UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

CEZAR AUGUSTO MEZZALIRA

IDENTIFICAÇÃO DE SIGLAS EM TEXTOS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

CEZAR AUGUSTO MEZZALIRA

IDENTIFICAÇÃO DE SIGLAS EM TEXTOS

Trabalho de Conclusão de Curso, apresentado ao III Curso de Especialização em Tecnologia Java, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2015

IDENTIFICAÇÃO DE SIGLAS EM TEXTOS

Por

Cezar Augusto Mezzalira

Esta monografia foi apresentada às 13h30 do dia 29 de setembro de 2015 como requisito parcial para a obtenção do titulo de ESPECIALISTA, no III curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composto pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Profa. Dra. Beatriz Brsoi
Orientadora
UTFPR – Campus Pato Branco

Prof. Dr. Dalcimar Casanova
Banca
UTFPR – Campus Pato Branco

Profa. Msc. Lucilia Yoshie Araki
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

RESUMO

MEZZALIRA, Cezar Augusto. Identificação de siglas em textos. 2015. 54 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Trabalhos acadêmicos (como relatórios de estágio e de projetos de pesquisa e extensão, monografias e dissertações) geralmente possuem listas de siglas (compostas pelas letras iniciais de palavras e expressões), abreviaturas (parte da palavra) e acrônimos (letras ou sílabas iniciais de expressões formando palavras pronunciáveis). Essas listas devem conter todos os termos (siglas, abreviaturas, acrônimos) dessa natureza que constam no texto e sua respectiva descrição. No texto, esse tipo de termo deve ser precedido, na primeira ocorrência, da sua descrição. A localização dos termos no texto e a sua descrição na primeira ocorrência é uma atividade trabalhosa, especialmente se o texto é extenso. Visando auxiliar nessa atividade, um aplicativo foi desenvolvido com o objetivo de identificar as siglas no texto, descrevê-las em sua primeira ocorrência e gerar a listagem desses termos com a respectiva descrição. O aplicativo desenvolvido é para ambiente *web* e foi implementado utilizando Java e tecnologias associadas ao desenvolvimento web, como o Spring. A Apache POI foi utilizada na implementação de uma das funcionalidades principais do sistema que é a identificação dos termos no texto. Para a consulta de termos visando identificar a sua descrição, um banco de dados é utilizado. As funcionalidades para a geração da lista e descrição das siglas foram implementadas em um aplicativo já existente desenvolvido com o objetivo de cadastrar siglas e seus significados e possibilitar a busca de siglas, gerando listagens. Como restrições ao trabalho implementado, destaca-se a de somente serem manipulados arquivos no formato .docx.

Palavras-chave: Apache POI. Java para web. Listagem de siglas.

ABSTRACT

MEZZALIRA, Cezar Augusto. Identification of acronyms in texts. 2015. 54 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Academic papers (such as training report and research projects and extension, monographs and dissertations) generally have list of acronyms (composed by the initial letters of words and phrases), abbreviations (of the word) and acronyms (letters or initial syllables of expressions forming pronounceable words). These lists must contain all terms (acronyms, abbreviations) that kind that are contained in the text and their respective description. In the text, these kinds of terms must to be preceded, in the first occurrence, from your description. The location of the terms in the text and the description in the first occurrence is a laborious activity, especially if is a long text. Aiming to assist in this activity, an application was developed in order to identify the acronyms in the text, describe them on their first occurrence and generate a list of these terms with their descriptions. The developed application is to web ambience and was implemented using Java and Technologies associated with web development, such as Spring. The Apache POI was used in the implementation in one of the main features of the system, which is the identification of the terms in the text. For the query terms to identify the description, a database is used. The features to generate a list of acronyms and their description have been implemented in a preexisting application, developed with the aim of register acronyms and their meanings and enable the search of the acronyms, generating listings. The main restriction for the implemented work is that the only handled files are those in the .docx format.

Keywords: Apache POI. Web Java. Acronyms.

LISTA DE FIGURAS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
API	*Application Programming Interface*
HTML	*HyperText Markup Language*
JSF	*JavaServer Faces*
MVC	*Model-View-Controller*
NBR	Norma Brasileira
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

Em textos acadêmicos, como relatórios, trabalhos de conclusão de curso e outros, por indicação das diversas normas que parametrizam apresentação desses trabalhos e são propostas pela Associação Brasileira de Normas Técnicas (ABNT) (FACEMA, 2013) é necessário que as siglas, as abreviaturas e os acrônimos sejam descritas em seu primeiro uso e sejam geradas listas agrupadas ou separadas por tipo, contendo o termo e a respectiva descrição. Nessas listagens e no texto, se a sigla, abreviatura ou acrônimo, genericamente denominado por termo neste texto, é de língua estrangeira deve ser grafado em itálico, no texto (em sua primeira ocorrência) e nas listagens.

Sigla é definida pela Norma Brasileira (NBR) como a "reunião das letras iniciais dos vocábulos fundamentais de uma denominação ou título." (ASSOCIAÇÃO…, 2003b, p.2). As siglas são apresentadas por meio de lista. Essa lista consiste na relação alfabética das abreviaturas e siglas utilizadas no texto, seguidas das palavras ou expressões correspondentes grafadas por extenso (ASSOCIAÇÃO…, 2006). A NBR 6029 recomenda-se a elaboração de lista própria para cada tipo. A NBR 6021 (ASSOCIAÇÃO…, 2003a) conceitua índice como uma lista de palavras ou frases, ordenadas segundo determinado critério, que localiza e remete para as informações contidas no texto. Quando da primeira ocorrência da sigla no texto, a forma completa do nome, ou seja, a descrição da sigla, deve precedê-la e a sigla colocada entre parênteses (ASSOCIAÇÃO…, 2003b).

De maneira simplificada, abreviatura é uma parte da palavra que representa a palavra como um todo, por exemplo, tec. para a palavra tecnologia. Siglas são formas de abreviaturas compostas pelas letras iniciais de palavras e de expressões, como, UTFPR para Universidade Tecnológica Federal do Paraná. E acrônimos são palavras formadas por letras ou sílabas iniciais de outras expressões formando uma palavra pronunciável, como Fortran que vem de Formula Tanslator (UNIVERSIDADE..., 2008). Nesse texto é utilizada a palavra termo quando a denominação se refere a qualquer um desses três termos ou aos três termos indistintamente.

Em trabalhos acadêmicos, compor essas listas e verificar se o primeiro uso do termo no texto está acompanhado pela sua descrição é uma atividade trabalhosa e é passível de erros. Um aplicativo computacional que permita cadastrar termos, selecionar termos de uma listagem e gerar a listagem dos mesmos facilita a realização dessa atividade. Outra

funcionalidade interessante para esse tipo de aplicativo é a “coleta”, por meio da identificação dos termos existentes no texto, e a geração de lista com a descrição do significado do termo. E ainda, a funcionalidade de inserir a descrição do termo no seu primeiro uso no texto. A primeira funcionalidade foi desenvolvida como trabalho de conclusão de curso de graduação pelo autor deste trabalho (MEZZALIRA, 2013). As outras duas foram implementadas como resultado da realização deste trabalho.

O aplicativo foi desenvolvido para *web* como forma de facilitar o acesso e para utilizar recursos da Apache POI. Por meio desse aplicativo é possível realizar a consulta do significado de termos e a elaboração de listagens. O aplicativo possibilita a categorização dos termos em áreas e subáreas. E, como funcionalidade principal, a identificação dos termos para gerar a listagem e a inclusão da descrição no primeiro uso do termo no texto. Como forma de redução do escopo e facilitar a implementação da funcionalidade somente siglas serão identificadas na busca pelas mesmas dentro do texto e pelas restrições da tecnologia utilizada são manipulados apenas arquivos no formato .docx.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Implementar um aplicativo para identificação de siglas em texto, gerando automaticamente a listagem das siglas.

1.1.2 Objetivos Específicos

- Identificar siglas em um texto e, se necessário, acrescentar a descrição da primeira ocorrência da sigla no texto.
- Facilitar o trabalho de geração de listagens de siglas em textos como os trabalhos acadêmicos de relatórios de estágios e trabalhos de conclusão de curso.
- Facilitar o trabalho de verificação, consistência entre as siglas existentes no texto e as constantes na listagem.

1.2 JUSTIFICATIVA

A geração de listas de siglas, abreviaturas e acrônimos em trabalhos acadêmicos pode ser uma atividade bastante trabalhosa se o texto é extenso e/ou se existem muitos desses termos no texto. Considerando que é uma atividade simples, embora trabalhosa, no sentido de não agregar valor em termos de conhecimento, o auxílio de um sistema informatizado pode ser bastante relevante.

Para que a busca em texto seja efetiva é necessário que sejam definidos padrões que caracterizam o que está sendo objeto de identificação. A definição de padrões de forma clara e não ambígua facilita a implementação de mecanismos de busca. Por exemplo, a definição de um padrão para identificação de abreviaturas caracterizadas por um termo seguido de um ponto como em “num.” para número pode não ser efetiva. Isso porque, o ponto “.” no final da abreviatura pode ser facilmente confundido com o ponto final de frase. Os acrônimos também podem ser difíceis de identificar, uma vez que podem ser confundidos com nomes próprios ou mesmo com palavras quaisquer. Para as siglas, teoricamente, a definição de um padrão parece ser mais clara, isso porque pode ser caracterizadas como um conjunto de caracteres grafados todos com letras maiúsculas (caixa alta).

A necessidade de estabelecer padrões evita a consulta de todas as palavras de um texto em um dicionário de termos. Contudo, mesmo que essa consulta exaustiva seja realizada, ainda é possível que nomes próprios ou mesmo palavras quaisquer sejam confundidas com termos.

Considerando a complexidade da identificação de abreviaturas e acrônimos, como primeira versão do aplicativo desenvolvido pela realização deste trabalho, somente siglas serão identificadas. E, em decorrência das restrições da Apache POI, utilizada para o tratamento do texto, apenas arquivos no formato .docx são utilizados. Apesar dessas restrições, ressalta-se que o algoritmo básico de busca está implementado, os complementos dependem de ajustes na forma de identificação (embora alguns, inicialmente, não pareçam tão simples) e de melhorias em tecnologias utilizadas.

1.3 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico que fornece a fundamentação conceitual para o aplicativo desenvolvido e, portanto, refere-se à descoberta de padrões em texto.

O Capítulo 3 apresenta as ferramentas e as tecnologias utilizadas para a modelagem e a implementação do sistema. Nesse capítulo também são apresentadas as principais atividades realizadas para o desenvolvimento do trabalho, compondo o método.

Os resultados da realização do trabalho são apresentados no Capítulo 4. Essa apresentação é feita por meio da modelagem, de telas com a explicação do funcionamento do sistema e exemplos da implementação.

Por fim estão as considerações finais sobre o desenvolvimento do trabalho e as referências utilizadas na fundamentação conceitual do trabalho.

2 REFERENCIAL TEÓRICO

O referencial teórico está centrado na descoberta de padrões em texto. Um termo genericamente referindo a sigla, acrônimo ou abreviatura, atende a um determinado padrão que foi estabelecido para a sua identificação dentro do texto. Esse padrão foi utilizado pelo mecanismo, algoritmo, de busca desenvolvido.

2.1 DESCOBERTA DE PADRÕES EM TEXTO

Mineração de texto é o processo de extrair informações consideradas relevantes, conhecimento ou padrões não estruturados em textos de diferentes fontes (NASA, 2012) e de grandes quantidades de documentos (SUBBAIAH, 2013). A mineração de texto se refere à descoberta de informação que não é previamente conhecida no texto (SUKANYAL; BIRUNTHA, 2012).

Sukanyal e Biruntha (2012) ressaltam que a mineração em texto é similar a mineração em dados em termos de objetivos e de técnicas, mas difere pelas ferramentas utilizadas na mineração de dados. Essas ferramentas são, geralmente, desenvolvidas para manipular dados estruturados, mas a mineração de texto pode trabalhar com conjunto de dados semi-estruturados ou não estruturados, como os encontrados em mensagens eletrônicas (*emails*), documentos em diferentes formatos, arquivos *HyperText Markup Language* (HTML), entre outros. Nesses documentos, a maior parte da informação estará armazenada no conteúdo do próprio texto, ou seja, não há a definição de metadados ou outras formas de informação que permitam caracterizar o conteúdo ou identificar aspectos relevantes do conteúdo do texto.

A mineração de texto está voltada para a descoberta de conhecimento interessante em documentos de texto (ASWINI; LAVANYA, 2014). Contudo, para esses autores ainda é um desafio encontrar o conhecimento que seja útil para o usuário e ao mesmo tempo confiável a partir de documentos compostos por texto não estruturado.

De acordo com Morais e Ambrósio (2007, p. 1), as principais contribuições da mineração de texto estão relacionadas à busca de informações específicas em documentos, à análise qualitativa e quantitativa de grandes quantidades de textos e a uma melhor

compreensão do conteúdo de documentos que estão no formato de texto. Loh (2001) ressalta que a mineração de texto se aplica às mais diversas áreas nas quais a identificação de conhecimento em texto possa ser utilizada. Aswini e Lavanya (2014) destacam que muitas aplicações, tais como análises de mercado e gerenciamento de negócios, experimentam os benefícios do uso da informação e do conhecimento extraído de documentos. Classificação de notícias e análise de *emails* indesejados são outras aplicações da mineração de texto citadas por Sukanyal e Biruntha (2012).

A mineração de textos é baseada na identificação de padrões que é uma técnica para extrair padrões de um documento de texto. Um padrão é composto por um conjunto de termos (ASWINI; LAVANYA, 2014). A identificação de padrões inclui identificar as regras específicas que descrevem os padrões nos dados. Os sistemas de mineração de textos são baseados em rotinas de pré-processamento, algoritmos para descoberta de padrões, que representam o núcleo de mineração, e os elementos para apresentação dos resultados (AZEVEDO; BEHAR; REATEGUI, 2010).

