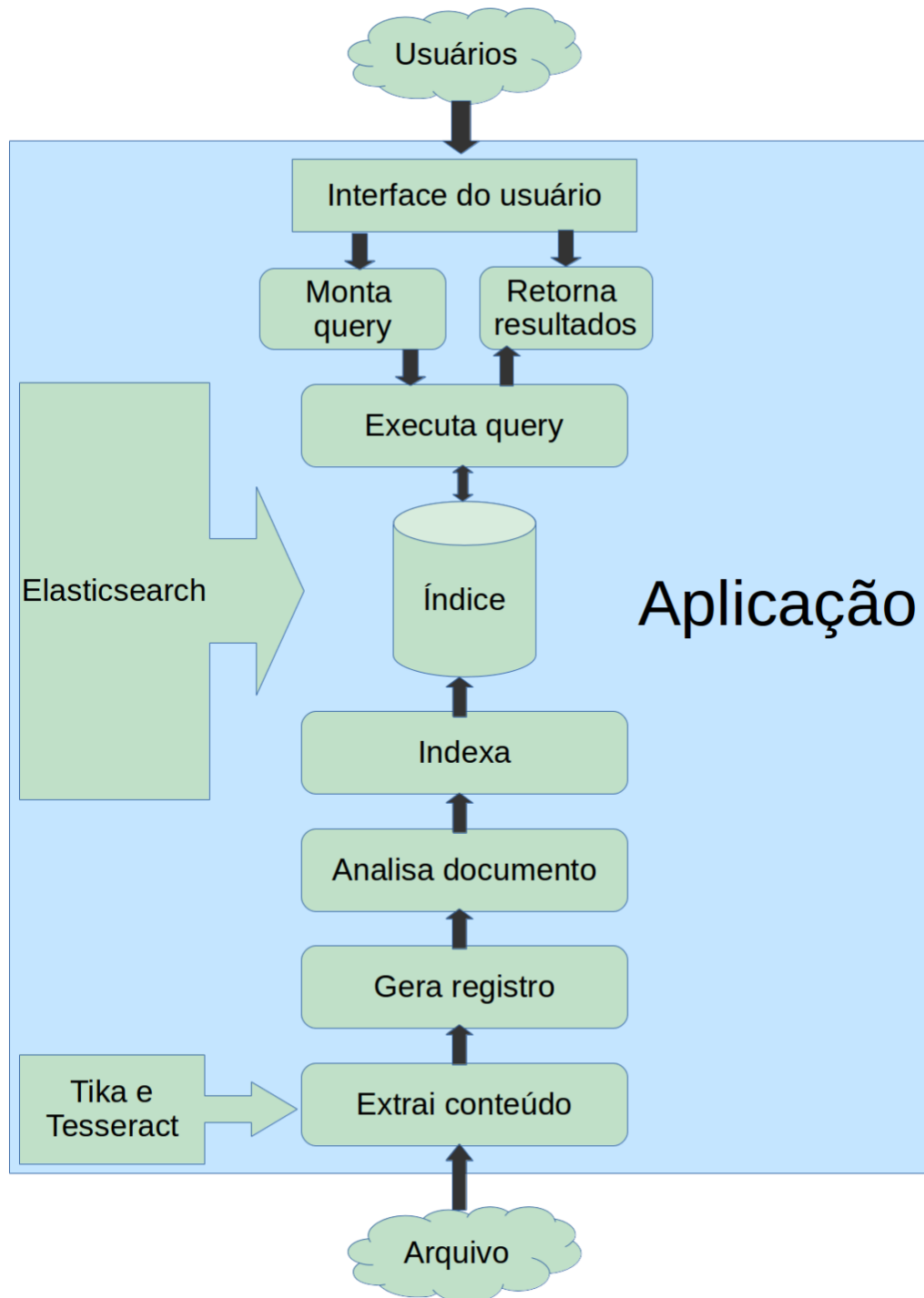
$$Medida F = 2 \times \left(\frac{\text{precisão} \times \text{cobertura}}{\text{precisão} + \text{cobertura}} \right)$$

De acordo com Costa (2008), a precisão é a capacidade de recuperar somente referências relevantes, eliminando as que não possuem relevância. Já a cobertura busca mensurar a capacidade de encontrar os resultados relevantes. Já Powers (entre 2014 e 2017), define a medida F como uma harmônica ponderada entre a precisão e cobertura.

5 MÉTODO

A Figura 1 traz uma visão geral que mostra as principais responsabilidades do sistema e como as ferramentas utilizadas no desenvolvimento da aplicação são utilizadas:

Figura 1 – Visão geral do sistema e responsabilidade dos módulos



Como pode ser percebido na Figura 1 a indexação é iniciada a partir de um arquivo que é recuperado pelo sistema para ser indexado. O Tika e o Tesseract atuam na extração de seu conteúdo de texto. O texto extraído é utilizado para gerar o registro que então é submetido para ser indexado pelo Elasticsearch.

