UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vitor Hugo de Castro Silva

Sistema de recuperação de informação para provas de vestibular

Uberlândia, Brasil

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vitor Hugo de Castro Silva

Sistema de recuperação de informação para provas de vestibular

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informção

Uberlândia, Brasil

2019

Vitor Hugo de Castro Silva

Sistema de recuperação de informação para provas de vestibular

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Rodrigo Sanches Miani
Orientador

Mauricio Cunha Escarpinati
Professor

Paulo Henrique Ribeiro Gabriel
Professor

Uberlândia, Brasil
2019

Dedico esse trabalho de conclusão de curso à minha família, por me dar estrutura para conseguir realizar meus estudos.

Agradecimentos

Agradeço aos meus professores, pelo conhecimento repassado, aos meus amigos pelo suporte e companheirismo nessa jornada e em especial ao meu orientador pela possibilidade da escolha do tema e apoio para a realização deste trabalho. Agradeço em especial ao meu amigo Matheus Manoel pelas incontáveis aulas de cálculo, estatística e outras coisas derivadas da matemática . Agradeço ao professor Dr. rer. nat. Daniel Duarte Abdala pelo conhecimento repassado e pela lição que não há almoço de graça.

Gostaria de agradecer imensamente aos professores que me deram direção ao longo da minha graduação e que esta direção me fez continuar no curso.

"Sou movido por duas filosofias principais, saber mais sobre o mundo hoje do que eu sabia ontem. E diminuir o sofrimento dos outros. Você ficaria surpreso quão longe que você pode chegar com isso."

Neil deGrasse Tysonn

Resumo

A principal motivação deste trabalho surge da necessidade de criação de um sistema para ajudar os possíveis ingressantes de Universidades a realizar seus estudos. O objetivo é desenvolver duas APIs (*Application Programming Interface*) uma para a captura e transformação das provas de vestibular para um formato que possa ser utilizado por um motor de busca e a segunda um motor de busca responsável por receber a consulta do usuário e retornar os documentos mais relevantes. Como estudo de caso, foram utilizadas as provas de vestibular da Universidade Federal de Uberlândia. Durante o trabalho foram utilizadas técnicas como *WebScraping*, *WebCrawler*, e a criação de lógica para transformar textos no formato PDF (*Portable Document Format*) onde estes são cifrados para texto claro. Foi possível gerar um banco de dados contendo as questões objetivas dos vestibulares da UFU. Estas provas foram agrupadas por período da prova, o conteúdo da questão (textos que precediam o enunciado da questão não foram possíveis de se recuperar), suas alternativas e o número da questão na prova. Também foi criado um motor de busca para este banco de dados utilizando o modelo vetorial de recuperação de informação. Testes com as APIs desenvolvidas mostram que o modelo vetorial foi corretamente implementado trazendo resultados relevantes para as consultas submetidas.

Palavras-chave:WebScraping, WebCrawler, Sistemas de informação, Motor de busca, Recuperação de Informação.

Lista de ilustrações

Lista de códigos

Lista de tabelas

Lista de abreviaturas e siglas

ORI	Organização e Recuperação de Informação
SE	Search Engine
TF	Term Frequency
IDF	Frequency–inverse document
API	Application Programming Interface
PDF	Portable Document Format
URL	Uniform Resource Locator
BSON	Binary JSON
DOM	Document Object Model
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
SRI	Sistema de Recuperação de Informação
POO	Programação Orientada a Objetos

Sumário

1 Introdução

A humanidade vem criando arranjos para guardar a informação e acessar a mesma de forma rápida e eficiente. Esse comportamento de categorizar, arranjar e prover fácil acesso tem evoluindo ao longo do tempo (BAEZA-YATES; RIBEIRO-NETO, 2013). As bibliotecas foram as primeiras instituições a adotarem sistemas de recuperação de informação permitindo buscas simples pelo título e nome do autor. Com o surgimento da *World Wide Web* a área de recuperação de informação tomou um grande impulso pois a quantidade de informação gerada foi gigantesca e os meios de se recuperar essa informação tiveram que evoluir. Os buscadores da *Web* surgiram, tentando recuperar documentos que eram relevantes com base na consulta do usuário. Destes buscadores os que ganharam maior destaque foram : Ask, Bing, Yahoo, DuckDuckGo, Baidu e Google.

Na data de realização deste trabalho existem poucas plataformas (Simulado Enem 2018, Simulado Já Enem, Só Exercícios, Aprova Concursos) que disponibilizam para vestibulandos opções de pesquisa utilizando questões de provas e, normalmente essa pesquisa é feita com base nas provas do ENEM. Provas como o vestibular independente não foram encontradas outras formas de serem pesquisadas se não manualmente. Em alguns casos a pesquisa manual não pode ser realizada pois o arquivo PDF (*Portable Document Format*) referente a prova está com problemas de codificação, restando apenas a tentativa de transformação deste PDF para texto acessível, o que pode ser uma tarefa difícil e onerosa.

O objetivo deste trabalho é criar uma API (Application Programming Interface) para que usuários submetam suas consultas e recuperem informações associadas a assuntos presentes nas provas de vestibular da UFU. É importante notar que, até o presente momento, a Universidade Federal de Uberlândia não possui esse tipo de serviço disponível para a comunidade.

Esta API de busca precisa ser alimentada pois não existe um banco de dados com as questões do vestibular da UFU disponível para fazer as consultas, e por isso foi necessário dividir este trabalho em dois módulos: i) API de Busca e Transformação das Provas e ii) API de Consultas.

O primeiro módulo do sistema é responsável por conter o código da API de Busca e Transformação das Provas e irá recuperar e realizar as transformações das provas objetivas que estão em PDF para uma coleção no banco de dados onde cada documento dessa coleção contém o texto das questões, as alternativas, o número da questão e o nome do processo seletivo a qual a questão pertence.

O objetivo do segundo módulo (API de Consultas) é realizar a recuperação dos documentos que sejam mais relevantes para determinada consulta do usuário, utilizando técnicas clássicas da teoria de recuperação de informação como o modelo vetorial e a ponderação TF-

IDF que serão descritas nos capítulos posteriores.

A API de Busca e Transformação das Provas foi implementada, sendo capaz de capturar, processar e armazenar o resultado do processamento das provas no banco de dados, junto com as ponderações (TF, IDF, TF-IDF).

A API de Consultas foi implementada e testada com algumas consultas de exemplos que foram geradas de forma aleatória em um conjunto de provas que foram escolhidas por apresentar menores erros de conversão. Estas consultas quando submetidas na API de Consultas trazem os documentos com maior peso TF-IDF e normas mais baixas no topo do ranking com grau de similaridade mais próximo a um, demonstrando que o modelo vetorial foi corretamente implementado.

O presente trabalho está dividido da seguinte forma. No capítulo 1 é apresentada uma breve descrição do trabalho realizado, no capítulo 2 são apresentadas as técnicas de recuperação de informação utilizadas e os trabalhos correlatos e pesquisas acadêmicas relacionadas a este trabalho. O capítulo 3 descreve as tecnologias e *frameworks* utilizados para programar os dois módulos das APIs. O capítulo 4 apresenta como cada tecnologia foi utilizada e seu propósito para atingir o objetivo de desenvolver as duas APIs (Busca e transformação das provas e Consultas do Usuário), no capítulo 5 é demonstrado a análise de resultados da API de Consultas e por último o capítulo 6 onde são detalhadas as conclusões deste trabalho e trabalhos futuros.

