CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA - CEFET/RJ

Sistema de Banco de Currículos

José Carlos Marques Rebelo

Orientador: Professor Eduardo Bezerra

Rio de Janeiro
Agosto de 2010

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA - CEFET/RJ

Sistema de Banco de Currículos

José Carlos Marques Rebelo

Projeto final apresentado como exigência da disciplina de Trabalho de Conclusão de Curso, sendo requisito para conclusão do Curso Superior de Tecnologia em Sistemas para Internet do CEFET/RJ

Orientador: Professor Eduardo Bezerra.

Rio de Janeiro
Agosto de 2010

R289 Rebelo, José Carlos Marques
Sistema de banco de currículos / José Carlos Marques Rebelo.—2010.
xii, 87f. : il. ; enc.

Projeto Final (Graduação) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca ,2010.
Bibliografia : f.75-76
Apêndices
Orientador : Eduardo Bezerra

1.Sistemas de computação 2.Internet 3.Banco de currículos 4.Recuperação da informação I.Bezerra, Eduardo (orient.)II.Título.

CDD 004

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois sem ele nada é possível. Agradeço também a todos os professores do curso, que nos souberam ministrar as disciplinas de forma a nos deixar preparados para exercer melhor nossa profissão. Não poderia também deixar de agradecer aos nossos colegas de curso, que estiveram ao nosso lado ao longo deste tempo. E por último, mas não menos importante, fica meu agradecimento especial ao coordenador do curso e professor orientador deste trabalho, Eduardo Bezerra, por ter orientado o caminho para a elaboração deste trabalho.

RESUMO

O sistema descrito neste trabalho tem o objetivo de proporcionar um ambiente via Internet no qual pessoas possam divulgar seus currículos e empresas possam divulgar vagas de emprego, de maneira a facilitar a comunicação entre as partes, além de ser também um estudo de caso sobre a utilização de mecanismos de recuperação de informação (mecanismos de busca). O sistema possui também ferramentas de apoio para alcançar este objetivo, como as seguintes: Cadastro de currículos e vagas, busca de currículos e vagas, exibição de estatísticas sobre usuários pessoa física, visualização de currículos de candidatos às vagas, candidatura à vaga, e visualização de candidaturas.

Palavras-chave: Recursos Humanos, Currículos, Mecanismos de Busca

.

ABSTRACT

The system described in this monograph aims to provide an Internet environment in which job candidates can register their CVs and companies can publicize job openings, so as to facilitate communication between the parties, as well as being a case study on the use mechanisms for information retrieval (search engines). The system also has support tools to achieve this goal, as the following: Registration of resumes and vacancies, search resumes and vacancies, view statistics on individual users, view resumes of job applicants, applying for job, and visualization of applications.

Keywords: Human Resources, Resumes, Search Engines

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AJAX – *Asynchronous Javascript And XML*

API – *Application Programming Interface*

ASP – *Active Server Pages*

BLL – *Business Logic Layer*

CPU – *Central Processor Unit*

CRUD – *Create Retrieve Update Delete*

CSS – *Cascade Style Sheet*

DAL – *Data Access Layer*

DAO – *Data Access Object*

DOM – *Document Object Model*

HTML – *Hypertext Markup Language*

J2EE – *Java to Enterprise Environment*

MVC – *Model View Controller*

MVP – *Model View Presenter*

SGBD – *Sistema de Gerenciamento de Banco de Dados*

SQL – *Structured Query Language*

W3C – *World Wide Web Consortium*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

Capítulo 1

Introdução

A necessidade de ferramentas para busca e gerenciamento de informações, de modo geral, é de extrema importância atualmente, devido à grande quantidade de informação que é produzida e armazenada e que necessita ser recuperada para sua utilização. O projeto descrito neste trabalho é um sistema voltado para a utilização por pessoas físicas e jurídicas que desejem publicar e fazer buscas de currículos ou vagas de emprego, e é ao mesmo tempo um exemplo da aplicação de mecanismos de busca para sistemas, especialmente via Internet e com um número potencialmente grande de usuários.

Este capítulo contextualiza o projeto apresentando a motivação do tema na Seção 1.1, as justificativas na Seção 1.2, os objetivos na Seção 1.3, a metodologia na Seção 1.4 e a organização dos capítulos na Seção1.5 .

1.1 Motivação

A área de recursos humanos das empresas é com certeza uma das que mais se beneficiou com o desenvolvimento dos sistemas de informação e da internet. Antes da popularização da Internet, por exemplo, no caso de contratações de pessoal, o contato entre um candidato a emprego e um empregador em potencial teria incluído, na maioria dos casos, uma combinação de ligações telefônicas, cartas de apresentação e entrevistas. Hoje, e cada vez mais, as pessoas estão procurando e se oferecendo para empregos virtualmente, contatando empregadores por meio da Internet e mandando currículos eletronicamente.

O recrutamento tem sido tradicionalmente uma questão de relacionamento interpessoal. Mas essa prática tradicional do mercado tem sido desafiada principalmente pelos sítios de recrutamento na Web, que se multiplicaram em milhares nos últimos anos e que tem se tornado um padrão nos métodos de procura e contratação utilizados pelos departamentos de RH para contratação de funcionários, por considerarem a Internet como a opção mais barata e direta. As empresas se sentem mais seguras quando seguem um processo convencional de solicitar currículos, embora muitas hoje já façam suas pesquisas em sítios especializados em empregos [1]. A importância dada a esse fato é tão grande que mesmo sítios de busca tradicionais como o *Yahoo* tem um braço especializado nesse segmento, no caso o *Hotjobs*

(http://hotjobs.yahoo.com/).

A despeito de problemas técnicos e a incapacidade, em alguns casos, de corresponder às expectativas de serviço de candidatos e clientes, o seu alcance, velocidade e a eficiência de custos sugerem que o recrutamento on-line se tornará um padrão definitivo, suplantando mesmo métodos tradicionais como os classificados de jornais.

A partir do que foi exposto acima, chegamos ao cerne da questão. Para as empresas que desejam publicar vagas de emprego e buscar currículos de candidatos, e para candidatos que desejam publicar seus currículos, é essencial que estes procedimentos sejam realizados com excelência e com o menor custo possível, com desempenho e confiabilidade. E tudo isso passa pela escolha de ferramentas corretas, que possam atender à demanda da empresa /candidato da forma mais simples e eficaz possível. Desse modo, o segmento de busca de currículos/vagas se torna uma área relevante para a aplicação de sistemas de mecanismos de busca e interessante como um estudo de caso sobre a aplicação desses mecanismos, especialmente por lidar com volumes grandes de informação e que necessitam utilizar critérios de busca refinados para as informações armazenadas.

1.2 Justificativas

O sistema de *software* deste trabalho de conclusão de curso foi criado para ser uma ferramenta capaz de proporcionar uma padronização na comunicação entre empresas (pessoas jurídicas) e pessoas que buscam vagas de emprego(pessoas físicas), e de demonstrar a importância do uso de mecanismos de busca como ferramenta de apoio. Através do alcance e da praticidade da Internet, usuários poderão fazer buscas e divulgar seus currículos e vagas de emprego além de ter ao seu dispor ferramentas de acompanhamento adicionais a essa tarefa.

1.3 Objetivos

O principal objetivo deste *software* é facilitar o gerenciamento e a busca de currículos e vagas de emprego, centralizando os dados. Em uma visão geral, o software pretende fornecer um ambiente para intercomunicação entre candidatos a vagas de emprego e empresas.

Através de funcionalidades como cadastro de currículos e vagas, busca de currículos e vagas, exibição de estatísticas sobre usuários pessoa física, visualização de currículos de candidatos às vagas, candidatura a vaga, e visualização de candidaturas, o *software* pretende atingir os objetivos listados acima.

A funcionalidade de cadastro de currículos permite que uma pessoa cadastre e edite o seu currículo, o qual é composto das seguintes informações: Área de atuação profissional do candidato, descrição da experiência profissional, descrição dos conhecimentos (habilidades, ferramentas de software que domina, etc.), descrição dos idiomas que domina, escolaridade, descrição da sua formação acadêmica (instituições que cursou) e observações gerais, no qual podem ser colocadas informações extras que o candidato julgar necessárias.

A funcionalidade de cadastro de vagas permite que uma empresa publique vagas de emprego. O cadastro de vaga possui as seguintes informações: Área profissional alvo, cargo, data de publicação, e observações opcionais relativas à vaga.

A funcionalidade de candidatura a vaga permite que uma pessoa se candidate a uma ou mais vagas publicadas, enviando opcionalmente uma mensagem de apresentação à empresa.

A funcionalidade de visualização de currículos de candidatos às vagas permite que uma empresa verifique quais pessoas se candidataram a uma determinada vaga publicada por ela, e acessem os currículos dessas pessoas.

A funcionalidade de visualização de candidaturas permite que uma pessoa veja uma listagem com as informações sobre as vagas às quais ela se candidatou recentemente.

A funcionalidade de exibição de estatísticas sobre usuários pessoa física, permite que sejam exibidas estatísticas sobre os usuários pessoa física cadastrados no sistema. Esta funcionalidade dá a empresa um panorama geral do universo que está sendo pesquisado, e dessa forma auxilia no processo de busca.

Por fim, a funcionalidade de busca de currículos e vagas permite que pessoas façam buscas por vagas e empresas façam buscas por currículos, utilizando palavras-chave, através de um mecanismo de busca especializado, o qual é a principal ferramenta do sistema.

1.4 Metodologia

Neste trabalho foram utilizados como parâmetros, na construção e análise das funcionalidades do sistema, os recursos e as características mais comuns relativas às ferramentas disponíveis na área de busca e recuperação de informações e, mais especificamente, na área de busca e cadastro de currículos e vagas de emprego, encontradas na internet. Procurou-se também fazer um análise de várias dessas características e recursos, visando encontrar a forma de implementação mais adequada ao trabalho, e aprimorar o uso desses recursos. Na implementação foram utilizadas ferramentas *open source*, devido à sua disponibilidade e adaptabilidade aos requisitos exigidos pela aplicação.

1.5 Organização dos Capítulos

No capítulo 0, que se segue, são apresentados os principais aspectos teóricos relacionados ao projeto. Em seguida, no capítulo 0, é mostrado todo o resultado da fase de modelagem do sistema, passando pela especificação dos requisitos funcionais e não funcionais, assim como das regras de negócio. Logo após, temos o modelo de casos de uso e a descrição de cada um deles, seguido dos modelos de classe e de interações. Finalizamos o capítulo com a descrição do projeto de banco de dados.

No capítulo 0, são demonstrados os aspectos de desenvolvimento e implantação do sistema, detalhando-se as tecnologias usadas durante o projeto, assim como os procedimentos e estratégias de implementação do mesmo.

Finalizando o trabalho, no capítulo 0, são apresentadas as conclusões e projetos de trabalhos futuros em cima deste tema.

Capítulo 2

Fundamentos Teóricos

Os fundamentos teóricos abordados neste capítulo são referentes a mecanismos de busca na seção 2.1, e características de sítios especializados que disponibilizam currículos e/ou vagas na seção 2.2.

2.1 Busca e Recuperação de Informação

Na seção 2.1.1 é apresentada uma introdução sobre a importância e origem dos mecanismos de recuperação de informação, na seção 2.1.2 são apresentados alguns dos principais conceitos na área e na seção 2.1.3 são descritos, como exemplo, os mecanismos de busca para a Web.

2.1.1 Introdução

A cada ano, uma quantidade muito grande de informação é criada e armazenada por computadores ao redor do mundo. Computadores estão em todo lugar, coletando dados relativos a quase todas as atividades humanas: Transações financeiras, notícias, dados governamentais, dados científicos. Estima-se que a cada ano produza-se em torno de 2 exabytes (1 exabyte = 10^{12} megabytes) de informação [2]. Se por um lado é extremamente importante ter uma grande quantidade de dados armazenada, por outro lado se torna cada vez mais difícil buscar algo específico nesse volume de dados armazenado. Nesta situação, pode-se dizer que há uma sobrecarga de informação. Como conseqüência, fica cada vez mais difícil extrair algo "útil" a partir dos dados existentes, como por exemplo, padrões, relacionamentos ou tendências relativas aos dados. E isto não se restringe apenas à Internet como um todo ou a ambientes acadêmicos ou empresariais; Mesmo pessoas usando seus computadores pessoais já se deparam com esse problema. Cada vez mais os computadores armazenam uma quantidade crescente de dados, em diversos tipos de arquivos diferentes. Como conseqüência disso, os sistemas operacionais mais modernos já trazem embutidos mecanismos de busca sofisticados que permitem ao usuário encontrar os arquivos desejados na sua máquina com a maior precisão e rapidez possível, a partir de um determinado critério de busca. A partir desta constatação, percebemos a importância crescente dos mecanismos de busca e técnicas de mineração de dados e suas áreas relacionadas (Recuperação da Informação, Lingüística Computacional, Processamento de Linguagem Natural, Aprendizado de Máquina, Banco de

dados, Estatística).

Historicamente, a necessidade de armazenar e recuperar informação escrita se tornou cada vez mais importante ao longo dos séculos, especialmente com a invenção do papel e da imprensa. Logo depois da invenção dos computadores foi percebido que eles poderiam ser úteis no armazenamento e busca de grandes quantidades de dados [3]. Em 1945, Vannevar Bush publicou um artigo intitulado "As We May Think" [12], considerado o nascimento da idéia do acesso automático a grandes volumes de dados. Em 1950 esta idéia se materializou em descrições concretas de como arquivos de texto poderiam ser buscados automaticamente e vários trabalhos surgiram em meados da década de 50 para consolidar a idéia de se fazer buscas de texto com computadores. Vários outros trabalhos apareceram na década seguinte, sendo o mais notável o desenvolvimento do SMART [13], por Gerard Salton, que permitiu pesquisas para o melhoramento da qualidade das buscas.

Os anos 60 e 70 viram um desenvolvimento ainda maior. Vários modelos de busca de documentos foram desenvolvidos, provando serem efetivos para as relativamente pequenas coleções de textos (alguns milhares de artigos) disponíveis para os pesquisadores na época. Entretanto, permaneceu a dúvida se esses modelos poderiam ser aplicados a coleções muito maiores de documentos. Isto mudaria em 1992 com a criação do TREC (*Text Retrieval Conference*), patrocinado por várias agências governamentais dos Estados Unidos, que fomentou o aperfeiçoamento e o desenvolvimento de muitas novas tecnologias aplicadas a grandes coleções de documentos. Esses algoritmos seriam os primeiros a serem usados nos mecanismos de buscas da Web de 1996 a 1998.

2.1.2 Conceitos

Mecanismos de busca podem ser definidos como sistemas construídos com a finalidade de coletar, analisar e armazenar dados, e realizar buscas posteriores nesses dados armazenados. Esses dados são normalmente oriundos de documentos textuais (como arquivos html, txt, pdf, LaTex, xml, Microsoft Word, Microsoft Excel) [3]. Os mecanismos de busca trabalham indexando os dados para otimizar e dinamizar a sua busca posterior (sem a indexação, todo o conjunto de dados teria que ser verificado inteiramente ao se fazer uma busca). Esses sistemas são utilizados em ambientes que vão desde computadores pessoais, passando por intranets até a própria Internet. O desenvolvimento e implementação desses sistemas envolve a utilização de conceitos de áreas diversas como lingüística, psicologia, matemática e ciência da computação. Existem vários fatores levados em conta no projeto da

arquitetura de um mecanismo de busca, sendo os principais [3]:

- Técnicas de armazenamento: Tratam de como armazenar os dados e quais informações armazenar. Por exemplo, se a informação deve ser filtrada, comprimida, etc.

- Tamanho do índice: Quanto espaço deve ser destinado ao índice.

- Manutenção: Como manter o índice ao longo do tempo, isto é, como manter a sua eficiência ao longo de seu uso.

- Tolerância a falhas: Capacidade de recuperação de falhas de hardware, verificação de integridade dos dados, isolamento de dados corrompidos .

- Velocidade de busca e atualização: A relação entre a velocidade com que um termo pode ser buscado no índice e a velocidade com que ele pode ser atualizado ou removido do índice. Leva em conta também a capacidade do sistema em atualizar o índice enquanto buscas são realizadas nele.

- Estrutura de dados utilizada no índice: Qual estrutura de dados utilizar no índice para obter o melhor resultado de acordo com ênfase dada ao projeto. Existem vários tipos de índices que podem ser utilizados, entre eles destaca-se o *suffix tree*, árvore B, índice invertido, *Citation index* e matriz de termos em documentos [8]. O índice invertido é usado pela maioria dos sistemas, e proporciona acesso rápido a uma lista de documentos que contém um determinado termo junto com outras informações (por exemplo o peso de cada termo no documento, isto é, o número de vezes que o termo ocorre no documento, a posição relativa de cada termo no documento, etc.) [3].