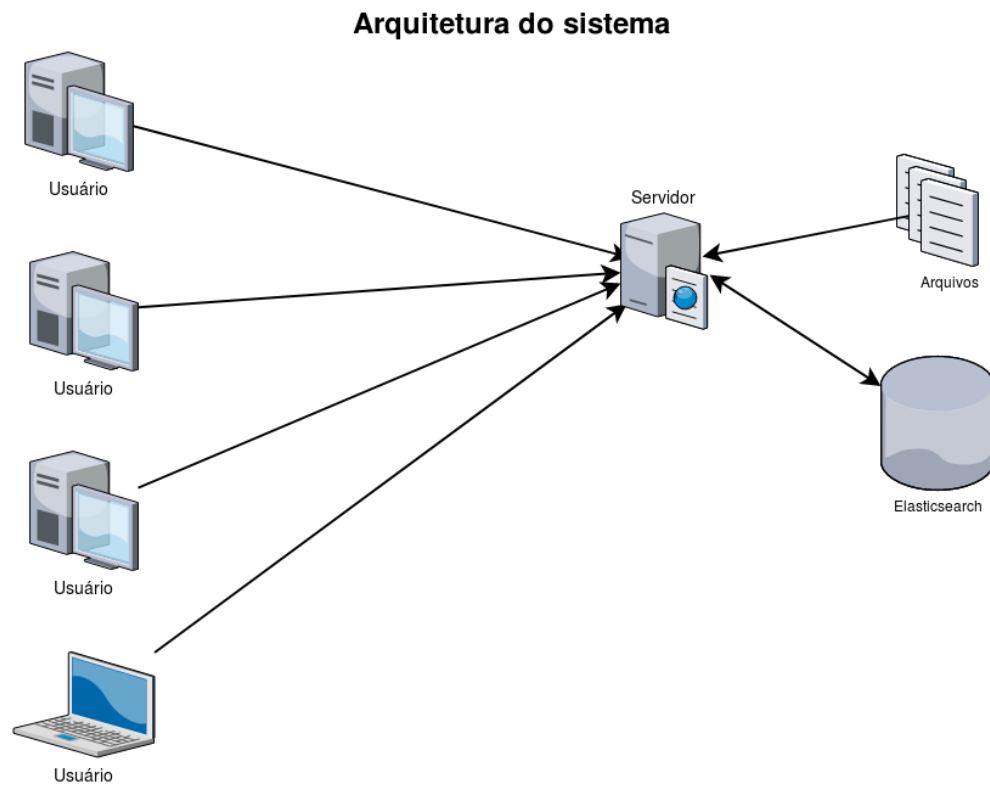
Uma vez que o conteúdo é indexado ele fica disponível para busca pelos usuários do sistema. Os usuários utilizam o sistema a partir de um *front end* desenvolvido para este propósito. Uma vez que as requisições feitas por um usuário autenticado chegam à aplicação, elas são processadas, e caso sejam uma busca por arquivo, geram uma query que é aplicada ao Elasticsearch. Essa *query* é então executada, e seu resultado é retornado à interface do usuário, onde então ele tem a opção de visualizar trechos em que o conteúdo foi encontrado, e caso queira ele pode realizar o download do arquivo.

As requisições de busca e listagem também podem ser feitas por outras aplicações que estejam devidamente autenticadas, uma vez que o resultado retornado pela API é no formato de JSON. Os tópicos abaixo, descrevem em detalhes a implementação do sistema.

5.1 Arquitetura do sistema

O sistema roda como uma única aplicação. Portanto, ele é responsável pela indexação do conteúdo monitorado, disponibilização da API de busca e disponibilização do lado cliente. A Figura 2 traz uma representação de sua arquitetura:

Figura 2 – Arquitetura do sistema



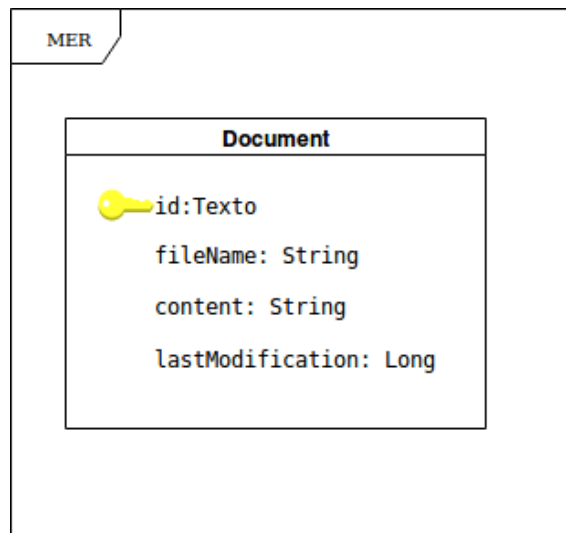
Fonte: Elaborado pelo autor (2017)

Como pode ser visto na Figura 2, tanto o sistema de indexação, quanto o *front end* são executados como uma única aplicação. Foi definido neste formato, uma vez que a divisão em duas aplicações separadas, aumentaria consideravelmente a complexidade do projeto.

5.2 Persistência dos dados

Os dados são persistidos no Elasticsearch com a estrutura da Figura 3:

Figura 3 – Estrutura do Documento



Fonte: Elaborado pelo autor (2017)

Conforme pode ser percebido na Figura 3, o id é do tipo texto. Seu conteúdo é o próprio *path* do arquivo. Este dado é do tipo “*not_analyzed*”, ou seja, ele deve ser buscado pelo conteúdo exato. Isto foi necessário, pois a marcação padrão do Elasticsearch, usa um *tokenizer* que faz com que o conteúdo entre cada barra fosse uma palavra diferente. Este comportamento impedia a checagem se o arquivo existia ou não.

Deste modo, o id representa o *path* do arquivo, o *fileName*, o seu nome, o *content* o conteúdo indexado, e por fim, *lastModification*, a data alteração no arquivo.

O repositório utilizado foi o Elasticsearch 2.4.6. Não foi utilizado a versão mais recente, pois a versão do Spring boot utilizada não garantia compatibilidade com as versões mais novas (SPRING, 2017).