2 Revisão bibliográfica

Neste capítulo são descritas algumas técnicas utilizadas para realizar a implementação dos dois módulos que compõem este trabalho.

2.1 Recuperação de informação

A recuperação de informação é uma área da computação que visa dar acesso fácil às informações de seu interesse (BAEZA-YATES; RIBEIRO-NETO, 2013). Originalmente, esta área cobria apenas a indexação de textos e busca por documentos úteis em uma coleção, atualmente seu escopo está muito mais expandido tratando da modelagem, classificação de textos, arquiteturas de sistemas e visualização de dados. Bibliotecas foram as primeiras instituições a adotarem sistemas de RI, e as primeiras implementações destes sistemas eram feitas primeiramente nas academias e posteriormente passaram a ser desenvolvidos por empresas privadas. Estes sistemas eram limitados a buscas por título da obra e nome do autor.

O surgimento da *World Wide Web* foi responsável pelo crescimento da área de recuperação de informação, pois a Web se tornou um abrigo mundial da cultura e conhecimento humano (BAEZA-YATES; RIBEIRO-NETO, 2013).

Um sistema de recuperação de informação é composto por um processo de coleta, um processo de indexação e um processo de recuperação de informação (BAEZA-YATES; RIBEIRO-NETO, 2013). Neste trabalho, foi criado um *WebCrawler* responsável pelo processo de coleta, um processo responsável por transformar e indexar os dados das provas coletadas, e um motor de busca para realizar as consultas e trazer os documentos mais relevantes para aquela consulta.

Para retornar os documentos mais relevantes precisamos de uma forma para decidir se o documento é relevante para a busca feita pelo usuário. O grande desafio para RI é quantificar o quanto um termo consegue descrever o conteúdo de um documento. Diferentes termos possuem diferentes graus de importância para descrever o conteúdo de um documento.

Para quantificar o quão relevante um documento é para uma determinada consulta, são aplicados os seguintes modelos de ponderação de termos: TF (*Term Frequency*), IDF (*Frequency-inverse Document*) e TF-IDF. A seguir cada um deles será brevemente explicado. Detalhes podem ser encontrados na referência (BAEZA-YATES; RIBEIRO-NETO, 2013).

2.1.1 TF

A ponderação TF, criada por Luhn (BAEZA-YATES; RIBEIRO-NETO, 2013) e se baseia na seguinte suposição:

O valor ou peso de um termo k_i que ocorre em um documento d_j é simplesmente proporcional a frequência do termo $f_{i,j}$. Isto é , quanto mais frequentemente um termo k_i ocorrer no texto do documento d_j maior será sua frequência de termo $TF_{i,j}$.

Termos de alta frequência no documento são importantes para descrever os tópicos chaves de um documento, o que nos leva a seguinte ponderação:

$$tf_{i,j} = \begin{cases} 1 + log f_{i,j} \; se, \; f_{i,j} > 0 \\ 0, \text{ caso contrário} \end{cases} \tag{2.1}$$

Onde f_{ij} é a frequência do termo no documento. Utilizamos base 2 no logaritmo que torna esse valor comparável com a ponderação IDF que também é feita com logaritmo na base 2. Esta ponderação melhorou os resultados da recuperação da informação, se comparada ao modelo booleano onde casamentos parciais não existem.

2.1.2 IDF

A ponderação IDF utiliza a noção de exaustividade (propriedade da descrição dos documentos, provendo a abrangência dos tópicos principais de um documento) e especificidade (o quão bem um termo consegue descrever um documento) dos termos.

De acordo com Baeza-Yates e Ribeiro-Neto (2013), ao adicionar termos presentes no vocabulário em um documento, sua exaustividade aumenta e assim a probabilidade de recuperação também aumenta, isto leva a ideia de um nível ótimo de exaustividade. Quanto mais termos de indexação são atribuídos a um documento, mais exaustiva fica sua descrição. Sua probabilidade de recuperação em resposta a um consulta selecionada aleatoriamente também aumenta. Contudo se muitos termos forem atribuídos a um documento ele irá ser retornado para consultas para quais ele não é relevante. Isso sugere que o número de termos de indexação por documento deve ser otimizado de modo que a probabilidade de relevância de um documento recuperado seja maximizada. Esse número ótimo de termos de indexação define a exaustividade ótima para descrições de tais documentos.

Em termos estatísticos, a exaustividade da descrição de um documento pode ser quantificada como o número de termos de indexação que ele possui. A partir daí, pode-se definir que a especificidade de um termo é uma função do inverso do número de documentos nos quais ele ocorre. Quanto mais longa a descrição do documento mais a sua especificidade dos termos será baixa. Logo se um termo aparece em muitos documentos ele não é muito útil para recuperação de informação.

A ponderação IDF é a relação entre em quantos documentos esse termo apareceu, sob a quantidade de documentos presentes na coleção.

$$IDF_i = log\frac{N}{n_i} \tag{2.2}$$

2.1.3 TF-IDF

A ponderação TF-IDF é o esquema mais popular para ponderação de termos, e combina as ponderações TF e IDF. O referido esquema foi proposto por Salton e Yang (1973).

$$w_{i,j} = \begin{cases} (1 + log f_{i,j}) \times log\frac{N}{n_1} \; se \; f_{i,j} > 0 \\ 0, \text{ caso contrário} \end{cases} \tag{2.3}$$

O pesos TF-IDF são eficazes para coleções genéricas de documentos, isto é para atribuir pesos aos termos de uma coleção de documentos sobre a qual não temos nenhuma informação (BAEZA-YATES; RIBEIRO-NETO, 2013). Essa ponderação é utilizada no modelo vetorial.

2.1.4 Modelo Booleano

Modelo simples que propõe recuperar todos os documentos que possuem os termos, de acordo com as condições que o usuário estabeleça e estas condições podem vir a utilizar operadores booleanos como o *or*,*and* e *not*. Este modelo apresenta a desvantagem de trabalhar com lógica binária, ou o documento é relevante ou não é, e não há nenhuma maneira de ordenar os resultados que satisfaçam a consulta do usuário (SOUZA, 2006).

2.1.5 Modelo Vetorial

O modelo vetorial para recuperação de informação foi proposto para suprir um problema do modelo booleano, onde casamentos parciais não existem. No modelo vetorial casamentos parciais são possíveis, e são realizados pela atribuição de pesos não binários para os documentos e estes pesos são utilizados para gerar o grau de similaridade entre o que o usuário está buscando e os documentos na coleção. Os documentos são ordenados pelo grau de similaridade de forma decrescente, e este modelo provê uma melhor resposta que se ajusta mais a necessidade do usuário do que a resposta que o modelo booleano gera. O grau de similaridade pode ser calculado a partir da seguinte fórmula:

$$\frac{\sum_{i=1}^{N} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} * \sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \tag{2.4}$$

Onde $w_{i,j}$ é o valor da ponderação TF-IDF do termo presente no documento e $w_{i,q}$ é o valor da ponderação TF-IDF do termo presente na consulta.

2.2 Índices

O índice foi criado como uma forma de se categorizar a informação, por meio de rótulos que facilitam a identificação do tópico associado e apontamentos para os documentos que discutem aquela categoria de informação(BAEZA-YATES; RIBEIRO-NETO, 2013). Estes índices eram criados e atualizados de forma manual por profissionais de biblioteconomia ou profissionais de ciências da computação.(BAEZA-YATES; RIBEIRO-NETO, 2013). Isto mudou com o advento da computação que ajudou a desenvolver índices de forma eficiente e automatizada.