Outro aspecto relevante a ser considerado é a análise de documentos (*Document Parsing*), utilizada para extrair palavras (termos) dos documentos para a formação dos índices. As palavras encontradas são chamadas de *tokens*. Durante esse processo são identificadas as seqüências de caracteres que representam palavras e outros elementos como pontuação, assim como estatísticas referentes aos *tokens* (freqüência, posição no documento, etc.). Também podem ser identificadas outras entidades significativas como endereços de

email, *urls*, números de telefone, códigos postais e outras, a critério do sistema. Palavras consideradas não informativas, chamadas de *stopwords*, como preposições e advérbios são frequentemente ignoradas no processo. A redução de palavras para a forma de radicais também é usada por muitos sistemas, método conhecido como *stemming*. A idéia principal por trás disso é a suposição que usuários buscando por uma palavra como *aviação* também estarão interessados em documentos que tiverem informações sobre *aviador* e *avião*. Mas como efeito colateral, um *stemming* pobre pode levar a erros como por exemplo o retorno de documentos com o termo *aviamentos* [3]. Opcionalmente, durante a análise de documentos, pode ser feito também um reconhecimento de linguagem para saber qual o idioma utilizado no documento (Um dos grandes desafios apresentados hoje para a análise e busca de documentos são justamente a ambigüidade de significado das palavras e a identificação do idioma do documento).

É interessante destacar que novos recursos tem sido desenvolvidos ao longo do tempo para auxiliar cada vez mais o refinamento das buscas pelo usuário. Como exemplo temos o uso de busca por proximidade (usando lógica *fuzzy* e algoritmos baseados na chamada distância de Levenshtein), que permite definir a distância, em termos de ortografia, entre as palavras-chave na busca e os termos nos resultados que se quer. Usando este recurso, por exemplo, uma busca por *castor* retornaria termos semelhantes em ortografia como *pastor* [14]. Outro recurso auxiliar comum utilizado hoje é o *highlight* (destaque) de palavras-chave, onde ocorrências das palavras-chave utilizadas na busca aparecem destacadas em fragmentos de texto exibido nos resultados. Os fragmentos contextuais que aparecem no resultado de cada busca ajudam o usuário a julgar se os resultados são apropriados e se é necessário acrescentar ou remover palavras-chave para refinar a busca. Destaca-se ainda a busca utilizando-se o conjunto dos resultados retornados por uma busca anterior e a busca por documento similares a um determinado documento escolhido pelo usuário.

Um dos aspectos primordiais a ser levado em consideração no desenvolvimento de mecanismos de busca é maximizar a relevância que o conteúdo retornado por uma busca tem para o usuário, o que define a “utilidade” do sistema do ponto de vista do usuário. A maioria dos sistemas cria um "ranking" dos documentos, baseada na estimativa de utilidade que terão para a busca do usuário. Existem diversos modelos para efetuar esse processo entre os quais o modelo probabilístico e o *vector space model* [3]. O primeiro é baseado no princípio de que documentos em uma coleção devem ser ranqueados pela diminuição da probabilidade de sua relevância numa busca. Assim, um sistema estimaria a probabilidade da relevância dos

documentos para uma busca, sendo que a implementação desta estimativa varia muito de sistema para sistema. No segundo, o texto é representado como um vetor de termos (tipicamente palavras ou frases). Se palavras são escolhidas como termos então cada palavra no vocabulário se torna uma dimensão independente num vetor muito grande (número de dimensões igual ao tamanho do vocabulário). Assim qualquer texto pode ser representado como um vetor. Se um termo pertence a um texto, ele toma um valor não-nulo no vetor. Desde de que qualquer texto contém um número limitado de termos, a maioria dos vetores se torna muito esparsa. Para a classificação de um documento, é medida a similaridade entre o vetor da *query* do usuário e do vetor do documento. Esta similaridade entre os vetores varia de sistema para sistema, mas geralmente é calculada pelo ângulo entre os dois vetores e o co-seno do ângulo é usado como o valor numérico da similaridade. Assim dois vetores idênticos teriam o co-seno igual a um e dois completamente diferentes teriam o co-seno igual a zero.

Existem também diversos parâmetros técnicos utilizados para se medir a eficiência de um sistema de busca relativamente à relevância do conteúdo retornado para o usuário [4]. Entre eles podemos destacar os mais importantes e utilizados de maneira geral:

- Precisão: Fração dos documentos retornados que são relevantes ao usuário. Pode também ser avaliado em apenas um subconjunto dos documentos retornados, considerando-se apenas os melhores resultados retornados. Pode ser visto como uma medida indicativa de exatidão ou fidelidade.

$$Precisão = \frac{|\{documentos\ relevantes\} \cap \{documentos\ retornados\}|}{|\{documentos\ retornados\}|}$$

- *Recall*: Fração dos documentos que são relevantes para o critério de busca (*query*) e que são retornados com sucesso. Pode ser entendido como a probabilidade de se retornar um documento relevante numa *query*.

$$Recall = \frac{|\{documentos\ relevantes\} \cap \{documentos\ retornados\}|}{|\{documentos\ relevantes\}|}$$

- *Fall-Out*: Fração dos documentos não relevantes que são retornados de todos os documentos não relevantes disponíveis. Pode ser entendido como a probabilidade de

que um documento não relevante seja retornado numa *query*.

$$Fall\text{-}Out = \frac{|\{documentos\ não - relevantes\} \cap \{documentos\ retornados\}|}{|\{documentos\ não - relevantes\}|}$$

Exemplificando o que foi exposto anteriormente, vamos supor uma busca por documentos que contenham um determinado termo, e que num conjunto de 1000 documentos, apenas 205 deles efetivamente contém esse termo. Supondo que a busca retorne 210 documentos, sendo que 200 deles contém o termo e 10 não contém o termo, então a precisão da busca seria de 0.95 (a fração 200/210) . O *recall* seria de 0.97 (a fração 200/205) e o *fall-out* seria de 0.012 (a fração 10/(1000-205)). Usualmente, a precisão e o *recall* não são discutidos isoladamente, mas ao invés disso um valor medido de um parâmetro é comparado com um valor fixo do outro parâmetro medido previamente. Muitas vezes também, há uma relação inversa entre estes dois parâmetros, sendo possível aumentar um ao custo de se diminuir o outro [4]. Um sistema de busca poderia aumentar o *recall* retornando mais documentos, ao custo de aumentar o número de documentos irrelevantes retornados.

2.1.3 Mecanismos de Busca e a Web

Como exemplo principal e mais difundido de mecanismos de busca destaca-se os sistemas de busca para a Web, que serão descritos aqui para exemplificar os mecanismos de busca de uma forma geral.

Os sistemas de busca para a Web trabalham armazenando informações detalhadas sobre páginas e arquivos com conteúdo textual de uma base de sítios pré-definida para ser indexada. Estas páginas e arquivos são recuperados por um mecanismo chamado *Web crawler*, um sistema que segue os *links* que encontra em cada página e segue por eles procurando outros arquivos e páginas relacionadas. De maneira geral, novos sítios não precisam ser necessariamente "enviados" aos sistemas de busca para serem listados. Um simples *link* vindo de um sítio já estabelecido fará com que os sistemas de busca visitem o novo sítio e inicie o mapeamento do conteúdo. O conteúdo de cada página é analisado para que se possa determinar como será indexado na base de busca (por exemplo, palavras extraídas de títulos, cabeçalhos ou campos *meta-tags*). Os dados sobre as páginas e arquivos são então armazenados na base de busca para uso posterior nas pesquisas. Alguns sistemas, como o do *Google*, por exemplo, armazenam parte da página original (um *cache*) assim como informações sobre as páginas. Quando um usuário faz uma busca o sistema procura no índice

e fornece um resultado com uma lista das páginas que melhor resolvem o critério de busca utilizado, normalmente com um breve resumo contendo o título do documento e, às vezes, partes do seu texto. Enquanto pode haver milhões de páginas que incluam uma palavra ou texto em particular, alguns resultados podem ser mais relevantes para o usuário do que outros. A maioria dos sistemas de busca da Web utiliza métodos para criar um *ranking* dos resultados, para retornar os "melhores" resultados de acordo com os critérios de busca do usuário, ordenando os resultados pela estimativa da utilidade que terão para esses usuários. A forma como um sistema de busca decide quais páginas são mais relevantes, e em qual ordem os resultados aparecerão, varia muito de um sistema para outro e também se alteram ao longo do tempo, enquanto o uso da Internet muda e novas técnicas evoluem.

Cho et al [5] descreveram alguns padrões para serem utilizados no processo de indexação e pesquisas de páginas da Web, os quais são listados a seguir. Esses são exemplos de padrões característicos dos mecanismos de busca para Web, mas servem perfeitamente para ilustrar a diversidade de parâmetros de classificação e pesquisa que podem ser utilizados pelos mecanismos de busca de forma geral, para obter conteúdo para indexação.

- Tempo de registro do domínio (Idade do domínio). Quanto mais antigo o domínio, mais “confiável” ele é.
- Idade do *link* e ranqueamento do sítio que aponta para ele.
- Idade do conteúdo. A preferência é por conteúdo mais antigo, que exista a mais tempo (desde que não seja desatualizado).
- Freqüência de atualização do conteúdo: regularidade com a qual novo conteúdo é adicionado. Neste caso quanto mais atualizado o conteúdo, melhor.
- Termos relevantes utilizados no conteúdo (os termos que os buscadores associam como sendo relacionados ao tema ou conteúdo principal da página).
- Quantidade de *links* externos. Quanto mais, melhor.
- Citações e fontes de pesquisa (indica que o conteúdo é de qualidade para pesquisa ou referência).
- Existência de termos relacionados na base de dados do mecanismo de busca (como financiar/financiamento).
- Ritmo de crescimento dos *links* de entrada: muitos ou incremento muito rápido pode indicar atividade de comercial de compra e venda de *links*.
- Textos descritivos próximos aos *links* que apontam para fora e dos *links* de entrada.

Um link acompanhado de palavras como "*links* patrocinados" ou “anunciantes” pode ser ignorado.

- Profundidade do documento na taxonomia do sítio. Quanto maior a profundidade, maior é a tendência de que o sítio apresente conteúdo relevante, isto é, os assuntos se desdobram em várias páginas e subcategorias.
- Métricas coletadas de outras fontes, tais como monitoramento da freqüência com a qual usuários retornam.
- Métricas coletadas de fontes consideradas como referência como Google AdWords/Adsense, etc.
- Métricas coletadas de compartilhamento de dados com terceiros (como provedores de dados estatísticos de programas utilizados para monitorar tráfego de sítios).
- Ritmo de remoção de *links* que apontam para o sítio. Uma grande atividade deste tipo pode indicar que o sítio tem algo de errado.
- IP do serviço de hospedagem e o número/qualidade dos demais sítios hospedados lá.
- Uso adequado do arquivo robots.txt, o qual é um arquivo que indica quais *links* devem ser excluídos da análise de indexação em um sítio. Também é conhecido como padrão de exclusão de robôs, e trata-se de uma convenção para prevenir que mecanismos de busca acessem parte ou todo o conteúdo de um sítio, seja por razões de privacidade, crença de que o conteúdo excluído é irrelevante para a categorização do sítio ou outros.
- Se o sítio mostra conteúdo diferente a diferentes tipos de usuários ou *crawlers* (indicação de manipulação).
- *Links* para páginas que não existem. Quanto maior a quantidade, menor a chance de indexação da página.
- Conteúdo inseguro, ilegal ou pornográfico diminui a chance de indexação.
- Qualidade da codificação do html da página e presença de erros no código. Quanto mais erros de html, mais difícil pode ser a indexação correta do conteúdo.
- Classificação de importância feita por pessoas nas páginas com acessos mais freqüentes.

2.2 Sítios Especializados que Disponibilizam Currículos e/ou Vagas

Existem vários sítios que disponibilizam serviços relativos a cadastramento e busca de vagas e currículos na Internet, podendo ser caracterizados assim como um sistema de busca

especializado nessa área. De forma geral todos são muito semelhantes em suas características, apresentando entretanto algumas diferenças operacionais. A seguir é exibido um resumo de alguns dos principais sítios do país e suas características gerais mais importantes do ponto de vista do usuário. Vale lembrar que estas características se apresentavam da forma descrita abaixo até a data de conclusão deste trabalho, mas pode ser que sejam alteradas ao longo do tempo de acordo com a política de cada sítio.

2.2.1 O Sítio Catho Online

Este sítio possui busca por área profissional, subdividida em categorias, por cargo, estado e cidade. O cadastro de currículos é gratuito, por tempo limitado. A busca de empregos e vagas é gratuita, mas a busca por currículos é gratuita apenas por um período de tempo limitado.

Categorias disponíveis: Administração, Agricultura, Pecuária e Veterinária, Artes, Arquitetura e Design, Comercial e Vendas, Comunicação, Comércio Exterior, Educação, Engenharia, Financeira, Hotelaria e Turismo, Industrial, Informática, jurídica, Saúde, Suprimentos, Telecomunicações, Telemarketing, Técnica, Outros. Cada categoria é dividida em subcategorias como por exemplo, a categoria "Agricultura, Pecuária e Veterinária" subdividida em: "Agronômica / Engenharia Agronômica/ Agribusiness", "Agropecuária/ Veterinária", "Biologia e Zootecnia".

2.2.2 O Sítio UOL Empregos

Este sítio possui busca de vagas por área de atuação, estado e por área profissional, subdividida em categorias. Adicionalmente, também atua na divulgação de notícias e artigos na área de gestão de RH. Categorias: Administrativa, Agronegócios, Arquitetura, Artes Gráficas, Biologia, Comercial e Vendas, Comércio Exterior, Compras, Comunicação e Publicidade, Contabilidade, Departamento de pessoal.

2.2.3 A Plataforma Lattes

A Plataforma Lattes, focada principalmente na área acadêmica, possui busca de currículos por estado e nível de escolaridade. Representa a experiência do CNPq na integração de bases de dados de currículos e de instituições da área de ciência e tecnologia em um único sistema de informações, cuja importância atual se estende, não só às atividades operacionais de fomento do CNPq, como também às ações de fomento de outras agências federais e estaduais.

2.2.4 O Sítio Timaster

Este é um exemplo de sítio especializado, focado na área de informática. Possui opções de busca de vagas por área de atuação, estado ou palavra-chave (em diversos campos, como título do cargo, cidade e outros). Permite o cadastro gratuito de currículos. Segundo a política de uso do sítio, uma empresa usuária do serviço só pode ler os currículos de profissionais interessados em uma oferta anunciada por ela própria, e a iniciativa do contato é sempre do candidato.

Categorias: Infra-Estrutura/Suporte, desenvolvimento/Sistemas, Criação/Conteúdo, Marketing/Vendas.

2.2.5 Comparação de Características

A TABELA 1 a seguir apresenta a comparação de algumas características funcionais dos sítios mencionados acima e do sistema de banco de currículos apresentado neste trabalho.

	Catho Online	UOL Empregos	Plataforma Lattes	Timaster	SBC (sistema de banco de currículos)
Possui busca de empregos ?	sim	sim	não	sim	sim
Possui busca de currículos ?	sim	sim	sim	sim	sim
Possui divisão de áreas profissionais em categorias ?	sim	sim	sim	sim	sim
Permite cadastro gratuito de currículos ?	sim (por tempo limitado)	não	sim	sim	sim
Focado em um segmento profissional específico?	não	não	não	sim	não
Possui recurso de destaque de palavras-chave na exibição dos resultados de busca ?	não	não	não	não	sim

Possui recurso de busca por similaridade dos resultados ?	não	sim	não	sim	não
Possui exibição de estatísticas sobre usuários ?	não	não	sim	não	sim
Possui algum sistema de divulgação de vagas via email ?	sim	não	não	sim	não

TABELA 1: Tabela comparativa de características dos sítios.

Capítulo 3

Especificação e Modelagem

Este capítulo apresenta o levantamento de requisitos do projeto na Seção 3.1, o modelo de casos de uso e a descrição de cada um deles na Seção 3.2. Apresenta também os modelos de classes na Seção 3.3, modelo de interações na Seção 3.4, e o projeto de banco de dados do sistema na Seção 3.5.

3.1 Especificação dos Requisitos

3.1.1 Descrição dos Requisitos Funcionais

- RF01 - O Administrador do sistema pode editar e excluir usuários.
- RF02 - O usuário pessoa física pode fazer pesquisas por vagas.
- RF03 - O usuário pessoa jurídica pode cadastrar, alterar e excluir vagas de emprego.
- RF04 - O usuário pessoa física pode se candidatar a vagas cadastradas por usuários pessoa jurídica.
- RF05 - O usuário pessoa jurídica pode visualizar um relatório de estatísticas sobre os usuários pessoa física.
- RF06 - O usuário pessoa física pode visualizar um relatório com as últimas candidaturas dele às vagas publicadas.
- RF07 - O usuário pessoa jurídica pode visualizar um relatório sobre os usuários pessoa física que se candidataram a uma determinada vaga publicada por ele.
- RF08 - O usuário pessoa jurídica pode fazer pesquisas por currículos.
- RF09 - Cada usuário, pessoa física ou jurídica, pode editar as suas informações de cadastro.
- RF10 – Visitantes podem realizar seu cadastro no sistema.

3.1.2 Descrição dos Requisitos Não Funcionais

- RNF01 - O sistema deve ser capaz de ser utilizado em pelo menos dois tipos de sistemas operacionais diferentes.
- RNF02 - O sistema deve ser "fácil" de usar. Operações pertinentes ao sistema devem ser ágeis, intuitivas e práticas.