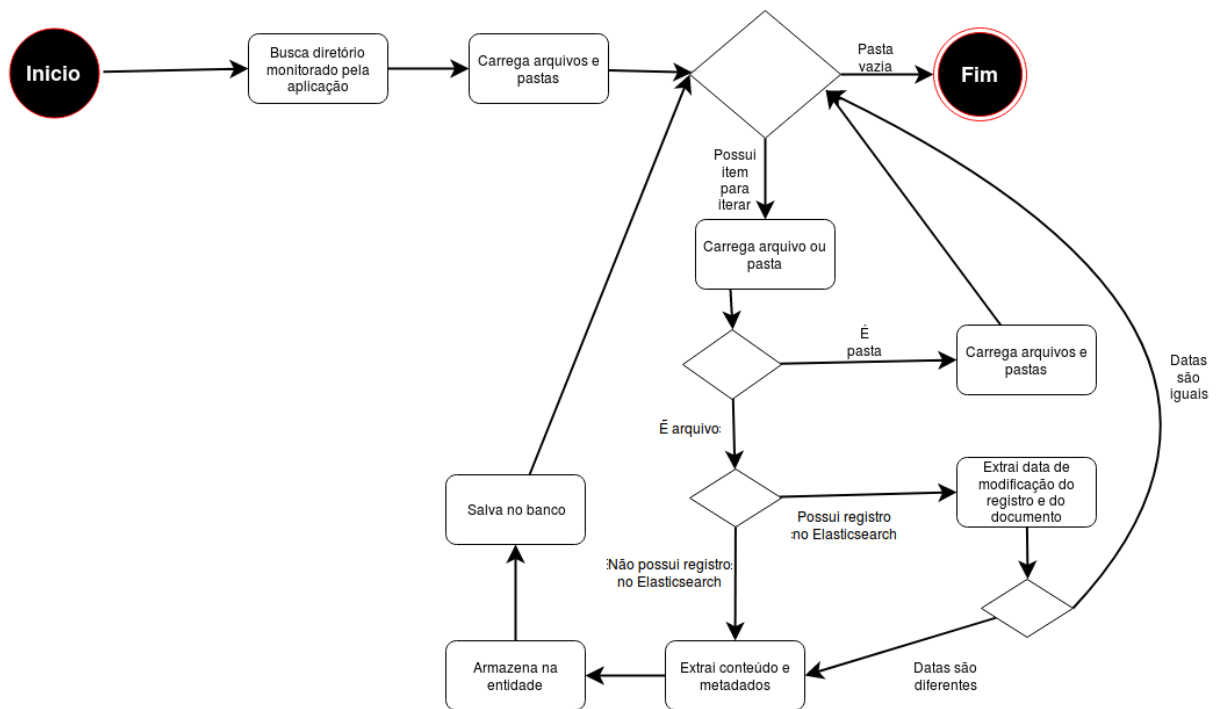
5.3 Implementação do lado servidor

O sistema foi desenvolvido, utilizando a linguagem Groovy, com o gerenciamento das dependências feito pelo Gradle. Como a aplicação utilizou o Spring boot, a injeção das dependências é feita por ele, e o gerenciamento da comunicação com o Elasticsearch é feita

pelo módulo Spring Data Elasticsearch, que foi importado como sub-dependência do Spring boot.

A extração dos dados do arquivo para serem indexados, podem ser inicializadas de duas formas: pela API, ou de forma automática a partir de um agendamento. O algoritmo utilizado para indexação pode ser representado pela figura 4:

Figura 4 – Algoritmo de indexação



Fonte: Elaborado pelo autor (2017)

Conforme pode ser percebido na Figura 4, o sistema busca os arquivos recursivamente na pasta e os indexa.

O sistema indexa todos os arquivos que estão na pasta monitorada, uma vez que há uma rotina que verifica a cada trinta minutos se há algum conteúdo novo para ser indexado. Essa rotina foi feita habilitando o agendamento na classe onde está o main (método que inicia o sistema) e anotando o método que executa a rotina com a anotação `@Scheduled`. Os arquivos que sofreram alteração também são atualizados.

O tempo das extrações feitas pelo Tika foi afetado pela aplicação do Tesseract. A aplicação do Tesseract em PDFs não é o comportamento padrão do Tika, que só aplica o OCR em arquivos que são explicitamente de imagens. A mudança desse comportamento foi feita adicionando esse extrator diretamente a classe `ParseContext` na inicialização de um serviço do sistema. Entretanto, este pequeno incremento de tempo não inviabiliza sua utilização.

A escolha destas duas ferramentas se justifica pela relevância e maturidade que as comunidades delas possuem, e pelas funcionalidades que elas disponibilizam, conforme foi descrito nos Tópicos 4.1.3 e 4.1.4.

Inicialmente foi considerado utilizar o Lucene diretamente como motor de busca e indexação, entretanto sua configuração é complexa e o seu acoplamento com a aplicação, fizeram o Elasticsearch se mostrar uma melhor opção.

O Elasticsearch utiliza o Lucene, em sua implementação, para fazer as buscas, e ao contrário do Lucene, o Elasticsearch permite ser utilizado como serviço, o que o torna desacoplado da aplicação e permite que ele possa ser escalado e personalizado de forma independente da aplicação. O Elasticsearch possui funcionalidades que são adequados ao sistema, por exemplo, permite que sejam aceitas variações do texto considerando o vocabulário da língua portuguesa e os *highlights* são úteis para que o usuário possa analisar se o documento é realmente o que deseja.

5.4 Implementação do *front end*

A implementação do lado cliente foi feita utilizando a biblioteca React, usando o javascript com o padrão ECMAScript 6. A arquitetura do lado cliente foi fortemente influenciada pelo uso do Mobx (Mobx, 2017), uma dependência que permite fazer as notificações de mudança para o virtual DOM do React. Nessa arquitetura, os dados costumam ficar centralizados na *store*.

O fato de o código ter sido escrito utilizando o padrão ECMAScript 6 faz com que a aplicação tenha problemas de ser executada na maioria dos navegadores, pois muitas funcionalidades que estão presentes nesse padrão ainda não estão implementadas nos navegadores. Deste modo, foi necessário utilizar um transpilador para converter o código para uma versão distribuível e utilizável pelos navegadores. A biblioteca que ficou responsável por isso, foi o *babel*.

Por o processo de transpilação ser bastante repetitivo e trabalhoso, foi utilizado o *webpack*, ao qual foi adicionado alguns plugins, que permitiram a automatização da transpilação e a cópia para o endereço onde o js transpilado seria distribuído.

O uso do *webpack* também permitiu que o css e html do projeto pudessem estar em uma pasta monitorada pelo git. Caso não fosse feito desta forma, os arquivos teriam que ser adicionados de forma forçada a cada necessidade de *commit* destes arquivos, pois a pasta onde

fica os arquivos distribuíveis para o lado cliente está marcada para ser ignorada, uma vez que parte dos arquivos são gerados automaticamente, e por isso não fazem sentido serem *commitados*.