De acordo com o Matos et al. (2008), um índice invertido pode ser definido como um banco de dados textual o qual é composto por uma coleção de documentos, que é representado por um conjunto de registros. Realizar comparações entre a busca do usuário com as palavras dentro dos registros pode ser oneroso. O índice invertido vem com o objetivo de acelerar estas buscas e possui dois componentes:

- Uma estrutura chamada vocabulário com todas as palavras presentes na coleção de dados.
- Para cada palavra presente no vocabulário uma lista que aponta para cada documento que possui aquela palavra

A figura 1 exemplifica como o índice invertido deste trabalho é composto, onde para cada termo presente no vocabulário do trabalho, há um apontamento para os *ids* dos documentos que possuem aquele termo.


```
"TERMO" : "FRANCES",
"DOCS" : [
    {
        "_id" : "5da4f7d7c167f614838453b7",
        "TF" : 1.0,
        "QT" : NumberLong(1),
        "TF-IDF" : 6.4093909361377
    },
    {
        "_id" : "5da4f7d6c167f61483845362",
        "TF" : 1.0,
        "QT" : NumberLong(1),
        "TF-IDF" : 6.4093909361377
    }
],
```


Figura 1 – Exemplo de índice invertido para o termo francês.

2.3 Trabalhos Correlatos

Nesta seção são abordados trabalhos objetivos semelhantes a proposta do presente trabalho que envolve permitir que o usuário realize consultas sobre assuntos presentes em provas de processos seletivos de universidades.

O aplicativo Simulado Enem 2018 (GeFilterTI, 2018) oferece simulados personalizados de questões que estavam em provas passadas do Enem e estatísticas do usuário, para o mesmo mensurar seu progresso e focar em áreas que não esteja evoluindo. O aplicativo não permite buscas por termos, somente a opção de montar um simulado personalizado com as questões listadas. Este aplicativo se encontra disponível na *Playstore* no link.

O aplicativo Simulado Já Enem (Sand Robot, 2018) oferece a criação de simulados com questões que já caíram nas edições passadas do Enem, você pode criar seu próprio simulado e acompanhar suas estatísticas de desempenho. Este aplicativo se encontra disponível na *Playstore* no link.

A plataforma Só Exercícios (2014) contém vestibulares de várias universidades e as provas do ENEM, onde você pode buscar por assuntos pré definidos na caixa de seleção, pode ser filtrado por questões que você já resolveu, acertou ou errou (recurso pago) e filtrar por ano de aplicação da prova, porém este recurso é pago. As questões são retornadas completas, e o aplicativo realiza a indexação de qualquer tipo de questão seja ela objetiva ou discursiva. Todos os recursos criados neste trabalho serão gratuitos diferentemente do Só Exercícios que contém recursos pagos. Este sistema se encontra disponível no link.

A plataforma Aprova Concursos (2019) é destinada a busca de questões de concursos públicos, podendo realizar buscas por concursos, assuntos ou disciplinas e acrescentar filtros como nome da prova, instituição e ano.

O retorno da consulta traz toda questão de acordo com os filtros pré estabelecidos, trazendo toda a questão relacionada ao que foi pesquisado. Outra funcionalidade do sistema é a opção de realizar simulados online. O sistema tem bastante abrangência no quesito de provas contendo questões de vários concursos e vestibulares aplicados e se diferenciando deste trabalho que foca em apenas em vestibular de forma mais específica o vestibular da UFU. Este sistema se encontra disponível no link.

O estudo feito por Leusin (2007) foi realizado com o objetivo de melhorar a precisão e revocação de pesquisas a documentos relacionados à arquitetura, engenharia e construção. O trabalho descreve a criação do sistema de captura, tratativas de *stop words*, normalização do vocabulário e seleção dos termos relevantes e classificação. Para a consulta realizada sob o sistema é realizado o processo de remoção de *stop words*, normalização do vocabulário e finalmente o resultado para a pesquisa. O sistema trouxe uma melhora de 35% na precisão média, melhorando a recuperação de informação (LEUSIN, 2007).

O trabalho de Souza (2006) visa dar uma visão geral sobre o que está sendo empregado em sistemas de recuperação de informação. É descrito que SRIs (Sistema de Recuperação de Informação) são interfaces entre uma coleção de recursos de informação e uma população de usuários e também mostra as atividades que são desenvolvidas por estes sistemas.

O trabalho cita algumas estratégias que vêm sendo empregadas para fazer a indexação de documentos, citando os modelos considerados clássicos de recuperação de informação onde cada documento é representado por um conjunto de palavras que o descreve. Dentre esses modelos clássicos se explica o modelo booleano e suas implementações alternativas como a lógica difusa e o modelo booleano estendido, o modelo vetorial e suas extensões como o modelo vetorial generalizado, indexação semântica latente e redes neurais e também cita o modelo probabilístico e suas extensões como as redes de inferência e redes de crença. É explicado que atualmente não tenta-se imitar as heurísticas intelectuais humanas para aplicar nas consultas e sim utilizar estratégias diferenciadas que demandam muito poder computacional, como a criação de interfaces mais intuitivas para os usuários onde se pode marcar o que é relevante ou não e aproveitar esse retorno para aprimorar os motores de busca, fazer utilização do perfil do usuário para fazer consultas e que o SRI aprenda com este perfil para aprimorar as buscas. O trabalho apresenta um diagrama de como podemos melhorar os SRIs e técnicas que podem ser empregadas para alcançar essas melhorias.

O trabalho de Magalhães e Souza (2019) foi realizado para trazer uma resposta de como as ontologias estão sendo utilizadas nos SRIs para trazer resultados melhores para as buscas. O trabalho consistiu de uma busca pela literatura existente presentes no Google Scholar,biblioteca eletrônica Scielo,Portal Capes e em Bancos de Teses e Dissertações, no período de outubro de 2017 a agosto de 2018 por termos como "recuperação de informação". Destaca-se a utilização da Lisa dentro do Portal Capes pelo fato desta base de dados ser voltada a profissionais de Biblioteconomia e Ciência da Informação.Foi realizada uma revisão da literatura encontrada e apresentada por ordem cronológica os trabalhos recuperados.

Este estudo demonstrou que ontologias melhoram os resultados das buscas, e que deve-se estabelecer para um vocabulário específico um domínio, para minimizar ambiguidades semânticas. Parcerias entre as áreas de ciência de informação e ciência da computação são necessárias para melhorar a precisão, relevância e eficácia dos SRis, buscando a construção de ontologias, aprofundando sobre sinônimos de uma determinada área do conhecimento.

3 Tecnologias Utilizadas

Neste capítulo são apresentadas as tecnologias que foram empregadas para desenvolver este trabalho, assim como a linguagem de programação das APIs e uma breve visão de como cada tecnologia funciona.

Para a linguagem de programação foi escolhido JAVA, pela facilidade de modelagem que a POO (Programação orientada a objetos) traz e pelo fato de ser a linguagem que o autor possui maior fluência e também os diversos *frameworks* de código livre escritos para essa linguagem. Destes *frameworks* foram utilizados o Gson utilizado para transformar os arquivos de configuração que foram persistidos em disco no formato *gson string*. Para realizar a parte de captura das provas foram utilizadas técnicas como *webscraping* e *webcrawler* e o jsoup para *parsar* o HTML (*Hypertext Markup Language*) e buscar pelas *tags* que foram utilizadas para capturar os *links* das provas. A transformação dos PDFs para texto foi realizada através do PDFBox, o Apache Commons para retirar acentos das palavras e o MongoDB para salvar o resultado do processamento das provas.