3.1.2.1 Requisitos de Segurança

- RS01 - Para entrar no sistema é necessário ter cadastro e estar logado.
- RS02 - As senhas dos usuários deverão ser guardadas no banco criptografadas usando o algoritmo Md5.
- RS03 - Somente o usuário do tipo administrador pode excluir usuários.

3.1.3 Descrição das Regras de Negócio

- RN01 - Cada pessoa jurídica pode cadastrar quantas vagas quiser.
- RN02 - Cada pessoa física pode se candidatar a quantas vagas quiser.
- RN03 - O usuário pessoa física visualiza um relatório com as 20 últimas vagas às quais ele se candidatou.
- RN04 - As estatísticas sobre os usuários pessoa física são apresentadas em forma de gráfico de porcentagens.
- RN05 - As estatísticas exibidas sobre os usuários pessoa física são: Porcentagens de candidatos por estado, por escolaridade, por faixa de idade e por área profissional.
- RN06 – As faixas de idade para a estatística por faixa de idade são : 18 a 25 anos, 26 a 35 anos, 35 a 45 anos, 46 a 55 anos, 56 a 65 anos e maior que 65 anos.
- RN07 - Os resultados de buscas devem conter o recurso de destaque das palavras-chave (*highlight*) no contexto dos resultados quando possível.

3.2 Modelo de Casos de Uso

Um Caso de Uso representa uma unidade discreta da interação entre um usuário (humano ou máquina) e o sistema. É uma unidade de um trabalho significante. Cada Caso de Uso tem uma descrição, a qual descreve a funcionalidade construída no sistema proposto [6]. Um Caso de Uso pode "incluir" outra funcionalidade de Caso de Uso ou "estender" outro Caso de Uso com seu próprio comportamento. São tipicamente relacionados a "atores". Um ator é um humano ou entidade máquina que interage com o sistema para executar um trabalho significante. Evitam o uso de termos técnicos, preferindo a linguagem do usuário final.
A FIGURA 1 mostra o modelo de casos de uso do sistema.

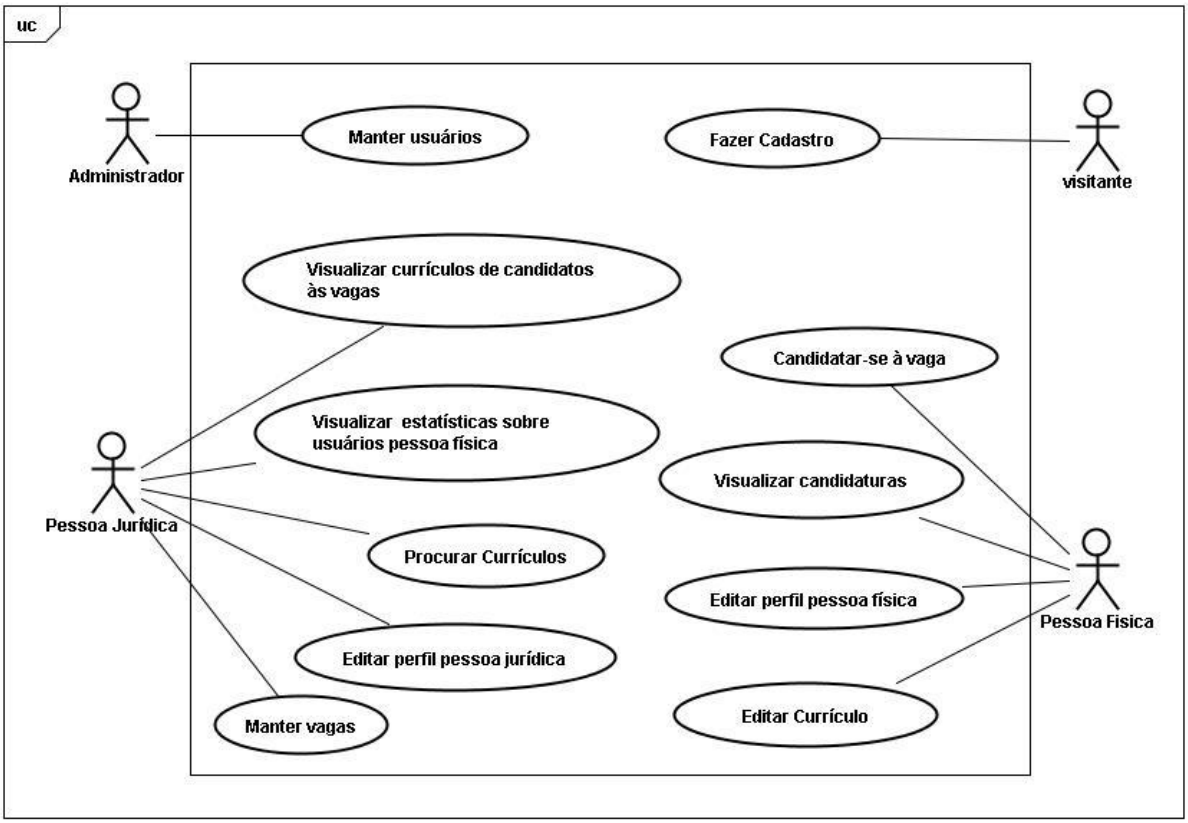

FIGURA 1: Diagrama de Casos de Uso

3.2.1 Descrição dos Atores

Administrador:

Representa o(s) administrador(es) do sistema.

Visitante:

Representa os usuários não-cadastrados no sistema. Podem apenas efetuar seu cadastro como pessoa física ou jurídica para posteriormente acessar as demais funcionalidades do sistema.

Pessoa Jurídica:

Representa os usuários do tipo pessoa jurídica cadastrados no sistema.

Pessoa Física:

Representa os usuários do tipo pessoa física cadastrados no sistema.

3.2.2 Descrição dos Casos de Uso

3.2.2.1 Manter usuário (CSU01)

Ator:

Administrador.

Sumário:

O Administrador pode efetuar as operações de CRUD para que seja mantido o cadastro do usuário.

Pré-condição:

O usuário do tipo administrador deve estar previamente cadastrado no sistema.

Fluxo Principal

1) O Administrador solicita a manutenção de usuário.

2) O sistema apresenta as operações de (CRUD).

3) O Administrador realiza a operação desejada.

4) Caso de uso é finalizado.

Fluxos Alternativos, a partir do passo 3:

Fluxo Alternativo 1 (Consultar)

3.1) O Administrador solicita a consulta de usuário.

3.2) O sistema apresenta um formulário para a busca do usuário.

3.3) O Administrador preenche os campos e confirma a consulta.

3.4) O sistema retorna uma lista com os usuários encontrados.

Fluxos Alternativos, a partir do passo 3.4:

Fluxo Alternativo 2 (Editar)

3.5) O Administrador seleciona a opção editar.

3.6) O Administrador altera os dados desejados e confirma a alteração.

Fluxo Alternativo 3 (Excluir)

3.5) O Administrador seleciona a opção excluir e confirma a exclusão.

3.2.2.2 Fazer cadastro (CSU02)

Ator:

visitante

Sumário:

O visitante pode fazer o seu cadastro no sistema.

Fluxo principal:

1) O visitante solicita a inserção de novo usuário.

2) O sistema apresenta um formulário para a inserção dos dados relativos ao usuário.

3) O visitante preenche o formulário e confirma o cadastro.

3.2.2.3 Editar currículo (CSU03)

Ator: Pessoa física

Sumário:

O usuário pessoa física pode editar o seu currículo no sistema

Pré-condição:

Usuário pessoa física previamente cadastrado.

Fluxo principal:

1) O usuário pessoa física solicita a edição do seu currículo

2) O sistema apresenta um formulário para a edição dos dados relativos ao seu currículo.

3) O usuário pessoa física altera os dados desejados e confirma a alteração.

3.2.2.4 Editar perfil pessoa física (CSU04)

Ator: Pessoa física

Sumário:

O usuário pessoa física pode editar o seu cadastro no sistema.

Pré-condição:

Usuário pessoa física previamente cadastrado.

Fluxo principal:

1) O usuário solicita a edição do seu cadastro

2) O sistema apresenta um formulário para a edição dos dados relativos ao cadastro do usuário.

3) O usuário altera os dados desejados e confirma a alteração.

3.2.2.5 Manter vagas (CSU05)

Ator:

Pessoa jurídica

Sumário:

O usuário pessoa jurídica pode efetuar as operações de CRUD no cadastro de vagas
Pré-condição:
O usuário do tipo pessoa jurídica deve estar previamente cadastrado no sistema.
Fluxo Principal:
1) O usuário pessoa jurídica solicita a manutenção de vagas.
2) O sistema apresenta as operações de CRUD.
3) O usuário pessoa jurídica realiza a operação desejada.
4) Caso de uso é finalizado.

Fluxos Alternativos, a partir do passo 3:
Fluxo Alternativo 1 (Cadastrar)
3.1) O usuário pessoa jurídica solicita a inserção de nova vaga.
3.2) O sistema apresenta um formulário para a inserção dos dados da vaga.
3.3) O usuário pessoa jurídica preenche o formulário e confirma o cadastro.

Fluxo Alternativo 2 (Consultar)
3.1) O usuário pessoa jurídica solicita a consulta de vagas.
3.2) O sistema retorna uma lista com todas as vagas cadastradas por ele.

Fluxo Alternativo 3 (Editar)
3.5) O usuário pessoa jurídica seleciona a opção editar.
3.6) O usuário pessoa jurídica altera os dados desejados e confirma a alteração.

Fluxo Alternativo 4 (Excluir)
3.5) O usuário pessoa jurídica seleciona a opção excluir e confirma a exclusão.

3.2.2.6 Procurar currículos (CSU06)

Ator:
Pessoa jurídica
Sumário:
O usuário pessoa jurídica pode efetuar buscas na base de currículos cadastrados.
Pré-condição:

Usuário pessoa jurídica previamente cadastrado.

Fluxo Principal:

1) O usuário pessoa jurídica solicita a busca de currículos.

2) O sistema apresenta um formulário para a busca de currículos.

3) O usuário pessoa jurídica preenche os campos e confirma a consulta.

4) O sistema retorna uma lista com os currículos encontrados.

3.2.2.7 Visualizar candidaturas (CSU07)

Ator: Pessoa física

Sumário:

O usuário pessoa física pode visualizar um relatório exibindo as últimas vagas às quais ele se candidatou.

Pré-condição:

Usuário pessoa física previamente cadastrado.

Fluxo principal:

1) O usuário pessoa física solicita a lista de vagas as quais ele se candidatou.

2) O sistema apresenta uma lista contendo os títulos das vagas e as datas nas quais o usuário se candidatou.

3.2.2.8 Visualizar estatísticas sobre usuários pessoa física (CSU08)

Ator:

Pessoa jurídica

Sumário:

O usuário pessoa jurídica pode visualizar as estatísticas relativas aos usuários pessoa física cadastrados no sistema.

Pré-condição:

Usuário pessoa jurídica previamente cadastrado.

Fluxo principal:

1) O usuário pessoa jurídica solicita a exibição de estatísticas ao sistema.

2) O sistema apresenta as opções disponíveis.

3) O usuário pessoa jurídica escolhe a opção desejada.

4) O sistema apresenta as informações solicitadas.

3.2.2.9 Visualizar currículos de candidatos às vagas (CSU09)

Ator:

Pessoa jurídica

Sumário:

O usuário pessoa jurídica pode visualizar os currículos de candidatos às vagas publicadas por ele.

Pré-condição:

Usuário pessoa jurídica previamente cadastrado.

Fluxo principal:

1) O usuário pessoa jurídica solicita a lista com as vagas publicadas por ele.
2) O sistema apresenta uma lista com as vagas.
3) O usuário pessoa jurídica escolhe uma vaga e solicita a lista dos candidatos a ela.
4) O sistema apresenta uma lista com os nomes dos candidatos.
5) O usuário pessoa jurídica escolhe um candidato e solicita a exibição do seu currículo.
6) O sistema exibe o currículo do candidato.

Fluxo alternativo a partir do passo 3:

3.1) Se não houver candidatos o sistema exibe uma mensagem comunicando o fato.

3.2.2.10 Candidatar-se à vaga (CSU10)

Ator:

Pessoa física

Sumário:

O usuário pessoa física pode buscar por vagas cadastradas e se candidatar a elas.

Pré-condição:

Usuário pessoa física previamente cadastrado.

Fluxo principal:

1) O usuário pessoa física solicita a consulta de vagas.
2) O sistema apresenta um formulário para a busca de vagas.
3) O usuário pessoa física preenche os campos e confirma a consulta.
4) O sistema retorna uma lista com as vagas encontradas.

5) O usuário pessoa física escolhe uma vaga e solicita a exibição das informações sobre ela.
6) O sistema retorna um formulário com as informações sobre a vaga.
7) O usuário pessoa física solicita o cadastro.
8) O sistema confirma o cadastro.

Fluxo alternativo a partir do passo 6:
6.1)Se o usuário não quiser efetuar o cadastro o caso de uso é encerrado.

3.2.2.11 Editar perfil pessoa jurídica (CSU11)

Ator: pessoa jurídica
Sumário:
O usuário pessoa jurídica pode editar o seu cadastro no sistema.
Pré-condição:
Usuário pessoa jurídica previamente cadastrado.
Fluxo principal:
1) O usuário solicita a edição do seu cadastro
2) O sistema apresenta um formulário para a edição dos dados relativos ao cadastro do usuário.
3) O usuário altera os dados desejados e confirma a alteração.

3.3 Modelo de Classes

Um diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para os objetos. É uma modelagem muito útil para o sistema, pois define todas as classes que o mesmo necessita possuir e é a base para a construção de outros diagramas [6].
A FIGURA 2 a seguir mostra o modelo de classes do sistema.

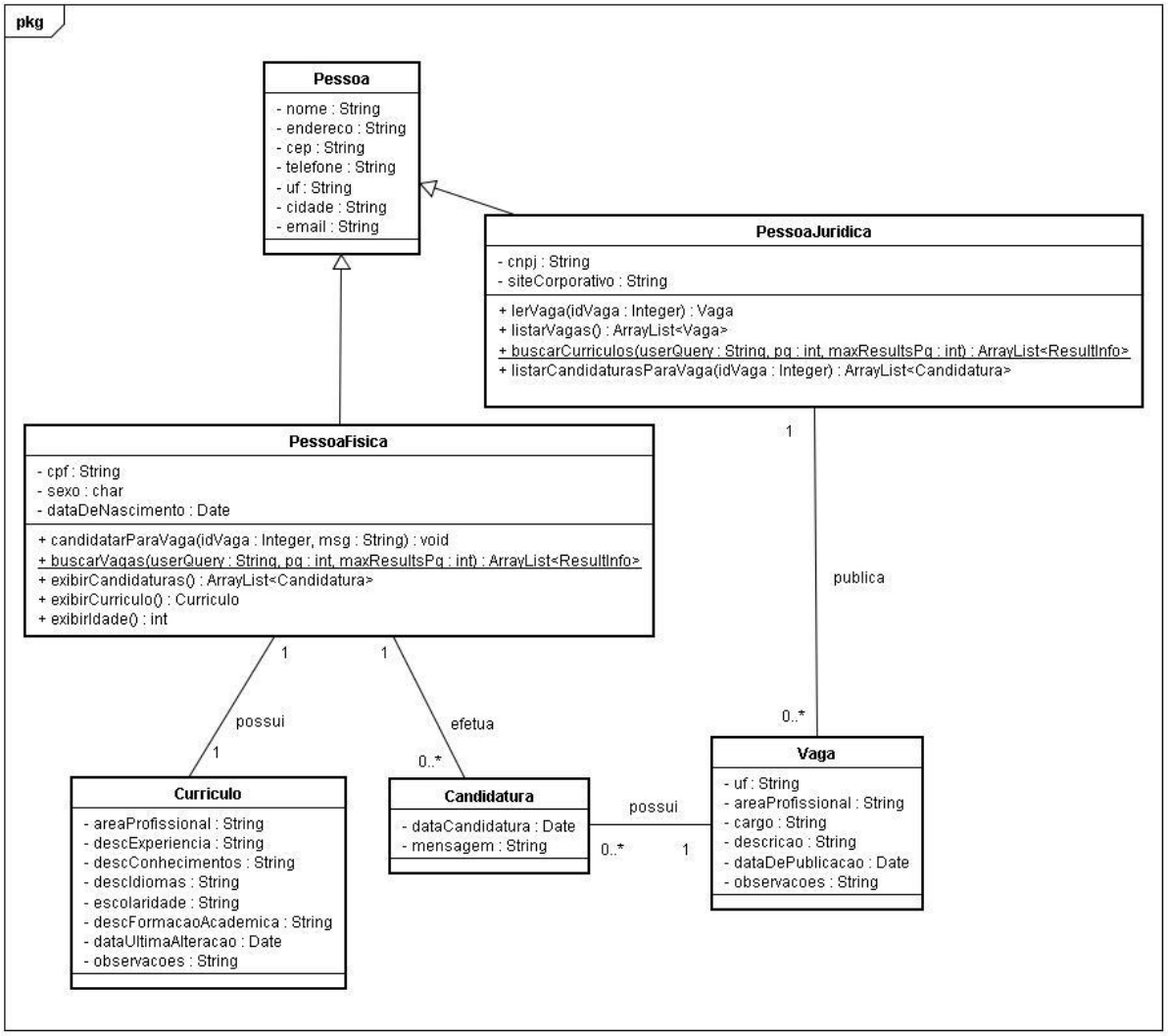

FIGURA 2: Diagrama de Classes

3.3.1 Dicionário de Classes do Diagrama Conceitual

Sumário das classes de negócio do sistema	
Pessoa	Classe que representa as pessoas (atributos comuns).
PessoaFisica	Classe que representa as pessoas físicas.
PessoaJuridica	Classe que representa as pessoas jurídicas.
Candidatura	Classe que representa as candidaturas de pessoas físicas às vagas.
Curriculo	Classe que representa os currículos de pessoas físicas.
Vaga	Classe que representa as vagas publicadas pelas pessoas jurídicas.