5.5 Configuração do ambiente

Para o desenvolvimento do sistema, foi utilizado um computador Linux Mint 18 - 64 bits, com os seguintes recursos:

- Java 1.8_151
- Groovy 2.4.9
- Gradle 4.3.1

O lado servidor utiliza o Gradle para gerenciamento das dependências, as quais são:

- Tika-core 1.16
- Tika-parsers 1.16
- Tesseract 3.04.01 (instalado como cliente na máquina)
- Spring-boot-starter-web 1.5.8-RELEASE
- Spring-data-elasticsearch
- Spring-actuator
- Spring-security
- jai-imageio-jpeg2000
- Spring gradle plugin
- jbig2

O lado cliente utilizou as seguintes dependências javascript:

- date-fns 1.29.0
- history 4.7.2
- mobx 3.3.1
- mobx-react 4.3.4
- react 16.1.1
- react-dom 16.1.1
- react-router 4.2.0
- syntec-apollo-11 0.4.20

- uuid 3.1.

Além disso, o lado cliente utilizou as seguintes dependências de desenvolvimento:

- Node 8.9.1
- babel-core 6.26.0
- babel-eslint 8.0.2
- babel-loader 7.1.2
- babel-plugin-transform-decorators-legacy 1.3.4
- babel-preset-es2015 6.24.1
- babel-preset-es2016 6.24.1
- babel-preset-react 6.24.1
- babel-preset-stage-0 6.24.1
- copy-webpack-plugin 4.2.1
- css-loader 0.28.7
- eslint 4.11.0
- eslint-config-prettier 2.7.0
- eslint-plugin-import 2.8.0
- eslint-plugin-prettier 2.3.1
- eslint-plugin-react 7.5.0
- html-webpack-plugin 2.30.1
- prettier-eslint 8.2.2
- prettier-eslint-cli 4.4.0
- style-loader 0.19.0
- webpack 3.8.1
- webpack-dev-server 2.9.4

E por fim, como repositório, foi utilizado o Elasticsearch 2.4.6.

Para que o sistema rode em um ambiente de produção, ele necessita dos seguintes recursos instalados no servidor:

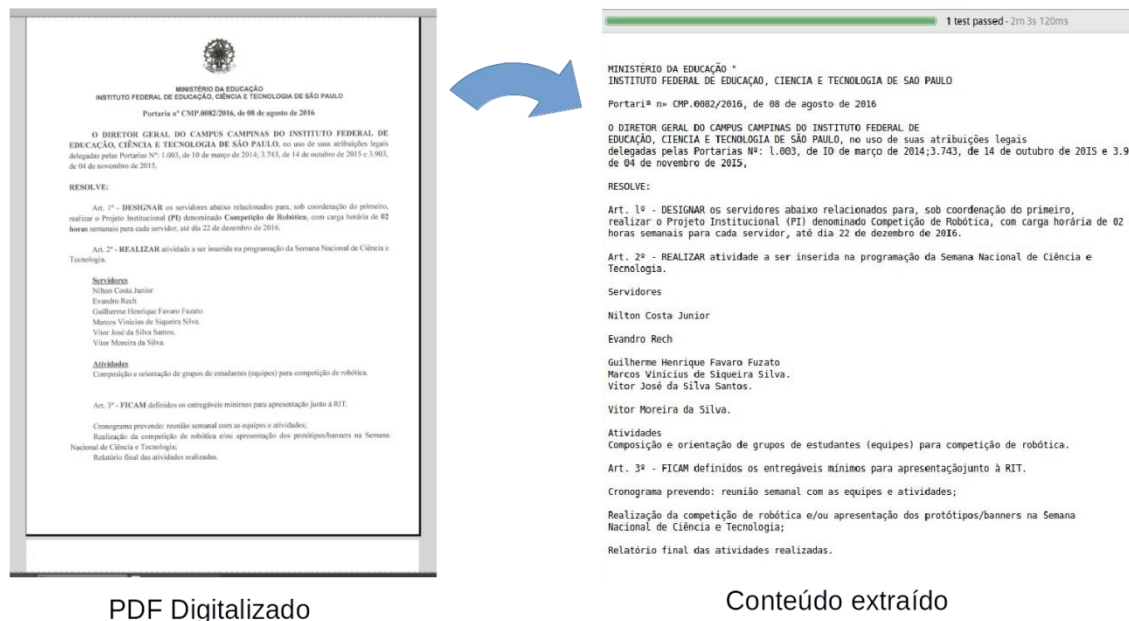
- Java 8
- Tesseract

Além disso a aplicação necessita do Elasticsearch, que pode estar instalado no mesmo servidor que a aplicação, ou em um servidor separado.

6 RESULTADOS OBTIDOS

Como resultado obtido, pode se considerar a extração de conteúdo digitalizado pelo Tika, conforme mostra a Figura 5:

Figura 5 – Extração de documento digitalizado



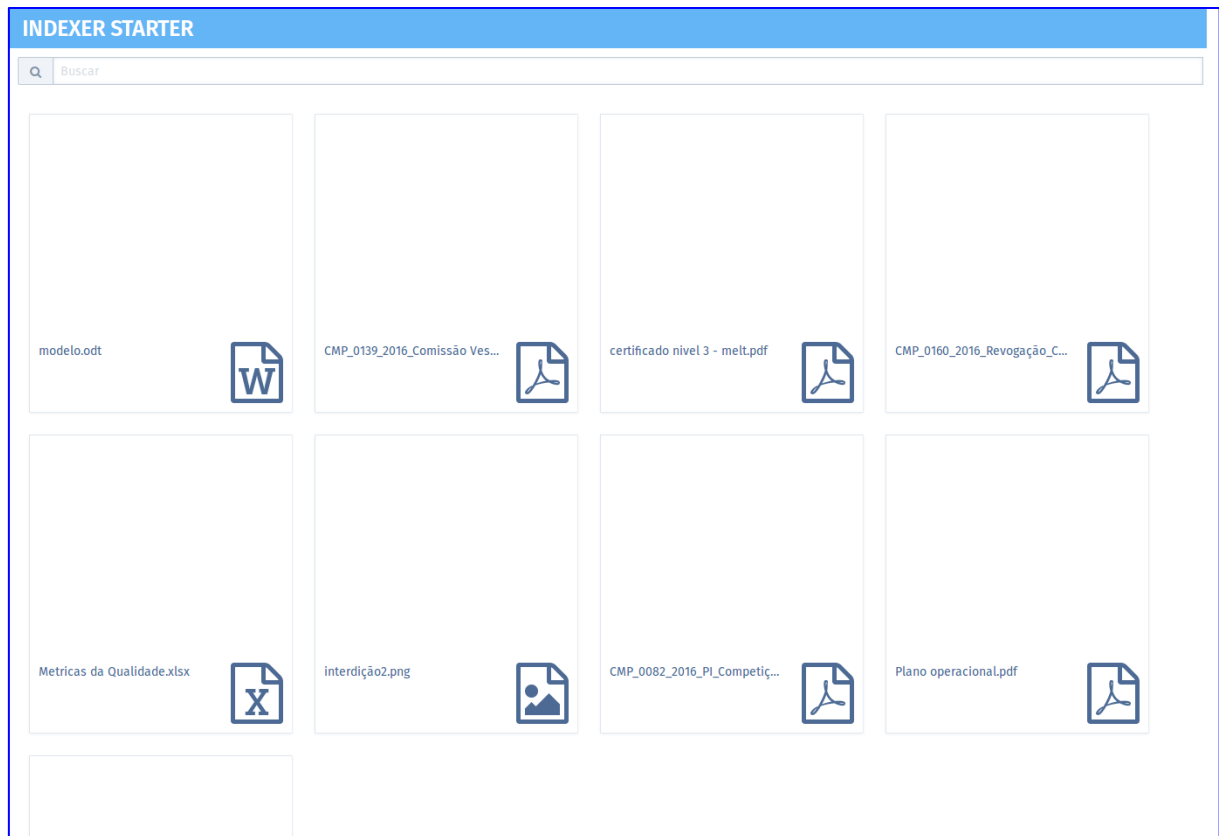
Fonte: Elaborado pelo autor (2017)

Outro resultado obtido é a API da aplicação, que possui os seguintes *end points*:

- *api/files/*: traz os documentos indexados não filtrados
- *api/files/download/{path}*: faz download do arquivo requisitado
- *api/files/refresh*: executa rotina que indexa novos arquivos e verifica se existe modificação nos existentes
- *api/files/search*: realiza busca em documentos por conteúdo. O valor a ser filtrado deve ser passado como parâmetro
- Outros *endpoints* criados pelo Spring actuator para coleta de informações da aplicação, que permite verificar a saúde e o status da aplicação.

Além da API também há a interface web desenvolvida para consumo dos dados indexados. Pela página é possível ver os arquivos, pesquisar por conteúdo e fazer download dos mesmos.

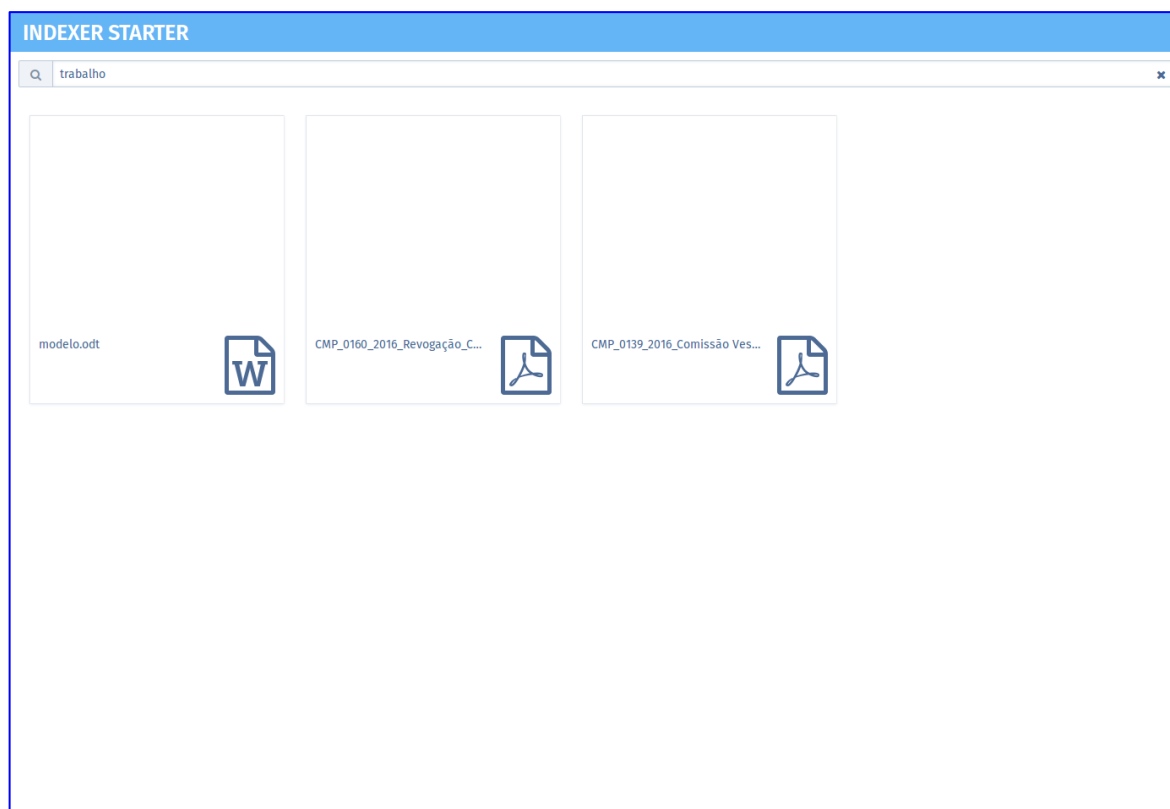
Figura 6 – Página inicial



Fonte: Elaborado pelo autor (2017)

A Figura 6 mostra a página inicial do sistema, onde são listados os arquivos que estão salvos no servidor. As buscas por conteúdo dos arquivos podem ser feitas pela caixa de busca com ícone de lupa na parte superior da página. Os arquivos são representados pelos retângulos, com o nome do arquivo no canto inferior esquerdo, e seu tipo no canto inferior direito. A Figura 7 mostra a reação da tela ao digitar alguma coisa na caixa de busca:

Figura 7 – Exibição de conteúdo filtrado

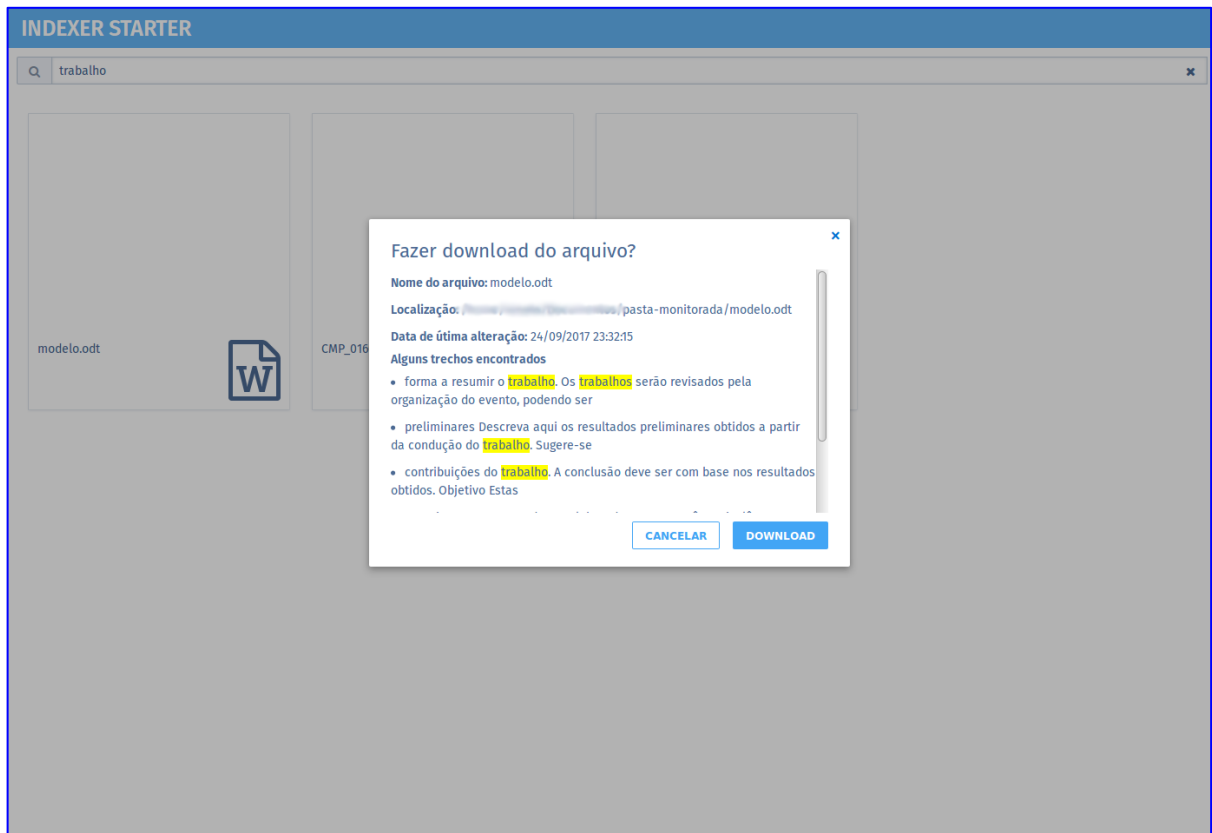


Fonte: Elaborado pelo autor (2017)

A Figura 7 mostra os arquivos filtrados pelo termo “trabalho”. Os resultados encontrados são listados logo abaixo da caixa de busca.

A Figura 8 traz o modal que é exibido quando o usuário seleciona um dos arquivos encontrados na busca. Ele traz o conteúdo de texto sublinhado e dá a opção de o usuário fazer o download do arquivo em sua máquina.