3.1 Java

Desenvolvida pela Sun Microsystems com o objetivo de ser uma linguagem de software para produtos eletrônicos e para isso tinha o desafio de ter código compacto. A linguagem não obteve sucesso ao entrar nesse mercado, no entanto os pesquisadores da Sun viram um potencial de adicionar conteúdo dinâmico, como interatividade e animações em páginas web, e mudaram o código fonte da linguagem para que ela pudesse funcionar em microcomputadores conectados a Internet. Desta aplicação surgiram programas como *Applets* e um navegador que fazia a interface entre o programa em Java e o sistema operacional do computador (INDRUSIAK, 1996).

O java foi escolhido pela sua portabilidade e pela aderência à programação funcional que facilita o trabalho com grandes listas de dados, aplicando operações de *map* e *reduce*. Toda a lógica das duas APIs (Busca e Transformação das Provas e Busca de Termos pelo usuário) foram desenvolvidas nesta linguagem e o controle de dependência e processo de construção do projeto foi realizado pelo maven.

Antes do Java Se 8 a linguagem trabalhava com os paradigmas de programação procedural, programação orientada a objetos e programação genérica. No entanto no Java Se 8 foi introduzido o paradigma de programação funcional (PAUL; HARVEY, 2014).

3.2 PDFBox®

Desenvolvida pela Fundação Apache, o PDFBox é uma biblioteca de código fonte aberto que permite a criação, edição e a extração de conteúdo dos arquivos PDF, assim como a possibilidade de combinar dois PDFs em um só, ou transformar um PDF em vários arquivos (Apache Software Foundation, 2019c).

3.3 Gson

Biblioteca desenvolvida pela Google para converter objetos Java para o formato de JSON. Também pode ser utilizada para converter uma *string json* para um objeto equivalente em Java. Suporta objetos complexos com altos níveis de hierarquia e tipos genéricos. Está sob a licença Apache 2.0 (Google, 2019).

3.4 MongoDB

O MongoDB é um banco que não usa o conceito de tabelas, esquemas, SQL ou linhas. Não há transações em conformidade com ACID, *joins* ou chaves estrangeiras. O MongoDB utiliza um formato de dados chamado BSON (*Binary Json*) para guardar os dados. Este formato é de código livre e foi desenvolvido pela equipe que criou o MongoDB (HOWS; MEMBREY; PLUGGE, 2015).

O MongoDB é um banco que está enquadrado na categoria de banco de dados NOSQL. Há certa discordância no que significa NOSQL, mais foi adotado o termo “Not Only SQL”, mas no geral cada um deles apresenta a maioria das seguintes características: não-relacional, distribuído, de código aberto e escalável horizontalmente, ausência de esquema ou esquema flexível, suporte à replicação nativa e acesso via APIs simples (DIANA; GEROSA, 2010).

3.5 *WebScraping*

Processo que visa extrair informação da Internet, utilizando-se de metadados e marcações específicas nas páginas para retirar os dados e informações que estão sendo procuradas (Krunal A, 2014).

3.6 *WebCrawler*

WebCrawler é um serviço da web e foi criado para auxiliar usuários na navegação pela web, pois muitos usuários não sabem onde encontrar a informação que eles estão procurando ou a URL (*Uniform Resource Locator*) que contenha a melhor informação dada uma certa busca. Este serviço auxilia o usuário automatizando a passagem de links e construindo um índice de

toda a web que possa ser consultado para atender certas pesquisas dos usuários. Tecnicamente *WebCrawler* é um nó na web que contém apontamentos (links) para sites e que consegue diminuir o caminho do usuário ao o que ele está buscando. (PINKERTON, 2000). A Google é um exemplo de empresa que faz uso intensivo de *crawlers*, em julho de 2019 a empresa tornou open source seu *crawler* que lê um arquivo chamado robots.txt, este arquivo que dita quais páginas de sites devem ser consideradas pelos mecanismos de buscas. Foi feito em C++ e possui códigos que foram produzidos na década de 90. Outro exemplo de *crawler* bem conhecido é o GNU Wget que é escrito em C e pode ser operado pela linha de comando. Utilizado para fazer espelho de um site inteiro ou simplesmente baixar informação de WebSites ou Servidores FTP (SINGH; VARNICA, 2014).

3.7 Jsoup

É uma biblioteca feita para trabalhar com HTML e provê uma API para manipular e extrair informação utilizando o melhor da DOM (*Document Object Model*),CSS (*Cascading Style Sheets*) e JQUERY. Esta biblioteca implementa a especificação WHATWG HTML5 e realiza a conversão do HTML para a mesma DOM que navegadores modernos. Biblioteca está sob licença MIT (Jonathan Hedley, 2019).

3.8 Apache Commons

As bibliotecas padrão do Java falham em trazer métodos suficientes para manipular as suas classes principais. Essa biblioteca nasce com o objetivo de fornecer extra métodos de manipulação. Provendo métodos de manipulação de *strings*, métodos de números, concorrência, criação e serialização de objetos. Licenciada sob Apache 2.0 (Apache Software Foundation, 2019a).

3.9 Maven

Maven é uma ferramenta para gerenciar e construir projetos JAVA, com o objetivo de facilitar o desenvolvimento de software. Ele foi desenvolvido em cima do conceito de POM (project object model).

Algumas das características do Maven são:

- Facilita o processo de construção.
- Sistema de construção único.
- Migração Fácil para novas funcionalidades.

Sua licença é a Apache 2.0 (Apache Software Foundation, 2019b).

4 Desenvolvimento

Neste capítulo são descritos os passos adotados para criar as duas APIs que compõem este trabalho: API de busca e captura das provas, API de consultas. Também é descrito como cada tecnologia, técnica e conceitos do referencial teórico e tecnologias foram empregados.

4.1 Visão Geral

Nesta seção é exibido e descrito o diagrama da arquitetura das duas APIs (API de busca e captura das provas e API de consultas).

