UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CEZAR AUGUSTO MEZZALIRA

SISTEMA WEB PARA CONSULTA E ELABORAÇÃO DE LISTAS DE SIGLAS, ABREVIATURAS E ACRÔNIMOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2013

CEZAR AUGUSTO MEZZALIRA

SISTEMA WEB PARA CONSULTA E ELABORAÇÃO DE LISTAS DE SIGLAS, ABREVIATURAS E ACRÔNIMOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2013

ATA Nº: **226**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **CEZAR AUGUSTO MEZZALIRA**.

Às 14:55 hrs do dia 20 de fevereiro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Robison Cris Brito (Convidado) e Luís Cassiano Goularte Rista (Convidado), para avaliar o Trabalho de Diplomação do aluno Cezar Augusto Mezzalira, matrícula 1116622, sob o título **Sistema Web para Consulta e Lista de Siglas, Abraviaturas e Acrônimos**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 15:30 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Robison Cris Brito, M.Sc.
Convidado

Prof. Luís Cassiano Goularte Rista, M.Sc.
Convidado

Eliane Maria de Bortoli Fávero, M.Sc
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

MEZZALIRA, Cezar Augusto. Sistema web para consulta e elaboração de listas de siglas, abreviaturas e acrônimos. 2013. 47 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2013.

Os trabalhos acadêmicos, como os 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. O uso de um aplicativo computacional para consulta de significado dos termos e geração dessas listas contribui para reduzir o tempo de elaboração das mesmas e o trabalho de conferência se os termos estão descritos no seu primeiro uso no texto. Assim, o resultado deste trabalho é o desenvolvimento de um sistema *web* para o armazenamento desses termos, a consulta de significados e a composição de listas. Optou-se pelo desenvolvimento de um aplicativo *web* para facilitar o acesso, seja para cadastro, para consulta de significado de termos ou para a composição de listas. Considerando que um mesmo termo pode ter significados distintos, o sistema permite a indicação de área e subárea para a categorização dos termos. Essa classificação facilita a escolha do significado adequado para o respectivo termo em decorrência da área do trabalho. O desenvolvimento foi realizado utilizando a linguagem Java com a interface implementada com componentes da biblioteca PrimeFaces.

Palavras-chave: Sistema web. Cadastro de siglas. Linguagem Java. PrimeFaces.

ABSTRACT

MEZZALIRA, Cezar Augusto. Web system to search and compose lists of acronyms and abbreviations. 2013. 47 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2013.

The academic works, such as internship reports, monographs and thesis, ordinarily have lists of acronyms and abbreviations. These lists are composed of all terms (words) of this nature that are in the text and their respective description (meaning). The use of software to register acronyms and abbreviations reduce the time to elaborate these lists and the time to verify if the terms are described in the first use in the text. Thus, the result of this academic work is the development of a web system to storage this terms and its description. If this software is a web application, the terms registered can be used to compose lists of distinct academics works and by many people (students and teachers). Considering that the same term (acronym or abbreviation) can have distinct meaning, the system developed as a result of this work enable the indication of field and subfield, favoring the adoption of the meaning more appropriated to the term in the context of the area. The development was made using Java language with the interface created with components of PrimeFaces library.

Keywords: Web system. Abbreviations and Acronyms. Java Language.

LISTA DE FIGURAS

FIGURA 1 – PADRÃO DE PROJETOS MVC 16
FIGURA 2 – DIAGRAMA DE CASOS DE USO DO SISTEMA 24
FIGURA 3 – DIAGRAMA DE ENTIDADES E RELACIONAMENTOS DO BANCO DE DADOS 28
FIGURA 4 – DIAGRAMA DE ENTIDADES E RELACIONAMENTOS DO BANCO DE DADOS 30
FIGURA 5 – PASSOS PARA GERAÇÃO DE UMA LISTA DE SIGLAS 31
FIGURA 6 – RELATÓRIO DE SIGLAS 32
FIGURA 7 – TELA INICIAL DOS CADASTROS 33
FIGURA 8 – TELA DE TERMOS NÃO APROVADOS 33
FIGURA 9 – FORMULÁRIO DE CADASTRO 34

LISTA DE QUADROS

QUADRO 1 – DIFERENÇAS ENTRE WEB 2.0 E WEB 3.0 14
QUADRO 2 – FERRAMENTAS E TECNOLOGIAS 18
QUADRO 3 – ETAPAS E ITERAÇÕES 20
QUADRO 4 – OPERAÇÃO INCLUIR DADOS DOS CASOS DE USO DE CADASTRO 25
QUADRO 5 – OPERAÇÃO CONSULTAR DADOS DOS CASOS DE USO DE CADASTRO 25
QUADRO 6 – OPERAÇÃO ALTERAR DADOS DOS CASOS DE USO DE CADASTRO 26
QUADRO 7 – OPERAÇÃO EXCLUIR DADOS DOS CASOS DE USO DE CADASTRO 27
QUADRO 8 – TABELA SIGLAS 28
QUADRO 9 – TABELA ÁREAS 29
QUADRO 10 – TABELA SUBÁREAS 29
QUADRO 11 – TABELA TIPOS DE SIGLAS 29
QUADRO 12 – TABELA LÍNGUAS 29
QUADRO 13 – TABELA USUÁRIOS 29

LISTAGENS DE CÓDIGO

LISTAGEM 1 - CLASSE ENTIDADE 35
LISTAGEM 2 - INTERFACE SIGLADATA 35
LISTAGEM 3 - INTERFACE SERVICE 36
LISTAGEM 4 - INTERFACE REPOSITORY 36
LISTAGEM 5 - CLASSE REPOSITORYIMPL 37
LISTAGEM 6 - CLASSE SERVICE IMPL 37
LISTAGEM 7 - MÉTODO PREPROCESSORSAVE 38
LISTAGEM 8 - MÉTODOS DO REPOSITORY 38
LISTAGEM 9 - CADASTRO DE USUÁRIOS 40
LISTAGEM 10 - VALIDAÇÃO PARA COLUNAS DE DATATABLE 40
LISTAGEM 11 - CABEÇALHO DA CLASSE CONTROLLER 40
LISTAGEM 12 - MÉTODO DE PESQUISA 41
LISTAGEM 13 - MÉTODO PARA EMISSÃO DE RELETÓRIOS 42
LISTAGEM 14 - MÉTODO ADICIONARNALISTA 42
LISTAGEM 15 - CLASSE SIGLACOMPARATOR 42

LISTA DE SIGLAS

Ajax	*Asynchronous Javascript and XML*
CDDL	*Common Development and Distribution License*
CSS	Cascading Style Sheet
CRUD	*Create Retrieve Update, Delete*
DOM	*Document Object Model*
GNU	*General Public License*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
JPA	*Java Persistence* API
JSF	*Java Server Faces*
MVC	*Model-View-Controller*
OWL	*Web Ontology Language*
PDF	*Portable Document Format*
POM	*Project Object Model*
RDF	*Resource Description Framework*
RIA	*Rich Internet Application*
URL	*Uniform Resorce Locator*
XHTML	*Extensible HyperText Markup Language*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 10
1.1 CONSIDERAÇÕES INICIAIS 10
1.2 OBJETIVOS 11
1.2.1 Objetivo Geral 11
1.2.2 Objetivos Específicos 11
1.3 JUSTIFICATIVA 11
1.4 ESTRUTURA DO TRABALHO 12
2 APLICAÇÕES PARA WEB 13
2.1 Desenvolvimento de Aplicações Web 13
2.2 Rich Internet Application 15
2.3 Desenvolvimento *Web* com Java 16
3 MATERIAIS E MÉTODOS 18
3.1 MATERIAIS 18
3.2 MÉTODOS 19
4 RESULTADO 22
4.1 ESCOPO DO SISTEMA 22
4.2 MODELAGEM DO SISTEMA 23
4.3 APRESENTAÇÃO DO SISTEMA 29
4.4 IMPLEMENTAÇÃO DO SISTEMA 35
4.5 DISCUSSÃO 43
5 CONCLUSÃO 44
REFERÊNCIAS 46

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa deste trabalho de conclusão de curso.

1.1 CONSIDERAÇÕES INICIAIS

No texto de determinados trabalhos acadêmicos, como relatórios de estágio e monografias de conclusão de curso, as siglas, abreviaturas e acrônimos devem ser descritas em seu primeiro uso e devem ser geradas listas agrupadas ou separadas por tipo, contendo o termo e a respectiva descrição. Caso a sigla, abreviatura ou acrônimo, genericamente denominado termo neste texto, seja de língua estrangeira deve ser grafado em itálico, no texto e na listagem.

Abreviatura é uma parte da palavra representando a palavra como um todo, por exemplo, tec. para representar a palavra tecnologia. Siglas são formas de abreviaturas compostas pelas letras iniciais de palavras e de expressões, como por exemplo, 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).

Compor essas listas em trabalhos acadêmicos é uma atividade que despende tempo. Um aplicativo computacional que permita cadastrar termos, selecionar termos de uma listagem e gerar a dos mesmos facilita a realização dessa atividade. Implementar um aplicativo que permita a elaboração dessas listagens e mesmo a consulta de significados de termos é o resultado da realização deste trabalho.

O aplicativo foi desenvolvido para *web* como forma de facilitar o acesso ao mesmo. Por meio desse aplicativo é possível realizar a consulta do significado de termos e a elaboração de listagens. Um usuário com permissão somente de consulta e composição de listas pode sugerir termos e as respectivas descrições para serem armazenados. Se essa sugestão é realizada por usuários que possuem apenas o perfil de consulta, o cadastro deverá ser validado por usuários com permissões para

edição para que o termo e o respectivo significado sejam incluídos no banco de dados.

O aplicativo desenvolvido possibilita a categorização dos termos em áreas e subáreas. Uma mesma sigla, por exemplo, pode possuir significados distintos para áreas distintas. A indicação se o termo é de origem estrangeira também é realizada para que possa ser grafado em itálico na elaboração da listagem.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Implementar um sistema *web* para cadastro, consulta e geração de listagens de siglas, abreviaturas e acrônimos.

1.2.2 Objetivos Específicos

. Facilitar a geração de listas de termos.

. Possibilitar consulta rápida ao significado de termos.

. Fornecer uma ferramenta *web* para auxiliar na elaboração de listagens termos para uso por acadêmicos e professores.

1.3 JUSTIFICATIVA

Um aplicativo para cadastro de termos como siglas, abreviaturas e acrônimos e a geração de listagens desses termos para constar em trabalhos acadêmicos pode reduzir o tempo de geração dessas listas, agilizar a localização da descrição do termo, facilitar a sua descrição no seu primeiro uso e auxiliar na verificação de consistência entre a listagem e a existência dos termos no texto.

Um aplicativo *web* torna o seu acesso mais facilitado pelo fato de o mesmo ser realizado por meio da Internet. Por ser um aplicativo que pode ter amplo uso no meio acadêmico considerou-se importante atender aspectos de usabilidade na elaboração da interface de interação com o aplicativo. Esses aspectos estiveram centrados em fornecer uma interface intuitiva com a composição da listagem de termos em uma única tela e a não repetição de termos na listagem que está sendo composta, mesmo que o usuário escolha o mesmo termo da mesma área e subárea mais de uma vez.

A escolha da tecnologia Java complementada por Java ServerFaces, Spring o *framework* PrimeFaces teve como fundamentação os recursos que as mesmas oferecem para o desenvolvimento de uma interface com recursos semelhantes às aplicações *desktop*.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. Este é o primeiro e apresenta a introdução do trabalho. O Capítulo 2 contém o referencial teórico que se refere ao desenvolvimento de aplicações para *web*. No Capítulo 3 estão os materiais utilizados para a modelagem e a implementação do aplicativo e o método, com as principais atividades realizadas para a modelagem e a implementação. O Capítulo 4 apresenta o sistema implementado, abrangendo a sua modelagem e exemplos dos códigos desenvolvidos. Por fim, no Capítulo 5, está a conclusão com as considerações finais.