As operações de pré-processamento visam identificar e extrair as características representativas do documento sendo analisado. Essas operações transformam dados não estruturados, armazenados em documentos, em uma estrutura que é representada por um modelo intermediário (FELDMAN; SANGER, 2007; TAN, 1999). As operações relacionadas à mineração em si incluem a descoberta de padrões, a análise de tendências e os algoritmos utilizados para a descoberta de conhecimento (AZEVEDO; BEHAR; REATEGUI, 2010). Consultas realizadas em bases de conhecimento podem ser utilizadas para melhorar a qualidade das operações em sistemas para mineração de texto (FELDMAN; SANGER, 2007). A apresentação dos resultados está relacionada com a interface de interação com o sistema e oferece funcionalidades para navegação e acesso à linguagem utilizada para consultas e a apresentação dos resultados das consultas (FELDMAN; SANGER, 2007; PURETSKIY; SHUTT; BERRY, 2010; TAN, 1999).

Existem diversas técnicas para mineração de textos. Nasa (2012) e Sukanyal e Biruntha (2012) citam algumas: sumarização do texto; categorização; classificador Naive Bayes; classificador de vizinho mais próximo; clusterização; algoritmo *k-means; e*xtração de informação.

Subbaiah (2013) destaca como vantagens dos métodos baseados em termos: o desempenho computacional eficiente e as teorias consistentes para atribuição de importância (pesos) para os termos.

3 MATERIAIS E MÉTODO

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias que foram utilizadas para desenvolver o aplicativo.

Ferramenta / Tecnologia	Versão	Referência	Finalidade
Java Platform (JDK)	1.8.0_51	http://www.oracle.com/	Linguagem de programação
MySQL	5.6.25	http://dev.mysql.com/downloads/mysql/	Sistema gerenciador de banco de dados
MySQL Workbench	6.2.3	http://dev.mysql.com/downloads/workbench/	IDE de gerenciamento de bases de dados MySQL
Glassfish	4.1	https://glassfish.java.net/download.html	Servidor web para a aplicação
Spring (MVC, web, data)	4.1.6.	http://www.spring.io/	*Framework* MVC de desenvolvimento *web*
Apache Maven	3.0.5	https://maven.apache.org/index.html	Ferramenta de automação de compilação de projetos Java
Apache POI API	3.12	https://poi.apache.org/	API para manipulação do texto

Quadro 1 – Ferramentas e tecnologias utilizadas

3.2 MÉTODO

A implementação das funcionalidades relacionadas aos objetivos deste trabalho foi realizada a partir de um aplicativo implementado como trabalho de conclusão de curso pelo autor desta monografia. Esse aplicativo (MEZZALIRA, 2014) é, basicamente, um repositório de termos (siglas, acrônimos, abreviaturas) e suas respectivas descrições permitindo selecionar termos cadastrados e gerar um relatório na forma de lista de termos. Tendo esse aplicativo como base, os seguintes complementos foram realizados:

1. Localizar em um documento .docx todas as siglas;
2. A partir dessa procura, pesquisar no banco de dados do sistema (o banco desenvolvido como trabalho de conclusão de curso), os significados das siglas que foram encontradas no documento;
3. Dar possibilidade para que o usuário selecione quais os significados corretos das siglas para substituí-los no documento;
4. Caso o usuário coloque na seção de lista de siglas a chave #LISTASIGLA, gerar automaticamente a lista de siglas e seus significados;
5. Ao solicitar o *download* do arquivo, o texto é atualizado com o acréscimo da descrição da sigla na sua primeira ocorrência no texto e é criada uma lista das mesmas em substituição à chave #LISTASIGLA;

A Figura 1 representa de forma gráfica e sequencial esses passos descritos.

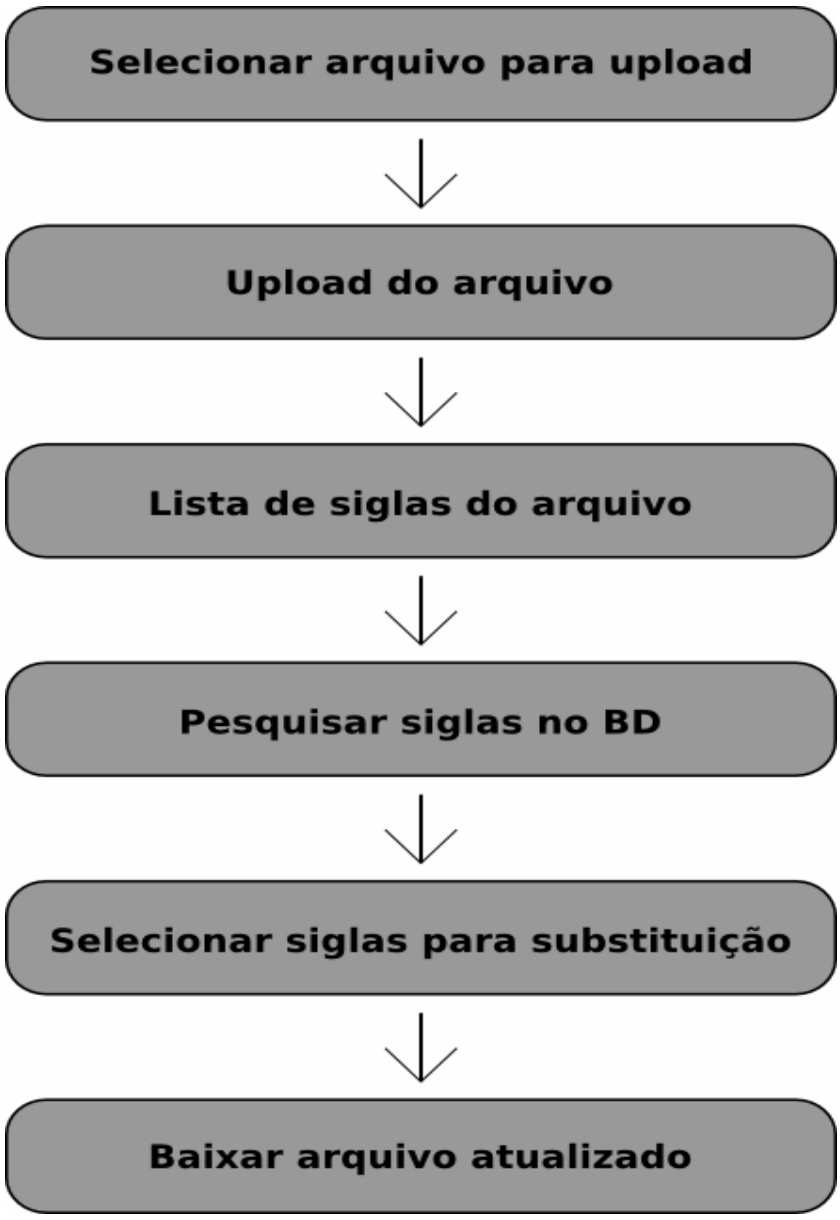

Figura 1 - Sequência de métodos para utilização da rotina de leitura e alteração de arquivo

A seguir está a descrição de procedimento para o método de procura de siglas e o método para acrescentar a descrição da sigla na sua primeira ocorrência no texto.

a) Método de procura de siglas

Para localizar as siglas existentes em um documento .docx é utilizada uma *Application Programming Interface* (API) Java chamada Apache POI. Essa API permite acessar o conteúdo de arquivos no formato padrão utiilizado pela Microsoft (APACHE POI, 2015).

Essa API foi utilizada neste trabalho para fazer a extração dos parágrafos por meio de um conjunto de classes específicas para o formato do pacote org.apache.poi.xwpf.

Cada parágrafo foi extraído para uma variável do tipo *string* e adicionado a uma lista do mesmo tipo, utilizando uma estrutura de repetição que percorre os parágrafos de cada seção do texto.

A regra básica para que o aplicativo determine se a palavra é uma sigla, é que ele deve estar entre parênteses e que todas as suas letras estejam grafadas em maiúsculas. Criou-se essa convenção baseado na determinação de como grafar um termo caracterizado como sigla disposta na publicação de formatação e apresentação de trabalhos acadêmicos da UTFPR (UNIVERSIDADE..., 2008). Essa publicação é utilizada para formatação de trabalhos acadêmicos.

Para encontrar as siglas foi criado um método que apanha cada parágrafo e o segmenta utilizando como referência o espaço entre as palavras, criando, assim, um vetor do tipo *string*. Em seguida uma estrutura de repetição percorre o vetor e executa os seguintes processos e validações em cada palavra:

1. Substitui pontos (ponto final) da *string* da palavra, por espaço em branco.
2. Substitui vírgulas da string da palavra, por espaço em branco.
3. Remove espaços da string da palavra, que podem ter sido pontos ou vírgulas.
4. Se a string da palavra não está vazia, por meio de uma estrutura condicional é verificado se o primeiro caractere da palavra está grafado em maiúsculo e se o último caractere da palavra é um fecha parênteses. Isso significa que a palavra é uma sigla.
5. Se a string da palavra está vazia, outra verificação é realizada por meio de uma estrutura condicional. Se o primeiro caractere da palavra é um abre parênteses e o último caractere é um fecha parênteses, significa que a palavra é uma sigla.

Com a lista de siglas encontradas, é possível, então, executar uma consulta no banco de dados que retorne os significados das mesmas, para que o usuário possa selecionar o significado correto e então baixar o arquivo com as siglas substituídas e a lista atualizada. Essa seleção é necessária para casos de uma mesma sigla possuir significados distintos em decorrência de pertencerem a áreas de conhecimento distintas.

b) Método de substituição de siglas

Utilizando um processo semelhante ao da leitura dos parágrafos por meio de estruturas de repetição e manipulação de *strings*, as siglas são substituídas e a lista de siglas constantes no texto é gerada.

Baseado na seleção das siglas pelo usuário no aplicativo, ao ser solicitado o *download* do arquivo são realizados os seguintes processos:

1. O arquivo original é recuperado na pasta de temporários gerados por *upload* para o servidor do aplicativo.
2. O arquivo é carregado utilizando a API Apache POI para um objeto da classe XWPFDocument.
3. Para fazer a substituição é recuperada a classe controladora a lista das siglas selecionadas.
4. No método de substituição, os parágrafos são percorridos e suas partes são tratadas pelo POI como objetos do tipo *run*.
5. A cada iteração de um *run* é percorrida a lista dos termos.
6. O termo é substituído apenas na sua primeira ocorrência no texto.
7. Caso o termo seja encontrado naquele *run*, ele é adicionado a uma lista de termos já substituídos, para que ele não seja mais substituído em nenhuma outra ocorrência no texto.
8. Depois de substituir todos os termos, é chamado um método para criar a lista de siglas. Esse método compõe, por meio de um objeto StringBuilder, a lista com as siglas selecionadas.
9. O texto já alterado é percorrido novamente procurando pela expressão "#LISTASIGLA".
10. Quando essa expressão é encontrada, ela é substituída pela lista gerada.
11. O arquivo, então, é salvo novamente no servidor e uma requisição de *download* é disparada para que usuário possa baixar o arquivo já atualizado.

c) Limitações do aplicativo

É importante ressaltar que arquivos da extensão ou formato .docx são do formato OpenDocument da Microsoft. Esse formato foi utilizado como padrão porque permite uma melhor portabilidade entre suítes de escritório, especificamente em se tratando da própria suíte da Microsoft e também da suíte aberta LibreOffice.

Outro fator importante para escolha desse formato é que o Apache POI lida melhor com ele para realizar operações que envolvam substituições de textos, sem perder a formatação do documento.

Em testes utilizando documentos no formato .doc, a substituição do texto era facilitada por alguns métodos da própria API, mas como existe o fato de que a sigla só precisa ser substituída uma única vez no texto todo, esse método acabava realizando a substituição em todas as ocorrências da sigla no parágrafo.

Outro problema que foi determinante para não utilização do formato .doc, foi o fato de que utilizando a substituição da sigla uma única vez no parágrafo e substituindo todo o conteúdo do parágrafo antigo pelo parágrafo novo já alterado, o texto inteiro perdia a formatação.

4 RESULTADOS

4.1 ESCOPO DO SISTEMA

Por padrão, uma sigla se refere a um conjunto de letras formando uma palavra, que pode ser ou não pronunciável, e é grafada toda em letras maiúsculas. De maneira mais formal, sigla é uma forma de abreviatura composta pelas letras iniciais de palavras de expressões (UNIVERSIDADE, 2008, p. 35).

Para identificar as siglas no texto é necessário procurar por (identificar) padrões nesse texto. Algumas padronizações foram estabelecidas para o escopo deste trabalho, que são descritas a seguir:

***a)* Identificação da sigla no texto**

Padrão esperado para a sigla no texto:

(SIGLA)

Significado (forma de identificação):

Abre parênteses.

Conjunto de caracteres grafados em letras maiúsculas até o fecha parênteses que não é considerado como parte da sigla.

Fecha parênteses.

b) Geração da lista de siglas

Padrão para a lista de siglas gerada:

Sigla *Descrição da sigla*

Significado:

Sigla grafada em formato normal (sem negrito e/ou itálico)

Marca de tabulação

Descrição da sigla grafada em itálico, se termo de língua estrangeira. A descrição é obtida do banco de dados e a sigla é identificada a partir de metadados se é de língua estrangeira ou não.

c) **Descrição do significado da sigla no texto, na sua primeira ocorrência**

Após identificada a sigla no texto e se a mesma não é antecedida pela descrição, acrescentar a descrição.

Padrão para a identificação da primeira ocorrência da sigla no texto:

Descrição (SIGLA)

Significado:

Descrição grafada de acordo com o armazenado no banco de dados.

Espaço

Sigla entre parênteses

d) **Padronização esperada das siglas no texto para descrição da sigla e identificada**.

A sigla deve ser descrita antes da colocação da sigla entre parênteses.

Padrão para descrição da sigla em sua primeira ocorrência no texto:

Significado da sigla (SIGLA)

Significado:

Abre parênteses

SIGLA

Fecha parênteses

4.2 MODELAGEM DO SISTEMA

Os requisitos funcionais identificados para o aplicativo são apresentados a seguir:

1. Efetuar *upload* de arquivo para que sejam extraídas as siglas do mesmo.
2. Siglas são extraídas do arquivo utilizando uma API específica, a Apache POI.
3. Efetuar pesquisa no banco de dados de siglas, pesquisando o significado das mesmas.
4. Apresentar uma listagem de significados, quando uma mesma sigla armazenada no banco de dados possui significados distintos, possibilitando que o usuário selecione o significado correto para o contexto e área de conhecimento do texto.
5. Substituir as siglas extraídas pelos significados selecionados.