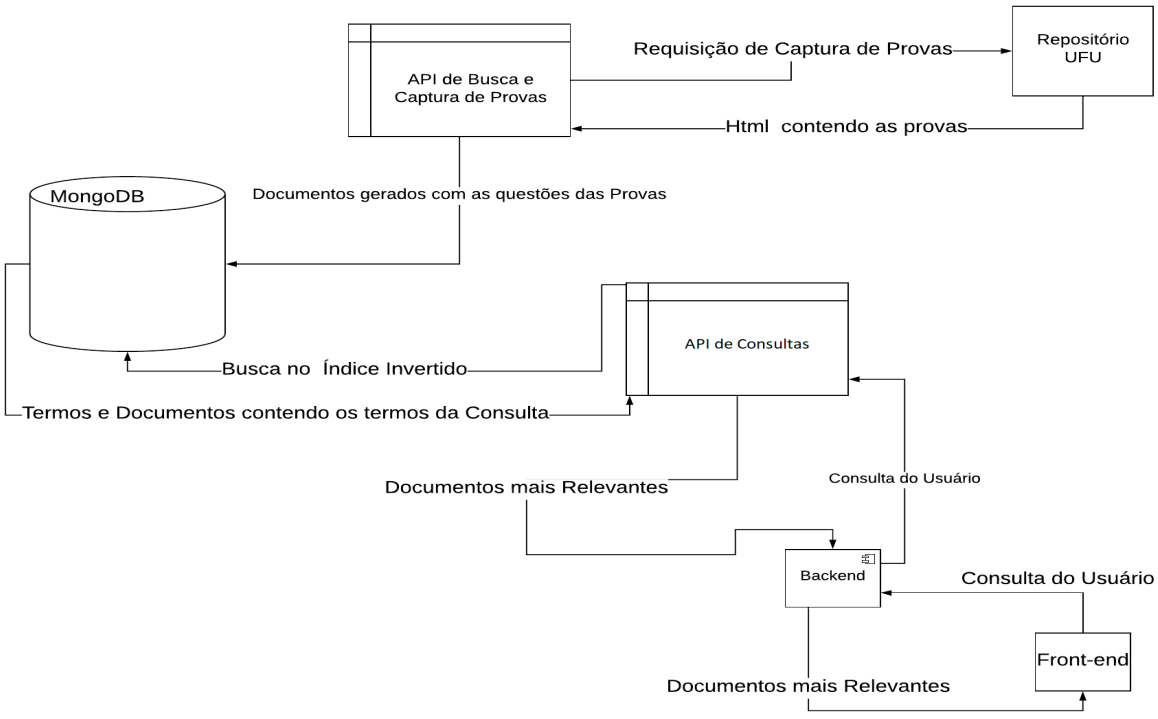


Figura 2 – Arquitetura das APIs.

A API de busca e captura das provas é feita para se conectar na URL fornecida, nesse caso foi o *link* do repositório da UFU e busca o HTML da página onde estão as provas, e realiza os trabalhos de identificar as provas por *tags* HTMLs e realiza a captura e o processamento dessas provas e por fim salvar esses dados gerados do processamento no MongoDB.

A API de consultas foi desenvolvida para ser importada em um *backend* java e ser chamada recebendo uma *string* contendo os termos que estão sendo buscados pelo usuário e ao receber esses termos, realiza uma consulta no MongoDB mais especificamente no índice

invertido criado neste trabalho e descrito posteriormente para achar os documentos que contém aqueles termos, pega-se o retorno desta consulta e aplica-se o modelo vetorial para retornar para o *backend* que fez a chamada da API os documentos ordenados por ordem de relevância.

4.2 API de busca e captura das provas

A coleta de dados do sistema ocorreu através de uma técnica chamada *WebCrawler* onde cada URL (caminho para cada prova de vestibular que a UFU já aplicou) de entrada foi informada para o *crawler* poder se conectar, foi realizada ampla utilização de operações lambdas, combinadas com as *streams* do java para realizar transformações dos objetos e filtragem de listas.

Para realizar este trabalho foi utilizado o jsoup para abrir uma conexão com cada url passada. Tal URL apontava para um ano de um processo seletivo específico da UFU, onde se buscava pela *tag* HTML de *link* e também é verificado se a mesma *tag* "a" possui a palavra "Objetiva" pois para esse trabalho foi-se considerado apenas provas de questões objetivas.

Código 4.1 – Exemplo de tag "a" que seria capturada pelo *crawler* pois possui a palavra "Objetiva".

```
<div class="collection">
    <div class="collection-item">
        <a href="/download/b139e104214a08ae3f2ebcce149cdf6e">
        Prova Historia Fase1 de QUESTOES OBJETIVAS</a>
        <div class="date-class">Criado em 15-12-2015 15:02</div>
    </div>
</div>
```

Feito esse filtro, para cada *tag* HTML que fosse encontrada de acordo com a condição pré estabelecida, é aberta uma conexão com aquele link e tenta-se capturar o PDF que se encontra naquela URL. O Código 4.2 é um trecho do *Crawler* que realiza as operações descritas anteriormente.

Código 4.2 – Conexão com a url fornecida via JSOUP para capturar as provas.

```
Connection c = Jsoup.connect(baseURl + processo);
Document arquivos = c.get();
String tituloArquivo = retornaTituloProcesso(arquivos);
```

```
String nomePasta = tituloArquivo.substring(29, 35);
Elements dados = retornaDadosDocument(arquivos);
dados.forEach(file -> {
    System.out.println(PEGANDO_PROVAS_EM + tituloArquivo);
    String nomeArquivo =
    tituloArquivo + " - " + file.text() + PDF;
    String url = baseURl + file.getAllElements().attr(HREF);
    if (podeBaixarArquivo(file.text().toLowerCase())) {
        salvaProva(url, nomeArquivo, nomePasta);
        Thread.sleep(1000);
    }
}
```

Todas as provas que o *webcrawler* consegue recuperar são salvas em diretórios com o referente ano daquele processo seletivo. Todo esse processo de percorrer as URLs e verificar os links foi facilitado pelo Jsoup, pois ele tem métodos que facilitam acesso via *tags* HTML aos documentos. No Código 4.3 os métodos que fazem a utilização do Jsoup para capturar certos elementos HTML.

Código 4.3 – Metódos para capturar as tags“a” e “h1”.

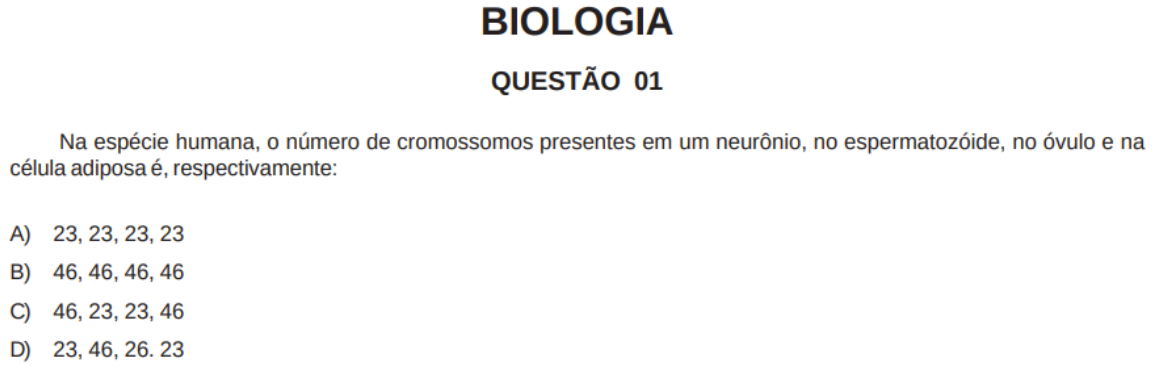
```
private Elements retornaDadosDocument(Document arquivos) {
        return arquivos.body().
        getElementById(CONTENT).getElementsByTag(A);
}


 private String retornaTituloProcesso(Document arquivos) {
    return arquivos
        .getElementsByTag(H_1)
        .get(arquivos.getElementsByTag(H_1).size() - 1)
        .text();
}
```

4.2.1 *Parser* das Provas

Após o processo de capturar as provas é iniciado um processo onde varre-se um diretório raiz que foi configurado para armazenar as provas por ano de realização da mesma, este processo é chamado de *Parser*.

Para cada processo seletivo que foi recuperado pelo *crawler*, é realizada a transformação do texto que está no arquivo em PDF, para o formato que é utilizado na API de Consultas e o resultado deste *Parser* é armazenado no banco de dados. Para ser realizada a extração do texto do PDF, foi utilizada de uma biblioteca chamada *PDFBox*, que provê acesso de forma fácil a estrutura e conteúdo do PDF, inclusive ao texto. Ao finalizar a captura do texto contido no PDF, é iniciado um processo que vai percorrer linha a linha do texto para conseguir identificar marcadores como início e fim da questão, início e fim das alternativas e o conteúdo da questão para posteriormente serem armazenados no formato que foi definido para ser persistido no *MongoDB*.