2 APLICAÇÕES PARA WEB

Este capítulo apresenta o referencial teórico que fundamenta a proposta deste trabalho.

2.1 Desenvolvimento de Aplicações Web

A *web* introduziu três conceitos fundamentais para a computação cliente-servidor (JAZAYERI, 2007): um método para nomear e referenciar documentos (*Uniform Resorce Locator* - URL), uma linguagem para escrever documentos que podem conter dados e vínculos para outros documentos (*HyperText Markup Language* - HTML) e um protocolo para máquinas cliente e servidor para comunicar-se entre si (*HyperText Transfer Protocol* - HTTP).

Porém, HTML foi originalmente criada como uma linguagem para Internet baseada em marcação de hipertexto que os navegadores *web* podem usar para interpretar e compor páginas *web* estáticas, sem permitir atualização parcial da página (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012). A *web* 1.0 é caracterizada como somente leitura (ALMEIDA; LOURENÇO, 2011). Os sistemas *web* criaram uma biblioteca crescente de informações publicadas em sites *web* estáticos, que os usuários podiam acessar diretamente por meio de *browsers* ou realizar buscas nos mesmos por meio de máquinas de busca.

Para Almeira e Lourenço (2011) a *web* 2.0 é caracterizada como *web* de leitura e escrita ou *web* social que facilitou o entendimento e uso da internet pela facilidade de desenvolvimento de conteúdo. As pessoas podiam compartilhar suas ideias por meio de *blogs*, *wikis* e sites de redes sociais. E para as empresas, agências do governo e outras organizações, a *web* 2.0 criou um movo modelo de negócio para usuários internos e externos (HOGG et al, 2006).

Para criar dinamismo com páginas *web* baseadas em HTML, que são naturalmente estáticas, é necessário criar o servidor (com uma linguagem de programação para gerar HTML) e criar o cliente usando tecnologias como Ajax (*Asynchronous Javascript and XML - eXtensible Markup Language)*, por exemplo,

para alterar dinamicamente partes da página e Cascading Style Sheet (CSS) para definir a interface. A atualização parcial de uma página pode ocorrer com o uso de JavaScript, por exemplo. Parcial se refere que apenas a porção da página que possui dados que são alterados com a interação do usuário ou decorrentes de operações realizadas no servidor deve ser atualizada (recarregada).

Pavlić, Pavlić e Jovanović (2012) indicam que para resolver este problema e não ter a necessidade de usar tecnologias diferentes, um tipo especial de plataforma *Rich Internet Application* (RIA) e uma abordagem têm sido desenvolvidos. A visão por trás dessa nova plataforma é um servidor que permite a execução de aplicações na Internet e em múltiplas plataformas. No lado cliente há um programa que faz a descrição da interface gráfica com o usuário e o tratamento de eventos. Aplicações que trabalham dessa forma são conhecidas como RIA. A tecnologia RIA representa um ambiente de desenvolvimento no qual a aplicação é programada pelo servidor e a mesma aplicação pode ser executada no cliente sem qualquer programação adicional para o lado cliente.

Em termos de disponibilidade de conteúdo, Almeida e Lourenço (2011), caracterizam a *web* 3.0 como a *web* inteligente e que resolverá os problemas relacionados à estrutura e organização da *web* 2.0. A *web* 3.0 será caracterizada como *web* semântica pelo uso de semântica para interpretar conteúdo e entregar conteúdo mais apropriado e relevante para o usuário. As principais diferenças entre *web* 2.0 e 3.0 são apresentadas no Quadro 1.

	Web 2.0	**Web 3.0**
Tarefa principal	Foca no poder da comunidade para criar conteúdo dinâmico e tecnologias de interação.	Dados, dispositivos e pessoas vinculados por meio da *web*.
Vínculos (linking)	Conteúdos isolados que inibem interoperabilidade.	Dados e dispositivos mais facilmente vinculados e em novas formas.
Conteúdo	Conteúdo criado por pessoas e organizações.	Pessoas, organizações e máquinas criam conteúdo que pode ser reusado.
Tecnologia	Ajax.	RDF (*Resource Description Framework*) e OWL (*Web Ontology Language*).
Site web	Google, Facebook, Wikipedia, eBay, Youtube.	Dbpedia (http://dbpedia.org/About), sioc-projetc.org (http://sioc-project.org/).

Quadro 1 – Diferenças entre web 2.0 e web 3.0

Fonte: Breslin e Decker (2007).

A *web* 3.0 introduziu novas técnicas para organizar conteúdo e novas ferramentas que tornarão possível para aplicativos coletar, interpretar e usar dados de maneiras que elas podem adicionar significado e estrutura à informação. A *web* 3.0 também oferece ferramentas para melhor gerenciar o fluxo de informação e disponibilizar conteúdo para o usuário de forma mais rápida e rica (HENDLER, 2009).

Em termos de conteúdo a *web* 3.0 está voltada para a estrutura e a organização da informação. É a *web* semântica ou baseada em ontologia que tem como base uma maior capacidade de o software interpretar o conteúdo disponibilizado na *web*, fornecendo assim resultados de pesquisa mais objetivos e personalizados (SABINO, 2007). Ao passo que a *web* 2.0 pode-se dizer que é caracterizada pela melhoria de usabilidade de interface de aplicações *web* e provendo mais interatividade, dinamismo e experiência mais satisfatória ao usuário (AMALFITANO et al., 2010). Essas aplicações são conhecidas como *Rich Internet Application* que se desenvolveram utilizando técnicas e tecnologias da *web* 2.0, como Ajax (GARRET, 2005).

2.2 Rich Internet Application

Aplicações *web* com Ajax exploram uma combinação de tecnologias *web* (tais como *Extensible HyperText Markup Language* (XHTML), JavaScript, XML e XMLHttpRequest) obtendo uma interação rica com o usuário da aplicação. A interface do usuário de uma aplicação Ajax é implementada por uma ou mais páginas *web* que são compostas por componentes individuais, que podem ser independentemente atualizados, excluídos ou adicionados em tempo de execução (AMALFITANO et al, 2010). A manipulação dos componentes da página é realizada por uma máquina Ajax escrita em JavaScript que é carregada pelo navegador *web* no início da sessão, acessa os componentes das páginas por meio de interface *Document Object Model* (DOM) (DOM, 2013) e é responsável pela comunicação entre o servidor e o usuário (GARRET, 2005).

Usando Ajax ou outras abordagens de desenvolvimento de interface caracterizadas como rica torna as aplicações *web* similares às aplicações *desktop*

em termos de recursos de interatividade do usuário, dentre outros aspectos. Contudo, enquanto as técnicas de implementação melhoram a usabilidade das aplicações há vários fatores que impactam negativamente na qualidade das suas características internas, tais como compreensibilidade e análise (AMALFITANO et al, 2010). As RIAs são implementadas usando uma ampla variedade de *frameworks* (AJAX, 2013; BUCHNER; BÖTTCHER; STORCH, 2012). Os *frameworks* aceleram o desenvolvimento, mas dificultam a análise do código gerado e a interação entre as diferentes tecnologias também pode ser dificultada (AMALFITANO et al, 2010).

2.3 Desenvolvimento *Web* com Java

As aplicações *web* baseadas na plataforma Java Enterprise Edition usando o padrão de projetos *Model-View-Controller* (MVC) são organizadas em três componentes arquiteturais (GAMMA, 2000): lógica de apresentação, lógica de controle e lógica de negócio. O padrão de projetos MVC tem permanecido um padrão fundamental, mesmo após grande evolução de arquiteturas de aplicações interativas (PRAJAPATI; DABHI, 2009).

O padrão de projetos MVC, como mostra a Figura 1, consiste de três tipos de objetos: modelo, visão e controle.

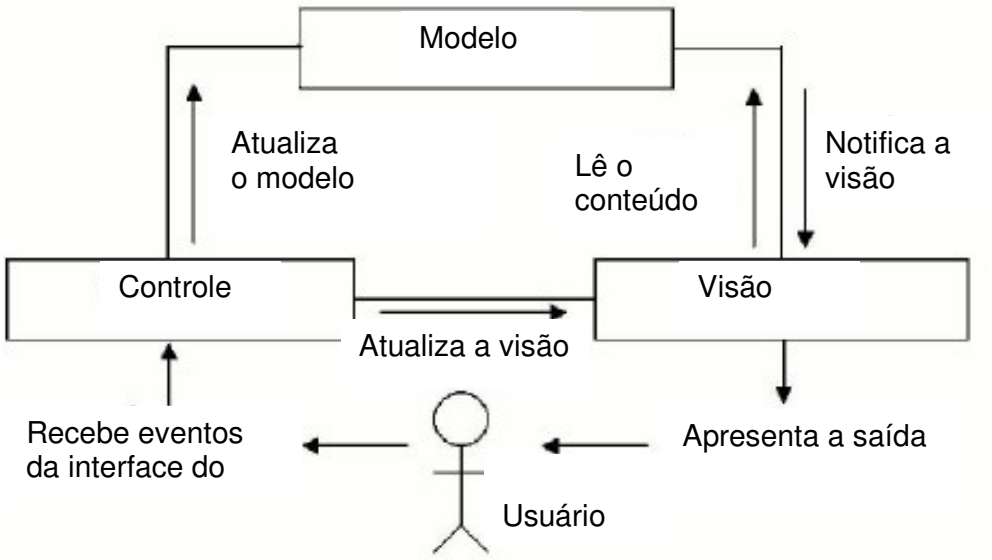

Figura 1 – Padrão de projetos MVC

Fonte: traduzido de Prajapati e Dabhi (2009, p. 1665).

Os objetos que compõem o padrão MVC possuem três responsabilidades básicas: entidades (dados), fronteira (apresentação) e controle (comportamento), respectivamente. O modelo encapsula os dados da aplicação e a lógica de negócio. A visão manipula a representação dos dados da aplicação e a interface visual com o usuário. O controle manipula a interação do usuário com a aplicação.

O padrão de projetos MVC separa visões e modelos por estabelecer um protocolo de aceitar/notificar entre eles. Um objeto da visão deve garantir que sua aparência reflita o estado do modelo. O objeto modelo é independente dos objetos de controle e da visão, assim é possível ter múltiplas visões (apresentações) do mesmo modelo (dados). Todas as visões associadas podem subscrever (*subscribe*) com o modelo e o modelo pode notificá-los sobre mudanças de estados. Na interação do usuário com esse modelo, todos os eventos são capturados pelo objeto controle. O controle, então, decide se o evento está relacionado à mudança de estado do modelo ou à mudança de estado da visão.

3 MATERIAIS E MÉTODOS

Este capítulo apresenta os materiais (tecnologias e ferramentas) e o método utilizados na modelagem e na implementação do aplicativo *web* que é o resultado da realização deste trabalho.

3.1 MATERIAIS

O Quadro 2 apresenta as ferramentas e as tecnologias utilizadas na modelagem e implementação do sistema.