TABELA 2: Sumário das Classes

3.3.1.1 Pessoa

Classe que representa as pessoas (atributos comuns), apresentada na TABELA 3.

Atributos		
Tipo	**Assinatura**	**Descrição**
String	nome	Nome da pessoa.
String	endereco	Endereço da pessoa.
String	cep	Código de endereçamento postal(cep) da pessoa.
String	telefone	Telefone da pessoa.
String	uf	Sigla da federação (estado) onde mora a pessoa.
String	cidade	Cidade onde mora a pessoa.
String	email	E-mail da pessoa.

TABELA 3: Estrutura da Classe Pessoa.

3.3.1.2 PessoaFisica

Classe que representa as pessoas físicas, apresentada na TABELA 4.

Atributos		
Tipo	**Assinatura**	**Descrição**
String	cpf	CPF da pessoa física.
char	sexo	Sexo da pessoa (M/F).
Date	dataDeNascimento	Data de nascimento da pessoa.
Curriculo	curriculo	Currículo da pessoa.
Collection <Candidatura>	candidaturas	Lista de candidaturas da pessoa.
Métodos		
Retorno	**Assinatura**	**Descrição**
Void	candidatarParaVaga(Integer idVaga , String msg)	Método para efetuar a candidatura da pessoa física.
ArrayList<ResultInfo>	buscarVagas(String userQuery , int pg, int maxResultsPg)	Método estático para efetuar a busca por vagas, com paginação de resultados.
int	exibirIdade()	Método que calcula e exibe a idade da pessoa.
ArrayList <Candidatura>	exibirCandidaturas()	Retorna uma lista com as candidaturas da pessoa.
Curriculo	exibirCurriculo()	Exibe o currículo da pessoa.

TABELA 4: Estrutura da Classe PessoaFisica

3.3.1.3 PessoaJuridica

Classe que representa as pessoas jurídicas, apresentada na TABELA 5.

Atributos		
Tipo	**Assinatura**	**Descrição**
String	cnpj	Cnpj da empresa.
String	sítioCorporativo	Endereço do sítio da empresa.
Collection<Vaga>	vagas	Vagas publicadas pela empresa.

Métodos		
Retorno	**Assinatura**	**Descrição**
Vaga	lerVaga(Integer idVaga)	Retorna a vaga, com o id igual a idVaga, publicada pela pessoa.
ArrayList<Vaga>	listarVagas()	Retorna uma lista com as vagas publicadas pela pessoa.
ArrayList <ResultInfo>	buscarCurriculos(String userQuery , int pg, int maxResultsPg)	Método estático para efetuar a busca por currículos, com paginação de resultados.
ArrayList <Candidatura>	listarCandidaturasParaVaga(Integer idVaga)	Retorna a lista de candidaturas, para a vaga com id igual a idVaga, publicada pela pessoa.

TABELA 5: Estrutura da Classe PessoaJuridica

3.3.1.4 Candidatura

Classe que representa as candidaturas, apresentada na TABELA 6.

Atributos		
Tipo	**Assinatura**	**Descrição**
Date	dataCandidatura	Data de realização da candidatura.
String	mensagem	Mensagem que o candidato pode enviar à empresa ao se candidatar.
PessoaFisica	pessoafisica	Pessoa física associada à candidatura.
Vaga	vaga	Vaga associada à candidatura.

TABELA 6: Estrutura da Classe Candidatura

3.3.1.5 Curriculo

Classe que representa os currículos, apresentada na TABELA 7.

Atributos		
Tipo	**Assinatura**	**Descrição**
String	areaProfissional	Área profissional do candidato.
String	descExperiencia	Descrição da experiência profissional do candidato.
String	descConhecimentos	Descrição dos conhecimentos profissionais do candidato.
String	descIdiomas	Descrição dos conhecimentos de idiomas do candidato.
String	escolaridade	Escolaridade do candidato.
String	descFormacaoAcademica	Descrição da formação acadêmica do candidato (instituições que cursou, ano de conclusão, etc.).
Date	dataUltimaAlteracao	Data da última alteração do currículo.
String	observacoes	Observações complementares que o candidato pode fazer.

int	idade	Idade do candidato.
PessoaFisica	pessoafisica	Pessoa física associada ao currículo.

TABELA 7: Estrutura da Classe Currículo

3.3.1.6 Vaga

Classe que representa as vagas publicadas, apresentada na TABELA 8.

Atributos		
Tipo	**Assinatura**	**Descrição**
String	uf	Sigla da federação (estado) do local de trabalho.
String	areaProfissional	Área profissional alvo da vaga.
String	cargo	Descrição do cargo.
String	descricao	Descrição geral da vaga.
Date	dataDePublicacao	Data de publicação da vaga.
String	observacoes	Observações sobre a vaga
PessoaJuridica	pessoajuridica	Empresa relacionada.
Collection <Candidatura>	candidaturas	Candidaturas relacionadas.

TABELA 8: Estrutura da Classe Vaga

3.3.2 Visões de Classes Participantes (VCP)

Uma Visão de Classes Participantes (VCP) é um diagrama das classes cujos objetos participam da realização de determinado caso de uso. Este diagrama está geralmente relacionado com diagramas de interações.

3.3.2.1 VCP Candidatar-se à vaga

A FIGURA 3 abaixo representa a visão de classes participantes denominada Candidatar-se à vaga, referente à funcionalidade descrita na seção 3.2.2.10. Os objetos da classe ResultInfo são objetos auxiliares para exibição dos resultados da busca para o usuário.

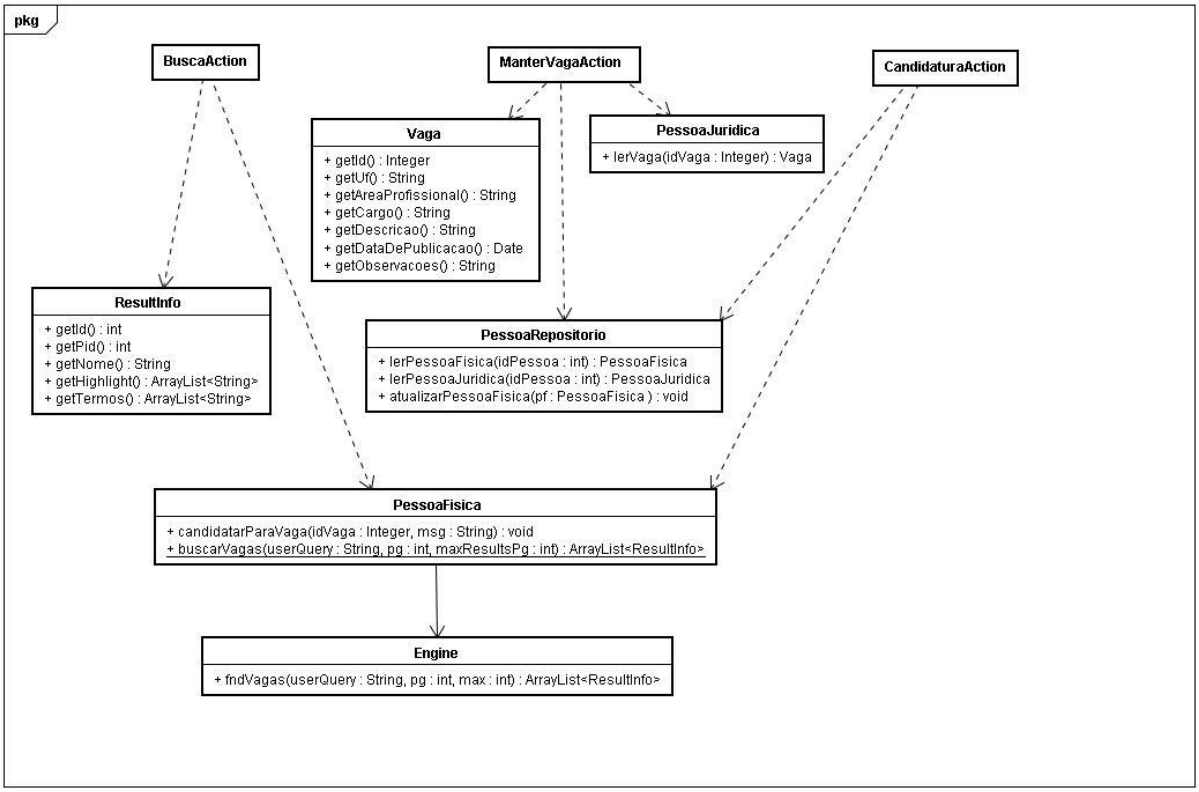

FIGURA 3: VCP – Candidatar-se à vaga

3.3.2.2 VCP Procurar currículos

A FIGURA 4 abaixo representa a visão de classes participantes denominada Procurar currículos, referente à funcionalidade descrita na seção 3.2.2.6. Os objetos da classe ResultInfo são objetos auxiliares para exibição dos resultados da busca para o usuário.

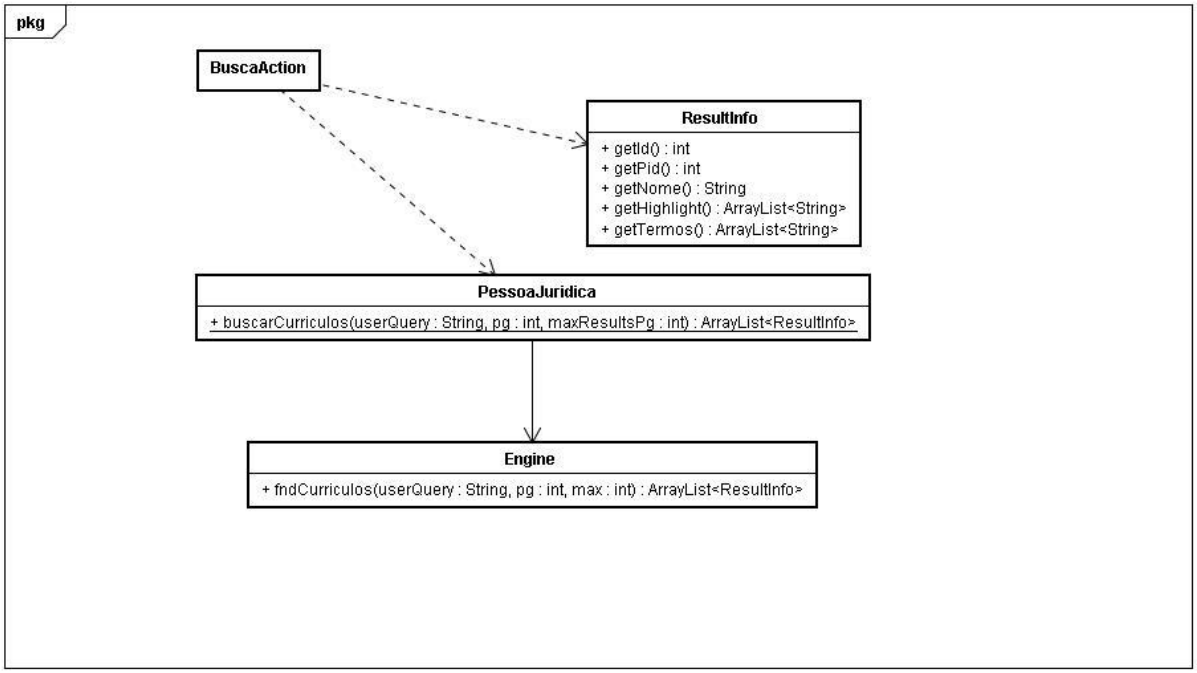

FIGURA 4: VCP – Procurar currículos

3.3.2.3 VCP Visualizar candidaturas

A FIGURA 5 abaixo representa a visão de classes participantes denominada Visualizar candidaturas, referente à funcionalidade descrita na seção 3.2.2.7.

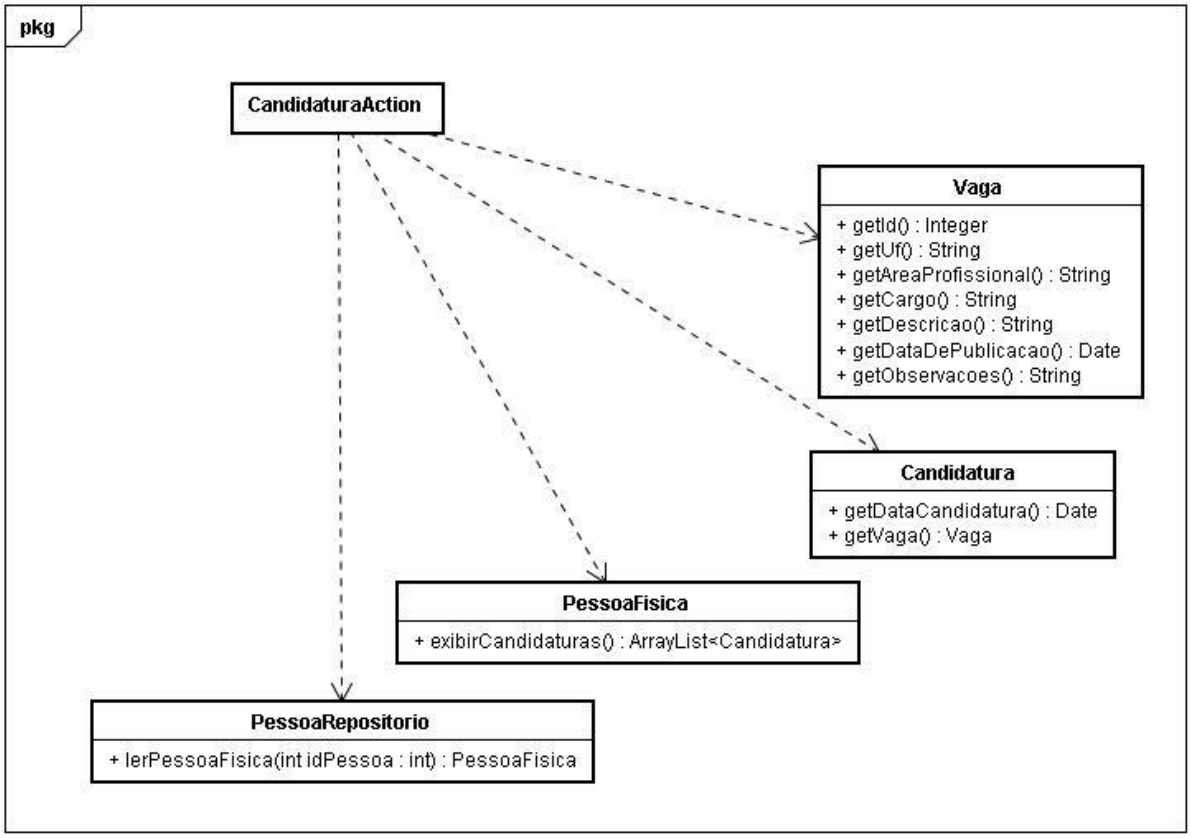

FIGURA 5: VCP – Visualizar candidaturas

3.3.2.4 VCP Visualizar currículos de candidatos às vagas

A FIGURA 6 abaixo representa a visão de classes participantes denominada Visualizar currículos de candidatos às vagas publicadas, referente à funcionalidade descrita na seção 3.2.2.9.