Sobre o formato que é armazenado no banco de dados, é persistido o nome do processo seletivo, o número da questão, o conteúdo da questão e suas alternativas. Foi escolhido tratar como conteúdo da questão tudo que aparece depois da marcação de texto 'Questão %s' onde %s seria o número da questão e finalizando a parte do texto que entra como conteúdo somente quando se acha uma *string* com "A) ". A figura 3 exemplifica uma questão presente no vestibular:



BIOLOGIA

QUESTÃO 01

Na espécie humana, o número de cromossomos presentes em um neurônio, no espermatozóide, no óvulo e na célula adiposa é, respectivamente:

A) 23, 23, 23, 23
B) 46, 46, 46, 46
C) 46, 23, 23, 46
D) 23, 46, 26. 23

Figura 3 – Formato de questão objetiva presente no vestibular da UFU que é processada pelo *Parser*

Todas as alternativas de respostas para as questões se iniciam com as seguintes *strings*:

- "A) "
- "B) "
- "C) "
- "D) "
- "E) "

Para cada questão é gerado um documento no formato BSON que será persistido no banco de dados, este formato é utilizado pela API do Mongodb que faz uso deste formato para fazer operações no banco de dados, dentre elas operações de inserção e consulta.

4.2.2 Vocabulário

O vocabulário é constituído das palavras presentes nas provas, palavras estas que precisam ter no mínimo três letras. Quando o processo de geração do vocabulário se inicia, busca-se todos os documentos da coleção de provas e estes são utilizados em processos que executam em paralelo. Utiliza-se todas as palavras presentes tanto no enunciado e nas alternativas da questão para construir o vocabulário.

No momento de inserção de um termo no vocabulário é acumulada a quantidade de vezes que essa palavra aparece para aquele documento específico que está sendo processado. Esta informação vai ser útil para gerar um vetor com os valores da ponderação TF-IDF para cada termo.

A estrutura de dados que armazena o vocabulário é um *ConcurretHashMap*, uma implementação de *HashMap* que evita problemas de concorrência, pois temos múltiplas *threads* sendo tratadas ao mesmo tempo e escrevendo no *ConcurretHashMap* de forma paralela.

Essa estrutura de dados foi pensada para a API como um índice invertido. Para cada termo presente no vocabulário é mapeado outro *ConcurretHashMap* que contém todos os documentos onde aquele termo aparece e, por sua vez, esse *ConcurretHashMap* armazena o ID do documento no banco de dados e um objeto onde é armazenado as ponderações pertinentes aquele documento.

Finalizando o processo de criação do vocabulário é iniciada a geração da ponderação TF-IDF.

4.2.3 Ponderações dos Termos

Para cada termo presente no vocabulário, é aberto um processo onde será gerado a ponderação TF e a ponderação IDF. Este processo recebe um termo, a coleção de documentos onde esse termo aparece.

A ponderação IDF é calculada assim que o processo se inicia, dado que os seguintes dados já estão disponíveis:

- Quantidade de documentos onde o termo aparece;
- Quantidade de documentos da coleção de provas.

Como anteriormente para cada documento já foi calculado quantas vezes o termo aparece, é calculada a ponderação TF do documento. Também é calculado o TF-IDF multiplicando o valor do TF pelo valor do IDF para cada documento onde aquele termo aparece.

No final do processamento de cada termo é realizada a inserção do objeto no banco de dados em uma coleção que funciona como um índice invertido. Os métodos presentes no Código 4.4 são utilizados para fazer as transformações necessárias para salvar no banco de dados.

Código 4.4 – *Parser* para transformar objetos no formato do JAVA para o *MongoDB*.

```
static Document paraMongo(FragmentoIndex fragmentoIndex) {
    Document beanToMongo = new Document();
    beanToMongo.put("DOCS", fragmentoIndex.getDocs()
        .stream().map(DocumentoIndexParser::paraMongo)
        .collect(Collectors.toList()));

    beanToMongo.put("TERMO", fragmentoIndex.getTermo());
    beanToMongo.put("IDF", fragmentoIndex.getIdf());
    return beanToMongo;
}

static Document paraMongo(DocumentoIndex documento) {
        return new Document(ID, documento.get_id())
                .append(TF, documento.getTf())
                .append(QT, documento.getQt())
                .append(TF_IDF, documento.getTf_idf());
}
```

Em uma coleção com um grande volume de documentos, o tamanho destes podem variar muito e com isso surge um problema para a recuperação de informação. Os documentos maiores tendem a ser recuperados mais frequentemente pelo fato que esses documentos contêm um número maior de termos. Para compensar esse efeito, foi utilizada uma técnica chamada normalização pelo tamanho do documento. Este processo para ser realizado de forma adequada é preciso definir como é computado o tamanho dos documentos. Para este trabalho definiu-se como tamanho do documento, o número de termos que este documento possui que estão no vocabulário. E segue os seguintes passos para o cálculo da norma:

1. Consulta-se a base de dados que contém o índice invertido referenciando termo aos documentos.

2. Agrupa os documentos pelo *id* do documento na coleção de questões.
3. Eleva-se o valor do TF-IDF ao quadrado de cada documento para cada termo e soma-se o resultado.
4. Retira-se a raiz quadrada do resultado do processo anterior para cada documento.
5. Salva-se a norma no banco de questões para cada documento.

4.3 API de Consultas

A API de consultas foi desenvolvida para receber uma *string* onde os termos são quebrados por espaço e o resultado armazenado em um vetor onde cada posição do vetor é um termo que estava contido na *string* recebida. São retirados todos os acentos pois nem sempre os termos da consulta vão estar com acentos tornando difícil a identificação dos termos e também são retiradas palavras com menos de 3 caracteres pois estas palavras tendem a ter alta frequência na coleção de documentos e não ajudam a descrever ou a diferenciar um documento do outro (MIYATA; KANO; DIGIAMPIETRI, 2013).

Feito isso, é iniciado o processo de geração do vocabulário utilizando os termos que foram passados como consulta. O Código 4.5 é responsável por gerar vocabulário da consulta.

Código 4.5 – Código utilizado para gerar o vocabulário.

```
String id = paraProcessar.get("_id").toString();
StripDocumento.retiraCamposProcessamento(paraProcessar);
paraProcessar.values().forEach(it -> {
 String[] words = StripDocumento.retornaPalavras(it);
 for (String palavra : words) {
  palavra = StringUtils.stripAccents(palavra);
  palavra = palavra.toUpperCase().replaceAll("[^A-Z]", "");
  System.out.println(palavra);
  if (palavra.trim().length() > 2) {
     if (enunciados.contains(palavra) ||
                 Objects.equals(palavra, ""))
        continue;
     VocabularioInstance.insereItem(palavra, id);
  }
 }
});
```

Após o processo de geração do vocabulário, a ponderação TF para cada um dos termos é calculada. O próximo passo envolve recuperar a ponderação IDF dos termos que já foi calculada no momento da criação do índice invertido. O valor do TF-IDF envolve, simplesmente, a multiplicação do valores TF e IDF. O processo é repetido para cada termo presente na consulta. O Código 4.6 realiza o processo descrito.