Tecnologia /Ferramenta	**Disponibiliade**	**Descrição de uso**
MySQL Workbench	http://www.mysql.com/products/workbench/	Modelagem do diagrama de entidades e relacionamentos do banco de dados.
Java	http://www.oracle.com	Linguagem de programação.
JavaServer Face	http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html https://javaserverfaces.java.net/	*Framework* para desenvolvimento da interface.
PrimeFaces	http://primefaces.org/	Biblioteca de componentes para desenvolvimento da interface.
IntelliJ IDEA 13	http://www.jetbrains.com/idea/	Ambiente de desenvolvimento para a linguagem Java.
GlassFish	https://glassfish.java.net/ http://docs.jelastic.com/pt/glassfish	Servidor de aplicação *web*.
MySQL	http://www.mysql.com	Gerenciador de banco de dados.
Hibernate	http://hibernate.org/	Mapeamento objeto relacional com o banco.
Spring	http://spring.io/	*Framework* de inversão de controle e injeção de dependências.
Apache Maven	http://maven.apache.org/	Ferramenta de gerenciamento, construção e implantação de projetos.
JasperReport	http://www.jaspersoft.com/	Para implementação dos relatórios

Quadro 2 – Ferramentas e tecnologias

O PrimeFaces é uma biblioteca de componentes de código aberto para o Java ServerFaces. Esses componentes permitem criar interfaces para aplicações *web* que são caracterizadas como ricas. Esses componentes foram construídos para trabalhar com Ajax. O PrimeFaces também permite a aplicação de temas (*skins*) com o objetivo de mudar a aparência dos componentes de forma simples.

O GlassFish é um projeto de servidor de aplicação de código aberto iniciado pela Sun Microsystems para a plataforma Java e que é, atualmente, patrocinado pela Oracle Corporation. O GlassFish é um software gratuito e licenciado sob a *Common Development and Distribution License* (CDDL) e a *GNU General Public License* (GPL).

O Spring é um *framework* de aplicação de código-fonte aberto que visa facilitar o desenvolvimento de aplicações Java. O Spring é composto por um contêiner, um *framework* para gerenciar componentes e um conjunto de serviços de para interfaces de usuário, transações e persistência da *web*. Esse *framework* oferece diversos módulos que podem ser utilizados de acordo com as necessidades de cada projeto. Dentre esses módulos estão os voltados para desenvolvimento *web*, persistência, acesso remoto e programação orientada a aspectos.

O Maven é uma ferramenta para gerenciamento e automação de projetos em Java. Esse aplicativo é utilizado para gerenciar as versões do projeto e utiliza uma construção denominada *Project Object Model* (POM) que descreve o processo de construção de um projeto de *software*, suas dependências em outros módulos e componentes e a sua sequência de construção. O Maven possui tarefas pré-definidas que realizam funções como a compilação e o empacotamento de código e armazena características do projeto como: nome, desenvolvedores, repositórios de código fonte (versionamento de código fonte), dependências de bibliotecas externas, *plugins* que auxiliam no desenvolvimento e testes realizados. A implantação do artefato gerado também é auxiliada pelo Maven.

3.2 MÉTODOS

O processo de modelagem e implementação do aplicativo para composição de listas de siglas, abreviaturas e acrônimos teve como base o processo unificado

(BLAHA et al., 2006). Algumas iterações foram definidas, como apresentado no Quadro 3.

Iteração / Etapas	1ª iteração	2ª iteração	3ª iteração	4ª iteração
Requisitos	Definição dos requisitos básicos.	Refinamento e complemento dos requisitos.	Avaliação e ajuste dos requisitos.	Redefinição da forma de listagem dos termos
Análise e projeto	Modelagem dos requisitos. Elaboração do diagrama de entidades e relacionamentos.	Revisão do diagrama de entidades e relacionamentos, com realização de ajustes no mesmo.	Revisão e complementação da modelagem.	
Implementação	Estudo e entendimento das tecnologias.	Definição da interface padrão do aplicativo (telas). Implementação de cadastros básicos	Implementação das demais funcionalidades.	Desenvolvimento dos relatórios.
Testes	De código, realizados pelo autor deste trabalho.	De modelo de interface, realizado pela orientadora do trabalho.	De código, realizados pelo autor deste trabalho.	Das funcionalidades do sistema, realizados pela orientadora.

Quadro 3 – Etapas e iterações

A seguir estão descritas as etapas definidas para o desenvolvimento do aplicativo que são apresentadas no Quadro 3.

a) Levantamento de requisitos

O levantamento dos requisitos iniciou com a orientadora fornecendo a visão geral do aplicativo, incluindo os principais requisitos pretendidos e as funcionalidades e aplicabilidades do mesmo. Dessa visão e a partir de diversas reuniões realizadas foram extraídos os requisitos. Esses requisitos foram complementados à medida que as iterações ocorriam e que as funcionalidades eram discutidas e melhor entendidas. A elicitação dos requisitos gerou uma listagem contendo as funcionalidades pretendidas para o sistema e as políticas aplicadas a esses requisitos. Essas políticas definiram restrições e permissões de acesso, os requisitos funcionais e não funcionais vinculados a cada tipo de usuário e a forma de interação com a funcionalidade principal do sistema que é a composição de listas de termos.

b) Análise e projeto do sistema

Com base nos requisitos foram definidos os casos de uso do sistema. Esses casos de uso foram documentados gerando informações para a definição do diagrama de entidades e relacionamentos do banco de dados. Esse diagrama foi elaborado juntamente com a descrição das tabelas. Para cada tabela foram explicitados os seus campos e para cada campo indicado o tipo de dado, uma descrição do conteúdo do campo e a indicação se o mesmo é de preenchimento obrigatório.

c) Implementação

A implementação foi realizada utilizando a IDE (*Integrated Development Environment*) IntelliJ. Na primeira iteração as tecnologias foram estudadas com o objetivo de aprendizado e entendimento do melhor uso da IDE. Em seguida, os cadastros simples foram implementados visando familiarização com ambiente e com as tecnologias. Na segunda iteração, o objetivo principal foi experimentar e testar a melhor maneira de compor os formulários e disponibilizar as informações na tela. Um protótipo da tela para a composição das listas foi gerado. Na terceira iteração, os cadastros e as demais funcionalidades foram implementadas. E em uma última iteração foram desenvolvidos os relatórios do sistema, mais especificamente, a listagem dos termos selecionados.

d) Testes

Os testes foram realizados com o objetivo da verificação do código. Esses testes foram informais (realizados sem um plano de testes) e realizados pelo autor deste trabalho. Os testes estiveram centrados na identificação de erros de código e na verificação do atendimento às funcionalidades definidas para o sistema.

4 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho que é um sistema para cadastro, consulta e emissão de listagens de siglas, abreviaturas e acrônimos, denominados genericamente de termos.

4.1 ESCOPO DO SISTEMA

O sistema para composição de listas de termos (listas, abreviaturas e siglas) desenvolvido como resultado da realização deste trabalho de conclusão de curso visa oferecer uma forma prática para professores e alunos da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, consultar o significado de siglas, abreviaturas e acrônimos e compor listas com o termo e com o respectivo significado.

Um termo pode ter significados distintos. Assim, quando do cadastro de um termo é necessário informar uma área de conhecimento à qual o termo pertence. É, também, possível cadastrar significados distintos dentro de uma mesma área de conhecimento para um mesmo termo, tornando-se necessária a indicação de uma subárea.

O sistema será disponibilizado por meio do portal acadêmico da área de Informática do Câmpus Pato Branco. Além de consultar, os usuários podem sugerir termos e seus significados. Os termos sugeridos devem ser validados por um usuário com perfil de professor ou de administrador para que possam ser efetivamente incluídas no banco de dados.

O sistema possui três níveis de acesso: administrador, professor e padrão. O administrador possui acesso a todas as funcionalidades do sistema. O professor inclui, exclui, altera, consulta e valida os termos sugeridos. O usuário que desejar permissões de professor faz um pré-cadastramento no sistema informado *login* e senha. Esse cadastro é validado pelo usuário administrador. O usuário padrão não precisa estar cadastrado. Ele pode consultar o significado de termos, sugerir termos e respectivos significados e compor listas. A denominação “padrão” é apenas para

indicar um nome específico para um usuário sem a necessidade de cadastro no sistema acessar determinadas funcionalidades do mesmo.

4.2 MODELAGEM DO SISTEMA

Os requisitos funcionais levantados foram:

a) Cadastrar termos e seu significado, indicando a língua nacional (português) ou estrangeira. Os termos podem ser pré-cadastrados (sugeridos) por usuários com permissões de usuário padrão e somente são inseridas no banco de dados após serem validadas por um usuário professor.

b) Cadastrar áreas, por exemplo: Informática, Administração ou Física.

c) Compor uma lista de termos no formato:

Termo<marca de tabulação> significado

“marca de tabulação” indica alguma forma de manter a coluna à direita alinhada.

d) Atribuir permissão de inclusão e exclusão para usuários do tipo professor.

e) Permitir a consulta do significado de um determinado termo. Nesse caso trazer todos os significados do respectivo termo, indicando a área à qual pertencem.

Como requisitos não funcionais foram levantados os seguintes:

a) Um termo pode ter significados distintos em uma mesma área ou para áreas distintas. Permitir a indicação da área no cadastro do termo e a inserção de significados diferentes para um termo da mesma área.

b) Na lista, o significado dos termos em inglês (ou outra língua que não o português) deve estar em itálico.

c) O sistema deverá possuir usuários distintos. Um usuário de consulta (denominado “padrão”) pode compor listas e sugerir termos e seus respectivos significados. Um usuário professor que pode incluir termos e significados e validar (aceitar para inclusão no banco) termos sugeridos por usuários de consulta. E um usuário administrador que concede permissões de inclusão e validação de termos a usuários professor.

d) Um termo pode ser indicado por um usuário, informando o termo, o significado e *email* (opcional). Não é necessário validar o *email* indicado. Contudo,

informar que é preciso indicar um *email* para retorno de que o termo foi cadastrado, mas que esse não é obrigatório.

e) Armazenar o termo sem pontuação.

f) No momento de geração das listas o usuário pode indicar se quer gerar listagens separadas por tipo de termo (siglas, acrônimos e abreviaturas) ou uma única lista. As listagens geradas devem estar em ordem alfabética.

Os requisitos funcionais do sistema foram representados sob a forma de um diagrama de casos de uso. Esse diagrama é apresentado na Figura 2.