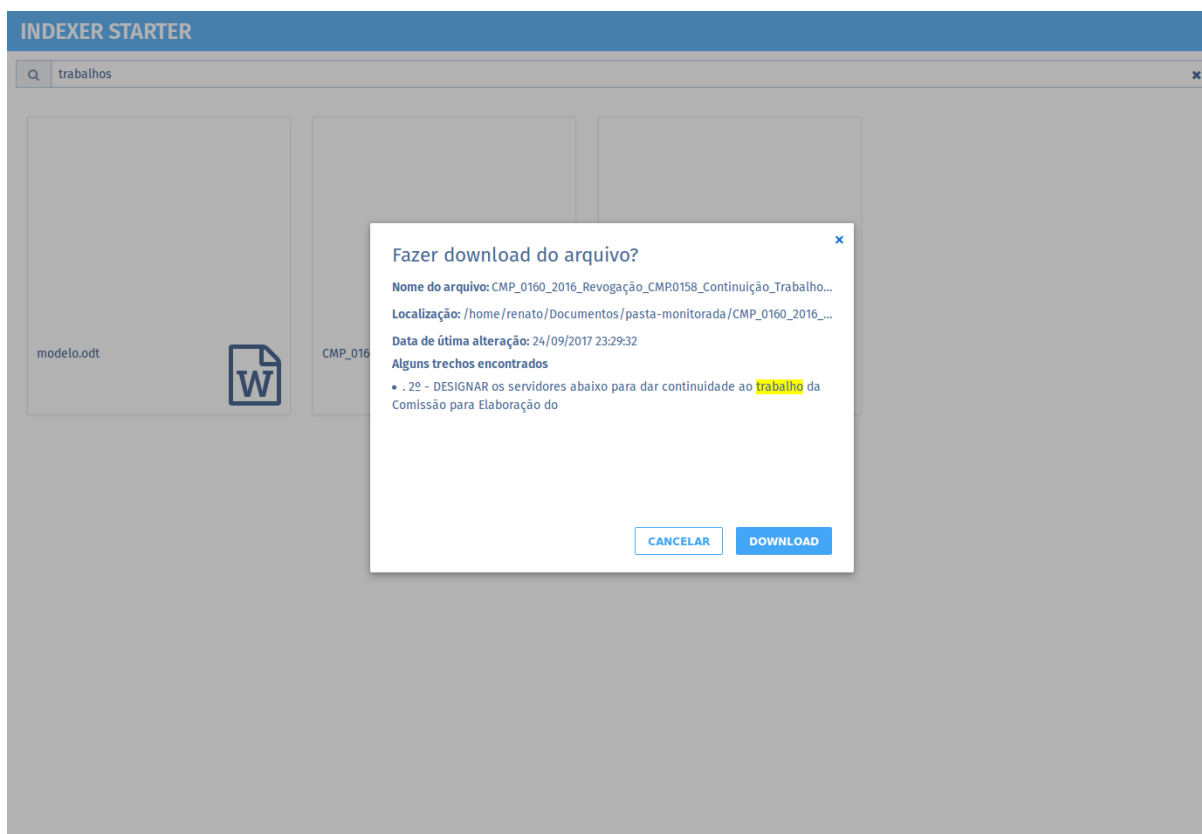
Figura 8 – Download do arquivo



Fonte: Elaborado pelo autor (2017)

E conforme descrito na metodologia, a busca no Elasticsearch aplica algumas variações da palavra, considerando o idioma utilizado como *analyzer*, o que pode ser verificado na Figura 9:

Figura 9 – Exibição de conteúdo filtrado



Fonte: Elaborado pelo autor (2017)

Buscando avaliar as métricas de eficiência da busca do sistema, foram feitos testes comparativos usando o Google Drive na versão gratuita, uma vez que possui funcionalidades semelhantes ao sistema desenvolvido. Foi utilizada uma massa de 43 arquivos, sendo 8 deles digitalizados. A massa foi composta pelas portarias disponíveis para o ano de 2017, no dia 13 de dezembro de 2017. Desse modo a massa foi composta pelas portarias da Secretaria de Saúde de São Paulo, ARTESP e Instituto Federal campus Campinas.

Foram realizados cinco casos de testes sobre a mesma massa, e utilizando os mesmos termos. Os resultados obtidos com o sistema desenvolvido neste projeto estão nas Tabelas 1, 2, 3, 4 e 5:

Tabela 1 – Resultado do teste com o termo “criação de fluxograma”

Precisão (%)	Cobertura (%)	Medida F (%)
33	100	50

Fonte: Elaborado pelo autor (2017)

Tabela 2 – Resultado do teste com o termo “coordenação do curso”

Precisão (%)	Cobertura (%)	Medida F (%)
50	100	67

Fonte: Elaborado pelo autor (2017)

Tabela 3 – Resultado do teste com o termo “pregão ar condicionado”

Precisão (%)	Cobertura (%)	Medida F (%)
100	100	100

Fonte: Elaborado pelo autor (2017)

Tabela 4 – Resultado do teste com o termo “convocação concurso público”

Precisão (%)	Cobertura (%)	Medida F (%)
67	100	80

Fonte: Elaborado pelo autor (2017)

Tabela 5 – Resultado do teste com o termo “compra campinas”

Precisão (%)	Cobertura (%)	Medida F (%)
29	100	44

Fonte: Elaborado pelo autor (2017)

A média dos resultados é demonstrada na Tabela 6:

Tabela 6 – Média das métricas no sistema desenvolvido

Precisão (%)	Cobertura (%)	Medida F (%)
56	100	68

Fonte: Elaborado pelo autor (2017)

Os resultados das buscas usando o Google Drive estão listadas nas Tabelas 7, 8, 9, 10 e 11:

Tabela 7 – Resultado do teste com o termo “criação de fluxograma”

Precisão (%)	Cobertura (%)	Medida F (%)
100	100	100

Fonte: Elaborado pelo autor (2017)

Tabela 8 – Resultado do teste com o termo “coordenação do curso”

Precisão (%)	Cobertura (%)	Medida F (%)
67	67	67

Fonte: Elaborado pelo autor (2017)

Tabela 9 – Resultado do teste com o termo “pregão ar condicionado”

Precisão (%)	Cobertura (%)	Medida F (%)
100	100	100

Fonte: Elaborado pelo autor (2017)

Tabela 10 – Resultado do teste com o termo “convocação concurso público”

Precisão (%)	Cobertura (%)	Medida F (%)
100	75	86

Fonte: Elaborado pelo autor (2017)

Tabela 11 – Resultado do teste com o termo “compra campinas”

Precisão (%)	Cobertura (%)	Medida F (%)
0	0	0

Fonte: Elaborado pelo autor (2017)

A média dos resultados está na Tabela 12:

Tabela 12 – Média das métricas no sistema desenvolvido

Precisão (%)	Cobertura (%)	Medida F (%)
67	60	70

Fonte: Elaborado pelo autor (2017)

Como pode ser observado comparando a Tabela 12 com a Tabela 6, o Google Drive, obteve melhor pontuação nos quesitos precisão e medida F. Entretanto, o sistema desenvolvido teve melhor resultado no quesito cobertura.

Parte deste resultado se deve ao fato de o sistema desenvolvido trouxe mais resultados que não são relevantes (apesar de os mais relevantes aparecerem entre os primeiros resultados); e por conta deste fato, a medida F também ficou abaixo dos testes executados no Google Drive.

No quesito cobertura, é observável uma diferença de aproximadamente 40% entre o sistema desenvolvido e o Google Drive. Enquanto no quesito precisão é possível observar que o Google Drive foi 16% mais eficiente. Por fim, na medida F, há uma vantagem de 3% do Google Drive em relação ao sistema desenvolvido.

7 CONCLUSÃO

No projeto foi desenvolvido um sistema web para guarda, indexação e busca de documentos por conteúdo. Ele foi desenvolvido com o intuito de agilizar a busca por arquivos, permitindo ao usuário realizar busca por conteúdo. Para tal, foi definido os objetivos específicos, os quais foram: realizar o controle de acesso por usuário, indexar o conteúdo em formatos legíveis por máquina, indexar conteúdo de texto encontrado nas imagens, em arquivos do tipo PDF, realizar a busca de arquivos por conteúdo, disponibilizar um *front-end* para utilização do sistema.