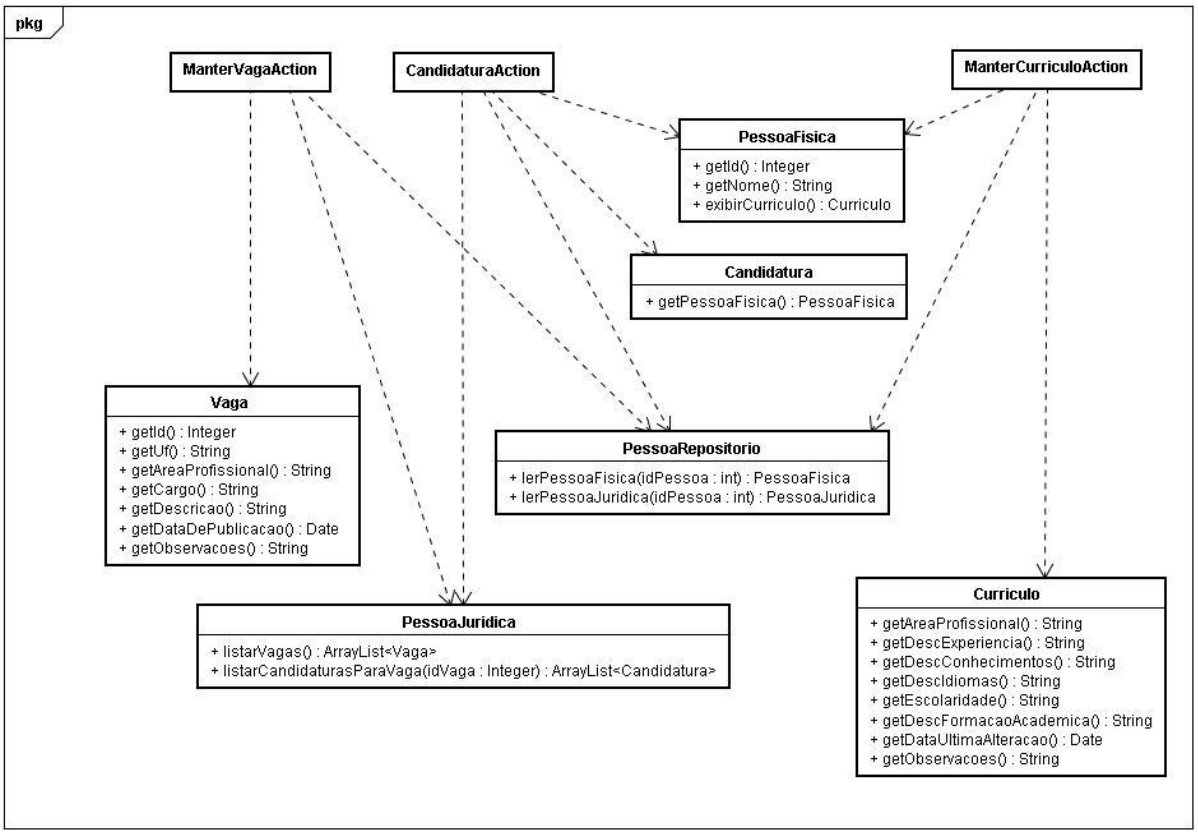

FIGURA 6: VCP – Visualizar currículos de candidatos às vagas .

3.3.2.5 VCP Visualizar estatísticas sobre usuários pessoa física

A FIGURA 7 abaixo representa a visão de classes participantes denominada Visualizar estatísticas sobre usuários pessoa física, referente à funcionalidade descrita na seção 3.2.2.8. Os objetos da classe JFreeChart são utilizados para a geração e exibição dos gráficos para o usuário.

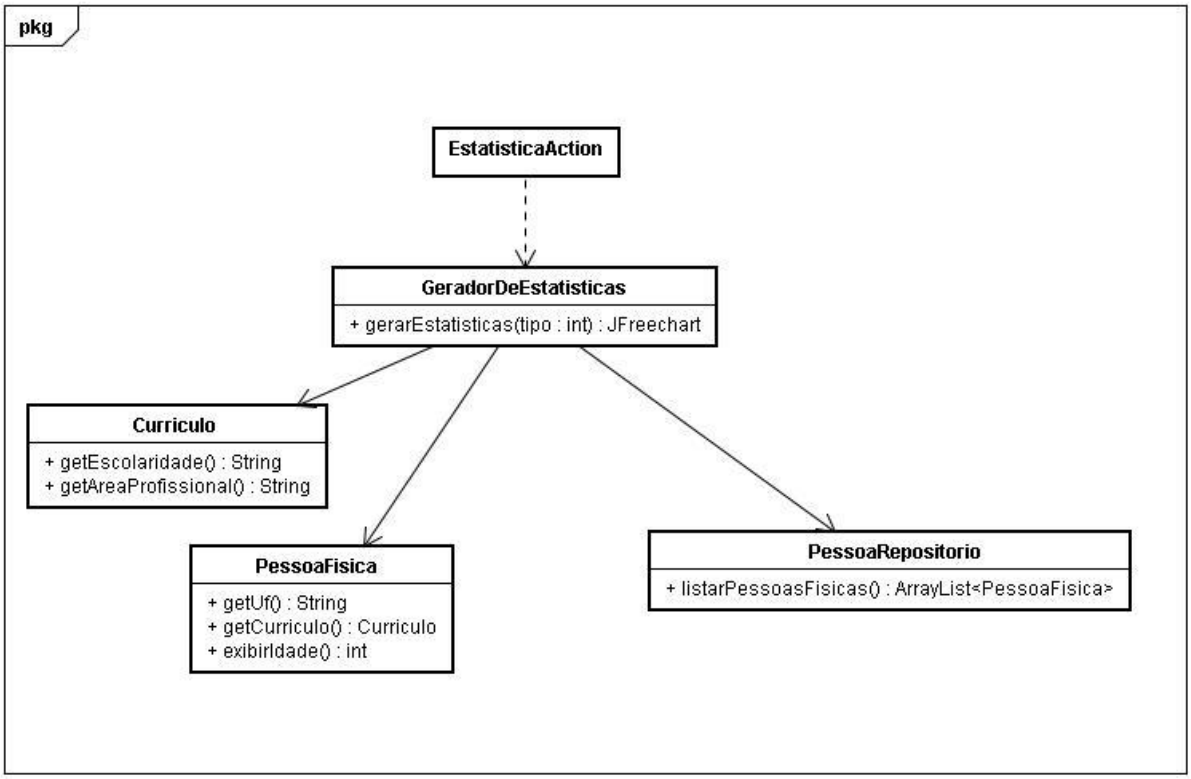

FIGURA 7: VCP – Visualizar estatísticas sobre usuários Pessoa Física

3.4 Modelo de Interações

Esse modelo tem o objetivo de detectar quais operações os objetos deverão realizar para realizar as transações previstas nos casos de uso ao qual se referem [6].

3.4.1 Diagrama de interação - Candidatar-se à vaga

Mostra a seqüência de mensagens trocadas entre os objetos participantes durante a execução da funcionalidade “Candidatar-se à vaga” descrita na seção 3.2.2.10. Os objetos da classe ResultInfo são objetos auxiliares usados para exibição dos resultados da busca para o usuário.

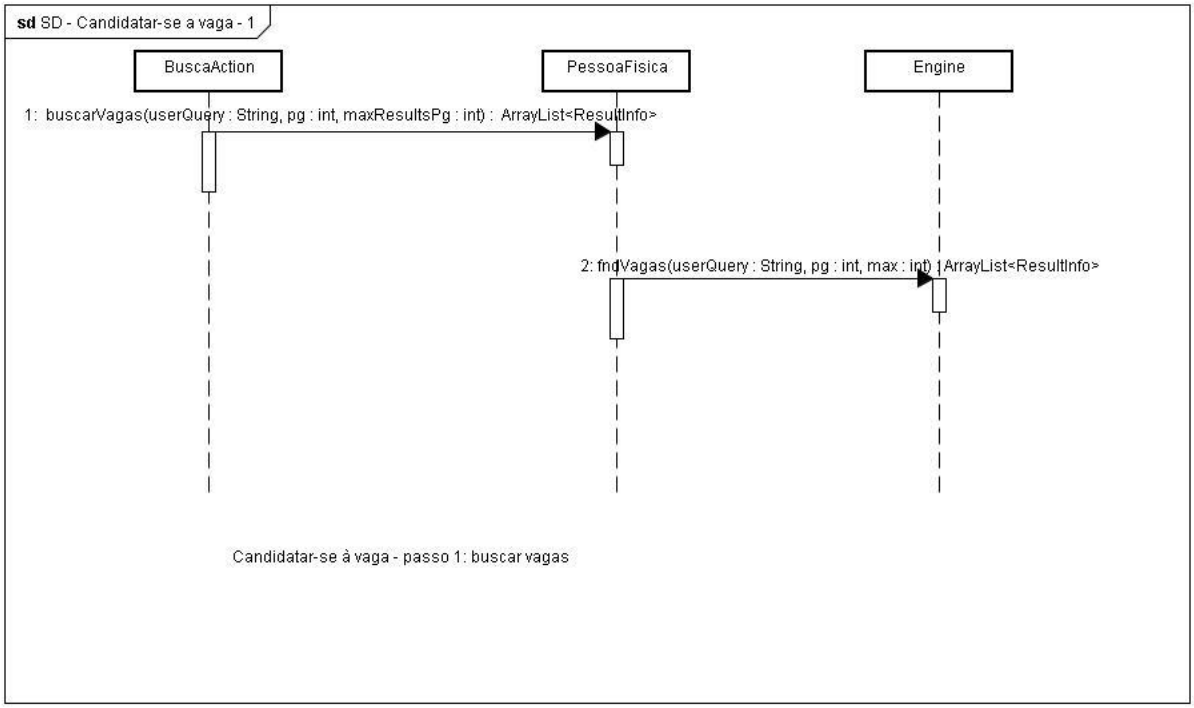

FIGURA 8: Diagrama de interação – Candidatar-se à vaga - passo 1

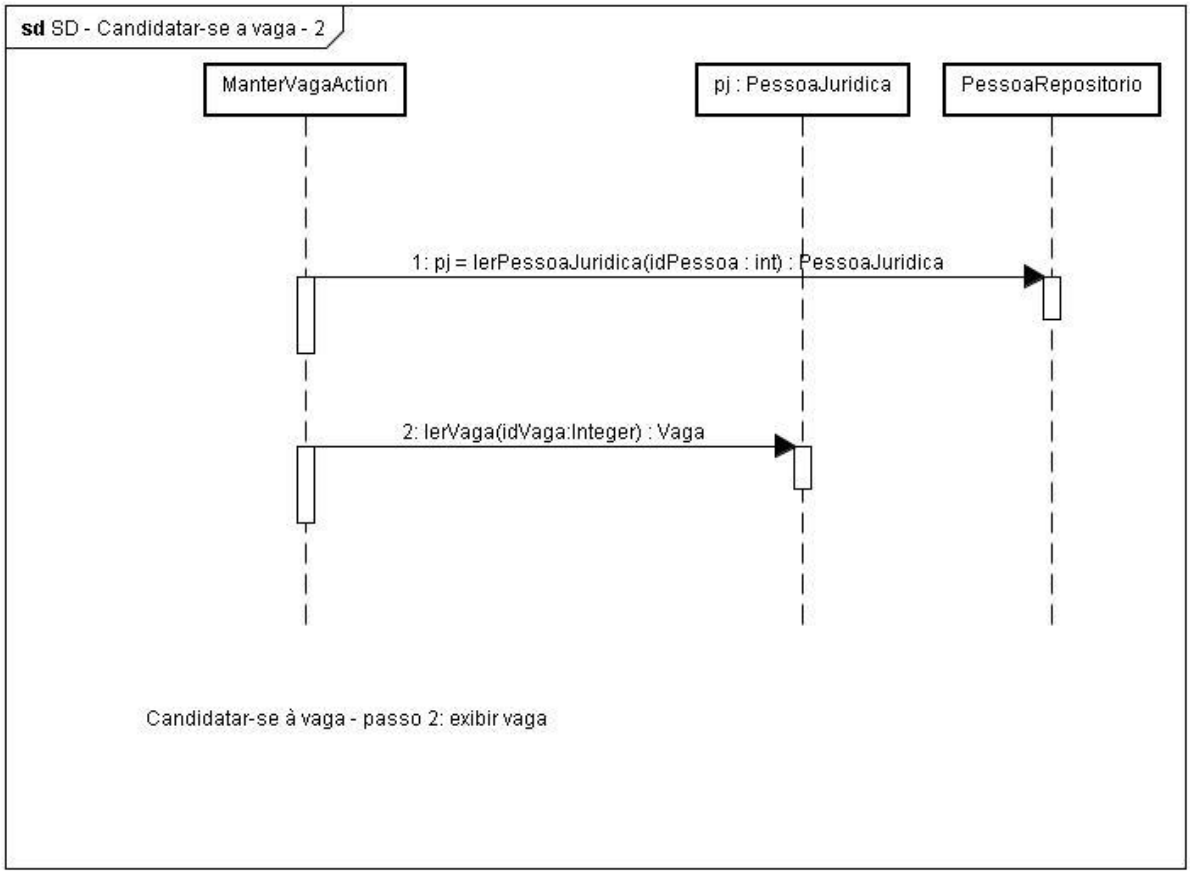

FIGURA 9: Diagrama de interação – Candidatar-se à vaga - passo 2

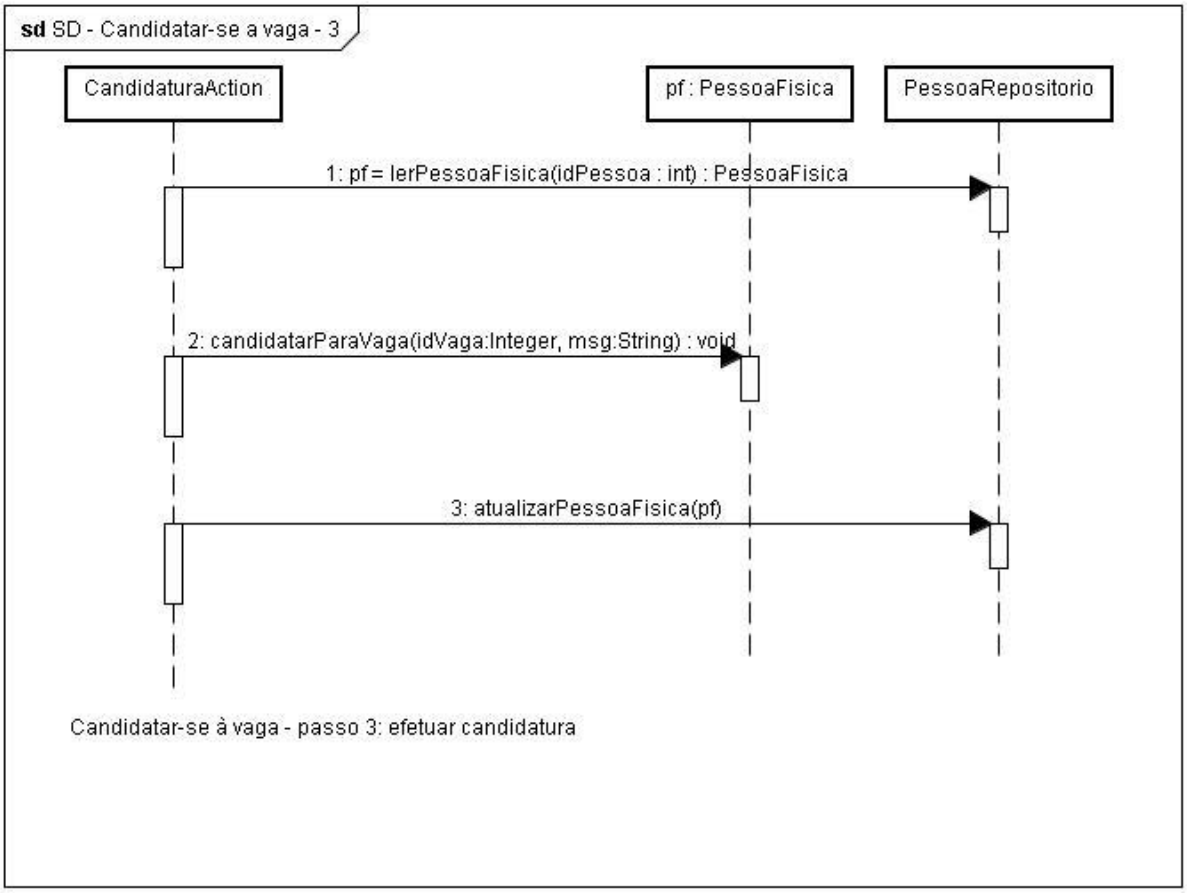

FIGURA 10: Diagrama de interação – Candidatar-se à vaga - passo 3

3.4.2 Diagrama de interação - Procurar currículos

Mostra a seqüência de mensagens trocadas entre os objetos participantes durante a execução da funcionalidade “Procurar currículos” descrita na seção 3.2.2.6. Os objetos da classe ResultInfo são objetos auxiliares usados para exibição dos resultados da busca para o usuário.