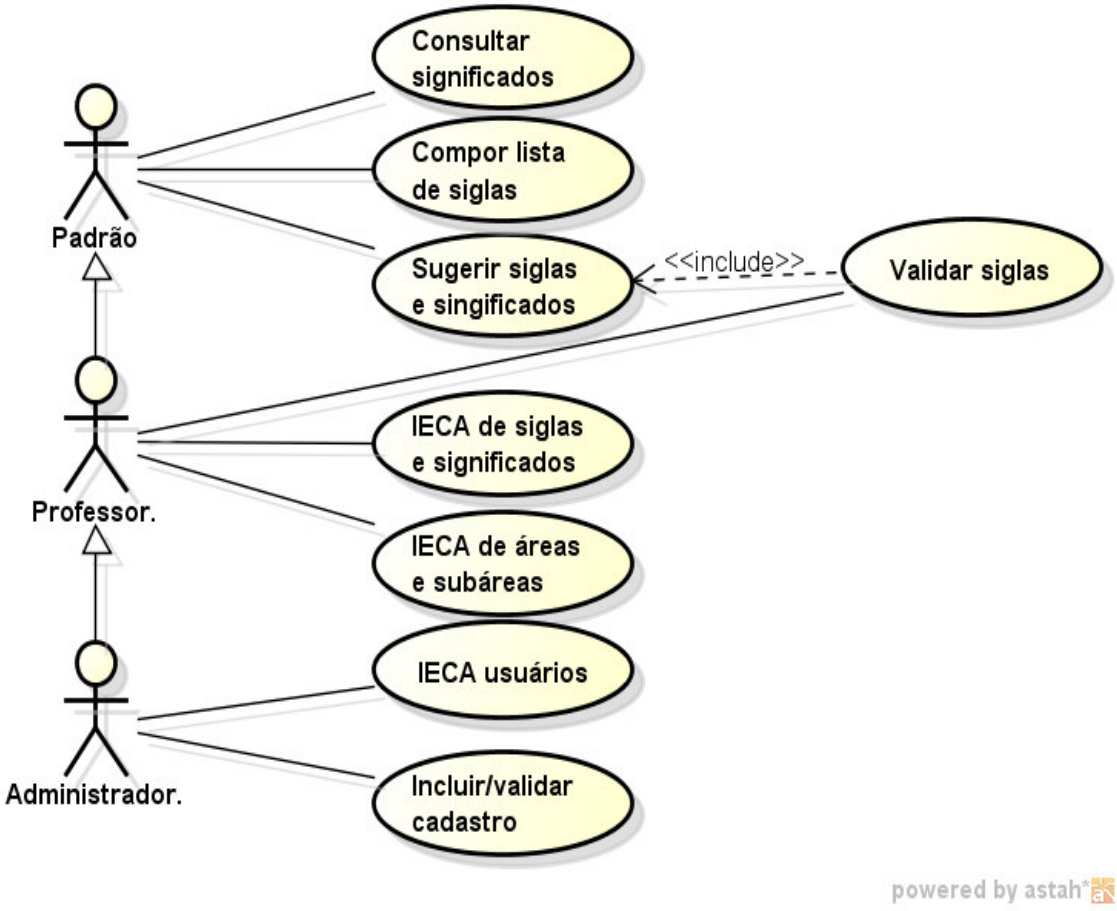

Figura 2 – Diagrama de casos de uso do sistema

Os quadros a seguir apresentam a descrição das operações dos casos de uso de cadastro. Na Figura 2, a sigla IECA significa Incluir, Excluir, Cadastrar e Alterar dados no banco de dados. Essas operações estão nos Quadros 4 a 7 a seguir. No Quadro 4 é apresentada a operação de inclusão.

<table>
<tr><td colspan="3">Identificador do requisito: Incluir dados.
Descrição: Caso de uso que permite ao ator efetuar cadastros.
Evento Iniciador: Qualquer tela de cadastro disponibilizada pelo software.
Atores: Qualquer ator com permissão para cadastro
Pré-condição: O usuário deve ter permissão para efetuar o cadastro solicitado.
Sequência de Eventos:
1 – O usuário informa os dados de entrada.
2 – O sistema valida os dados inseridos pelo usuário e caso estejam de acordo com a estrutura esperada pelo banco de dados insere os mesmos no banco.
3 – O sistema retorna uma mensagem informando se o cadastro foi efetuado ou alguma mensagem de erro que auxilie o usuário no cadastro.
Pós-Condição: Os dados inseridos devem ser validados.
Extensões: Caso ocorra erro nos dados informados o sistema deve solicitar a correção ou inserção de alguma informação faltante.</td></tr>
<tr><td colspan="2">Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td colspan="2">3.1 Dados inválidos</td><td rowspan="2">3.1 Se dados inválidos são informados para campos para os quais há validação, é solicitado ao usuário informá-los novamente.
3.2 O sistema retorna para o passo 2 solicitando novamente os dados.</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">Requisitos não funcionais:</td></tr>
<tr><td>Identificador</td><td>Nome</td><td>Descrição</td></tr>
<tr><td rowspan="2">RNF1.1</td><td>Informações inválidas</td><td rowspan="2">Os dados só devem ser inseridos no banco se forem validados.
Se dados inválidos são informados, o usuário deve informá-los novamente até que atendam aos critérios de validação estabelecidos.</td></tr>
<tr><td></td></tr>
</table>

Quadro 4 – Operação incluir dados dos casos de uso de cadastro

A operação de consulta é realizada pela descrição apresentada no Quadro 6.

<table>
<tr><td colspan="3">Identificador do requisito: Consultar dados.
Descrição: Este caso de uso permite consultar dados disponíveis no sistema.
Evento Iniciador: Telas de consulta e relatórios disponíveis.
Atores: Qualquer usuário com permissão de consulta.
Pré-condição: Dados disponíveis para consulta.
Sequência de Eventos:
1 – O usuário solicita consulta de dados.
2 – O sistema retorna os resultados da consulta.
Pós-Condição: Listagem dos dados é apresentada.</td></tr>
<tr><td colspan="2">Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td colspan="3">Requisitos não funcionais:</td></tr>
<tr><td>Identificador</td><td>Nome</td><td>Descrição</td></tr>
</table>

Quadro 5 – Operação consultar dados dos casos de uso de cadastro

A operação de alteração é apresentada no Quadro 5.

Identificador do requisito: Alterar dados. **Descrição**: O caso de uso permite ao ator efetuar alterações em dados já armazenados no banco de dados. **Evento Iniciador**: Qualquer tela que permita alteração nos dados. **Atores**: Usuário com permissões de alteração. **Pré-condição**: O usuário deve ter permissão para efetuar alterações em dados cadastrados. **Sequência de Eventos**: 1 – O usuário pesquisa o registro que deseja alterar. 2 – O usuário altera os dados em tela que são apresentados no formulário do respectivo cadastro. 3 – O sistema valida se as informações que o usuário deseja alterar permitem esse tipo de mudança. 4 – O sistema retorna uma mensagem ao usuário dizendo se foi possível ou não efetuar a alteração solicitada. **Pós-Condição**: As informações a serem alteradas devem ser inseridas no banco de dados e estar disponíveis para consulta. **Extensões**: Se não for possível realizar as alterações, o sistema deve retornar uma mensagem ao usuário informando-o da situação e retornar ao estado anterior à solicitação da operação de alteração.

Nome do fluxo alternativo (extensão)	Descrição
3.1Impossível realizar alterações	3.1 Os dados sendo utilizados em outros cadastros (chaves estrangeiras) não podem ser alterados.

Requisitos não funcionais:		
Identificador	**Nome**	**Descrição**
RNF1.1	Alteração	3.1 Não poderão ser feitas alterações em dados que são utilizados em outras operações realizadas pelo aplicativo. Esses dados podem ser chaves estrangeiras que estão em outras tabelas. 3.2 Se os dados não podem ser alterados o usuário é informado dessa impossibilidade. 3.3 Sistema retornar ao passo 3 e não realiza a operação de alteração realizada. O estado do sistema retorna para a condição anterior à solicitação de alteração dos dados.

Quadro 6 – Operação alterar dados dos casos de uso de cadastro

No Quadro 7 está a descrição da operação de exclusão dos casos de uso de cadastro.

Identificador do requisito: Excluir dados **Descrição**: Caso de uso que possibilita a exclusão de dados. **Evento Iniciador**: Qualquer tela que possibilite a exclusão de dados. **Atores**: Qualquer usuário com permissão de exclusão de dados. **Pré-condição**: O usuário deve ter permissão para excluir os dados na tela em questão. **Sequência de Eventos**: 1 – O usuário solicita a exclusão dos dados. 2 – O sistema valida se o usuário tem permissão para excluir os dados. 3 – O sistema valida se os dados que serão excluídos não são necessários em outras partes do sistema, ou seja, se eles podem ser excluídos. 4 – O sistema retorna a mensagem de sucesso na exclusão ou de erro na exclusão **Pós-Condição**: Dados excluídos ou mensagem de não possibilidade de exclusão apresentada para o usuário. **Extensões**: Se a exclusão estiver bloqueada por algum motivo, o sistema deverá apresentar uma mensagem com o motivo desse bloqueio ao usuário.

Nome do fluxo alternativo (extensão)	Descrição
4.1 Exclusão bloqueada	4.1 O sistema deverá apresentar uma mensagem ao usuário com o motivo do bloqueio da exclusão caso ela não seja possível. 4.2 O sistema retornar ao passo 3 e o seu estado é retornado para a condição anterior à solicitação de exclusão.

Requisitos não funcionais:		
Identificador	**Nome**	**Descrição**
RNF 1.1	Excluir dados	4.1 Os dados só poderão ser excluídos se o usuário possuir permissão para a exclusão e se os dados não forem necessários em outras partes do sistema.

Quadro 7 – Operação excluir dados dos casos de uso de cadastro

O diagrama de entidades e relacionamentos que representa o banco de dados é apresentado na Figura 3. A tabela principal é a de siglas que representa um termo (sigla, acrônimo, abreviatura ou outra denominação cadastrada no próprio sistema). Assim um termo é de um tipo, permitindo emitir listagens separadas por tipos. Há um cadastro de tipos de siglas para que possam ser definidas outras denominações para o termo, além de sigla, abreviatura e acrônimo, como utilizado neste texto. Uma sigla é classificada em área e subárea. A linguagem permite identificar se a descrição da sigla será grafada em itálico, no caso de siglas em língua estrangeira. O usuário permite identificar quem foi o autor do cadastro ou da validação da sigla.

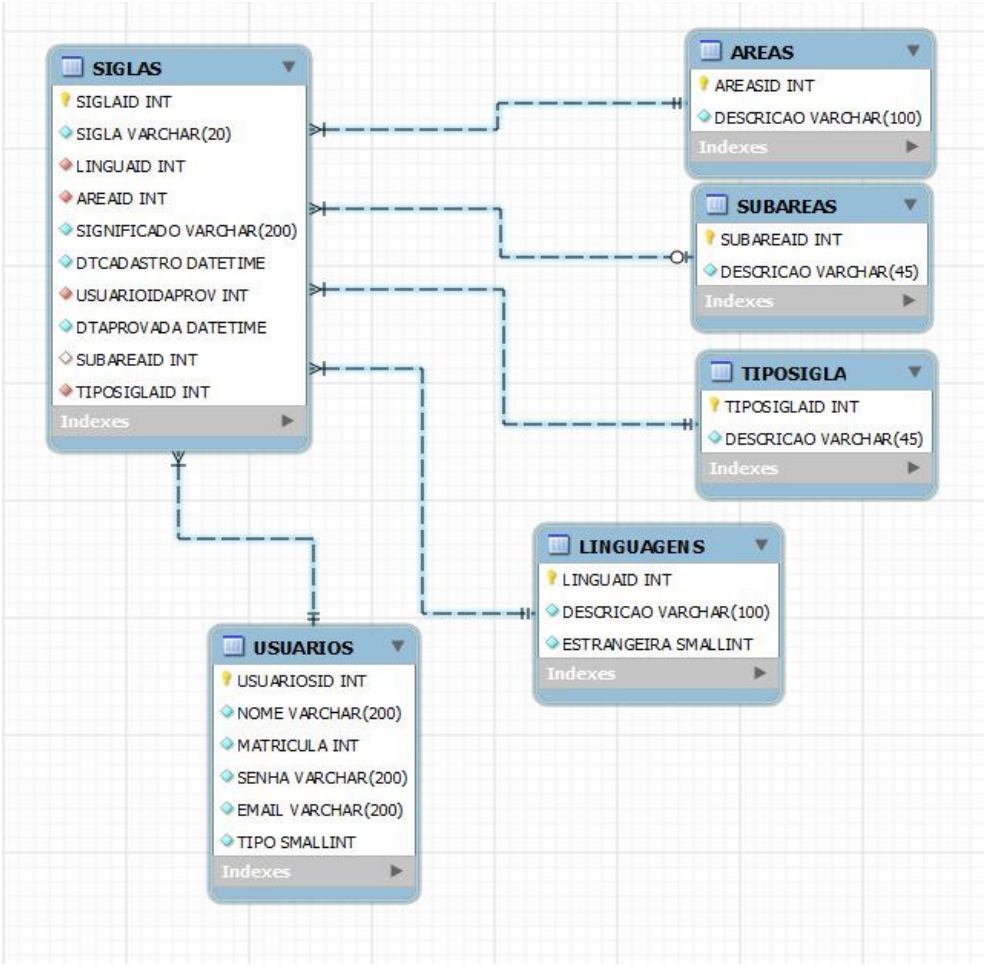

Figura 3 – Diagrama de entidades e relacionamentos do banco de dados

Os Quadros 8 a 13 apresentam a descrição das tabelas constantes na Figura 3. No Quadro 8 estão os campos da tabela siglas.