6. Substituir a expressão #LISTASIGLAS, por uma lista das siglas com seu significado ordenadas alfabeticamente.
7. Disponibilizar para o usuário um arquivo atualizado com a substituição das siglas e a geração da lista.

A Figura 2 representa por meio de um diagrama de casos de uso os requisitos funcionais atribuídos a seus respectivos atores.

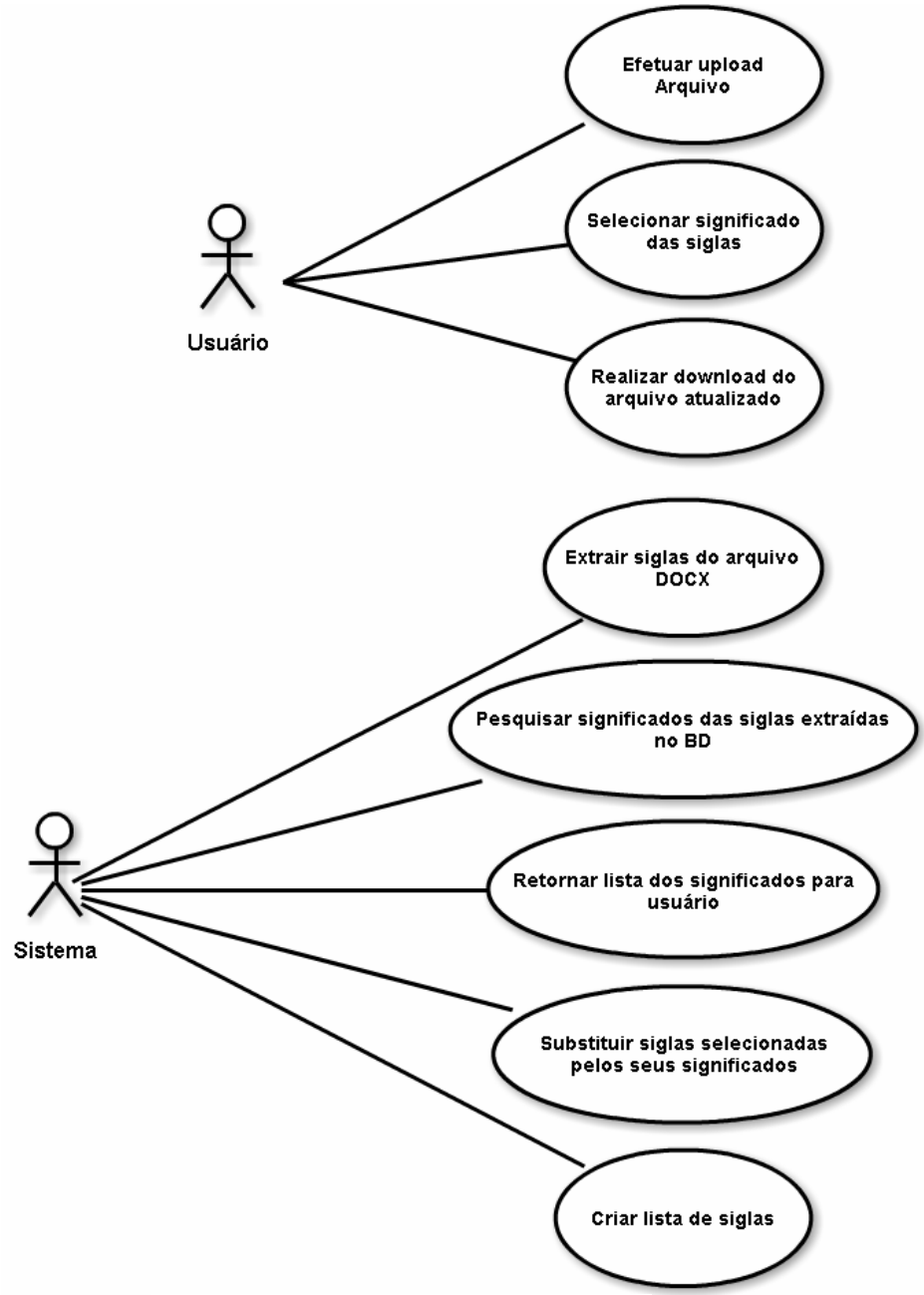

Figura 2 - Diagrama de casos de uso da rotina de leitura

Os casos de uso apresentados na Figura 2 seguem a seguinte sequência de execução dentro do sistema:

1. Usuário efetua upload de um arquivo no formato .docx.
2. Sistema abre o arquivo e efetua a extração das siglas.
3. Usuário requisita a pesquisa das siglas que foram encontradas no texto.
4. Sistema retorna as siglas com o significado em uma lista para seleção.
5. Usuário efetua a seleção dos significados corretos das siglas que foram obtidos por meio da pesquisa anterior.
6. Usuário solicita *download* do arquivo atualizado.
7. Sistema efetua a substituição das siglas selecionadas pelo seu significado.
8. Sistema cria a lista com as siglas e seu significado se encontrar a expressão "#LISTASIGLA" e então disponibiliza o arquivo para *download* para o usuário.

Das funções listadas anteriormente, a maior parte do trabalho é realizada pelo sistema, que automatiza todo um processo que seria manual e moroso, tendo em vista que o usuário teria de procurar todas as siglas no texto manualmente, possivelmente marcando as siglas com um estilo de formatação diferente, para então criar a lista e procurar o significado de cada uma delas individualmente.

Todas as fases de implementação foram executadas seguindo essa sequência. Assim, o processo de desenvolvimento foi progredindo de forma que cada nova rotina implementada era ligada às anteriores com as quais havia vínculo, ou seja, cada passo dependia do passo anterior.

4.3 APRESENTAÇÃO DO SISTEMA

Nesta seção é apresentado o aplicativo e a forma de utilizá-lo. Essa apresentação é realizada por figuras e explicações simples sobre cada passo executado. Será utilizada como base a ordem de execução listada no final da seção anterior.

A Figura 3 mostra a tela inicial do aplicativo. Essa interface, que representa a funcionalidade principal do sistema, é apresentada quando o usuário acessa o sistema.

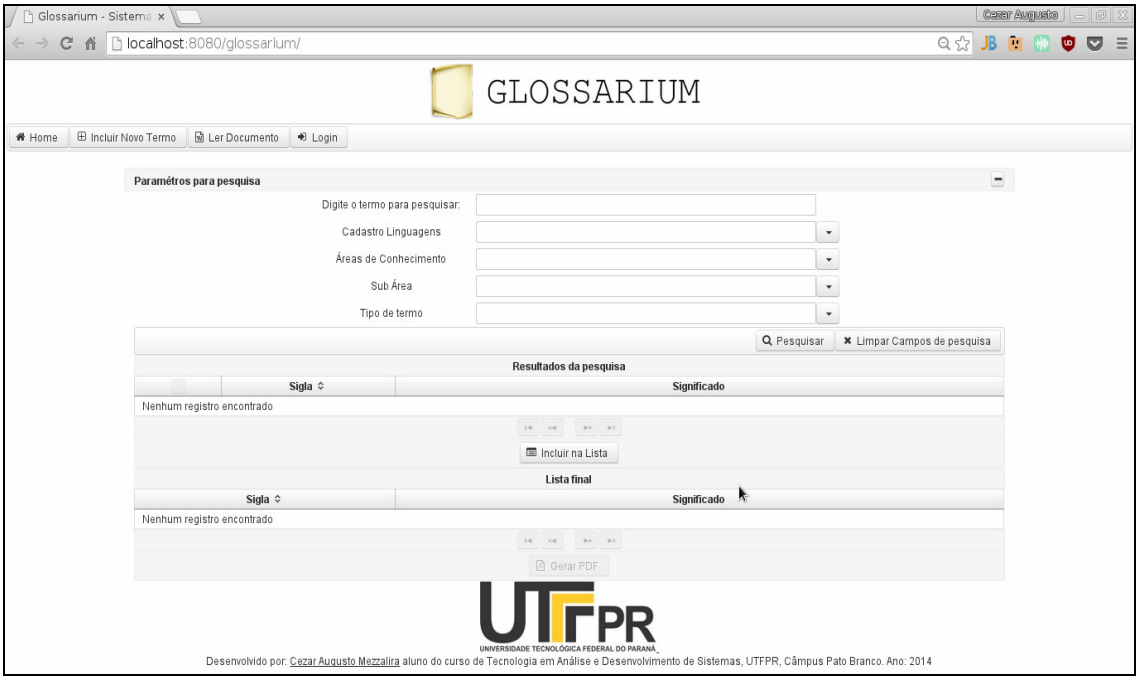

Figura 3 - Tela inicial Glossarium

Para acessar a rotina de leitura de documentos o usuário deve utilizar o botão na barra superior com a descrição “Ler Documento”, conforme indicado pelo destaque na imagem da Figura 4.

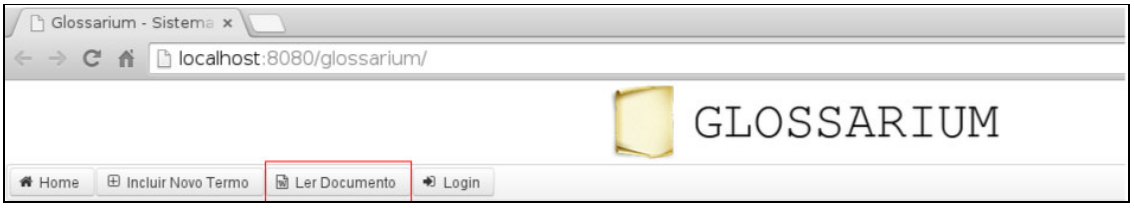

Figura 4 - Acesso à rotina de leitura de documentos

Na Figura 5 está a rotina de leitura de documentos. Ela mantém por padrão o cabeçalho, que contém o nome do aplicativo e a barra de ferramentas com as rotinas principais do sistema.

Figura 5 - Tela da rotina de leitura de documentos

A Figura 6 apresenta a forma de interação do usuário com a funcionalidade principal do sistema, que é a localização de siglas no texto, descrição da sigla em sua primeira ocorrência e geração da lista de siglas. Nessa tela, o usuário pode efetuar o *upload* do arquivo (1), pesquisar os significados (2) e então fazer a seleção dos corretos (3), para que então o sistema devolva o arquivo com as siglas substituídas quando for solicitado o *download* do arquivo (4).

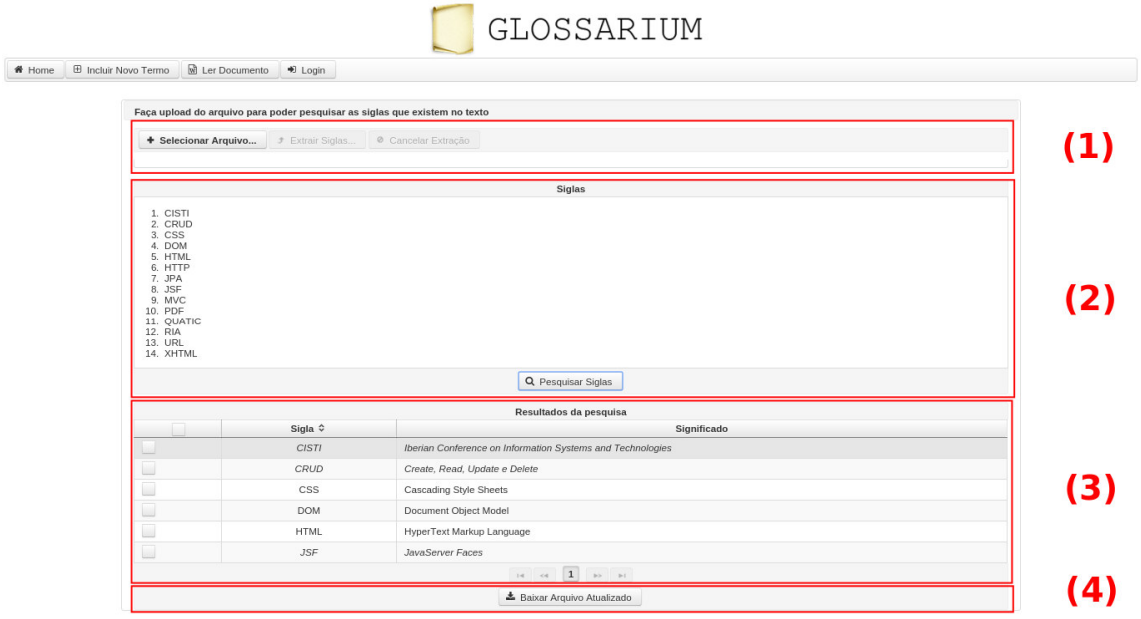

Figura 6 - Disposição das funcionalidades da rotina

O primeiro passo que o usuário deve fazer dentro dessa rotina, é efetuar o *upload* do arquivo, seguindo os seguintes passos:

1. Clicar no botão *Selecionar Arquivo...*

2. Selecionar e abrir um arquivo no formato .docx.

3. Clicar no botão *Extrair Siglas*.

A Figura 7 destaca, dentro do retângulo vermelho, o botão no qual o usuário deverá clicar para que seja apresentada a caixa de diálogo para que o arquivo possa ser escolhido.

Figura 7 - Botão para escolher arquivo

A Figura 8 mostra a tela de diálogo para selecionar o arquivo após clicar no botão *Selecionar Arquivo*.