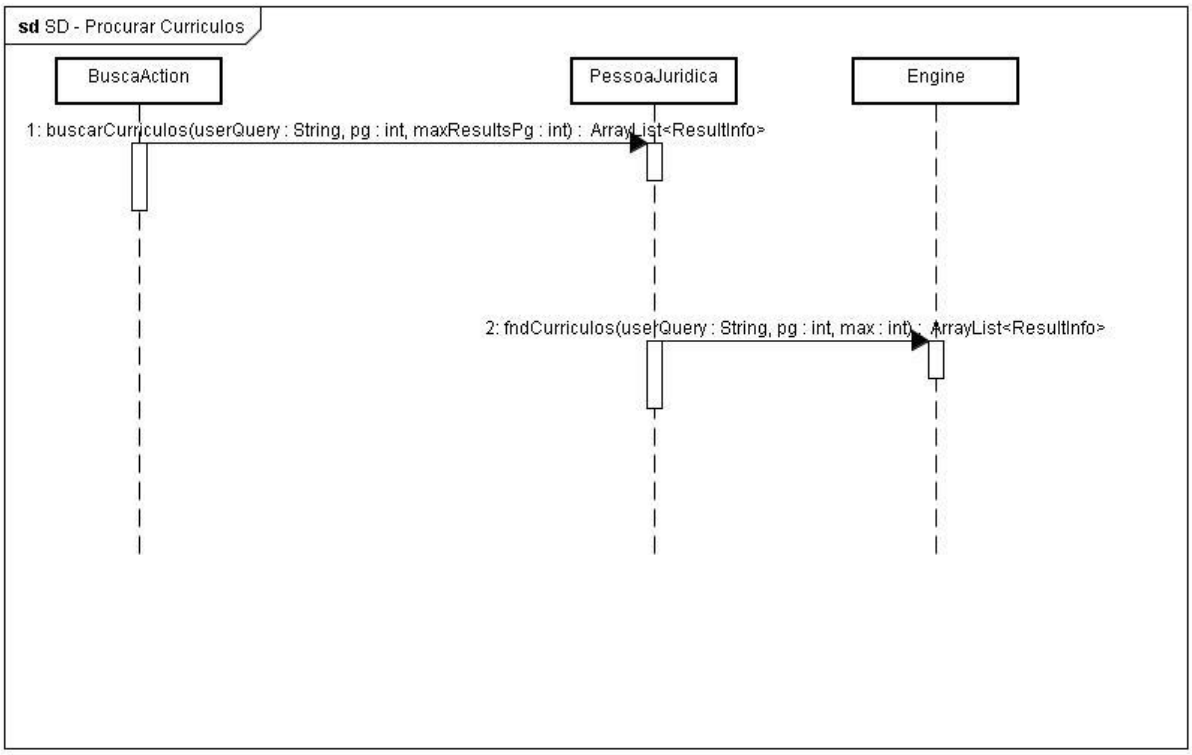

FIGURA 11: Diagrama de interação – Procurar Currículos

3.4.3 Diagrama de interação - Visualizar candidaturas

Mostra a seqüência de mensagens trocadas entre os objetos participantes durante a execução da funcionalidade “Visualizar candidaturas” descrita na seção 3.2.2.7.

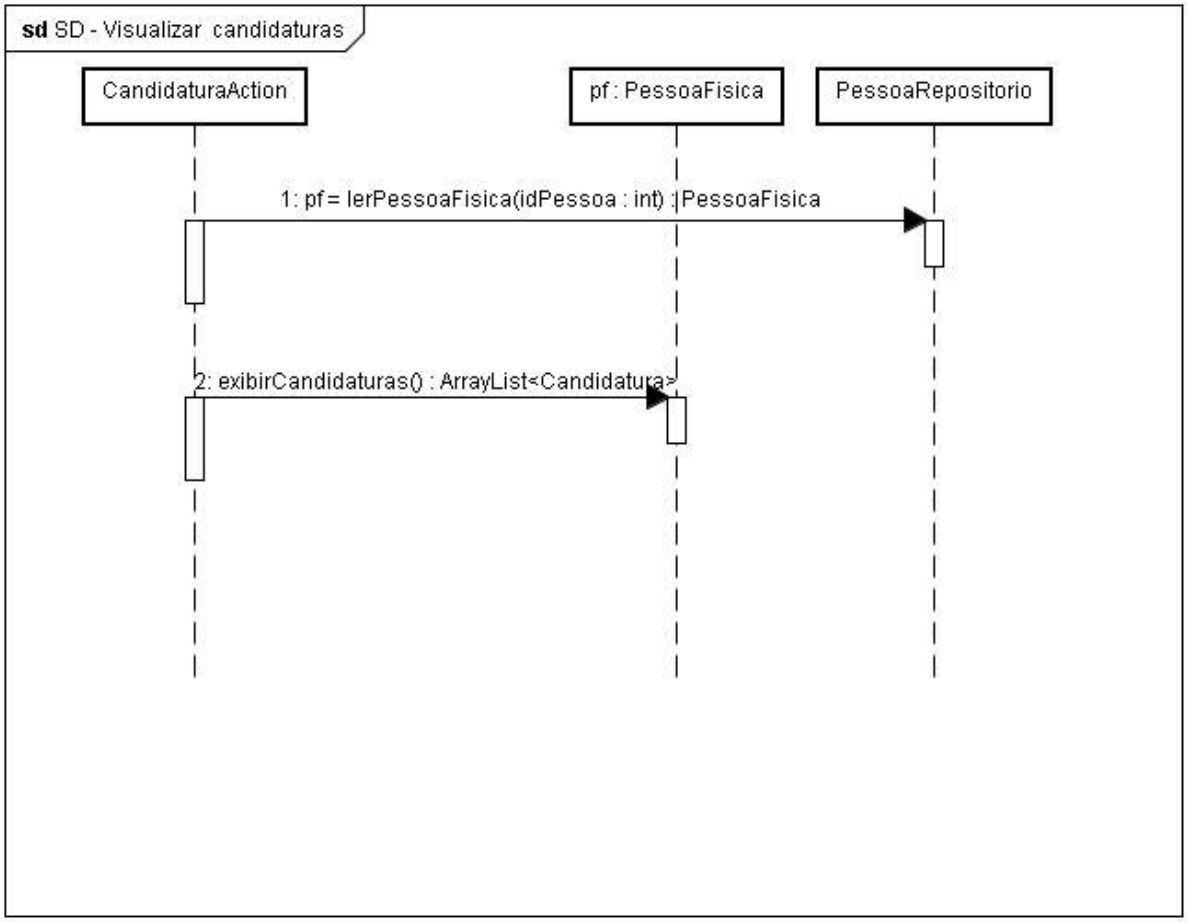

FIGURA 12: Diagrama de interação – Visualizar candidaturas

3.4.4 Diagrama de interação - Visualizar currículos de candidatos às vagas

Mostra a seqüência de mensagens trocadas entre os objetos participantes durante a execução da funcionalidade "Visualizar currículos de candidatos às vagas" descrita na seção 3.2.2.9.

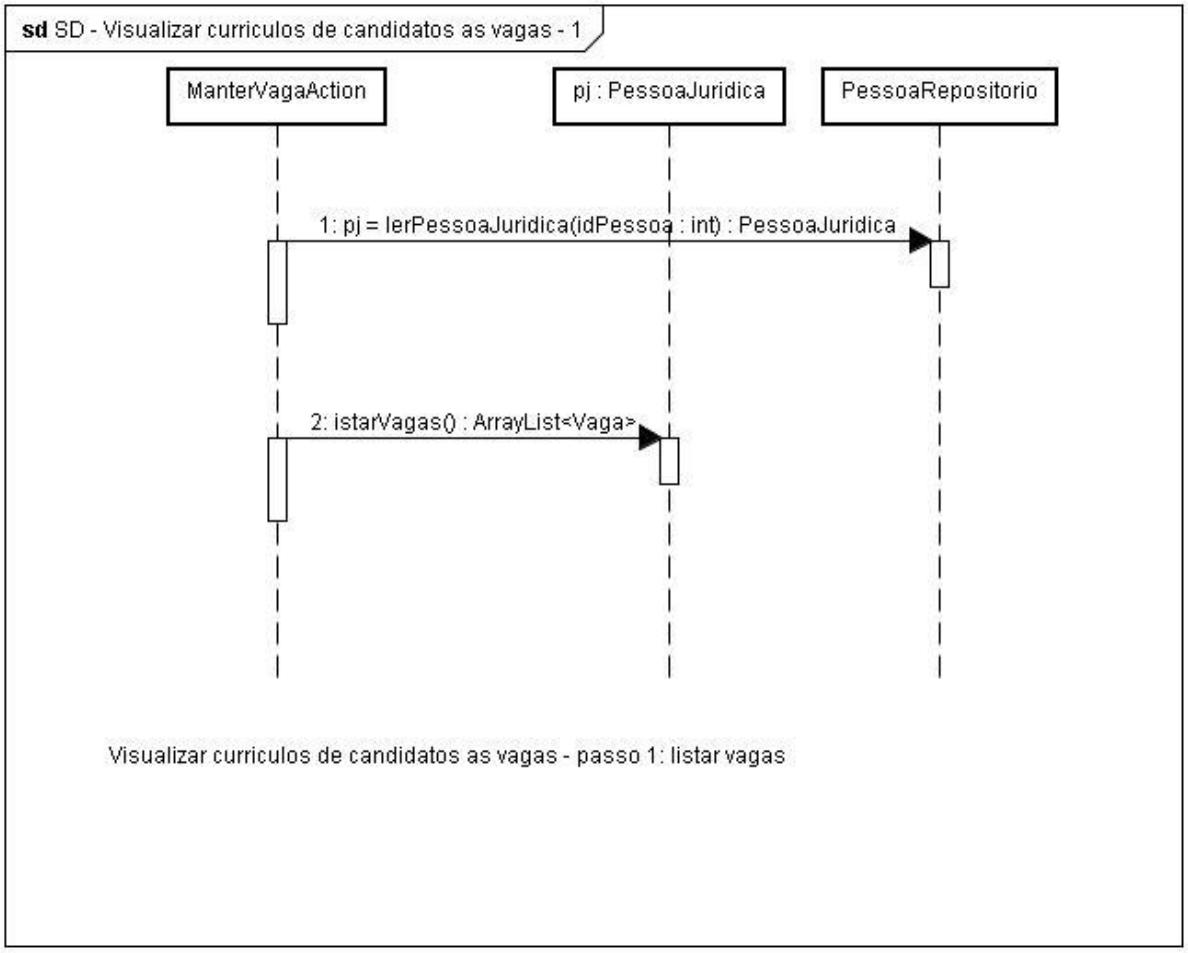

FIGURA 13: Diagrama de interação – Visualizar currículos de candidatos às vagas – passo 1

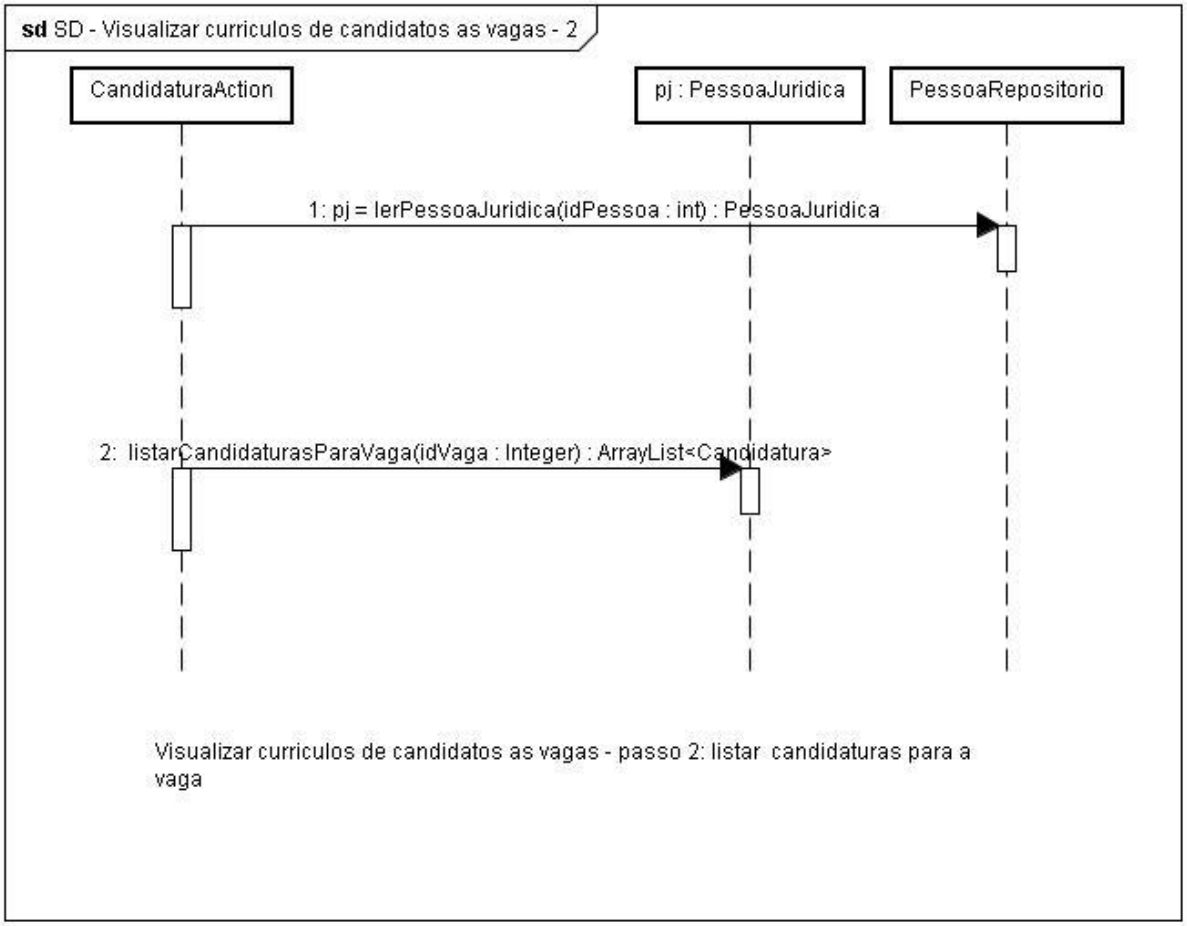

FIGURA 14: Diagrama de interação – Visualizar currículos de candidatos às vagas - passo 2

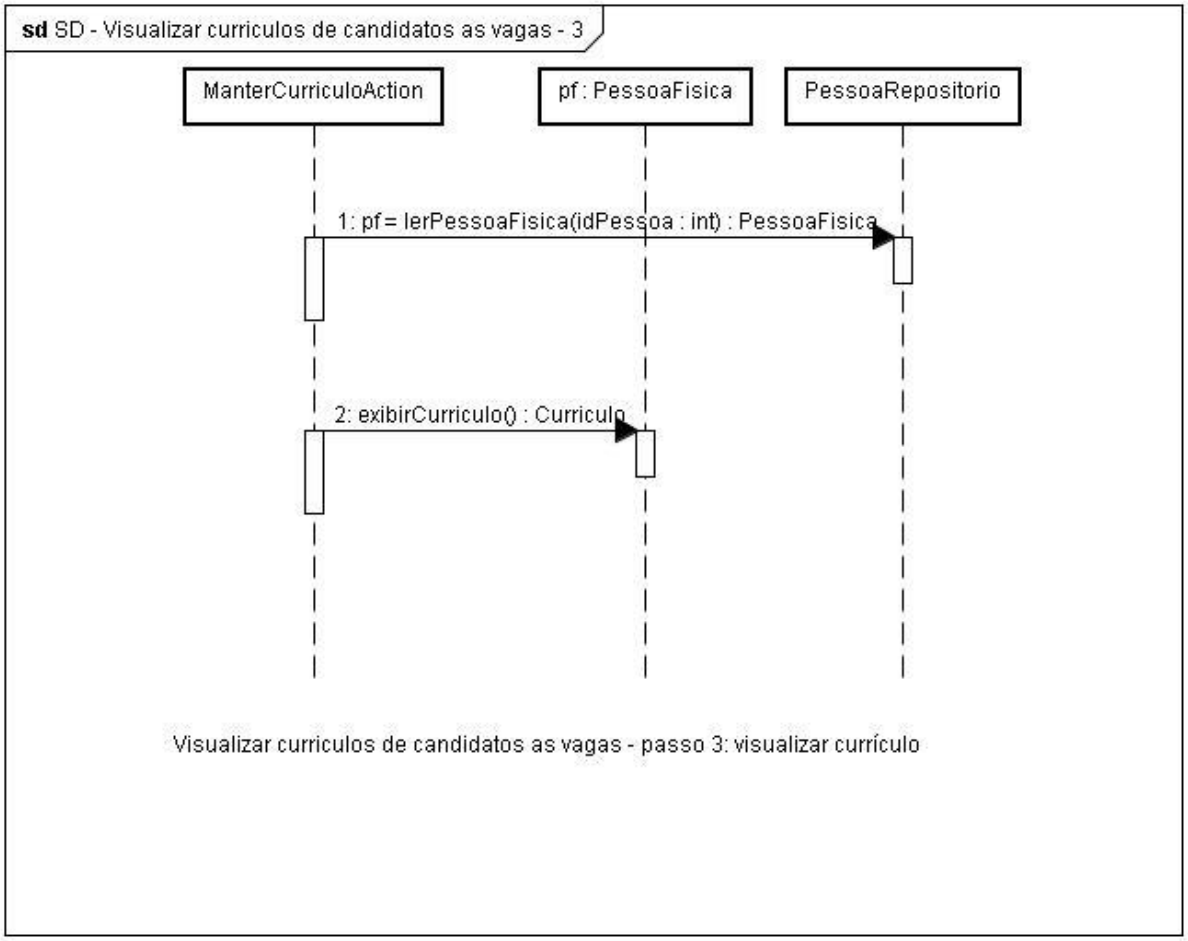

FIGURA 15: Diagrama de interação – Visualizar currículos de candidatos às vagas - passo 3

3.4.5 Diagrama de interação - Visualizar estatísticas sobre usuários pessoa física

Mostra a seqüência de mensagens trocadas entre os objetos participantes durante a execução da funcionalidade "Visualizar estatísticas sobre usuários pessoa física" descrita na seção 3.2.2.8. Os objetos da classe JFreeChart são utilizados para a geração e exibição dos gráficos para o usuário.