Dado	Descrição	Tipo	Obrigatório
SIGLAID	Chave primária do cadastro de termos	Inteiro	Sim
SIGLA	Termo	Texto	Sim
LINGUAID	Chave estrangeira. Indica a língua de origem do termo	Inteiro	Sim
AREAID	Chave estrangeira. Indica a categoria por área do termo	Inteiro	Sim
SIGNIFICADO	Descrição do termo.	Texto	Sim
DTCADASTRO	Data de cadastro do termo	Data	Inserido pelo sistema
USUARIOIDAPROV	Chave estrangeira. Usuário que cadastrou ou aprovou o termo. É o usuário logado no sistema.	Inteiro	Sim
DTAOPROVADA	Data de aprovação do cadastro	Data	Não
SUBÁREA	Chave estrangeira. Subárea de classificação da sigla.	Inteiro	Não
TIPOSIGLA	Chave estrangeira. Tipo de termo.	Inteiro	Sim

Quadro 8 – Tabela siglas

Os campos da tabela áreas estão descritos no Quadro 9.

Dado	Descrição	Tipo	Obrigatório
AREASID	Chave primária do cadastro de áreas	Inteiro	Sim
DESCRICAO	Descrição da área	Texto	Sim

Quadro 9 – Tabela áreas

O Quadro 10 apresenta os campos da tabela de subáreas.

Dado	Descrição	Tipo	Obrigatório
SUBAREASID	Chave primária do cadastro de subáreas	Inteiro	Sim
DESCRICAO	Descrição da subárea	Texto	Sim

Quadro 10 – Tabela subáreas

Os campos da tabela de tipos de termos são apresentados no Quadro 11.

Dado	Descrição	Tipo	Obrigatório
TIPOSIGLA	Chave primária do cadastro de tipos de termos	Inteiro	Sim
DESCRICAO	Descrição do tipo de termo	Texto	Sim

Quadro 11 – Tabela tipos de siglas

O Quadro contém os dados da tabela de línguas.

Dado	Descrição	Tipo	Obrigatório
LINGUAID	Chave primária do cadastro de áreas	Inteiro	Sim
DESCRICAO	Descrição da área	Texto	Sim
ESTRANGEIRA	Indica se o termo é em língua estrangeira	Inteiro	sim

Quadro 12 – Tabela línguas

Os dados da tabela de usuários são apresentados no Quadro 12.

Dado	Descrição	Tipo	Obrigatório
USUÁRIOID	Chave primária do cadastro de usuários	Inteiro	Sim
NOME	Nome do usuário	Texto	Sim
MATRÍCULA	Registro acadêmico ou funcional do usuário	Inteiro	Não
SENHA	Senha para acesso ao sistema	Texto	Sim
EMAIL	Conta de *email* do usuário	Texto	Não
TIPO	Indica o tipo de usuário (ator) definindo os seus privilégios de acesso às funcionalidades do sistema.	Inteiro	sim

Quadro 13 – Tabela usuários

4.3 APRESENTAÇÃO DO SISTEMA

A Figura 4 apresenta a página principal do aplicativo *web* desenvolvido. É através dessa página que serão realizadas as pesquisas para a geração das listas de termos.

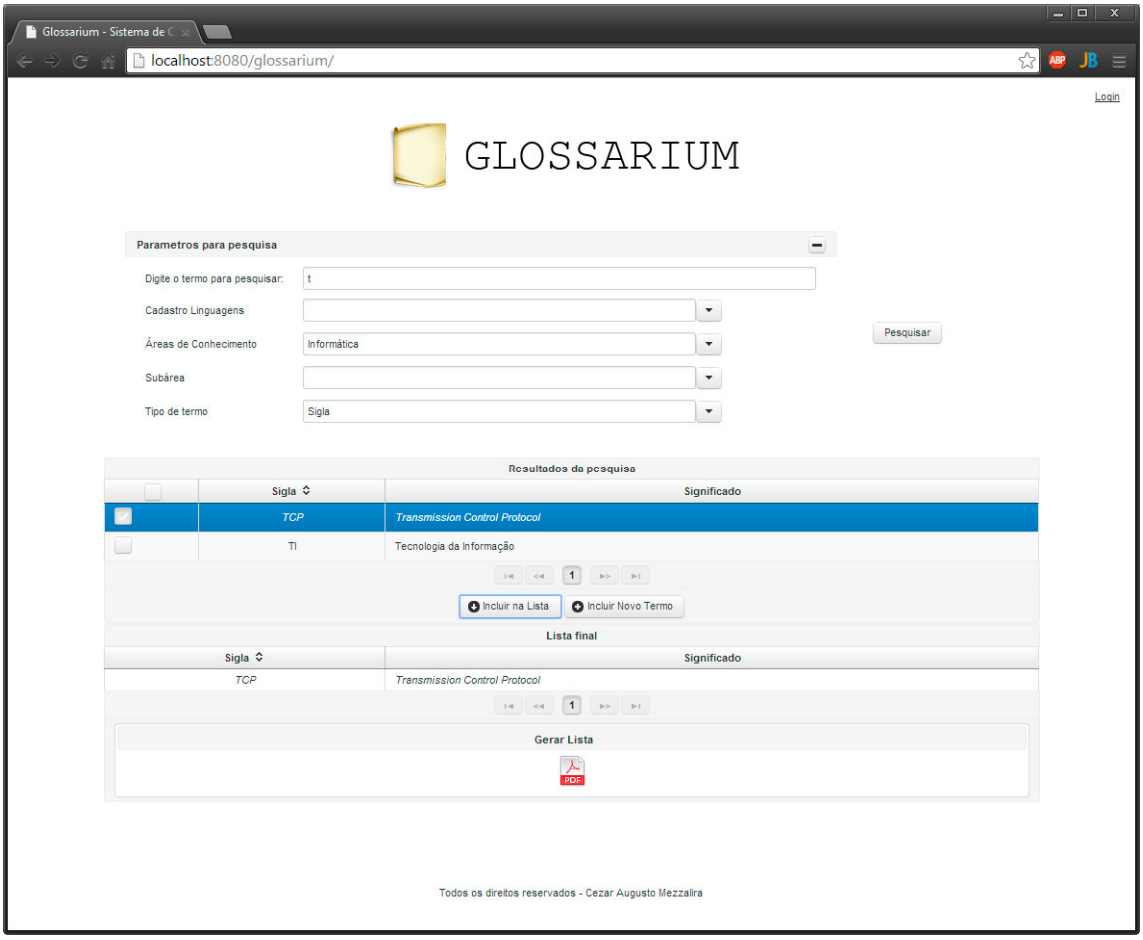

Figura 4 – Diagrama de entidades e relacionamentos do banco de dados

Para obter a lista o usuário deverá seguir o formulário que está divido em 4 passos:

1 - informar parâmetros de pesquisa.

2 - selecionar itens.

3 - incluir na lista final.

4 - gerar a lista.

A Figura 5 apresenta a tela inicial para a composição de uma lista de siglas, abreviaturas, acrônimos e outros. Essa tela é a apresentada quando o sistema é acessado, independentemente do tipo de usuário. Um usuário não cadastrado no sistema pode consultar o significado dos termos, compor lista de termos e gerar o relatório das mesmas, além de, sugerir termos. Nesse caso, os termos indicados devem ser validados por um usuário com permissões para realizar esse tipo de operação.

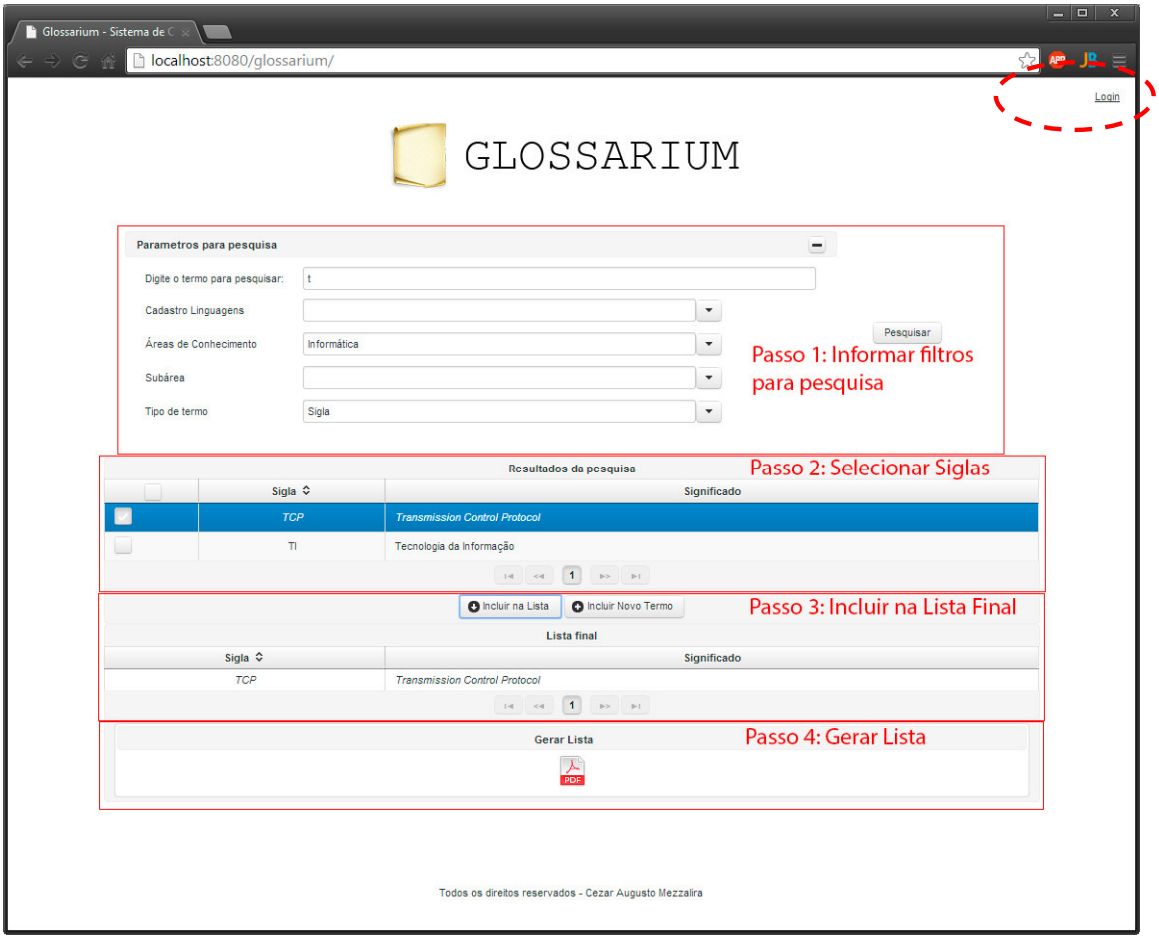

Figura 5 – Passos para geração de uma lista de siglas

Conforme mostrado na Figura 5, o primeiro passo para gerar uma lista é informar o termo ou parte dele. Caso seja necessária uma pesquisa mais especifica é possível também informar qual linguagem, área de conhecimento, subárea ou tipo do termo deve fazer parte dos critérios de pesquisa.

A execução do passo 2 depende dos resultados listados de acordo com os critérios informados no passo 1. Dessa forma, o usuário poderá selecionar somente os itens que ele desejar e então ir para o passo três.