Para realizar a extração do conteúdo de texto dos arquivos, foi utilizado o Tika junto ao Tesseract. O uso destas ferramentas atingiu o resultado esperado, uma vez que elas extraíram o conteúdo de texto dos formatos que foram definidos nos objetivos específicos (Tópico 2.2).

O *front end* criado para consumo da aplicação também foi desenvolvido com sucesso, permitindo ao usuário realizar buscas por conteúdo e realizar o download de arquivos que estão indexados.

As métricas sobre a eficiência de busca obtidas demonstraram que o sistema possui diferenciais que podem ser úteis e demonstraram que ele possui eficácia adequada para ser utilizado.

O sistema também implementou a segurança de forma a restringir o acesso ao sistema apenas aos usuários autorizados.

Deste modo, pode-se assumir a aplicação obteve sucesso, no objetivo geral e obteve resultados compatíveis com os objetivos específicos definidos.

Como trabalhos futuros, poderia ser adicionado o filtro por data de modificação, melhoria no algoritmo de segurança para que ele seja mais flexível e otimizações na *query* para que ela obtenha resultados mais precisos. A aplicação do sistema em um ambiente real certamente seria um trabalho futuro interessante para melhoria da interface e implementação de novas funcionalidades, de acordo com a necessidade dos usuários.

REFERENCIAS

BABEL. **Babel is a JavaScript compiler**. 2017. Disponível em: <<https://babeljs.io/>>. Acesso em: 13 dez. 2017.

BYRON, Lee. **How is Facebook's React JavaScript library? How does it compare with other popular JavaScript libraries?** 2013. Disponível em: <<https://www.quora.com/React-JS-Library/How-is-Facebooks-React-JavaScript-library-How-does-it-compare-with-other-popular-JavaScript-libraries/answer/Lee-Byron?srid=3DcX>>. Acesso em: 13 dez. 2017.

COSTA, Belzik. **Revocação (recall) e precisão (precision) no processo de recuperação de informação da biblioteca do icex da UFMG através da amostra do acervo de teses e dissertações**. UFMG, 2008. 18-20 p. Disponível em: <<http://www.bibliotecadigital.ufmg.br/dspace/handle/1843/BUBD-9AXNLD>>. Acesso em: 15 dez. 2017.

DOCUMENTAÇÃO DO DESENVOLVEDOR. **Tesseract OCR**. 2017?. Disponível em: <<https://github.com/tesseract-ocr/tesseract>>. Acesso em: 20 nov. 2017.

DOCUMENTAÇÃO DO DESENVOLVEDOR. **Tesseract OCR**. 2017?.. Disponível em: <<https://github.com/tesseract-ocr/tesseract/blob/master/doc/tesseract.1.asc>>. Acesso em: 20 nov. 2017.

DOCUMENTAÇÃO DO DESENVOLVEDOR. **Tesseract(1) Manual Page**. 2017?. Disponível em: <<https://github.com/tesseract-ocr/tesseract/blob/master/doc/tesseract.1.asc>>. Acesso em: 20 nov. 2017.

ECMA. **Standard ECMA-262**. Disponível em: <<http://www.ecma-international.org/publications/standards/Ecma-262.htm>>. Acesso em: 13 dez. 2017.

FUNDAÇÃO APACHE. **Lucene features**. Disponível em: <<https://lucene.apache.org/core/features.html>>. Acesso em: 20 nov. 2017.

GORMLEY, Clinton; TONG, Zachary. **Elasticsearch The Definitive Guide: a distributed real-time search and analytics engine**. 1 ed. Sebastopol, Estados Unidos: Oreilly, 2015.

KÖNIG, Dierk; KING, Paul. **Groovy in action**. 2 ed. New York, Estados Unidos: Manning Publications Co., 2015. 5 p.

MATTMANN, Chris A.; ZITTING, Jukka L. **Tika in action**. 20 Baldwin Road PO Box 261 Shelter Island, NY, Estados Unidos: Manning Publications Co., 2012.

MCCANDLESS Michael et al. **Lucene in action**. 2th ed. Estados Unidos. Ed. Manning Publications Co., 2010.

MICROSOFT. **What Is Included in the Index**. Disponível em:
<[https://msdn.microsoft.com/en-us/library/windows/desktop/bb266513\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/bb266513(v=vs.85).aspx)>.
Acesso em: 20 nov. 2017.

MOBX. **Mobx**. 2017. Disponível em: <<https://mobx.js.org/>>. Acesso em: 18 dez. 2017.

MOZILLA DEVELOPER TOOLS. **Handling common javascript problems**. [2017]
Disponível em: <https://developer.mozilla.org/en-us/docs/learn/tools_and_testing/cross_browser_testing/javascript>. Acesso em: 22 nov. 2017.

POWERS M.W. David. **What the F --- measure doesn't measure...** [entre 2014 e 2017]
Disponível em: <<https://arxiv.org/ftp/arxiv/papers/1503/1503.06410.pdf>>. Acesso em: 15 dez. 2017.

SPRING. **Appendix F. Dependency versions**. Disponível em: <<https://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/reference/html/appendix-dependency-versions.html>>. Acesso em: 20 nov. 2017.

THOUGHTWORKS. **React.js**. Disponível em:
<<https://www.thoughtworks.com/radar/languages-and-frameworks/react-js>>. Acesso em: 20 nov. 2017.

WEBPACK. **Webpack**. Disponível em: <<https://webpack.js.org/>>. Acesso em: 13 dez. 2017.

WILLIAMS, Bill. **The Economics of Cloud Computing**: an overview for decision makers. Indianapolis, Estados Unidos: Cisco Press, 2012. p. 2.

YANG, TAO. **Search Evaluation**. [entre 2000 e 2017] Disponível em:
<<http://www.cs.ucsb.edu/~tyang/class/293S17/slides/Topic4Evaluation.pdf>>. Acesso em: 15 dez. 2017.