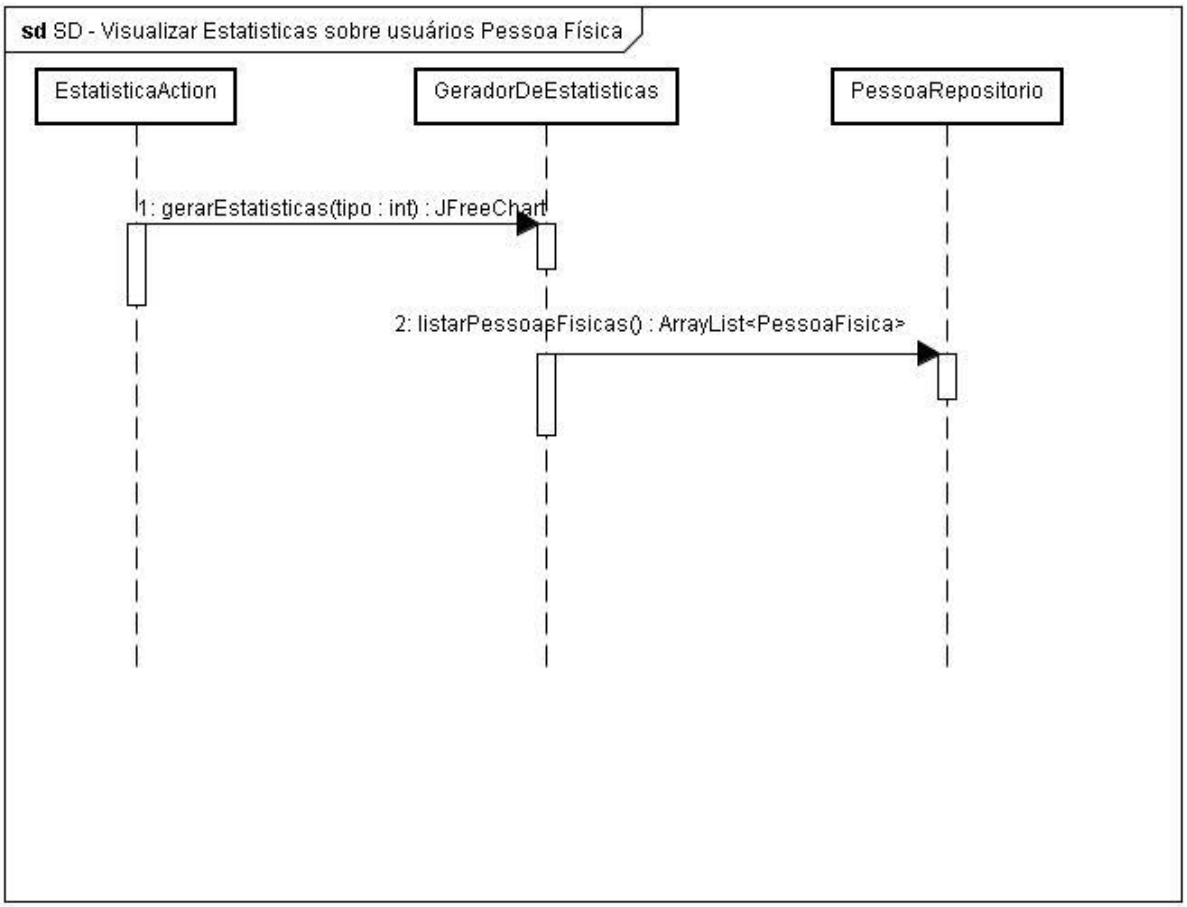

FIGURA 16: Diagrama de interação – Visualizar estatísticas sobre usuários Pessoa Física

3.5 Projeto de Banco de Dados

3.5.1 Projeto lógico de banco de dados

O projeto lógico de banco de dados é uma das fases do projeto de modelagem de um banco de dados relacional. Um esquema lógico é uma descrição da estrutura do banco de dados que pode ser processada por um Sistema Gerenciador de Banco de Dados (SGBD) [7].
O Projeto lógico de banco de dados consiste em criar um modelo lógico de dados a partir do modelo conceitual.

3.5.1.1 Usuario

Tabela que guarda dados relativos aos usuários do sistema.

Modelo Lógico: Usuario (**pessoa_id**, login, senha, tipo).

Atributos:

- pessoa_id (Chave primária e estrangeira) : Atributo do tipo inteiro que identifica a pessoa associada ao usuário. Deve possuir valor único no domínio e não pode ser nulo.
- login: Atributo do tipo texto que guarda o login do usuário. Deve possuir valor único no domínio e não pode ser nulo.
- senha: Atributo do tipo texto que guarda a senha do usuário. Não pode ter valor nulo.
- tipo: Atributo do tipo inteiro que informa o tipo do usuário. Não pode ter valor nulo.

3.5.1.2 Pessoa

Tabela que guarda os dados em comum das pessoas física e jurídica.

Modelo Lógico: Pessoa (**id** , nome, endereco, cep, telefone, uf, cidade, email)

Atributos:

- id (Chave primária): Atributo do tipo inteiro que identifica a pessoa. Deve possuir valor único no domínio e não pode ser nulo.
- nome: Atributo do tipo texto que guarda o nome da pessoa. Não pode ter valor nulo.
- endereco: Atributo do tipo texto que guarda o endereço da pessoa.
- cep: Atributo do tipo texto que guarda o CEP da pessoa.
- tefone: Atributo do tipo texto que guarda o telefone da pessoa.
- uf: Atributo do tipo texto que guarda a sigla do estado da federação onde mora a pessoa.
- cidade: Atributo do tipo texto que guarda o nome da cidade onde mora a pessoa.
- email: Atributo do tipo texto que guarda o email da pessoa.

3.5.1.3 Pessoafisica

Tabela que guarda os dados adicionais das pessoas do tipo pessoa física.

Modelo Lógico: Pessoafisica (**pessoa_id** , cpf, sexo, dataDeNascimento)

Atributos:

- pessoa_id (Chave primária e estrangeira): Atributo do tipo inteiro que identifica a pessoa relacionada. Deve possuir valor único no domínio e não pode ser nulo.
- cpf: Atributo do tipo texto que guarda o cpf da pessoa. Não pode ter valor nulo e deve possuir valor único no domínio.
- sexo: Atributo do tipo char que guarda o sexo da pessoa.
- dataDeNascimento: Atributo do tipo date que guarda a data de nascimento da pessoa.

3.5.1.4 Pessoajuridica

Tabela que guarda os dados adicionais das pessoas do tipo pessoa jurídica.

Modelo Lógico: Pessoajuridica (**pessoa_id** , cnpj, sítioCorporativo)

Atributos:

- pessoa_id (Chave primária e estrangeira): Atributo do tipo inteiro que identifica a pessoa relacionada. Deve possuir valor único no domínio e não pode ser nulo.
- cnpj: Atributo do tipo texto que guarda o cnpj da empresa. Deve possuir valor único no domínio e não pode ser nulo.
- sítioCorporativo: Atributo do tipo texto que guarda o sítio da empresa.

3.5.1.5 Candidatura

Tabela que guarda os dados relativos à candidatura da pessoa física à vaga.

Modelo Lógico: Candidatura (**pessoafisica_pessoa_id** , **vaga_id**, dataResposta, mensagem)

Atributos:

- pessoafisica_pessoa_id (Chave primária e estrangeira): Atributo do tipo inteiro que identifica a pessoa relacionada. Não pode ter valor nulo.
- vaga_id (Chave Primária e estrangeira): Atributo do tipo inteiro que identifica a vaga relacionada. Não pode ter valor nulo.
- dataResposta: Atributo do tipo datetime que guarda a data/hora em que foi efetuada a candidatura.
- mensagem: Atributo do tipo texto que guarda a mensagem de apresentação do candidato à empresa.

3.5.1.6 Vaga

Tabela que guarda os dados relativos às vagas.

Modelo Lógico: Vaga (**id**, pessoajuridica pessoa_id, uf, areaProfissional, cargo, descricao, dataDePublicacao, observacoes)

Atributos:

- id (Chave primária): Atributo do tipo inteiro que identifica a vaga. Deve possuir valor único no domínio e não pode ser nulo.
- pessoajuridica_pessoa_id (Chave Estrangeira): Atributo do tipo inteiro que identifica a pessoa relacionada. Não pode ter valor nulo.
- uf: Atributo do tipo char que guarda o local de trabalho para a vaga (estado da federação).
- areaProfissional: Atributo do tipo texto que guarda a área profissional alvo da vaga.
- cargo: Atributo do tipo texto que guarda o cargo da vaga.
- descrição: Atributo do tipo texto que guarda a descrição da vaga.
- dataDePublicacao: Atributo do tipo datetime que guarda a data/hora da publicação da vaga.
- observacoes: Atributo do tipo texto que guarda observações sobre a vaga.

3.5.1.7 Curriculo

Tabela que guarda os dados relativos aos currículos.

Modelo Lógico: Curriculo (**pessoafisica_pessoa_id**, areaProfissional, descExperiencia, descConhecimentos, descIdiomas, escolaridade, dataUltimaAlteracao, descFormacaoAcademica, observacoes)

Atributos:

- pessoafisica_pessoa_id (Chave primária e estrangeira): Atributo do tipo inteiro que identifica a pessoa física relacionada. Deve possuir valor único no domínio e não pode ser nulo.
- areaProfissional: Atributo do tipo texto que guarda o nome da área profissional da pessoa.
- descExperiencia: Atributo do tipo texto que guarda a descrição da experiência profissional da pessoa.
- descConhecimentos: Atributo do tipo texto que guarda a descrição dos conhecimentos profissionais da pessoa.
- descIdiomas: Atributo do tipo texto que guarda a descrição dos conhecimentos em idiomas da pessoa.
- escolaridade: Atributo do tipo texto que guarda a escolaridade da pessoa.
- dataUltimaAlteracao: Atributo do tipo datetime que guarda a data/hora da última alteração do currículo
- descFormacaoAcademica: Atributo do tipo texto que guarda a descrição da formação acadêmica da pessoa.
- observacoes: Atributo do tipo texto que guarda observações gerais relativas ao currículo / vida profissional.

3.5.2 Diagrama de Tabelas do Sistema

A FIGURA 17 a seguir apresenta o diagrama contendo todas as tabelas do sistema, com seus respectivos relacionamentos.

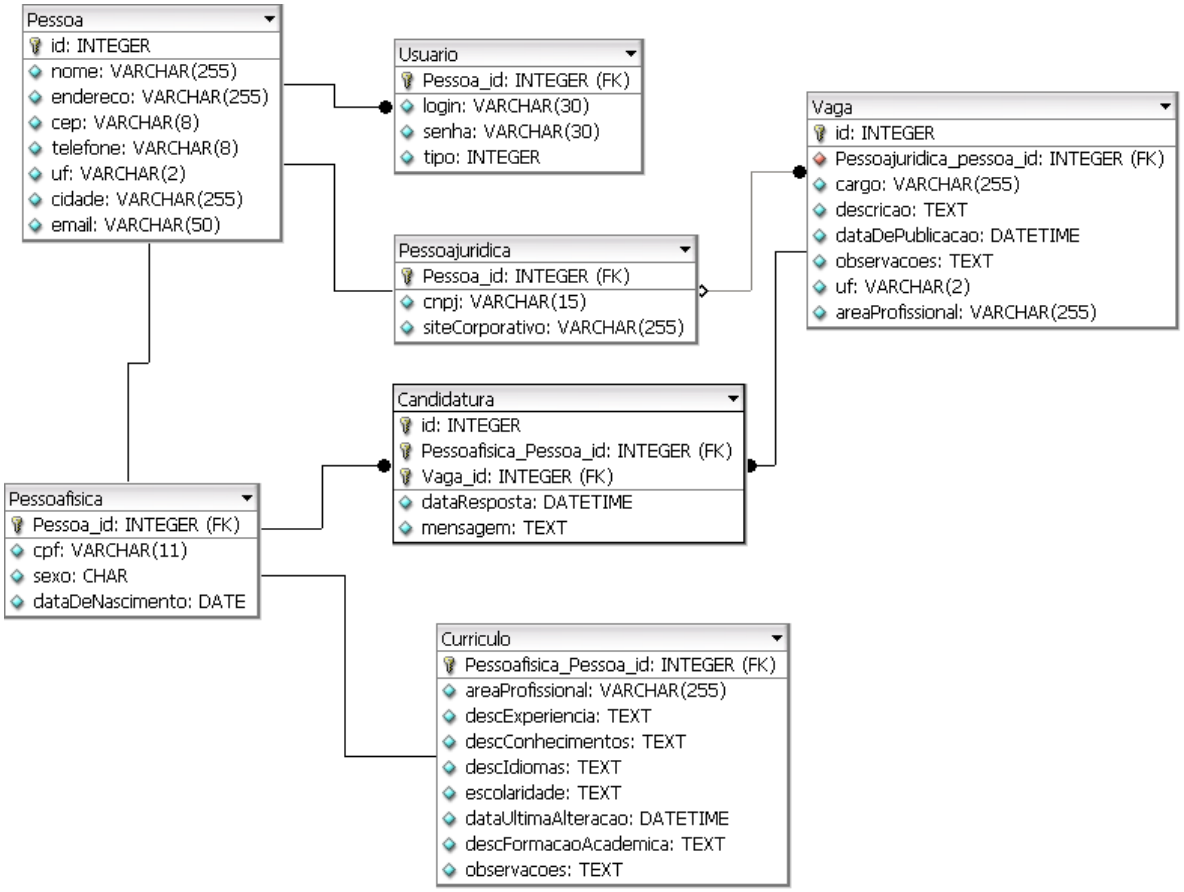

FIGURA 17: Tabelas do sistema e seus relacionamentos

Capítulo 4

Aspectos de Implementação

Neste capítulo, são apresentados os principais aspectos sobre a implementação do projeto. São abordadas também, as tecnologias usadas na implementação. Na Seção 4.1 é descrita a arquitetura utilizada na implementação do sistema. Na Seção 4.2 é abordada a implementação da persistência de dados no sistema. Na Seção 4.3 é descrito o módulo responsável por gerar estatísticas. Na Seção 4.4, é descrito o módulo do mecanismo de busca.

4.1 Arquitetura do Sistema

Nesta seção, é descrita a arquitetura do sistema e seus aspectos principais. Na seção 4.4.1 é descrita a implementação do sistema e na seção 4.1.2 o padrão *MVC*.

4.1.1 Implementação

Neste trabalho utilizou-se o padrão *MVC*, por sua flexibilidade e facilidade que proporciona para a manutenção do código-fonte. Aliado ao padrão *MVC*, utilizou-se o padrão de projeto *DAO*, escolhido com o objetivo de separar a lógica de acesso dos dados da lógica de negócio. Para implementar o padrão *MVC* foi escolhido o *framework Struts*, versão 2.1, distribuído e mantido pela *Apache Software Foundation*.

As classes responsáveis pela implementação da camada *controller*, chamadas no *Struts* de *actions*, estão localizadas no diretório \WEB-INF\classes\action\ da aplicação. Estas classes são como *java beans* (possuem métodos *get* e *set* para os atributos) e cujos nomes dos atributos correspondem aos nomes das variáveis que serão passadas pelos métodos POST e/ou GET. As variáveis enviadas pelo usuário, via GET ou POST são recebidas pelo controlador do *Struts* e repassadas para as actions conforme o mapeamento interno feito no arquivo *Struts.xml* [10]; Conforme a necessidade, os *actions* invocam os objetos da camada de negócio e fazem o redirecionamento dos resultados para a componente *view*, no caso deste projeto, páginas jsp localizadas no diretório raiz da aplicação.

Em relação à componente *view* e exibição da página para o usuário final, utilizou-se os padrões correntes para desenvolvimento Web *client side*, como o padrão html *tableless*, *Css*, e códigos javascript implementados com a biblioteca *Jquery*.

A TABELA 9 abaixo mostra parte do conteúdo do arquivo *Struts.xml*, exibindo o mapeamento do *action* responsável pelo *login* e *logout* na aplicação, a classe *LoginAction*.

Para se fazer o mapeamento, basicamente define-se o nome do *action* (na *tag* action, no atributo name), o nome da classe que o implementa e o diretório onde está localizada (na *tag* action, no atributo class), e os componentes *view* associados (nas *tags* result). O atributo namespace da *tag* package é utilizado para organizar e associar um nome de subdiretório a partir da raiz da url da aplicação a um conjunto de classes action e seus resultados, de acordo com a necessidade.

```
<package name="default" namespace="/" extends="struts-default">
        <action name="login" class="action.LoginAction">
        <result name="input">login.jsp</result>
        <result>index.jsp</result>
      </action>
...
  </package>
```

TABELA 9: O mapeamento do *action* responsável pelo login na aplicação

Na TABELA 10 a seguir é exibido como exemplo o fragmento de código principal da classe *LoginAction*, responsável pelas funcionalidades de *login* e de *logout* do sistema. O código é explicado logo em seguida.

```
...
public class LoginAction extends ActionSupport implements SessionAware {
   private String usrname=null;
   private String pswd=null;
   private String logout = null;
   private Map session=null;

   public String execute() throws Exception {
      if (logout != null) {
         getSession().remove("nome");
         getSession().remove("id");
         getSession().remove("tipo");
         return INPUT;
      }
      Cadastro cad = new Cadastro();
      Usuario u = cad.validarLogin(usrname,pswd);
      if (u != null) {
         getSession().put("nome", u.getPessoa().getNome());
         getSession().put("id", u.getPessoa().getId());
         getSession().put("tipo", u.getTipo());
         return SUCCESS;
      } else {
         addActionError("Usuário ou senha inválidos");
         return INPUT;
      }
   }
...
```

TABELA 10: Fragmento de código da classe *LoginAction*

Toda a classe *action* por padrão estende a classe ActionSupport e possui o método *execute*, que é chamado depois que o controlador carrega as propriedades da *action* com os valores passados via GET/POST. Quando o usuário faz o *login*, as variáveis usrname e pswd são carregadas pelo *Struts* com os valores passados pelo usuário via método POST. A senha (pswd) e o nome de usuário (usrname) são verificados, e caso o *login* esteja correto o usuário é direcionado para a página de sucesso (index.jsp). Caso contrário, o usuário é direcionado novamente para a página de *login*, onde será exibida a mensagem "Usuário ou senha inválidos". Ao fazer o *logout* o usuário também é redirecionado para a página de *login*.