No passo 3, com os itens selecionados na lista, o usuário pode incluir na lista final ou caso ele não tenha encontrado o termo, pode sugeri-lo através do botão “Incluir Novo Termo”.

É importante ressaltar que os três primeiros passos podem se repetir até que a lista esteja completa.

No quarto e último passo o usuário poderá emitir a lista em formato *Portable Document Format* (PDF) com os termos que estão na lista final. Os termos que forem de origem estrangeira serão listados sempre em itálico. A Figura 6 apresenta a listagem dos termos conforme a seleção realiza (apresentada na Figura 5).

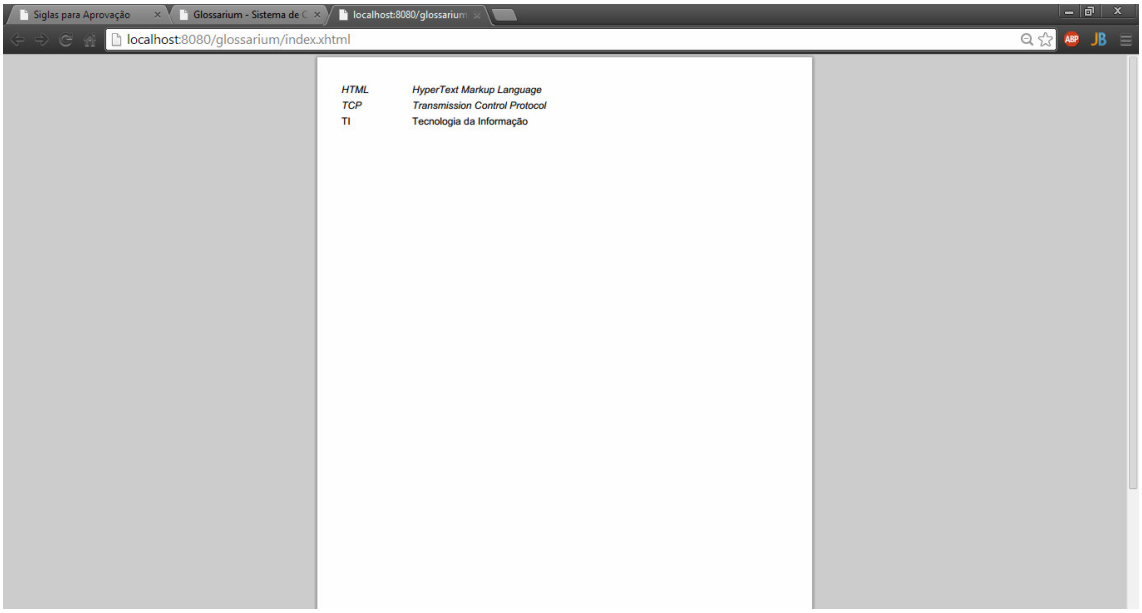

Figura 6 – Relatório de siglas

Ainda na tela principal, existe a possibilidade de um usuário professor acessar área administrativa por meio do link de *login*, que pode ser encontrado no canto superior direito da tela (área destacada na Figura 5).

A tela de *login* de professores exige a matrícula e a senha. Caso o *login* seja bem sucedido, serão apresentados os cadastros do aplicativo. Nesse cadastro é possível efetuar a manutenção dos mesmos, conforme mostra a Figura 7.

Nessa tela inicial, na região central são listados os termos que ainda não foram aprovados e no lado esquerdo está o menu com todas as funcionalidades do sistema com acesso permitido para o usuário com perfil de professor. Com o rótulo de “Manutenção” é possível acessar a listagem dos termos que precisam ser verificados para serem incluídos no banco de dados. Os demais itens de menu se referem aos cadastros de linguagens (a língua de origem do termo), áreas de conhecimento, subáreas de conhecimento, tipos de termos e o acesso para o cadastro de um novo termo. O item de menu cadastro permite acesso às operações de inclusão, exclusão e alteração. E por meio da consulta é possível verificar se um determinado item está cadastrado ou obter uma listagem dos cadastrados.

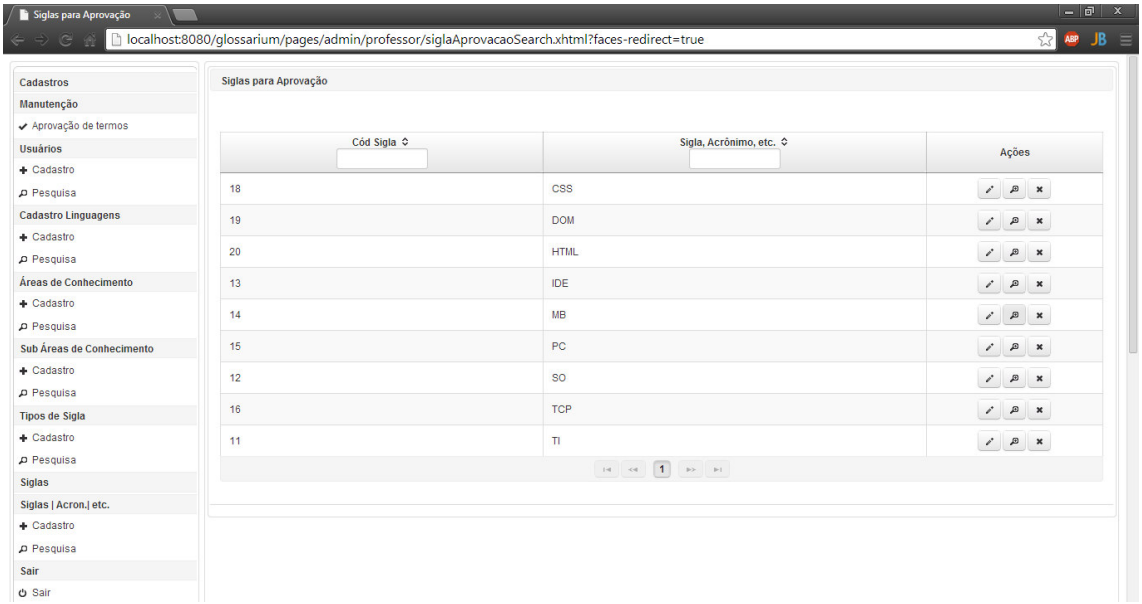

Figura 7 – Tela inicial dos cadastros

A lista dos termos para aprovação (Figura 8) mostra todos os termos cadastrados por um usuário com acesso somente de consulta. Para aprovar o termo é necessário apenas editar o cadastro, clicando no ícone de “Lápis”. Ao salvar o registro, é feita uma validação na qual é verificado se realmente o registro existe no banco e se os campos de data de validação e usuário de validação estão vazios. Caso a validação retorne um resultado verdadeiro, o registro é aprovado e deverá aparecer nas pesquisas feitas.

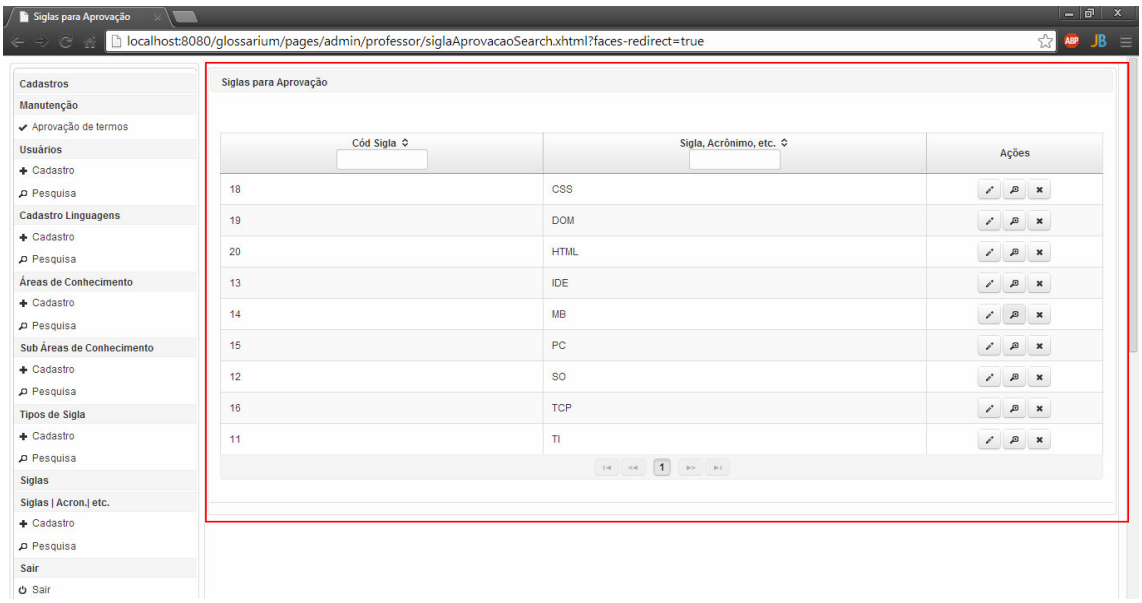

Figura 8 – Tela de termos não aprovados

No que se refere aos cadastros, o aplicativo segue um padrão para todos, possuindo em comum uma tela de cadastro (*Form*) e uma tela de pesquisa (*Search*). Será apresentado a seguir o cadastro de siglas, que é o principal cadastro do aplicativo.

Para acessar cadastro de siglas, o usuário devera acessar o menu Siglas> Siglas | Acron | Etc > Cadastro. Será exibido então o cadastro conforme mostra a Figura 9.

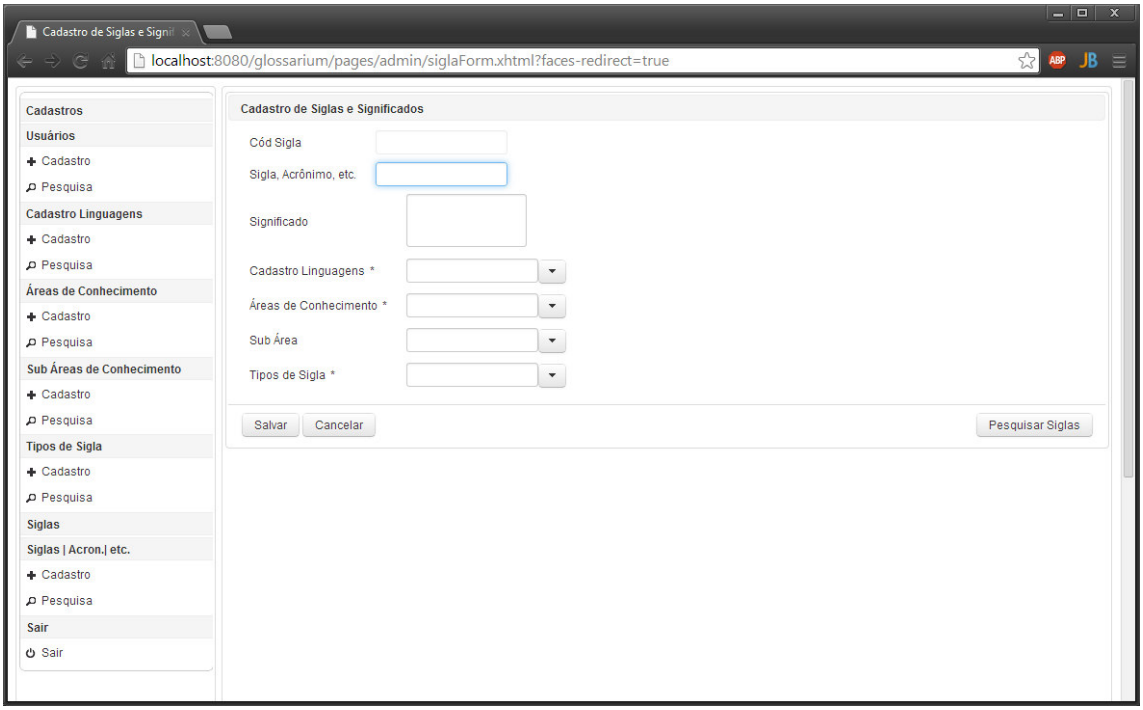

Figura 9 – Formulário de Cadastro

Para cadastrar uma nova sigla serão necessários os seguintes dados: sigla, significado, língua (Português ou outra língua estrangeira), área de conhecimento, subárea e tipo do termo (sigla, abreviatura e outros). Nos campos de sigla e significado, o usuário deverá digitar as informações. Já nos campos de língua, área e subárea de conhecimento e tipo de termo, o usuário deverá selecionar um item da lista. Caso o usuário não encontre algum termo, ele poderá efetuar o cadastro por meio do menu de cadastro de cada um dos campos, que se encontra do lado esquerdo da tela.