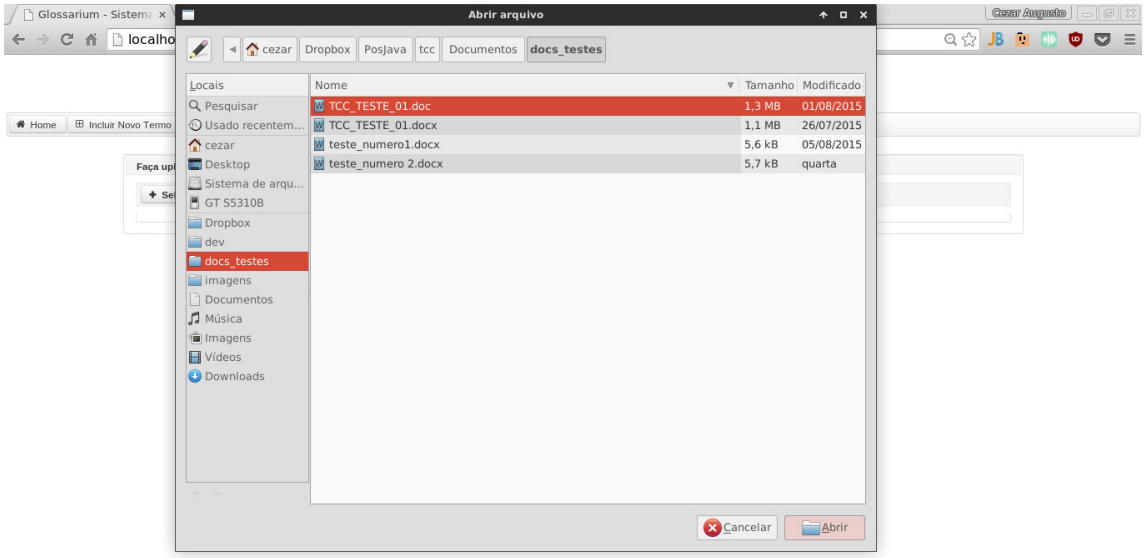

Figura 8 - Tela de seleção de arquivo

A Figura 9 apresenta a tela que é apresentada após a seleção do arquivo, destacando, por meio de áreas circuladas na cor vermelha, o arquivo que foi selecionado e o botão de *Extrair Siglas*.

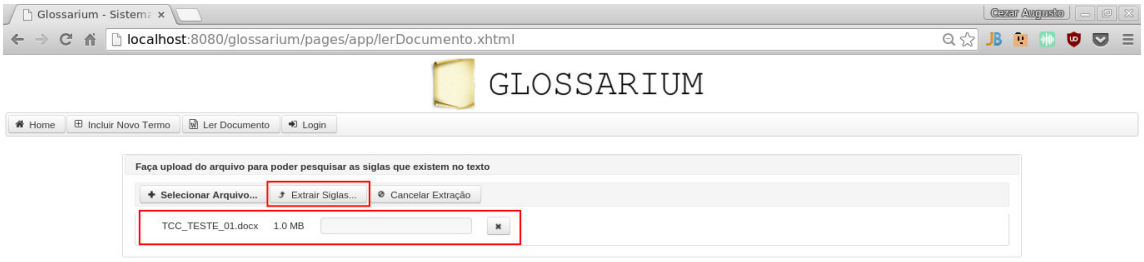

Figura 9 - Arquivo selecionado para upload

Ao clicar em *upload*, se o arquivo é um .docx, então o sistema irá lê-lo e extrairá do texto as siglas existentes, conforme apresentado na Figura 10.

Figura 10 - Lista de siglas extraídas do texto

Após isso, o usuário pode, então, solicitar a busca dos significados das siglas clicando no botão pesquisar siglas, conforme mostrado na Figura 11. A área destacada nessa Figura apresenta o botão para a pesquisa por siglas.

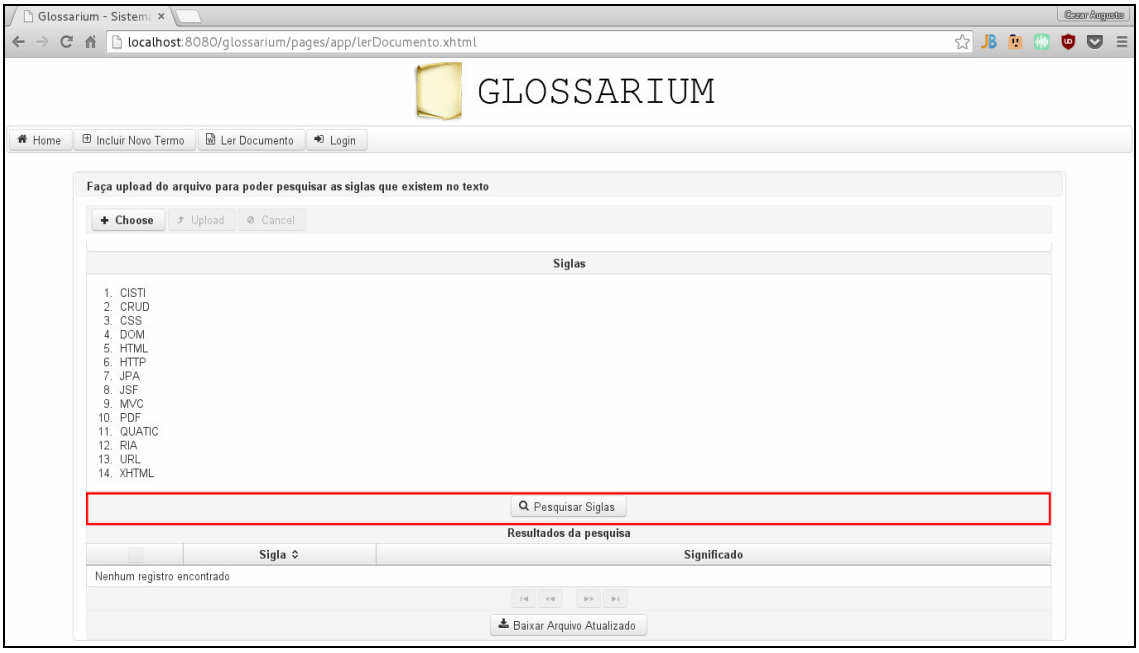

Figura 11 - Botão para pesquisa das siglas extraídas

Se a pesquisa encontrar significados armazenados no banco de dados, então eles serão listados em uma lista de seleção. Um exemplo dessa lista é apresentado na Figura 12.

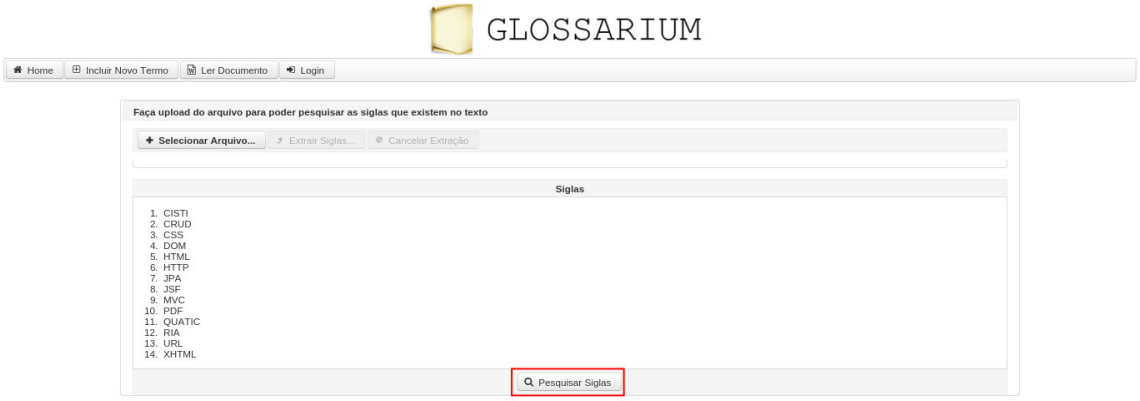

Figura 12 - Lista de siglas extraídas e lista de siglas encontradas

O próximo passo a ser executado pelo usuário é selecionar os significados corretos, de acordo com as siglas encontradas e o contexto do texto. Esses significados são utilizados para substituir as siglas no texto, quando o usuário requisitar o *download* do arquivo.

Na Figura 13 é apresentada somente a lista de siglas pesquisadas e já selecionadas para serem substituídas no texto.

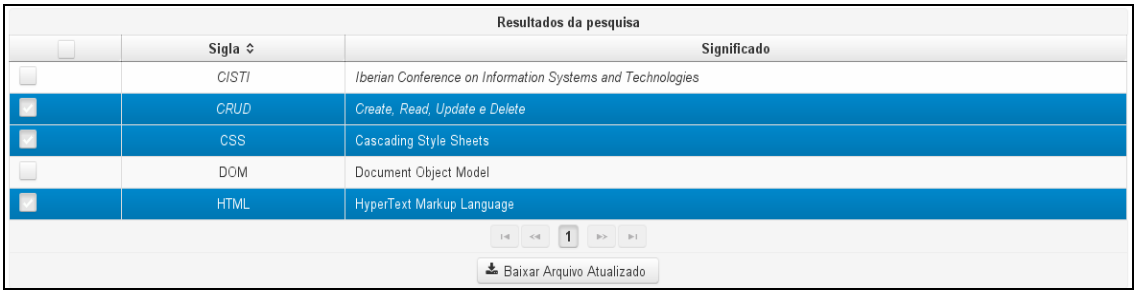

Figura 13 - Lista de siglas selecionadas

O último passo é solicitar o *download* do arquivo atualizado, clicando no botão baixar arquivo atualizado. Nesse momento são realizadas duas operações com o arquivo. Inicialmente, são substituídas todas as siglas pelos seus significados. Por exemplo, a sigla CSS é substituída pela expressão "*Cascading Style Sheet* (CSS)". Na sequência, se a expressão "#LISTASIGLA" for encontrada no texto, então é criada uma lista com as siglas e seus significados e a expressão é substituída por essa lista.

Na Figura 14 está a janela de diálogo para salvar o arquivo já atualizado. É possível notar que o nome do arquivo é alterado para identificar que ele foi manipulado pelo aplicativo.

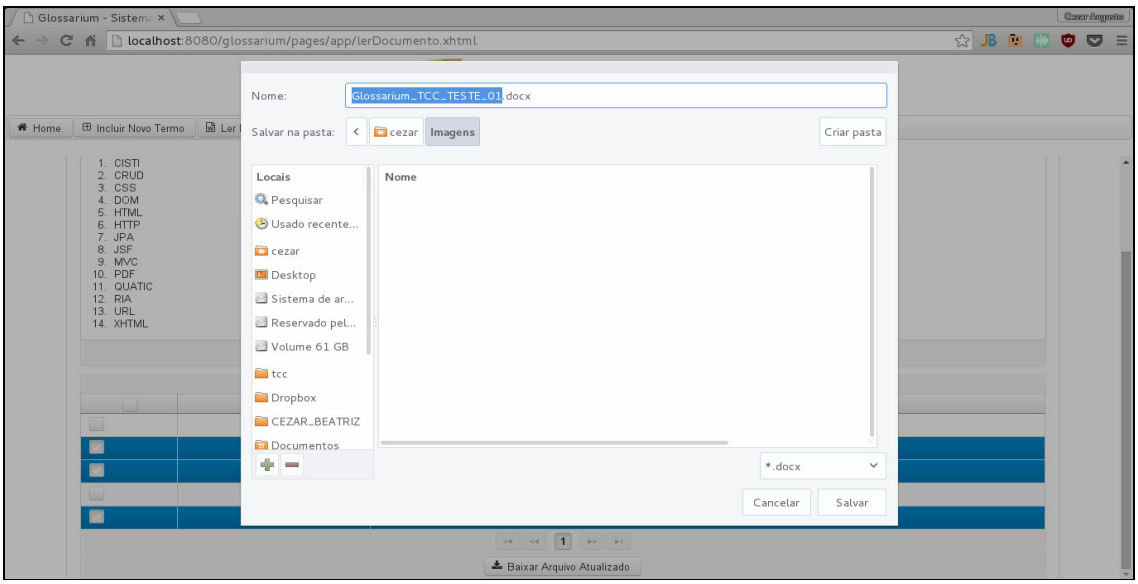

Figura 14 - Download do arquivo atualizado

No Apêndice A está um exemplo de uso do sistema.

4.4 IMPLEMENTAÇÃO DO SISTEMA

a) Componentes Primefaces e eventos ajax

A partir das implementações já realizadas como trabalho de graduação (MEZZALIRA, 2013), foi necessário prover acesso para a *API Apache POI* por meio de algumas classes que permitissem a leitura e a manipulação de arquivos.

Antes de iniciar o desenvolvimento, foram realizadas algumas melhorias no *backend*, tais como, atualização das versões dos principais pacotes de *frameworks* e bibliotecas, dentre as quais o *Spring* (*Data, Model-View-Controller* (MVC), *Web*), *Primefaces* e *Jasper Reports, além* da inclusão dos pacotes da *API Apache POI*.

No *frontend* foram implementadas melhorias de usabilidade, organização da interface e implementação de utilização de *template* de telas. Visando, assim, padronizar a implementação das telas do sistema, facilitando a criação de novas telas.

Na Listagem 1 é possível verificar o código fonte do *template* padrão de telas para rotinas que podem ser acessadas sem efetuar *login* no sistema. Essa tela insere um cabeçalho com os menus e define quais partes devem ser inseridas quando esse *template* for utilizado para criar outra tela. Também está codificado um *dialog* que permite a inserção de siglas por meio de um botão que existe no menu que é inserido no *header* da página.

Outra implementação agregativa em termos de interface foi a utilização da fonte *Awesome* do projeto *Twitter BootStrap*. Essa fonte disponibiliza caracteres em formato de ícones, que são redimensionados quando o conteúdo da página é redimensionado. Para utilização dessa fonte, foi necessária a adição do pacote *maven* "org.webjars:font-awesome:4.3.0-2".