Código 4.6 – Código que realiza o cálculo do TF, recupera o IDF da coleção e calcula o TF-IDF.

```
private double retornaIDFColecao(String termo) {
 Document query = new Document("TERMO", termo);
 Document resultado = MongoProducer.retornaInstanciaIndex()
 .find(query).first();
 return (resultado == null) ? 0.0 : resultado.getDouble("IDF");
}

public DocumentoIndex call() {
 termoQuery.
    setTf(1 + (Math.log10(termoQuery.getQt()) / Math.log10(2)));
 termoQuery.
    setTf_idf(termoQuery.getTf() * (retornaIDFColecao(termo)));
 termoQuery.
    set_id(termo);
 return termoQuery;
}
```

Após ponderar os termos da consulta, é iniciada uma busca no índice invertido para os documentos que contém aqueles termos.

Utilizou-se um *hashmap* para guardar o valor da multiplicação do TF-IDF do termo da consulta com o valor do TF-IDF do termo no documento. A chave deste *hashmap* é o *id* do documento da coleção de provas.

Após iterar os termos da consulta, encontrar os documentos que contém aqueles termos e armazenar o *id* do documento da coleção de provas e o valor dos TFs-IDfs multiplicados, itera-se esse *hashmap* e para cada documento presente é capturado seu valor e dividi-se pela norma do documento presente na coleção de provas. Este processo é realizado no Código 4.7.

Código 4.7 – Somatório da norma para cada documento que possui os termos da consulta.

```
docsQuery.forEach(it -> {
 normaQuery += Math.pow(it.getTf_idf(), 2);
```

```
 getByTermoIndex(it.get_id()).getDocs()
 .forEach(doc -> {
   if (similaridade.containsKey(doc.get_id())) {
     Double simiLaridadeCalculada =
        similaridade.get(doc.get_id());
     similaridade.put(doc.get_id(),
           simiLaridadeCalculada +
                (doc.getTf_idf() * it.getTf_idf()));
   } else {
     similaridade.
           put(doc.get_id(), doc.getTf_idf() * it.getTf_idf());
   }});
});

normaQuery = Math.sqrt(normaQuery);
```

Feito isso, é ordenado de forma decrescente o resultado e retornado os *ids* e seus graus de similaridade. Isso conclui o processo do modelo vetorial em que serão exibidos os documentos mais relevantes de uma coleção dado uma consulta usando o grau de similaridade. O Código 4.8 realiza este processamento.

Código 4.8 – Cálculo do grau de similaridade do documento com a consulta.

```
similaridade.forEach((key, value) -> {
            Double normaDoc = getByIDDoc(key);
            value = value / (normaDoc * normaQuery);
            similaridade.put(key, value);
});

return similaridade.entrySet()
.stream().sorted((o1, o2) -> {
  if (o1.getValue() >= o2.getValue()) {
   return -1;
  } else
   return 0;
}).collect(Collectors.toMap(
                        Map.Entry::getKey,
                        Map.Entry::getValue,
                        (e1, e2) -> e1, LinkedHashMap::new)
);
```

5 Resultados

Neste capítulo é apresentada uma análise dos resultados obtidos pela API de Consultas, descrevendo a amostra de dados, o processo de escolha dos termos, a escolha das questões e, por fim, a validação se o modelo vetorial foi devidamente implementado.

5.1 Amostra de Dados

Foram escolhidas sete provas para os testes do sistema proposto:

- Provas de 1999-1:
  - Prova de Biologia Fase 1
  - Prova de Filosofia Fase 1
- Provas de 2007-1:
  - Prova de Biologia Fase 1
  - Prova de Filosofia Fase 1
  - Prova de Geografia Fase 1
- Provas de 2015-2:
  - Biologia, Física, Geografia, Língua Portuguesa, Literatura e Sociologia
  - Língua Estrangeira (opção por Espanhol, Francês ou Inglês)
  - Filosofia, História, Matemática e Química.
- Provas de 2016-2:
  - Biologia, Física, Geografia, Língua Portuguesa, Literatura e Sociologia
  - Língua Estrangeira (opção por Espanhol, Francês ou Inglês)
  - Filosofia, História, Matemática e Química.

Estas provas foram escolhidas pois ao serem transformadas para texto claro, as mesmas apresentam um grau menor de erros ao serem transformadas para a estrutura do sistema. Estes erros estão relacionados a codificação das *strings* onde em alguns casos não se consegue identificar o que está escrito.

5.2 Seleção dos Termos

Foram selecionados dois termos de forma aleatória no índice invertido contido no banco de dados. Foi conferido em quais documentos possuíam o termo A, quais documentos possuíam o termo B, e em quais documentos possuíam os dois termos. Cada par de termo selecionado se transformou em uma consulta realizada nas provas escolhidas como amostras de dados. Foi consultado quais questões das provas continham os termos e estas foram ordenadas pelo grau de relevância.

As próximas seções apresentam os resultados das duas consultas usadas para validar o sistema.

5.2.1 Consulta ANTIGO ESQUEMA

A Tabela 1 exibe os documentos que contém os termos da consulta em ordem de relevância.

ID Documento	TF-IDF "Antigo"	TF-IDF "Esquema"	Norma
5da4f7d6c167f61483845316	-	15.05	53.90
5da4f7d6c167f6148384532c	6.41	-	61.76
5da4f7d6c167f6148384535a	6.41	-	67.07
5da4f7d6c167f61483845322	-	5.82	68.47
5da4f7d6c167f61483845320	-	5.82	95.75

Tabela 1 – Resultado Consulta Manual Antigo Esquema

5.2.2 Consulta "NOVOS RESIDENTES"

A tabela 2 exibe os documentos que contém os termos da consulta em ordem de relevância.

ID Documento	TF-IDF "Novos"	TF-IDF "Residentes"	Norma
5da4f7d7c167f614838453b2	5.41	7.41	77.15
5da4f7d6c167f61483845385	10.82	-	70.70
5da4f7d6c167f61483845363	5.41	-	71.48
5da4f7d6c167f61483845361	5.41	-	88.82

Tabela 2 – Resultado Consulta Manual Novos Residentes.

5.3 Análise dos resultados da API

A seguir é descrito o resultado das consultas desenvolvidas com os termos selecionados, quando estas consultas são submetidas na API de busca.

5.3.1 Resultados da consulta ANTIGO ESQUEMA

Quando aplicado esta consulta sobre a API desenvolvida, é obtido o seguinte resultado:

```
Ponderação QUERY ::
------------------------------------------------------------
_id : ANTIGO | QT : 1 | TF  : 1,000000 | TF-IDF : 6,409391
_id : ESQUEMA | QT : 1 | TF  : 1,000000 | TF-IDF : 5,824428
------------------------------------------------------------
RANKING
------------------------------------------------------------
Ranking :: 1 , ID  : 5da4f7d6c167f61483845316  , SIMILARIDADE :  0,19
Ranking :: 2 , ID  : 5da4f7d6c167f6148384532c  , SIMILARIDADE :  0,08
Ranking :: 3 , ID  : 5da4f7d6c167f6148384535a  , SIMILARIDADE :  0,07
Ranking :: 4 , ID  : 5da4f7d6c167f61483845322  , SIMILARIDADE :  0,06
Ranking :: 5 , ID  : 5da4f7d6c167f61483845320  , SIMILARIDADE :  0,04
------------------------------------------------------------
```

Figura 4 – Resultado da consulta ANTIGO ESQUEMA

5.3.2 Resultados da consulta “NOVOS RESIDENTES”

```
Ponderação QUERY ::
------------------------------------------------------------
_id : RESIDENTES | QT : 1 | TF  : 1,000000 | TF-IDF : 7,409391
_id : NOVOS | QT : 1 | TF  : 1,000000 | TF-IDF : 5,409391
------------------------------------------------------------
RANKING
------------------------------------------------------------
Ranking :: 1 , ID  : 5da4f7d7c167f614838453b2  , SIMILARIDADE :  0,12
Ranking :: 2 , ID  : 5da4f7d6c167f61483845385  , SIMILARIDADE :  0,09
Ranking :: 3 , ID  : 5da4f7d6c167f61483845363  , SIMILARIDADE :  0,04
Ranking :: 4 , ID  : 5da4f7d6c167f61483845361  , SIMILARIDADE :  0,04
------------------------------------------------------------
```

Figura 5 – Norma do documento 4.