4.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção serão apresentados trechos de código que compõe o aplicativo. São trechos que implementam funcionalidades especificas do aplicativo, tais como métodos de pesquisa no banco de dados e validações no *Create Retrieve Update, Delete* (CRUD). Serão mostrados na seguinte ordem: *model*, *view* e *controller*.

O *model* compreende toda a parte de interação do sistema com o banco de dados, implementação de CRUD e classes entidade, dentre outras operações.

Na Listagem 1 está uma classe entidade padrão que implementa anotações *Java Persistence* API (JPA) “@Entity” e “@Table”. Na anotação “@Table” existem os atributos *name*, *catalog* e *schema*. O atributo *name* representa o nome da tabela no banco, já o atributo *catalog* representa o nome do banco e por fim o atributo *schema* está em branco, pois o banco não utiliza *schemas* para separar dados dentro do banco.

```
@Entity
@Table(name = "areas", catalog = "glossarium", schema = "")
public class Area implements Serializable {}
//
```

Listagem 1 - Classe entidade

Para criação do CRUD são necessárias a criação de três classes: *Data*, *Service* e *ServiceImpl*. Existe ainda a possibilidade de codificar uma quarta interface chamada *Repository* e implementar uma classe *RepositoryImpl*.

A interface Data estende da interface *JpaRepository*, que por sua vez é uma interface padrão do Spring Framework. Essa é implementada na classe *ServiceImpl* que é apresentada na Listagem 3. Na Listagem 2 é possível observar a codificação da interface.

```
public interface SiglaData extends JpaRepository<Sigla, Integer>{
    List<Sigla> findBySigla(String sigla);
    @Query(value = "select * from siglas s where LOWER(s.sigla)
like(:sigla)", nativeQuery = true)
    List<Sigla> findBySiglaLike(@Param("sigla") String sigla);

}
```

Listagem 2 - Interface SiglaData

A interface *Service* tem por objetivo definir o cabeçalho dos métodos padrão que deverão ser implementados na classe ServiceImpl. Aqui existem dois pontos importantes para o aplicativo: o primeiro deles é que todo e qualquer método de pesquisa no banco precisa ter sua assinatura codificada nesta interface. Outro ponto importante é que métodos de consulta no banco precisam, também, estar com a sua assinatura codificada ou na interface Data ou na interface *Repository*.

Na Listagem 3 estão codificadas a assinatura de três métodos que são responsáveis por consultas ao banco. O primeiro deles é um método que se utiliza da injeção de dependência do Spring Framework para fazer a consulta, por isso não possui implementação. Os outros dois métodos são codificados dentro do repositório, representados pela interface *Repository* e a classe *RepositoryImpl*.

A interface ainda estende de uma outra interface que possui a assinatura de todos os padrões de um CRUD, tais como incluir, atualizar e excluir um registro além de busca por ID (o campo chave identificador de um registro) e busca de todos os registros.

```
public interface SiglaService extends ICrudService<Sigla, Integer> {
    List<Sigla> findBySigla(String sigla);

    List<Sigla>  findBySiglaOrLinguaIdOrAreaIdOrSubareaIdOrTipoSiglaId(Sigla
sigla);

    List<Sigla> findByDataAprovada();
}
```

Listagem 3 - Interface Service

A interface *Repository* e a classe *RepositoryImpl* tem como funcionalidade servir de repositório de métodos de pesquisa no banco. Na Listagem 4 está um exemplo de codificação de uma interface *Repository*.

```
public interface SiglaRepository {

    List<Sigla>  findBySiglaOrLinguaIdOrAreaIdOrSubareaIdOrTipoSiglaId(Sigla
sigla);

    List<Sigla> findByDataAprovada();
}
```

Listagem 4 - Interface Repository

Na Listagem 5 está a implementação dos dois métodos codificados na interface apresentada na Listagem 4 através da classe *RepositoryImpl*. Ela estende de uma classe chamada *BaseRepository* que contém métodos para injetar o

entityManager e então gerar *queries*. É importante também lembrar que a classe precisa estar anotada com a anotação “@Repository” para poder funcionar.

```
@Repository
public class SiglaRepositoryImpl extends BaseRepository<Sigla> implements
SiglaRepository{}
```

Listagem 5 - Classe RepositoryImpl

Em relação às operações CRUD há ainda a classe *ServiceImpl*. Essa classe estende da classe *CrudService* que implementa métodos como inclusão, alteração, exclusão e consultas por ID e de todos os itens. É necessário anotar essa classe com a anotação “@Service”, pois, assim, o Spring entende que a classe é uma implementação do CRUD.

Na Listagem 6 existem dois objetos com a anotação “@Autowired”, que indica que esses objetos serão injetados em tempo de execução pelo Spring, ou seja, serão instanciados e ao final da execução serão destruídos automaticamente pelo *Garbage Collector*.

```
@Service
public class SiglaServiceImpl extends CrudService<Sigla, Integer>
implements SiglaService {

    @Autowired
    private SiglaData siglaData;

    @Autowired
    private SiglaRepository siglaRepository;
//Outros métodos
.
.
.
```

Listagem 6 - Classe Service Impl

Também existem alguns métodos do CRUD que podem ser sobrescritos. Eles têm funções específicas, tais como o método da Listagem 7, que é executado dentro do método salvar, antes de o registro ser salvo. Nesse momento, é possível validar valores e, caso necessário, alterá-los dentro do objeto que está sendo inserido ou atualizado. No exemplo foram feitas as seguintes validações:

1 - Para que o termo sempre seja salva em maiúsculo.

2 - Quando estiver inserindo seja salva a data de inclusão.

3 - Quando um termo estiver sendo alterado por um usuário professor o termo seja marcado como aprovado.

```
    @Override
    public Sigla preProcessorSave(Sigla entity) {
        //Cezar Mezzalira - 10/02/2014
        //Feita validação para não mudar a data toda  a vez que salva, mas
sim somente na primeira vez.
        if (entity.getDtcadastro() == null) {
            entity.setDtcadastro(new
Date(Calendar.getInstance().getTimeInMillis()));
        }

        //Cezar Mezzalira 10/02/2014
        //Feita validação para que quando o registro já esteja persistido e
o usuario seja administrador
        //o registro é aprovado automaticamente.
        if ((entity.getSiglaid() != null) &&
                (entity.getDtaprovada() == null) &&
                (entity.getUsuarioidaprov() == null)) {
            Usuario             usuario             =             (Usuario)
JsfUtil.getAttributeSession("usuario");
            entity.setDtaprovada(new
Date(Calendar.getInstance().getTimeInMillis()));
            entity.setUsuarioidaprov(usuario);
        }

        //Cezar Mezzalira - 21/01/2014
        //Irá salvar todas as siglas como maiusculas...
        entity.setSigla(entity.getSigla().toUpperCase());
        return entity;
    }
```

Listagem 7 - Método preProcessorSave

Ainda dentro da classe *ServiceImpl*, são implementados as chamadas dos métodos do *Repository*. Dentro dos métodos são chamados os métodos já implementados dentro da classe *RepositoryImpl*. O *ServiceImpl* faz a interligação entre o *Repository* e o *Controller*. A Listagem 8 contém a implementação dos métodos citados.

```
    @Override
    Public List<Sigla>
findBySiglaOrLinguaIdOrAreaIdOrSubareaIdOrTipoSiglaId(Sigla sigla) {
        return
siglaRepository.findBySiglaOrLinguaIdOrAreaIdOrSubareaIdOrTipoSiglaId(sigla
);
    }

    @Override
    public List<Sigla> findByDataAprovada() {
        return siglaRepository.findByDataAprovada();
    }
```

Listagem 8 - Métodos do Repository

V*iew* é a camada da aplicação na qual ocorre a interação do usuário com o sistema. A *view* interage com a camada *controller*, que por sua vez interage com o a camada *model* por meio de um *service*.

A Listagem 9 contém a codificação da tela de cadastro de usuário. Nessa tela são utilizados componentes do PrimeFaces e também componentes nativos do *Java Server Faces* (JSF).

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:components="http://java.sun.com/jsf/composite/components"
      xmlns:h="http://java.sun.com/jsf/html">
<ui:decorate template="template/padrao.xhtml">
    <ui:define name="titulo">
        <h:outputText value="#{form.cadastroDeUsuario}"/>
    </ui:define>

    <ui:define name="conteudo">
        <p:panelGrid columns="2">
            <p:outputLabel value="#{form.usuariosid}" for="codigo"/>
            <p:inputText value="#{usuarioController.entity.usuariosid}"
                         label="#{form.usuariosid}" id="codigo"
disabled="true"/>

            <p:outputLabel value="#{form.nome}" for="nome"/>
            <p:inputText value="#{usuarioController.entity.nome}"
                         label="#{form.nome}" id="nome" maxlength="100"
required="true"/>

            <p:outputLabel value="#{form.matricula}" for="matricula"/>
            <p:inputText value="#{usuarioController.entity.matricula}"
                         label="#{form.matricula}" id="matricula"
maxlength="100"
                         required="true"/>

            <p:outputLabel value="#{form.senha}" for="senha"/>
            <p:password value="#{usuarioController.entity.senha}"
                         label="#{form.senha}" id="senha" maxlength="100"
                         required="true"/>

            <p:outputLabel value="#{form.email}" for="email"/>
            <p:inputText value="#{usuarioController.entity.email}"
                         label="#{form.email}" id="email" maxlength="100"
required="true"/>

            <p:outputLabel value="#{form.tipo}" for="tipo"/>
            <p:selectOneMenu id="tipo"
value="#{usuarioController.entity.tipo}" required="true">
                <f:selectItem itemLabel="#{form.selecione}" itemValue="" />
                <f:selectItem itemLabel="#{form.admin}" itemValue="0" />
                <f:selectItem itemLabel="#{form.aluno}" itemValue="1" />
            </p:selectOneMenu>
        </p:panelGrid>
    </ui:define>

    <ui:define name="rodape">
```

```
        <components:commandButtonForm update="@form"
controller="#{usuarioController}"/>
        <p:commandButton value="#{form.pesquisarUsuarios}"
action="/pages/admin/usuarioSearch.xhtml?faces-redirect=true"
                         immediate="true" ajax="false" style="float:
right"/>
        <div style="clear: both"></div>
    </ui:define>
</ui:decorate>
</html>
```

Listagem 9 - Cadastro de Usuários

Existem muitas funcionalidades que foram utilizadas dentro da aplicação para melhorar a interação do usuário com a mesma. Uma delas é na tela principal na pesquisa de termos. Termos em língua estrangeira serão grafados em itálico tanto na lista de pesquisa, quanto na lista final. Na Listagem 10 é apresentado o código para a validação dentro do formulário.