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:h="http://xmlns.jcp.org/jsf/html"
       xmlns:p="http://primefaces.org/ui"
       xmlns:f="http://xmlns.jcp.org/jsf/core"
       xmlns:components="http://java.sun.com/jsf/composite/components"
       xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
 <h:head>
     <title>
         <h:outputText value="Glossarium - Sistema de Consulta e Lista De Siglas,
Abreviaturas E Acrônimos"/>
     </title>
     <h:outputStylesheet name="webjars/font-awesome/4.3.0/css/font-awesome-
jsf.css"/>
```

```
     <h:outputStylesheet name="css/index.css"/>
 </h:head>
<!-- Corpo padrão para todas as telas de acesso comum-->
 <h:body>
     <f:view>
      <!--Componente de layout para a tela toda-->
            <p:layout fullPage="true">
            <!--Crio o topo, que contém os botões das rotinas e de login-->
      <p:layoutUnit id="topo" position="north" styleClass="panelNoBorder"
size="125">
      <!-- através da tag ui:insert é feita a inserção de uma pagina ou recurso-->

                <ui:insert name="topo">
                    <ui:include src="/resources/template/header.xhtml"/>
                </ui:insert>
            </p:layoutUnit>
      <!-- Crio o centro, aonde serão inseridos o conteudo e o rodape e também
serão exibidas as mensagens do sistema.
            <p:layoutUnit id="centro" position="center"
styleClass="panelNoBorder">
                <p:messages id="messages" closable="true"/>
                <ui:insert name="conteudo"/>
                <ui:insert name="rodape"/>
            </p:layoutUnit>
        </p:layout>
      <!-- posso inserir também outros componentes não visuais-->
        <ui:insert name="foraDoForm"/>

      <!--Declaro o dialog de inclusão de siglas aqui, pois ele podera ser
acessado a partir de qualquer lugar do sistema aonde possua o cabeçalho padrão-->
        <p:dialog widgetVar="cadSigla" modal="true" draggable="false"
closable="true" resizable="false"
                  header="Novo Termo" appendTo="@(body)" width="600"
height="400">
            <h:form>
                <p:panelGrid columns="2">
                    <p:outputLabel value="#{form.siglaid}" for="siglaid"/>
                    <p:inputText value="#{siglaController.entity.siglaid}"
                                 label="#{form.siglaid}" id="siglaid"
disabled="true"/>
                    <p:outputLabel value="Sigla" for="c_sigla"/>
                    <p:inputText value="#{siglaController.entity.sigla}"
                                 label="#{form.sigla}" id="c_sigla"/>
                    <p:outputLabel value="#{form.significado}"
for="c_significado"/>
                    <p:inputTextarea
value="#{siglaController.entity.significado}"
                                     label="#{form.significado}"
id="c_significado"/>
                    <p:outputLabel value="Lingua" for="c_linguagem"/>
                    <p:autoComplete value="#{siglaController.entity.linguaid}"

completeMethod="#{siglaController.completeLinguagem()}"
                                    converter="#{linguagemConverter}"
                                    label="#{form.linguagem}" id="c_linguagem"
var="linguagem"
```

```
                                    itemLabel="#{linguagem.descricao}"
itemValue="#{linguagem}"
                                    dropdown="true" forceSelection="true"
required="true"/>
                    <p:outputLabel value="Área de Conhecimento" for="c_area"/>
                    <p:autoComplete value="#{siglaController.entity.areaid}"
completeMethod="#{siglaController.completeArea()}"
                                    converter="#{areaConverter}"
                                    label="#{form.area}" id="c_area" var="area"
                                    itemLabel="#{area.descricao}"
itemValue="#{area}"
                                    dropdown="true" forceSelection="true"
required="true"/>
                    <p:outputLabel value="Subárea de Conhecimento"
for="c_subarea"/>
                    <p:autoComplete value="#{siglaController.entity.subareaid}"
completeMethod="#{siglaController.completeSubArea()}"
                                    converter="#{subAreaConverter}"
                                    label="#{form.subarea}" id="c_subarea"
var="subarea"
                                    itemLabel="#{subarea.descricao}"
itemValue="#{subarea}"
                                    dropdown="true" forceSelection="true"/>
                    <p:outputLabel value="Tipo de Termo" for="c_tiposigla"/>
                    <p:autoComplete value="#{siglaController.entity.tiposiglaid}"

completeMethod="#{siglaController.completeTipoSigla()}"
                                    converter="#{tipoSiglaConverter}"
                                    label="#{form.tiposigla}" id="c_tiposigla"
var="tiposigla"
                                    itemLabel="#{tiposigla.descricao}"
itemValue="#{tiposigla}"
                                    dropdown="true" forceSelection="true"
required="true"/>
                </p:panelGrid>
                <components:commandButton update="@form"
controller="#{siglaController}"/>
            </h:form>
        </p:dialog>
    </f:view>
 </h:body>
</html>
```

Listagem 1 - Código template default.xhtml

Ainda no *frontend*, foi criada uma nova tela para leitura de documentos, que a tela da principal implementação deste trabalho. Essa tela foi criada utilizando o *template* listado anteriormente como base. Nessa tela foram utilizados componentes mais complexos em relação aos utilizados na primeira versão do sistema.

O primeiro componente utilizado nessa tela foi o *FileUpload* do *Primefaces*. Esse componente é mais completo em termos de propriedades do que o componente padrão de

upload do *JavaServer Faces* (*JSF*). Na Listagem 3 está a implementação do componente na tela de leitura documento.

```
<p:fileUpload fileUploadListener="#{siglaController.upload}"
      allowTypes="/(\.|\/)(docx)$/" sizeLimit="5000000"
      description="Selecione o arquivo"
      mode="advanced" dragDropSupport="true"
      fileLimit="1"
      fileLimitMessage="Só é possível efetuar upload de um arquivo"
      update="@form" process="@form" styleClass="panelNoBorder"
      style="margin-top: 10px; !important;" uploadLabel="Extrair Siglas..."
```

Listagem 2 - Componente de upload Primefaces

Outro componente do *Primefaces* utilizado foi o *DataList* para listar as siglas encontradas em documento carregado para o sistema. A Listagem 4 mostra a utilização do componente na tela de leitura de documentos.

```
<!-- Data List responsável por mostrar a lista das siglas existentes no texto -->
<p:dataList id="listTermos" value="#{siglaController.terms}"
            emptyMessage="#{msg.nenhum_registro_encontrado}"
            var="term" type="ordered"
            style="margin-top: 10px;">
      <f:facet name="header">
                  Siglas
      </f:facet>
      #{term}
      <f:facet name="footer">
                  <p:commandButton id="pesquisaSiglas" value="Pesquisar Siglas"
                        icon="fa fa-search"
                        process="@form" update="@form"
                        action="#{siglaController.pesquisarSiglas()}"
                        disabled="#{siglaController.terms == null}"/>
      </f:facet>
</p:dataList>
```

Listagem 3 - Componente DataList do Primefaces

Um componente utilizado em várias partes do sistema é o *DataTable* que também é do *Primefaces*. Nesse componente foi tratado o evento de seleção de item na lista por meio da utilização de eventos por *Ajax*. Os seguintes eventos foram tratados: *rowSelectCheckbox*, *rowUnselectCheckbox* e *toggleSelect*. Esses três eventos atualizam a lista de siglas selecionadas para substituição.

No evento *rowSelectCheckbox* é chamado o método *onRowSelect d*a classe controladora SiglaController. Isso ocorre, quando o *checkbox* do item da lista de siglas e significados é marcado. Nesse método é validado se o item já existe na lista e caso não exista, ele é adicionado a lista. Na Listagem 5 está o método implementado.

```
public void onRowSelect(SelectEvent event) {
     if (!itensAdicionados.contains(((Sigla) event.getObject()))) {
         itensAdicionados.add(((Sigla) event.getObject()));
     }
}
```

Listagem 4 - Método onRowSelect chamado via ajax

O evento *rowUnselectCheckbox chama o método onRowUnselect* da classe controladora SiglaController, quando o *checkbox* do item da lista de siglas e significados é desmarcado. O método então exclui da lista de itens selecionados (itens desmarcados pelo usuário). A Listagem 5 apresenta a implementação do método *onRowUnselect*.

```
//método responsável por excluir da lista de itens selecionados o item desmarcado
public void onRowUnselect(UnselectEvent event) {
     itensAdicionados.remove(((Sigla) event.getObject()));
}
```

Listagem 5 - Método onRowUnselect chamado via ajax

O evento *toggleSelect* é executado todas as vezes que for marcado ou desmarcado o *checkbox* que está no cabeçalho da coluna de seleção dos itens da lista. Quando esse *checkbox* estiver marcado, todos os itens da lista serão marcados e então será chamado, para cada um dos itens, o método *onRowSelect*. Já quando esse *checkbox* estiver desmarcado, todos os itens serão desmarcados e será executado, para cada item da lista, o método *onRowUnselect*.

b) Implementações utilizando Apache POI

Nesta seção serão listados trechos de código das implementações dos métodos de leitura e manipulação de arquivos .docx utilizando a API Apache POI.

A primeira utilização da API ocorreu com a leitura de um arquivo carregado para o sistema com a utilização da classe ReadWordUtil. Essa classe em um primeiro momento lia documentos .doc e .docx. Por fim, optou-se apenas por utilizar o formato .docx que é mais adequadamente manipulado pela API, tanto para leitura quanto para edição. A sequência a seguir lista os passos para extração das siglas do texto:

1. O arquivo carregado para o servidor é aberto pela API Apache POI.
2. É criado um objeto do tipo XWPFDocument que contém todo o texto, separado por seções e parágrafos.
3. A primeira iteração ocorre percorrendo as seções, pois elas contém os parágrafos.

4. A segunda iteração ocorre percorrendo os parágrafos, extraindo seu texto na íntegra para uma variável do tipo *string* que, em seguida, é adicionada a uma lista do tipo ArrayList.

5. A terceira iteração ocorre em método específico que percorre o ArrayList dos parágrafos, sendo que cada parágrafo é quebrado em um vetor de palavras. O identificador para a quebra é o espaço entre as palavras.

6. A quarta iteração ocorre com o vetor de palavras. Cada palavra recebe um tratamento antes de ser validada se é uma sigla ou não. Esse tratamento consiste em primeiramente substituir pontos e vírgulas por espaços na *string* da palavra e, por fim, retirar todos os espaços da mesma utilizando o método *trim*.

7. Para a primeira validação são removidos todos os parênteses da palavra e então é verificado se todos os caracteres estão em maiúsculo. Se a condição for verdadeira, então existem duas condições: a primeira é se a primeira letra da palavra é maiúscula e o último caractere é um fecha parênteses; a segunda é se o primeiro caractere é um abre parêntese e o último caractere é uma letra maiúscula. Se uma dessas duas condições for atendida, então significa que a palavra identificada é uma sigla. Em sendo uma sigla, ela é adicionada para um ArrayList com as siglas sem os parênteses.

O método readDocxFile implementa o descrito nos itens de 1 a 4 desses passos listados. Esse método tem como único objetivo extrair os parágrafos para uma lista de variáveis do tipo string, compreendendo que cada variável representa um parágrafo. Na Listagem 6 é possível verificar como foi feita a implementação desse método.

```
public List<String> readDocxFile(InputStream inputStream) {
     List<String> listTerms = null;
     try {
            //Crio uma instancia do objeto  XWPFDocument para armazenar o arquivo
que veio
      //através do stream de entrada do arquivo que foi carregado
      XWPFDocument xwpfDocument = new XWPFDocument(inputStream);

      //Crio uma lista dos paragrafos do texto
        List<XWPFParagraph> paragraphs = xwpfDocument.getParagraphs();

      //crio uma lista para armazenar os paragrafos em forma de string
      listTerms = new ArrayList<>();

      //extraio os paragrafos para string e adiciono um a um na lista
      for (XWPFParagraph para : paragraphs) {
            listTerms.add(para.getText());
        }
     } catch (Exception e) {
        e.printStackTrace();
```

```
    }
    //retorno a lista dos paragrafos
    return listTerms;
}
```

Listagem 6 - Método readDocxFile

Os métodos *findTerms*, *addTerm* e *validTerm* compreendem aos passos 5 a 7 da listagem anterior. O método *findTerms* utiliza o método validTerm para validar se todas as letras da palavra estão em maiúsculo e utiliza o método *addTerm* para adicionar o termo em uma lista de escopo global da classe. Há, ainda, um método auxiliar chamado *removeParenthesis* que tem apenas o intuito de remover os parênteses da palavra.

A Listagem 7 mostra como foi codificado o método *findTerms*. Esse método percorre a lista de parágrafos, quebrando-os em vetores de palavras por meio dos espaços entre elas. Então, a partir desse vetor, é realizada uma segunda iteração que percorre todo o vetor procurando pelas siglas.

```
public List<String> findTerms(List<String> paragraphs) {
    terms = new ArrayList<>();
    int countWords = 0;
    //percorre a lista de parágrafos e transforma eles em listas de palavras...
    for (int i = 0; i < paragraphs.size(); i++) {
        String paragraph = paragraphs.get(i);
        //separa as palavras em
        String[] words = paragraph.split(" ");
        for (int j = 0; j < words.length; j++) {
            String word = words[j];
            countWords++;
            //remove os pontos em caso de palavras que estão em finais de
parágrafos.
            word = word.replace('.', ' ');
            //remove os virgulas em caso de palavras que estão em finais de
orações.
            word = word.replace(',', ' ');
            //elimina os espaços
            word = word.trim();
            /*Cezar 07/09/2015
            * Criada variavel com palavra sem os parenteses
            * para diminuir tempo de processamento na validação.
            *
            * */
            String tempWord = removeParenthesis(word);
            if (validTerm(tempWord) && !tempWord.isEmpty()) {
                ///Se o primeiro caractere for maiusculo e o ultimo for fecha
parenteses...
                //Evita numeros e listas de letras
                if ((Character.isUpperCase(word.charAt(0))) &&
                        (Character.compare(word.charAt(word.length() - 1), ')')
== 0)) {
                    addTerm(tempWord);
```

```
                } //Se o primeiro caractere for abre parenteses e o ultimo for
fecha parenteses...
                else if ((Character.compare(word.charAt(0), '(') == 0) &&
                        (Character.compare(word.charAt(word.length() - 1), ')')
== 0)) {
                    addTerm(tempWord);
                }
            }
        }
    }
    return terms;
 }
```

Listagem 7 - Implementação do método findTerms

Na Listagem 8 está a implementação do método validTerm, que valida se todas as letras da palavra estão grafadas em letras maiúsculas.