O ranking da API de Consultas, manteve os mesmos documentos e a ordenação de relevância para os documentos que a consulta que foi feita manualmente nas provas escolhidas como amostras de dados. O modelo vetorial utilizado para criar o motor de busca foi correta-

mente implementado pois os documentos que haviam maior valor de TF-IDF e normas menores estão no top do ranking com um grau de similaridade mais próximo a um.

6 Conclusão

O objetivo deste trabalho foi criar duas APIs para capturar, transformar e disponibilizar um meio de buscar por termos as provas de vestibular da Universidade Federal de Uberlândia foi realizado com o auxílio de diferentes *frameworks* de código livre.

O maior desafio deste trabalho foi encontrar um padrão nas provas de vestibular para poder separar as questões e suas respectivas alternativas e popular o banco de dados para permitir as buscas dos usuários. Para encontrar um padrão nas provas objetivas utilizadas neste trabalho, foram desenvolvidos vários *parsers* para processar os diferentes tipos de prova. Os *parsers* que geraram os melhores resultados foram combinados para gerar um *parser* único para o sistema. Essa versão permite realizar o processamento das provas com maior eficiência.

Diversos trabalhos futuros poderiam surgir deste estudo. Um exemplo seria a implementação de melhorias na API de busca por termos para a utilização de tesauros. Isso permitiria a inserção de palavras similares na consulta em uma tentativa de ampliar e melhorar a recuperação de informação. Outro trabalho futuro importante seria desenvolver um sistema para atender as requisições dos usuários e exibir, de maneira amigável, as respostas para as consultas. Outro ponto que vale ser citado como trabalho futuro seria solicitar a UFU que implemente melhorias nos metadados dos PDFs. Uma possível melhoria seria adicionar marcadores indicando onde começa e termina uma questão.

Referências

Apache Software Foundation. *Apache Commons*. 2019. Disponível em: <https://commons.apache.org/proper/commons-lang/>. Acesso em: 18-11-2019. Citado na página 23.

Apache Software Foundation. *Maven*. 2019. Disponível em: <https://maven.apache.org/>. Acesso em: 13-11-2019. Citado na página 24.

Apache Software Foundation. *PDFBox*. 2019. Disponível em: <https://pdfbox.apache.org/>. Acesso em: 19-11-2019. Citado na página 22.

Aprova Concursos. *Quetões de Concurso*. 2019. Disponível em: <https://www.aprovaconcursos.com.br/questoes-de-concurso/questoes>. Acesso em: 03-01-2020. Citado na página 19.

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Recuperação de Informação Conceitos e Tecnologia das Máquinas de Busca*. [S.l.: s.n.], 2013. 590 p. Citado 5 vezes nas páginas 13, 15, 16, 17 e 18.

DIANA, M. D.; GEROSA, M. A. Nosql na Web 2.0: Um estudo comparativo de bancos Não-Relacionais para Armazenamento de Dados na Web 2.0. *IX Workshop de Teses e Dissertações em Banco de Dados - WTDBD*, p. 8, 2010. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wtdbd/2010/sbbd_wtd_12.pdf>. Citado na página 22.

GeFilterTI. *Simulado Enem 2018*. 2018. Disponível em: <https://play.google.com/store/apps/details?id=nf.co.gefilter.simuladoenem2015free>. Acesso em: 05-01-2020. Citado na página 19.

Google. *Apache Commons*. 2019. Disponível em: <https://github.com/google/gson>. Acesso em: 25-11-2019. Citado na página 22.

HOWS, D.; MEMBREY, P.; PLUGGE, E. *Introdução ao MongoDB*. [S.l.]: Novatec Editora, 2015. Citado na página 22.

INDRUSIAK, L. S. Linguagem java. *Grupo JavaRS JUG Rio Grande do Sul*, 1996. Disponível em: <http://www.cin.ufpe.br/~arfs/introjava.pdf>. Citado na página 21.

Jonathan Hedley. *Jsoup*. 2019. Disponível em: <https://jsoup.org/>. Acesso em: 16-11-2019. Citado na página 23.

Krunal A, V. Content Evocation Using Web Scraping and Semantic Illustration. *IOSR Journal of Computer Engineering*, v. 16, n. 3, p. 54–60, 2014. Disponível em: <https://pdfs.semanticscholar.org/4452/fec111652edd0ec96c6dfe179946d681a8c8.pdf>. Citado na página 22.

LEUSIN, S. R. Sistema de Indexação e Recuperação de Informação em construção baseado em ontologia. n. 1, 2007. Disponível em: <http://noriegec.cpgec.ufrgs.br/tic2007/artigos/A1115.pdf>. Citado na página 19.

MAGALHÃES, L. H. de; SOUZA, R. R. SISTEMA DE RECUPERAÇÃO DA INFORMAÇÃO: uma abordagem baseada em ontologias. *PontodeAcesso*, v. 13, n. 2, p. 63–85, 2019. Citado na página 20.

MATOS, T. et al. Índice invertido para recuperação de imagens baseada em conteúdo. In: *Congresso Nacional de Matemática Computacional Aplicada. Belém:[sn]*. [S.l.: s.n.], 2008. Citado na página 18.

MIYATA, B. K.; KANO, V. Y.; DIGIAMPIETRI, L. A. Combinando mineração de textos e análise de redes sociais para a identificação das áreas de atuação de pesquisadores. In: SBC. *Anais do II Brazilian Workshop on Social Network Analysis and Mining*. [S.l.], 2013. p. 79–90. Citado na página 31.

PAUL, D.; HARVEY, D. *Java: how to program 10. ed.* [S.l.]: Prentice Hall, 2014. 1248 p. Citado na página 21.

PINKERTON, B. *Webcrawler: Finding what people want*. [S.l.]: Citeseer, 2000. Citado na página 23.

SALTON, G.; YANG, C.-S. On the specification of term values in automatic indexing. *Journal of documentation*, MCB UP Ltd, v. 29, n. 4, p. 351–372, 1973. Citado na página 17.

Sand Robot. *Simulado Já Enem*. 2018. Disponível em: <https://play.google.com/store/apps/details?id=com.sandrobot.simuladoja_enem/>. Acesso em: 05-01-2020. Citado na página 19.

SINGH, M. S. A. D. J.; VARNICA, B. Web crawler: Extracting the web data. *International Journal of Computer Trends and Technology*, v. 13, n. 3, p. 132–137, 2014. Citado na página 23.

SOUZA, R. R. Sistemas de recuperação de informações e mecanismos de busca na web: panorama atual e tendências. *Perspectivas em Ciência da Informação*, v. 11, 08 2006. Citado 2 vezes nas páginas 17 e 20.

Só Exercícios. *Só Exercícios*. 2014. Disponível em: <https://soexercicios.com.br/plataforma/busca-de-exercicios-vestibular>. Acesso em: 15-12-2019. Citado na página 19.