```
<p:column width="70%">
      <f:facet name="header">Significado</f:facet>
      <h:outputText style="font-style: italic"
                          rendered="#{item.linguaid.estrangeira == 1}"
                          value="#{item.significado}"/>
      <h:outputText rendered="#{item.linguaid.estrangeira == 0}"
                          value="#{item.significado}"/>
</p:column>
```

Listagem 10 - Validação para colunas de DataTable

Controller nada mais é que a ponte entre a camada *view* e a camada *service*. O papel do *controller* é tratar as chamadas feitas pelo *view*, utilizar métodos fornecidos pelo *service* e retornar os resultados para a *view* novamente.

A Listagem 11 mostra o inicio da implementação da classe *controller*. Essa classe estende da classe abstrata *CrudController*. Para que a classe seja identificada como *controller* e possa ser visualizada pela *view* é necessária a anotação “@Controller”. A anotação “@Scope” define em que escopo será utilizado essa classe. Também é utilizado o conceito de injeção de dependências para os objetos *service* que são utilizados pela classe.

```
@Controller
@Scope("view")
public class PesquisaTermoController extends CrudController<Sigla, Integer>
{
    @Autowired
    private com.mezzalira.model.service.SiglaService siglaService;
... //Outros metodos.
```

Listagem 11 - Cabeçalho da classe Controller

Na Listagem 12 está a chamada do método de pesquisa de termos feita na tela principal do aplicativo. Observa-se que no *controller* é apenas chamado um método do *service* que irá retornar os dados para uma lista.

```
    public void pesquisar() {
        lsEntity =
siglaService.findBySiglaOrLinguaIdOrAreaIdOrSubareaIdOrTipoSiglaId(entity);
        siglaDataModel = new SiglaDataModel(lsEntity);
    }
```

Listagem 12 - Método de pesquisa

Para emitir relatórios, existe um método implementado dentro da classe *controller*, conforme apresentado na Listagem 13.

```
    public void gerarRelatorio() {
        //Crio a lista com os itens da lista final
        List<SiglaReport> siglasReport = new ArrayList<SiglaReport>();
        for (Sigla itensAdicionado : itensAdicionados) {
            siglasReport.add(new SiglaReport(itensAdicionado));
        }


        String nomeRelatorio = "Siglas";

        ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
        ServletContext servletContext = (ServletContext)
externalContext.getContext();
        String arquivo = servletContext.getRealPath("WEB-INF/relatorios/" +
nomeRelatorio + ".jasper");
        JRDataSource jrds = new JRBeanCollectionDataSource(siglasReport);
        FacesContext context = FacesContext.getCurrentInstance();
        HttpServletResponse response = (HttpServletResponse)
context.getExternalContext().getResponse();

        try {
            DefaultJasperReportsContext jContext =
DefaultJasperReportsContext.getInstance();

JRPropertiesUtil.getInstance(jContext).setProperty("net.sf.jasperreports.de
fault.font.name", "Arial Sans");

JRPropertiesUtil.getInstance(jContext).setProperty("net.sf.jasperreports.de
fault.pdf.embedded", "true");

JRPropertiesUtil.getInstance(jContext).setProperty("net.sf.jasperreports.de
fault.pdf.font.name", "Arial Sans");

            ServletOutputStream servletOutputStream =
response.getOutputStream();
            JasperRunManager.runReportToPdfStream(new FileInputStream(new
File(arquivo)), servletOutputStream, null, jrds);
            response.setContentType("application/pdf");
            response.setHeader("Content-Disposition", "attachment;
filename=\"" + nomeRelatorio + ".pdf\"");

            servletOutputStream.flush();
```

```
            servletOutputStream.close();
            context.renderResponse();
            context.responseComplete();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Listagem 13 - Método para emissão de reletórios

Para a geração da lista final foi implementado o método mostrado na Listagem 14. Esse método valida para que não seja inserida uma mesma sigla mais de uma vez na lista e após a inserção, faz a ordenação por meio do método *sort* da classe *Collections*.

```
    public void adicionarNaLista() {
        for (Sigla sigla : siglas) {
            if (!itensAdicionados.contains(sigla)) {
                itensAdicionados.add(sigla);
            }
        }
        //Ordeno a lista depois de inserir os dados para que a visualização
do relatório fique correta.
        Collections.sort(itensAdicionados, new SiglaComparator());
    }
```

Listagem 14 - Método adicionarNaLista

Para fazer a ordenação de uma lista com um tipo é necessário passar um parâmetro extra para o método *sort*. Esse parâmetro é um *comparator*, uma classe que tem um método chamado compareTo que faz a comparação de dois itens passados no cabeçalho do método. A implementação da classe é simples e apenas precisa que ela estenda da classe *Comparator* e implemente o método citado anteriormente conforme a Listagem 15.

```
package com.mezzalira.web.comparator;
import com.mezzalira.model.entity.Sigla;
import java.util.Comparator;

public class SiglaComparator implements Comparator<Sigla>{
    @Override
    public int compare(Sigla sigla, Sigla outraSigla) {
        return sigla.getSigla().compareTo(outraSigla.getSigla());
    }
}
```

Listagem 15 - Classe SiglaComparator

4.5 DISCUSSÃO

Durante o desenvolvimento do trabalho foram enfrentados diversos problemas para implementar determinadas partes do sistema, principalmente em relação à criação da tela principal de pesquisa, pequenas validações para uma mesma tela, validações de *login* e na construção e apresentação de relatórios.

Levando em conta o tempo de implementação do sistema, comparadas aos métodos tradicionais de programação para web e *desktop*, é possível afirmar que o ganho de tempo é real. Isso porque existe uma lógica na implementação, principalmente por conta da utilização do MVC, da injeção de dependências com Spring para a redução sensível de escrita de código desnecessário e que acaba tornando o sistema, em termos de implementação, mais flexível. E, ainda, pelo uso do Maven que por meio da sua configuração por XML gerencia todas as bibliotecas e dependências necessárias para o funcionamento do sistema.

Outra grande vantagem das tecnologias utilizadas é a portabilidade do sistema. No decorrer do trabalho, várias vezes houve necessidade de troca de máquina utilizada para implementação, sendo apenas necessário levar para a nova máquina as pastas de configuração do GlassFish, do Maven, ter uma instância do MySQL para executar o *script* de criação do banco e os arquivos do projeto. Para utilizar o projeto, bastava apenas configurar o caminho do GlassFish, do Maven e por fim compilar com o Maven para que as bibliotecas do projeto fossem baixadas.

Outra grande vantagem dessas tecnologias é a possibilidade de alterar código em tempo de execução e somente atualizar as classes dentro do pacote *war* e o sistema já estará funcionando. Não há a necessidade de fazer um novo *deploy* ou ter que reiniciar o GlassFish.

5 CONCLUSÃO

O objetivo deste trabalho foi implementar um sistema *web* para cadastro, consulta e geração de listagens de termos como siglas, abreviaturas e acrônimos. Visando, assim, facilitar o trabalho de geração das listas desses termos, bem como a consulta pelo seu significado. A opção de desenvolvimento de um aplicativo para web tem como justificativa a facilidade de acesso. Para esse desenvolvimento foram utilizadas tecnologias que caracterizam as aplicações Java como ricas, incluindo Java ServerFaces e PrimeFaces. Outras ferramentas e tecnologias também foram utilizadas visando facilitar e agilizar o desenvolvimento do sistema. Dentre essas está o Maven.

Como forma de fundamentar conceitualmente o trabalho foi apresentado, no capítulo do referencial teórico, sobre desenvolvimento de aplicações *web*. Um breve histórico desse tipo de aplicação, seguido por conteúdo relacionado às *Rich Internet Applications* e o desenvolvimento de aplicações *web* utilizando a linguagem Java, foi apresentado.

O sistema implementado tem como principal funcionalidade a composição de listas de termos. Para que essas listas possam ser compostas é necessários que os termos estejam cadastrados juntamente com a descrição, que é o seu significado. Como forma de facilitar a escolha do termo correto para o tipo e o contexto do trabalho, os termos são categorizados em áreas e subáreas, que também são cadastradas. O tipo de termo também é cadastrado, permitindo outras categorias além das tradicionais denominações sigla, acrônimo e abreviatura. A indicação se o termo é de língua estrangeira permite que o mesmo seja grafada em itálico, como é solicitado pelas normas de formatação de trabalhos utilizadas pela UTFPR.

Outra funcionalidade importante do sistema é a possibilidade de indicação de termos para serem incluídos no banco de dados por parte de usuários que não precisam estar logados no sistema. Esses termos devem ser validados por usuários com permissão para realizar esse tipo de ação.

Dentre os trabalhos futuros, como forma de complementar as funcionalidades implementadas, destaca-se a coleta automática dos termos que estão no texto, colocando a descrição somente no primeiro uso do termo, e a composição da listagem. Para isso é necessário ler um arquivo texto e fazer a

identificação do termo, a inclusão da descrição do termo no corpo do texto na sua primeira ocorrência e a geração da listagem, sem que sejam realizadas alterações na formatação do texto.

REFERÊNCIAS

AJAX. *Ajax Frameworks*. Disponível em <http://ajaxpatterns.org/Ajax_Frameworks>. Acesso em: 28 dez. 2013.

ALMEIDA, Fernando L. F.; LOURENÇO, Justino M. R.. **eCreation of value with web 3.0 technologies**. 6th Iberian Conference on Information Systems and Technologies (CISTI), 2011, p. 1-4.

AMALFITANO, Domenico; FASOLINO, Anna Rita; POLCARO, Armando; TRAMONTANA, Porfirio. **Comprehending Ajax web applications by the DynaRIA tool.** Conference on the Quality of Information and Communications Technology (QUATIC), 2010 Seventh International, 2010, p. 122-131.

BLAHA, Michael; JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2**. 2ª ed. Rio de Janeiro: Elsevier, 2006.

BRESLIN, John; DECKER, Stefan. The future of social networks on the internet: the need for semantics.,**IEEE Internet Computing**, v. 11, 2007, p. 86-90.

BUCHNER, Björn; BÖTTCHER, Axel; STORCH, Christian. **Evaluation of java-based open source web frameworks with ajax support**. 14th IEEE International Symposium on Web Systems Evolution, 2012, p. 45-49.

DOM. **Document Object Model (DOM), W3C**. Disponível em: <http://www.w3.org/DOM/>. Acesso em 28 dez. 2013.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GARRETT, Jesse James. **AJAX: A new approach to Web applications**, Adaptive, 2005, Disponível em: <http://www.adaptivepath.com/publications/essays/archives/000385print.php>. Acesso em: 28 dez. 2013.

HENDLER, Jim. Web 3.0 emerging. **Computer**, v. 42, n. 1, 2009, p. 111-113.

HOGG, Roman; MECKEL, Miriam; SLABEVA, Stanoevska; MARTIGNONI, Robert. **Overview of business models for web 2.0 communities**. In: Proceedings of GeNeMe, 2006, p. 23-37.

JAZAYERI, Mehdi. **Some trends in web application development**. Future of Software Engineering (FOSE'07), 2007, p. 199-213.

PAVLIĆ, Daniel; PAVLIĆ, Mile; JOVANOVIĆ, Vladan. **Future of internet technologies.** MIPRO 2012, p. 1366-1371.

PRAJAPATI, Harshad B.; DABHI, Vipul K. **High quality web-application development on Java EE platforn**. 2009 WiEE International Advance Conputing Conference (IACC 2009), p. 1164-1169.

UNIVERSIDADE FEDERAL DO PARANÁ. **Normas para apresentação de trabalhos acadêmicos**. Curitiba: Editora da UFPR, 2008.