```
private boolean validTerm(String word) {
  //converte a palavra para array
  char[] letters = word.toCharArray();
  //termina em -1 para verificar se a palavra é inteira maiuscula
  boolean isTerm = true;
  //Verifica se na palavra todos os caracteres são números.
  if (word.matches("[0-9]*\\d+.*")) {
    isTerm = false;
  } else {
    //se não existirem numeros, então faz a validação se todos os caracteres são maiusculos
    for (int j = 1; j < letters.length - 1; j++) {
      //se o caracter não for maiusculo, a flag é marcada como false e o loop é interrompido
      if (!Character.isUpperCase(letters[j])) {
        isTerm = false;
        break; //termino o laço para evitar processamento desnecessario
      }
    }
  }
  //retorna o valor da flag
  return isTerm;
}
```

Listagem 8 - Implementação do método validTerm

Na sequência, a Listagem 9 mostra a implementação dos métodos *addTerm* *removeParenthesis*, que são métodos auxiliares ao método *findTerms*.

```
private void addTerm(String word) {
  //se for um termo
  if (validTerm(word)) {
    //se o termo não estiver na lista
    if (!terms.contains(word)) {
      //adiciona o termo
      terms.add(word);
    }
  }
}
private String removeParenthesis(String word) {
  //retorno a propria palavra sem os parenteses e sem espaços
  return word.replace('(', ' ').replace(')', ' ').trim();
```

Listagem 9 - Implementação dos métodos auxiliares addTerm e removeParenthesis

Na parte de edição do arquivo, quando o usuário já fez as seleções dos significados das siglas, é utilizada a classe *WordPOIUtil*. Essa classe contém todos os métodos necessários para abrir um arquivo e editá-lo de acordo com as premissas definidas no escopo do projeto.

Essa classe é utilizada diretamente pelo controlador *SiglaController* que a acessa utilizando o método atualizarDocumento, mostrado na Listagem 10. Esse método cria uma instância da classe *WordPOIUtil*.

```
public void atualizarDocumento() {
  //crio uma variavel com o caminho do arquivo no servidor
  String path = destination + fileName;
  //Crio uma variavel com o caminho do novo arquivo depois de alterado
  String pathDownload = destination + "Glossarium_" + fileName;
  //chamo o método adicionar na lista para adicionar os itens
  // selecionados na tela na lista de significados selecionados
  adicionarNaLista();
  //Crio uma instancia da classe WordPOIUtil que é responsavel por manipular e
alterar o documento.
  WordPOIUtil wordPOIUtil = new WordPOIUtil();
  //Chamo o método replaceAndGenerateWord, passando por parametro o path no servidor,
  // o path para salvar o novo arquivo, qual chave deve ser procurada para
substituição
  // e o conteudo para ser substituído.
  boolean generateFile = wordPOIUtil.replaceAndGenerateWord(path, pathDownload,
"#LISTASIGLA", itensAdicionados);
  try {
    //Se o método de substituição retornar verdadeiro, significa que o arquivo foi
alterado com sucesso
    // e então chamo a tela para download
    if (generateFile) {
      downloadFile(pathDownload);
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Listagem 10 - Implementação do método atualizarDocumento

Então é chamado o método *replaceAndGenerateWord* que passa por referência o caminho do arquivo que foi carregado para o servidor e que está em uma pasta temporária que um caminho temporário para salvar o novo arquivo, a chave “#LISTASIGLA”. Identifica, assim, que deve ser criada uma lista com os termos para ser substituída por essa chave. E por fim a lista das siglas selecionadas. Caso a substituição seja realizada com sucesso, o método retornará verdadeiro e o arquivo será disponibilizado para *download*.

Dentro da classe *WordPOIUtil* o método *replaceAndGenerateWord* verifica qual o formato do arquivo e, então, chama o método correto de acordo com o tipo do arquivo. Como está sendo utilizado somente o formato .docx, serão utilizados somente os métodos que tratam esse formato especifico.

Para arquivos .docx o método sequencial é o método *replaceOnDocx*, que tem por objetivo abrir o arquivo e chamar os métodos *replaceTerms* e *replaceSectionListOnDoc. A*lém de salvar o novo arquivo com o conteúdo e nome atualizados, conforme mostrado na implementação descrita na Listagem 11.

```
private boolean replaceOnDocx(String srcPath, String destPath,
            String sectionName, List<Sigla> lsEntity) {
  try {
    //Crio uma instancia de XWPFDocument e abro o documento salvo no servidor
    XWPFDocument document = new XWPFDocument(
        POIXMLDocument.openPackage(srcPath));
    // Substituir os parágrafos de texto especificados
    document = replaceTerms(document, getListToReplace(lsEntity));
    //Se o documento for vazio, o método retornara false
    if (document == null) {
      return false;
    } else {
      //se as siglas foram substituidas com sucesso, então também cria a lista atraves da
substituição
      // da chave passada por parametro, como por exemplo #LISTASIGLA
      document = replaceSectionListOnDocx(document, sectionName, lsEntity);
    }
    //crio um stream de saida
    FileOutputStream outStream = new FileOutputStream(destPath);

    //escrevo as alterações no documento novo
    document.write(outStream);

    //fecho o novo documento
    outStream.close();
    return true;
  } catch (Exception e) {
    e.printStackTrace();
    return false;
  }
}
```

Listagem 11 - Implementação do método replaceOnDocx

O método *replaceTerms* é responsável por substituir no texto as siglas e os seus significados, conforme a seleção das mesmas pelo usuário antes de executar o *download*. Ele recebe por parâmetro um objeto *XWPFDocument*, que é o próprio documento aberto pelo POI, e recebe, também, a lista de objetos da classe auxiliar Termo, que possui os atributos termo (que é a sigla em si), significado (que é a descrição do termo) e estrangeiro (que diz se a sigla é de língua estrangeira ou não). Essa lista especifica é criada por meio do método

getListToReplace que recebe a lista de siglas e monta uma lista simplificada para facilitar a substituição das siglas.

O *Apache POI* trata os parágrafos quebrando-os em pedaços chamados *run*. Dessa forma, são necessárias duas iterações: primeiro entre os parágrafos e depois entre as partes dos parágrafos.

O texto de cada *run* é extraído para uma variável do tipo *String*, pois será nela que será substituída a sigla, para então ser atualizado o texto da *run*. São retirados os parênteses dessa *string*, para que apenas o termo seja substituído.

Para agilizar o processo, em cada *run* é percorrida a lista de siglas, validando se existe o termo da referida iteração. Caso o termo exista nessa *string*, então ele é substituído pelo seu significado e a sigla é colocada entre parênteses, isso somente na primeira ocorrência da sigla no texto. Em seguida, a sigla substituída é adicionada em uma lista secundária, para que ela não seja mais substituída em nenhuma outra ocorrência no texto. No fim da operação de substituição, é retornado o documento em forma de objeto do *Apache POI*, para que não seja necessário abrir o documento duas vezes. A Listagem 12 apresenta a implementação desse método.

```
private XWPFDocument replaceTerms(XWPFDocument document, List<Termo> terms) {
  //recebo por parametro o documento original
    XWPFDocument docChanged = document;


  try {
    //crio um iterator para usar na iteração entre os paragrafos
    Iterator<XWPFParagraph> itPara = docChanged
        .getParagraphsIterator();
    //crio uma lista para armazenar os itens já substituidos
    List<String> replacedTerms = new ArrayList<>();
    //percorro os paragrafos enquanto existirem paragrafos
    while (itPara.hasNext()) {
      //crio uma instancia de um paragrafo
      XWPFParagraph paragraph = (XWPFParagraph) itPara.next();
      //recupero as partes dos paragrafo, tratadas pelo POI como run
      List<XWPFRun> runs = paragraph.getRuns();
      //percorro as partes do paragrafo
      for (int i = 0; i < runs.size(); i++) {
        //recupero em uma variavel string o texto do run
        String oneparaString = runs.get(i).getText(
            runs.get(i).getTextPosition());
        //se não foi marcado nenhum termo, então
        if (terms == null) {
          break;
        } else if (oneparaString != null) {
          //retira todos os parenteses para substituir apenas o termo
          oneparaString = oneparaString.replace("(", "");
          oneparaString = oneparaString.replace(")", "");
          for (Termo termo : terms) {
            //se não estiver na lista de substituidos e conter o termo na string
            if ((!replacedTerms.contains(termo.getTermo())) &&
                (oneparaString.contains(termo.getTermo()))) {
              String term = termo.getTermo();
              //crio um string builder para facilitar a criação da string que substituirá o
termo
              StringBuilder significado = new StringBuilder();
```

```
significado.append(termo.getSignficado()).append("(").append(termo.getTermo()).append(")");
            //substitui o termo na primeira vez que aparece
            oneparaString = oneparaString.replaceFirst(term, significado.toString());
            //atualiza o texto do run
            runs.get(i).setText(oneparaString, 0);
            //adiciono para uma lista de itens já substituidos
            replacedTerms.add(termo.getTermo());
          }
        }
      }
    }
   }
   //retorno o documento alterado
   return docChanged;
  } catch (Exception ex) {
   ex.printStackTrace();
   return null;
  }
}
```

Listagem 12 - Implementação do método replaceTerms

O método replaceSectionListOnDocx cria a lista das siglas baseado na seleção do usuário, substituindo a chave "#LISTASIGLA" pela nova lista criada. Assim como na substituição das siglas, também são percorridos os parágrafos e as *runs*, mas quando é encontrada a chave, a iteração é interrompida para que sejam removidas todas as *runs* do parágrafo e então sejam criadas novas com as siglas e seus significados. A Listagem 13 apresenta o código para essas operações.

```
private XWPFDocument replaceSectionListOnDocx(XWPFDocument document, String sectionName,
List<Sigla> terms) {
  //recebo por parametro o documento já alterado
  XWPFDocument docChanged = document;
  //crio um iterator para percorrer os paragrafos
  Iterator<XWPFParagraph> itPara = docChanged
     .getParagraphsIterator();
  //enquanto houver paragrafos para serem percorridos
  while (itPara.hasNext()) {
   //crio uma variavel com o paragrafo atual
   XWPFParagraph paragraph = itPara.next();
   //extraio do paragrafor as partes (runs)
   List<XWPFRun> runs = paragraph.getRuns();
   //Crio uma flag que indica se a chave foi encontrada
   boolean findSection = false;
   //crio uma variavel de controle para verificar qual run em que foi encontrada a chave
   int posRun = 0;
   //percorro os runs...
   for (int i = 0; i < runs.size(); i++) {
    //recupero o texto do run
    String oneparaString = runs.get(i).getText(
       runs.get(i).getTextPosition());
    ///verifico se a chave está nessa run
    if ((oneparaString != null) &&
       (oneparaString.contains(sectionName))) {
     //se estiver, marco que a chave foi encontrada
     findSection = true;
     //valorizo a variavel de controle indicando a posição da run dentro do paragrafo
     posRun = i;
     //finalizo o loop
     break;
    }
```

```
    }
    //se encontrar a seção, vai remove-la e adicionar novas com as siglas
    if (findSection) {
      //removo efetivamente a run
      paragraph.removeRun(posRun);
      //percorro a lista de termos selecionados e para cada um crio um run
      for (Sigla term : terms) {
        //crio um novo run
        XWPFRun tempRun = paragraph.createRun();
        //crio um string builder para facilitar a concatenação do texto
        StringBuilder termText = new StringBuilder();
        //concateno a sigla, duas tabulações e o significado

termText.append(term.getSigla().toUpperCase()).append("\t\t").append(term.getSignificado());
        //seto o texto do run
        tempRun.setText(termText.toString());
        //se o termo for estrangeiro, o texto ficará em italico
        boolean isItalic = term.getLinguaid().getEstrangeira() == 1;
        tempRun.setItalic(isItalic);
        //adiciono uma quebra de linha
        tempRun.addCarriageReturn();
      }
    }
  }
  //retorno o documento com a lista das siglas
  return docChanged;
}
```

Listagem 13 - Implementação do método replaceSectionListOnDocx

c) Métodos de pesquisa em banco de dados

Para a implementação do aplicativo foi utilizada apenas uma implementação relacionada à pesquisa em banco de dados. Houve a necessidade de pesquisar todas as siglas extraídas do texto de uma só vez, utilizando o operador *in*.

Para isso o Spring Data facilita muito na criação de consultas, sendo necessário criar as assinaturas dos métodos nas interfaces de *repository* e *service*, implementar o método de pesquisa na classe *repository* e chamar o método da classe *repository* na classe *service*.

Utilizando a interface *CriteriaBuilder* é possível o operador *in* para a coluna ou as colunas que forem necessárias. Para adicionar os itens a cláusula *where* basta apenas chamar o operador *in* e método *value* percorrendo o *set* de objetos e depois adicioná-lo ao *where* chamando o método *and* passando por parâmetro o operador.

Na Listagem 14 está detalhado o método *findBySiglaIn* que é a implementação do método de pesquisa na classe *repository* chamada *SiglaRepositoryImpl*.

```
@Override
public List<Sigla> findBySiglaIn(Set<String> termos) {
  //Inicializo os objetos usados para criar a query
  initialize();
  //crio um objeto Path, para indicar uma das colunas do banco
  Path<String> columnSigla = root.get("sigla");
  //crio a query baseada na tabela Sigla, ordenando em ordem alfabetica atraves da coluna
sigla
  query.select(root).orderBy(criterio.asc(columnSigla));
  //Crio um predicate In, utilizado para criar o where
```

```
  CriteriaBuilder.In<Object> in = criterio.in(columnSigla);
  //percorro a lista dos termos
  for(String termo : termos){
    //adiciono a lista in do where o termo
    in.value(termo);
  }
  //crio a clasula where siglas.sigla in (termo1, termo2...termoN)
  and(in);
  //retorno os significados encontrados.
  return entityManager.createQuery(query).getResultList();
}
```

Listagem 14 - Implementação do método findBySiglaIn

5 CONCLUSÃO

Os objetivos pretendidos com a realização do trabalho representados pela identificação de siglas em um texto, geração da lista de siglas e inclusão do significado de uma sigla na sua primeira ocorrência no texto foram alcançados. Em decorrência de restrições de própria API Apache POI, somente são utilizados arquivos no formato .docx.

Para a realização deste trabalho, além de utilizar diversos conhecimentos obtidos no decorrer das aulas do curso de especialização, especificamente das aulas de interfaces ricas e aplicações para Internet, foram necessárias, também, muitas pesquisas e testes em torno do funcionamento da API Apache POI. Os fóruns de desenvolvimento Java como *Stackoverflow*, *Code Ranch*, *Primefaces*, *Spring* e GUJ foram bastante úteis para auxílio na resolução de dúvidas e dificuldades com as tecnologias.

As maiores dificuldades giraram em torno de como manipular os arquivos utilizando a API, pois há pouquíssima documentação oficial por parte da Apache. Apenas alguns exemplos mais simples são encontrados. Foram, assim, necessárias muitas implementações de teste para se chegar ao estágio final de desenvolvimento.

Outra dificuldade encontrada está relacionada ao formato padrão de documento utilizado. Devido ao formato *.doc* ser um arquivo binário, ao fazer as substituições o arquivo perdia a formatação. No formato *docx* a formatação não era perdida. E como uma solução não foi encontrada, optou-se pelo uso do formato .docx.

Como próximo passo, serão feitos testes com outra API de manipulação de arquivos *Microsoft Open XML* chamada *docx4j*. Ela trabalha exclusivamente com a manipulação de arquivos *DOCX*, *PPTX* e *XLSX*.

Contudo, as dificuldades para implementar soluções baseadas em uma *API* de terceiros só foram superadas pelo aprofundamento do conhecimento na mesma antes de iniciar o desenvolvimento, aliado ao estudo de várias formas de utilizá-la com base em problemas e respectivas soluções reportadas em fóruns por outros desenvolvedores.

De acordo com as premissas de desenvolvimento definidas no escopo do projeto, alcançou-se com sucesso o objetivo de ler, extrair, manipular e devolver um arquivo atualizado para o usuário de forma simples e intuitiva e ao mesmo tempo gratuita devido a utilização de *frameworks* e *APIs* livres.

Como implementações futuras para complementar esse trabalho, destacam-se a possibilidade de leitura de documentos no formato .doc e a inclusão de outras padronizações para a identificação de siglas no texto.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6021. Informação e documentação. Publicação periódica científica impressa – Apresentação.** ABNT: Rio de Janeiro, maio 2003a.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6022: Informação e documentação: Artigo em publicação periódica científica impressa - Apresentação**. ABNT: Rio de Janeiro, maio de 2003b.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6029: Informação e documentação — Livros e folhetos — Apresentação**. ABNT: Rio de Janeiro, 2006.

APACHE POI. Disponível em: <http://mvnrepository.com/artifact/org.apache.poi>. Acesso em: 20 set. 2015.

ASWINI, Vaidya; LAVANYA, Sri K. **Pattern discovery for text mining**. International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC), 2014, p. 412-416.

AZEVEDO, Breno Fabrício Terra; BEHAR, Patricia Alejandra; REATEGUI, Eliseo Berni. Aplicação da mineração de textos em fóruns de discussão. ***Novas Tecnologias na Educação,*** CINTED-UFRGS *v*. 8, n. 3, dezembro, 2010, p. 1-10.

FACEMA. **Catálogo da ABNT: NBRs vigentes: trabalhos acadêmicos científicos.** Organizado por Conceição Boavista. Assessoria Técnico-Científica. Caxias, MA: FACEMA, 2013. Disponível em: <>. Acesso em: 30 set. 2015.

FELDMAN, Ronen; SANGER, James. **The text mining handbook: advanced approaches in analyzing unstructured data**. MA: Cambridge University Press, 2007.

HOVY, Eduard. **Automated text summarization,** 2005, Capítulo 32. disponível em: <http://www.isi.edu/natural-language/people/hovy/papers/05Handbook-Summ-hovy.pdf>. Acesso em: 10 mar. 2015.

LOH, Stanely. **Abordagem baseada em conceitos para descoberta de conhecimento em textos**. PhD thesis, Universidade Federal do Rio Grande do Sul, Instituto de Informática, 2001. Disponível em: < http://www.lume.ufrgs.br/handle/10183/1849>. Acesso em: 30 mar. 2015.

MEZZALIRA, Cezar A. **Sistema web para consulta e elaboração de listas de siglas, abreviaturas e acrônimos**. Trabalho de conclusão de Curso. Curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. 2013.

MORAIS, Edison Andrade Martins; AMBRÓSIO, Ana Paula L. **Mineração de textos**. Relatório técnico, 2005. Disponível em: <http://www.portal.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_005-07.pdf>. Acesso em: 30 mar. 2015.

NASA, Divya. **Text mining techniques: a survey.** International Journal of Advanced Research in Computer Science and Software Engineering, v. 2, n. 4, April 2012.

PURETSKIY, Andrey A.; SHUTT, Gregory L.; BERRY, Michael W**.** Survey of text visualization techniques. In: BERRY, Michael; KOGAN, Jacob. **Text mining**: applications and theory. John Wiley & Sons Ltd, 2010, p. 107-127.

SUBBAIAH, Suresh. **Extracting knowledge using probabilistic classifier for text** Mining. International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013, p. 440 – 442.

SUKANYA, Manna; BIRUNTHA, Sujeevan. **Techniques on text mining**. 2012 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), p. 269 - 271, 2012.

TAN, Ah-hwee. **Text mining: the state of the art and the challenges**. In: Workshop on Knowledge Discovery from Advanced Databases (PKDAD'99), 1999. Beijing. Proceedings. Beijing: 1999, p.71-76.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ. **Normas para elaboração de trabalhos acadêmicos**. Curitiba: UTFPR, 2008.

APÊNDICE A - TESTES E ETAPAS DE EXECUÇÃO

Para efetuar os testes na rotina de leitura de documentos, foram utilizadas algumas amostras de texto. A seguir está a apresentação por meio de telas obtidas da execução do aplicativo do teste realizado com o trabalho de conclusão da graduação (MEZZALIRA, 2013) que serviu de base para este trabalho.

O conteúdo desse texto é composto de capa, contra capa, sumário, listagens, seção para criação da lista de siglas e parágrafos nos quais estão as siglas de acordo com as premissas definidas no escopo do sistema.

Na Figura 1 é possível verificar a parte do documento de testes que tem destacada a chave #LISTASIGLA. Essa chave determina a localização das siglas encontradas no texto.

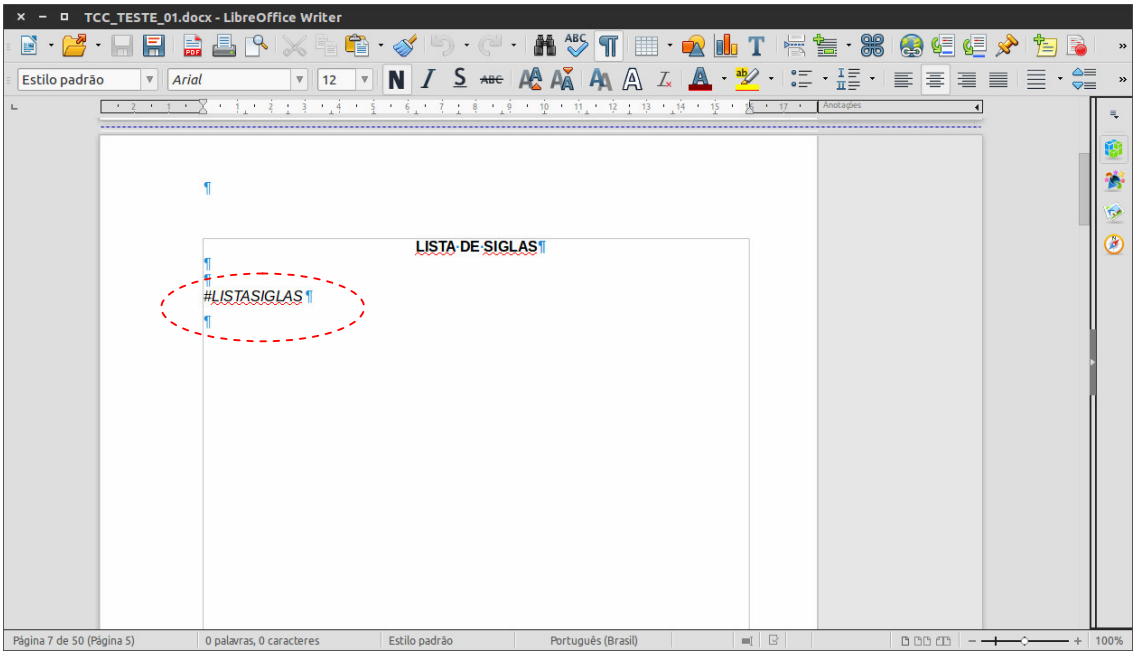

Figura 1 – Chave de substituição #LISTASIGLA

Na Figura 2 são mostrados, em destaque do tipo marca texto, alguns parágrafos que possuem siglas sem o seu significado, ou seja, a sigla aparece entre parênteses no parágrafo.

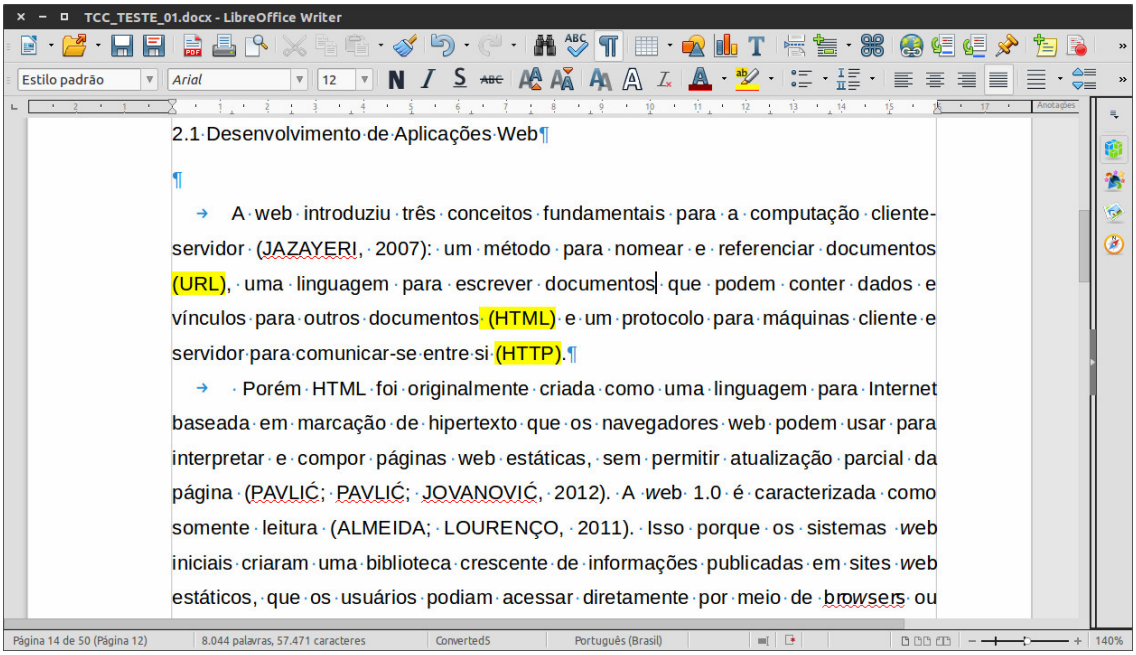

Figura 2 – Parágrafos com as siglas sem significado

Tendo um texto base, progressivamente foram desenvolvidas as seguintes funções:

- leitura de um arquivo docx;
- extração das siglas;
- validações para determinar se a palavra é sigla;
- pesquisa dos significados das siglas extraídas.
- manipulação de arquivo docx;
- disponibilização para *download*;

Em cada uma dessas funções, o mesmo texto foi aplicado, para que se mantivesse uma homogeneidade no resultado final, obtido apenas quando o arquivo é alterado e disponibilizado para *download*.

A primeira função testada foi a de extração de siglas. Essa rotina depende do *upload* do arquivo que é feito pelo usuário. O usuário abre a rotina de leitura de documentos (Figura 3), clica no botão “Selecionar Arquivo...” (Figura 4), escolhe e abre o arquivo (Figura 5) e então clica no botão “Extrair siglas...” (Figura 6).

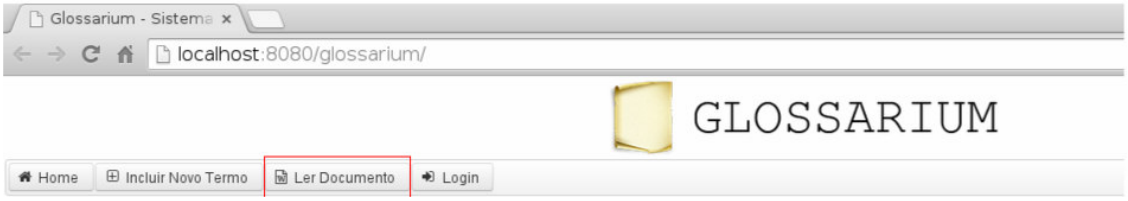

Figura 3 – Acesso a rotina de ler documento

Figura 4 – Botão para selecionar o arquivo

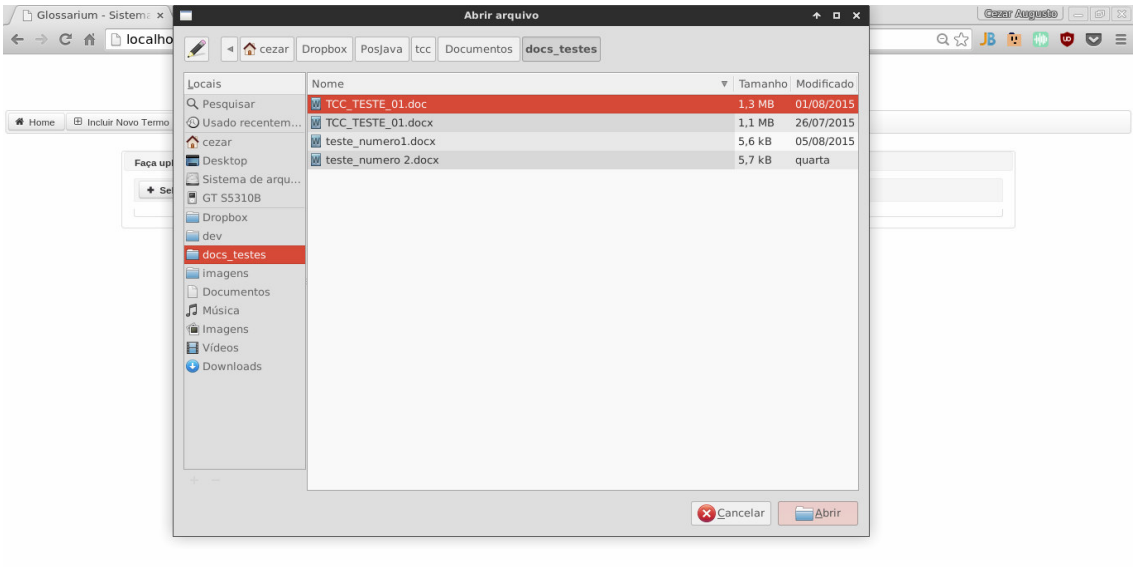

Figura 5 – Escolha do arquivo de texto

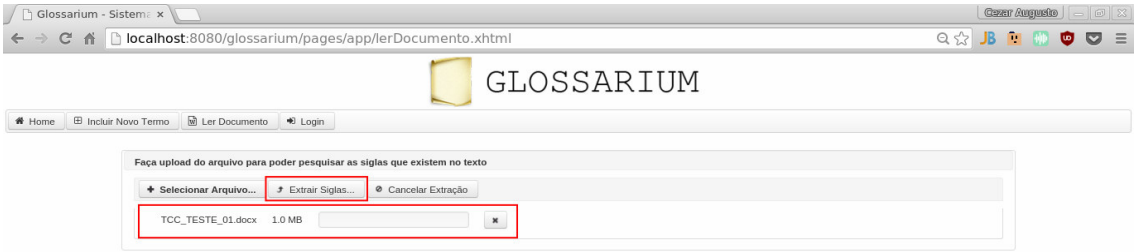

Figura 6 – Botão Extrair Siglas

Ao clicar em “Extrair Siglas...”, o sistema carrega o arquivo para o servidor, cria uma cópia e então executa os processos de leitura do conteúdo do texto, percorrendo parágrafo por parágrafo, palavra por palavra e letra por letra, para validar se a palavra é ou não uma sigla. O sistema retorna, então, uma lista das siglas (se existirem), conforme mostrado na Figura 7.

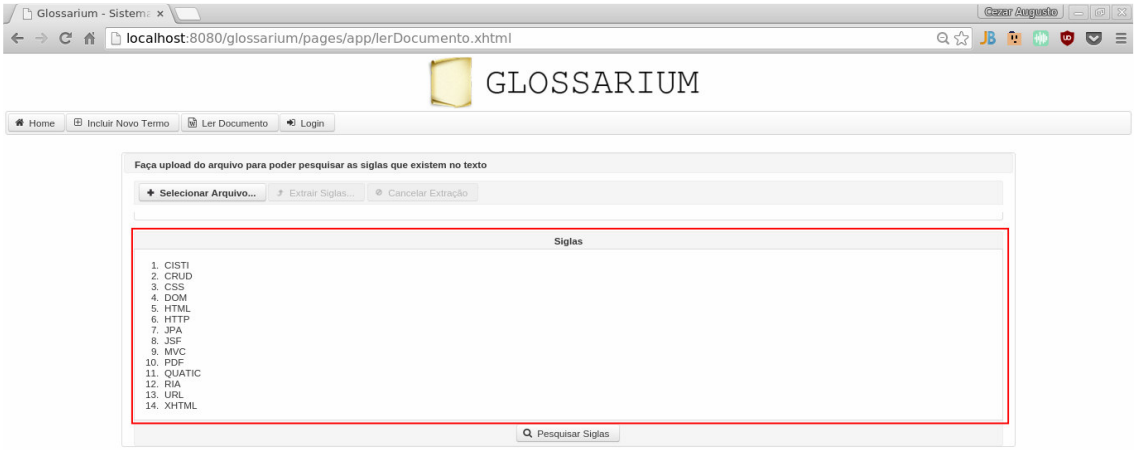

Figura 7 – Lista das siglas extraídas.

Na sequência, o usuário clica no botão “Pesquisar Siglas”, para que seja feita uma busca no banco de dados do aplicativo, visando localizar se existem significados para as siglas listadas. A Figura 8 destaca a localização do botão, o qual está logo abaixo da lista das siglas extraídas.

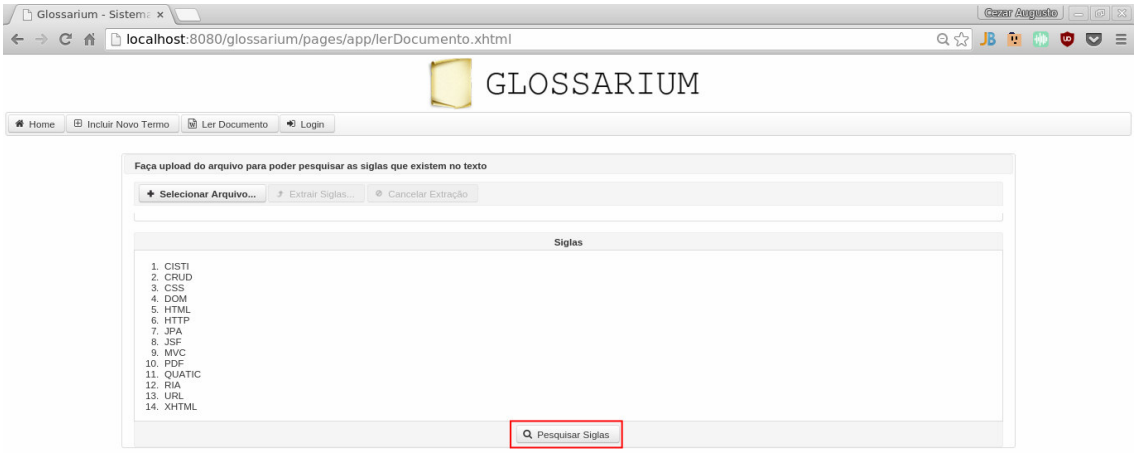

Figura 8 – Botão pesquisar siglas

O resultado da pesquisa dos significados é mostrado em uma lista de seleção, pois o usuário tem a possibilidade de selecionar quais os significados corretos das siglas, caso sejam apresentados mais de um significado para mesma sigla, conforme mostra a Figura 9.

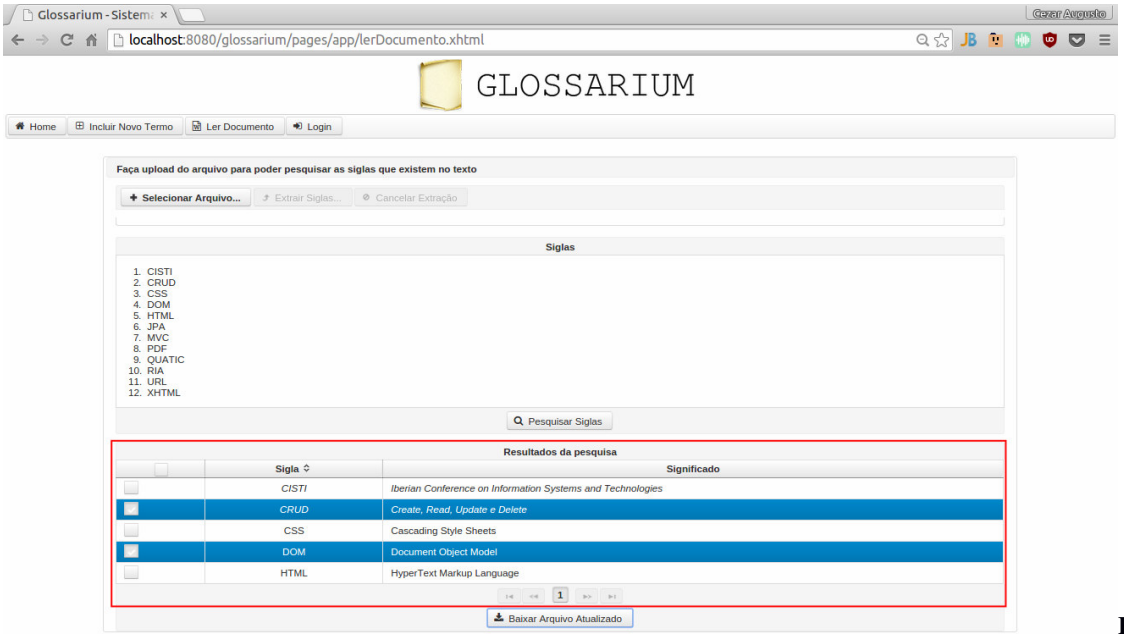

Figura 9 – Lista de significados para seleção

Com os significados selecionados, ao solicitar para baixar o arquivo, as siglas são substituídas pelo seu significado acrescido na sequência pela sigla, e também é criada a lista de siglas no lugar da chave "#LISTASIGLA", baseando-se nos significados selecionados.

A Figura 10 demonstra o mesmo parágrafo que foi mostrado acima com as siglas alteradas.

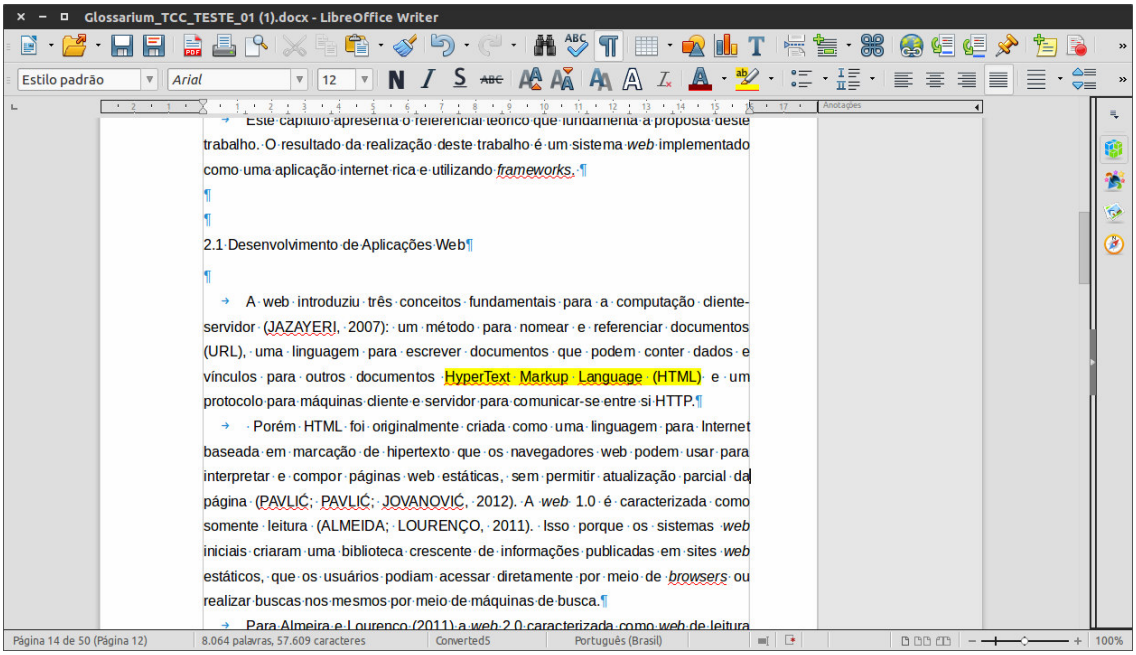

Figura 10 – Trecho do texto de testes com alteração

A Figura 11 apresenta a lista de siglas criada a partir da seleção dos significados das siglas.

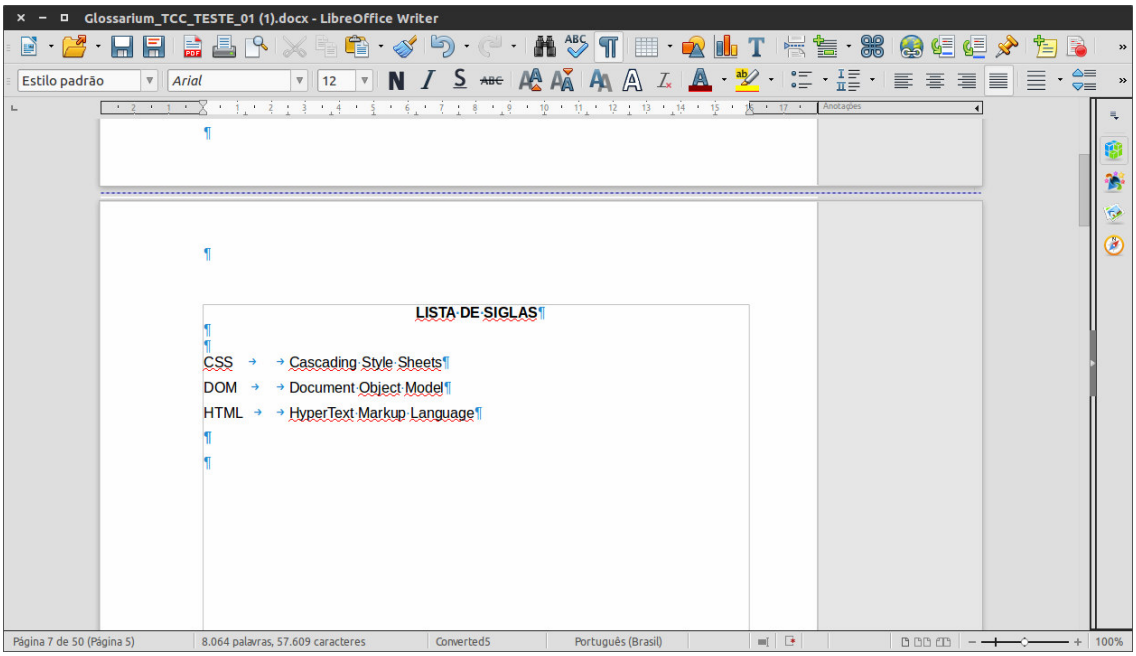

Figura 11 – Lista das siglas criada através do aplicativo