Na TABELA 11, é mostrado o código do componente *view* (a página de *login*)

correspondente. É interessante notar o uso da *taglib* padrão do *Struts* (cujo prefixo é *s*), responsável por implementar as funcionalidades desse *framework* para as componentes *view* (renderização de componentes html, exibição de valores de variáveis, e outras funcionalidades diversas).

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<jsp:include page="INCLUDES/head.jsp" />
<div class="main">
<div class="areaEsquerda">
<p style="margin:8px">
Digite ao lado o seu username e a sua senha para entrar no sistema.
</p>
<p style="margin-top:20px;">Fazer novo cadastro</P>
<ul>
<li style="margin:3px;"><a href="manterUsuarioAddPj.jsp">Pessoa jurídica</a>.</li>
<li style="margin:3px;"><a href="manterUsuarioAddPf.jsp">Pessoa física</a>.</li>
</ul>
<br />
</div>
<div class="conteudo">
<div class="breadcumb"></div>
<div class="divlogin">
<s:actionerror cssClass="msgerro" />
<s:form namespace="/" action="login" validate="true" method="POST" >
<s:textfield name="usrname" key="usrname" label="username"/>
<s:password name="pswd" key="psdw" label="senha" />
<s:submit value="Entrar"  align="center"  cssStyle="margin-top:10px;"/>
</s:form>
</div>
</div>
</div>
<jsp:include page="INCLUDES/footer.jsp" />
```

TABELA 11: Código do componente *view* do login

Por último, finalizando esta seção, é exibida a Tabela 12 abaixo, que resume esquematicamente os módulos do sistema e algumas observações sobre eles:

Módulo/Subsistema	**Observações**
Geração de estatísticas	Responsável pelas funcionalidades relacionadas à geração de estatísticas.
Busca/indexação	Responsável pelas funcionalidades relacionadas à busca e indexação dos currículos e vagas.
Gerenciamento de usuários	Responsável pela funcionalidade de gerenciamento dos usuários.
Gerenciamento de vagas	Responsável pelas funcionalidades relacionadas às vagas.
Gerenciamento de currículos	Responsável pelas funcionalidades relacionadas aos currículos.
Persistência de dados	Responsável pela persistência de dados no sistema. Todos os outros módulos/subsistemas se relacionam com este módulo.

TABELA 12: Divisão esquemática dos módulos do sistema

4.1.2 O padrão Model-View-Controller (MVC)

O MVC é um padrão de arquitetura de projeto para desenvolvimento de sistemas que tem como objetivo tornar a construção da aplicação mais organizada e facilitar a sua alteração e manutenção ao longo do tempo, causando o menor impacto no resto da aplicação, através da separação da lógica de negócio da lógica de apresentação [9,11]. Inicialmente desenvolvido para a linguagem Smalltalk em 1979, este padrão funciona basicamente separando a lógica de negócio da lógica de apresentação, usando uma camada de controle para isso [9]. Em termos práticos de desenvolvimento para a Internet, isto significa por exemplo que o código fonte da aplicação não fica misturado com o código html (ou xml) responsável pela renderização das páginas.

O fluxo no padrão MVC pode ser descrito assim: O controlador recebe a entrada do usuário (em um ambiente Web a requisição viria pelos métodos GET ou POST, por exemplo) e baseado nela promove atualizações no modelo; O modelo então aplica as regras de negócio

existentes e utiliza a camada de persistência de dados, caso exista; Baseado na atualização do modelo e na entrada do usuário o controlador escolhe o componente *view* apropriado; O *view* transforma os dados do modelo atualizado em uma forma de visualização adequada ao usuário. No padrão Java EE o componente *view* é implementado normalmente usando-se a tecnologia *java server pages*, e o controlador por um *servlet*, o qual é responsável por receber a requisição dos usuários e interagir com o modelo [11].

A definição das componentes é a seguinte:

• Model: responsável por armazenar informações de estado da aplicação, manipula e gerencia as regras de negócio e se comunica com a camada de acesso a dados.

• View: responsável por transformar as informações do modelo em uma forma apropriada para exibição ao usuário.

• Controller: Processa a entrada do usuário (via GET ou POST), e baseado nessa entrada define qual ação é requerida. Atualiza o modelo e escolhe o *view* apropriado.

As principais vantagens no uso do padrão MVC são [9]:

• Desenvolvimento paralelo: cada desenvolvedor pode ficar responsável por uma das componentes, aumentando-se a produtividade e eficiência.

• Reusabilidade: a componente *view* pode ser reutilizada em interfaces diferentes.

• Manutenibilidade: devido à separação em componentes, atualizações e correções nos sistemas são mais fáceis de serem efetuadas.

• Independência da Interface com o Usuário (IU): As componentes *controller, model e view* podem ser reutilizadas, apenas adaptando-se a componente *view* à nova IU.

4.2 Persistência de Dados

Nesta seção é descrita a persistência de dados no sistema. Na seção 4.2.1 é descrita a implementação da persistência de dados, na seção 4.2.2 é apresentado o *framework Hibernate*, e na seção 4.2.3 o banco de dados MySql.

4.2.1 Implementação

A persistência de dados no sistema, implementada no padrão *DAO*, é feita utilizando-se o *framework* para mapeamento objeto-relacional *Hibernate* e o banco de dados *MySql*. As classes responsáveis pela persistência de dados estão localizadas no diretório WEB-INF/classes/dao, e são *PessoaDao* e *UsuarioDao*. Estas classes são responsáveis pelas

funcionalidades CRUD do sistema. As classes de negócio que necessitam trabalhar com os objetos persistidos comunicam-se com as classes de repositórios (*PessoaRepositorio* e *UsuarioRepositorio*), localizadas no diretório WEB-INF/classes/repositorio, que por sua vez se comunicam com as respectivas classes DAO. Assim, por exemplo, a classe PessoaRepositorio se comunica com a classe PessoaDao. Existe ainda uma classe utilitária para o uso com o *Hibernate*, a classe *HibernateUtil*, localizada no diretório WEB-INF/classes/dao e utilizada para ler configurações de inicialização usadas no *Hibernate*. A TABELA 13 a seguir exibe o trecho do arquivo de configuração do *Hibernate* para a aplicação, hibernate.cfg.xml, contendo o mapeamento das classes de domínio e tabelas, assim como outras configurações.

```
<property name="hibernate.connection.datasource">java:/comp/env/jdbc/SBCDB</property>
        <property name="connection.useUnicode">true</property>
        <property name="connection.characterEncoding">UTF-8</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="current_session_context_class">thread</property>
<property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</prope
rty>
        <property name="show_sql">false</property>
        <mapping resource="Usuario.hbm.xml"/>
        <mapping resource="Pessoa.hbm.xml"/>
        <mapping resource="PessoaFisica.hbm.xml"/>
        <mapping resource="PessoaJuridica.hbm.xml"/>
        <mapping resource="Curriculo.hbm.xml"/>
        <mapping resource="Vaga.hbm.xml"/>
        <mapping resource="Candidatura.hbm.xml"/>
```

TABELA 13: O arquivo *hibernate.cfg.xml* da aplicação

A seguir é mostrado na TABELA 14, como exemplo, os métodos mais importantes da classe PessoaDao. Essa classe possui métodos para criar, atualizar, excluir e ler objetos das classes PessoaFisica e PessoaJuridica (e assim indiretamente os objetos associados), e ilustra a utilização do *Hibernate* para realizar o mapeamento objeto-relacional e a persistência de dados no sistema.

```
...
 public Integer criarPessoa(PessoaFisica pf) {
     Session ss = HibernateUtil.getSession();
     Transaction tx = null;
     Integer id = null;
     try {
        tx = ss.beginTransaction();

        Curriculo c = new Curriculo();
        Calendar now = Calendar.getInstance();
        c.setDataUltimaAlteracao(now.getTime());
        c.setPessoaFisica(pf);
        pf.setCurriculo(c);

        id = (Integer) ss.save(pf);

        tx.commit();

     } catch (Exception ex) {
        Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
        tx.rollback();
     } finally {
        ss.close();
     }

     return id;
  }

  public Integer criarPessoa(PessoaJuridica pj) {
     Session ss = HibernateUtil.getSession();
     Transaction tx = null;
     Integer id = null;
     try {
        tx = ss.beginTransaction();

        id = (Integer) ss.save(pj);

        tx.commit();

     } catch (Exception ex) {
```

```
            Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
            tx.rollback();
        } finally {
            ss.close();
        }

        return id;
    }

    public void excluirPessoa(Integer[] pessoaIds) {

        String sql = null;

        Session ss = HibernateUtil.getSession();
        Transaction tx = null;
        try {
            tx = ss.beginTransaction();

            Query q = ss.createQuery("DELETE FROM PessoaFisica pf WHERE pf.id IN (:ids)");
            q.setParameterList("ids", pessoaIds);
            q.executeUpdate();

            q = ss.createQuery("DELETE FROM PessoaJuridica pj WHERE pj.id IN (:ids)");
            q.setParameterList("ids", pessoaIds);
            q.executeUpdate();

            tx.commit();
        } catch (Exception ex) {
            Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
            tx.rollback();
        } finally {
            ss.close();
        }

    }

    public void atualizarPessoaFisica(PessoaFisica pf) {

        Session ss = HibernateUtil.getSession();
        Transaction tx = null;
```

```
        try {
            tx = ss.beginTransaction();
            ss.update(pf);
            tx.commit();
        } catch (Exception ex) {
            Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
            tx.rollback();
        } finally {
            ss.close();
        }

    }

    public void atualizarPessoaJuridica(PessoaJuridica pj) {

        Session ss = HibernateUtil.getSession();
        Transaction tx = null;
        try {
            tx = ss.beginTransaction();
            ss.update(pj);
            tx.commit();
        } catch (Exception ex) {
            Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
            tx.rollback();
        } finally {
            ss.close();
        }

    }

    public PessoaFisica lerPessoaFisica(int idPessoa) {

        PessoaFisica p = null;
        Session ss = HibernateUtil.getSession();
        try {
            Query q = ss.createQuery("from PessoaFisica p left join fetch p.candidaturas where p.id=:id");
            q.setInteger("id", idPessoa);
            p = (PessoaFisica) q.uniqueResult();

        } catch (Exception ex) {
```

```
            Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            ss.close();
        }
        return p;
    }

    public PessoaJuridica lerPessoaJuridica(int idPessoa) {

        PessoaJuridica p = null;
        Session ss = HibernateUtil.getSession();
        try {
            Query q = ss.createQuery("from PessoaJuridica p left join fetch p.vagas v left join fetch
v.candidaturas where p.id=:id");
            q.setInteger("id", idPessoa);
            p = (PessoaJuridica) q.uniqueResult();
        } catch (Exception ex) {
            Logger.getLogger(PessoaDao.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            ss.close();
        }
        return p;
    }

    public ArrayList<PessoaFisica> listarPessoasFisicas() {
        ArrayList<PessoaFisica> pessoasFisicas = new ArrayList<PessoaFisica>();
        Session ss = HibernateUtil.getSession();
        try {
            Query q = ss.createQuery("from PessoaFisica");
            List lista = q.list();

            pessoasFisicas.addAll(lista);
        } catch (Exception ex) {
            Logger.getLogger(UsuarioDao.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            ss.close();
        }


        return pessoasFisicas;
```

```
    }

    public ArrayList<PessoaJuridica> listarPessoasJuridicas() {
      ArrayList<PessoaJuridica> pessoasJuridicas = new ArrayList<PessoaJuridica>();
      Session ss = HibernateUtil.getSession();
      try {
         Query q = ss.createQuery("from PessoaJuridica");
         List lista = q.list();

         pessoasJuridicas.addAll(lista);
      } catch (Exception ex) {
         Logger.getLogger(UsuarioDao.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
         ss.close();
      }

      return pessoasJuridicas;
    }
...
```

TABELA 14: Métodos da classe PessoaDao

4.2.2 O *framework* Hibernate

O *Hibernate* é um *framework* implementado em Java para mapeamento objeto relacional que auxilia na persistência de dados nas aplicações que o utilizam. Neste projeto, utilizou-se o *hibernate* para fazer o mapeamento objeto-relacional das classes de domínio, de modo a facilitar o trabalho de codificação relativo à persistência de dados.

O *Hibernate* trabalha fazendo o mapeamento de atributos das classes permitindo que dados sejam persistidos como se fossem objetos, não necessitando da utilização de SQL. Usando o *Hibernate*, o desenvolvedor se livra de escrever muito do código de acesso a banco de dados e de SQL que ele escreveria não usando a ferramenta, acelerando a velocidade do seu desenvolvimento [15]. O *Hibernate* faz conversões de métodos para salvar, alterar, excluir, etc. vinda de objetos para chamadas SQL, isso o torna independente de um banco de dados específico e o fato de ser configurado através de um arquivo XML, a alteração de uma base de dados para outra é relativamente simples de ser feita, contudo isso gera um aumento no tempo de execução do sistema. Para se efetuar consultas especializadas o *Hibernate* possui

uma linguagem específica, que se assemelha muito com o SQL padrão, mas tem como característica ser orientada a objeto, o que inclui paradigmas de herança, polimorfismo, etc [15]. Apesar de o *Hibernate* possuir uma linguagem específica para consultas, também é possível a utilização de SQL.

Mas o *framework* não é uma boa opção para todos os tipos de aplicação. Sistemas que fazem uso extensivo de recursos como *stored procedures*, *triggers* ou que implementam a maior parte da lógica da aplicação no banco de dados não vai se beneficiar com o uso do *Hibernate*. Ele é mais indicado para sistemas onde a maior parte da lógica de negócios fica na própria aplicação Java, dependendo pouco de funções específicas do banco de dados.

Neste trabalho utilizou-se o *Hibernate* com o banco de dados MySql, e a conexão entre eles ficando a cargo do *connection pool* do servidor. O *connection pool* é uma técnica que permite a criação e o gerenciamento de conexões de banco de dados usando o JDBC, e é baseada no fato de que a maioria das aplicações só fazem uso ativo de uma conexão JDBC apenas quando um *thread* precisa efetuar uma transação e depois disso essas conexões se tornam ociosas [15]. Dessa forma essa conexão que ficaria ociosa é reaproveitada e fica a disposição de um outro *thread* de outra aplicação. Na prática quando um *thread* de uma aplicação precisa trabalhar com um banco de dados usando JDBC ele faz uma requisição de uma conexão ao pool e quando termina de usar essa conexão ela retorna ao pool para ser usada novamente. A principal vantagem do uso dessa técnica é o controle e otimização do uso de recursos do sistema, já que a criação e o uso de conexões com banco de dados demandam consideráveis recursos de processamento e memória.

4.2.3 O banco de dados MySql

O sistema de gerenciamento de dados relacional MySql foi escolhido para este trabalho por ser um dos mais utilizados SGBD´s do mercado, isto é, possui uma base instalada de milhares de servidores, dando suporte de persistência de dados aos mais diversos tipos de aplicações e negócios, além de estar disponível para os principais sistemas operacionais usados atualmente [8]. Além disso, é um sistema de gerenciamento de dados poderoso e confiável que fornece recursos robustos, proteção de dados e desempenho e escalabilidade para clientes com vários tipos de demanda, seja, por exemplo, em aplicativos Web ou armazenamento de dados local ou remoto.

Algumas características incluem o fato de possuir fácil instalação e manutenção, possuir várias ferramentas gratuitas para gerenciamento desenvolvidas pela própria fabricante e por

terceiros, ser escrito em C e C++, poder usar processamento em múltiplas CPU´s, possuir APIs para desenvolvimento de aplicações cliente em várias linguagens de programação como C, C++, Eiffel, Java, Perl, PHP, Python e Ruby. Para a plataforma java a conectividade para clientes é provida por um driver chamado MySQL Connector/J, o qual é um driver java puro que se comunica diretamente com o servidor MySql utilizando o próprio protocolo do MySql. Neste trabalho utilizou-se a versão 5.1 do MySql, por sua simplicidade e facilidade de configuração e adaptação às outras tecnologias usadas no projeto.

4.3 Geração de Estatísticas

Nesta seção é discutida a geração de estatísticas no sistema. A seção 4.3.1 apresenta a implementação e a seção 4.3.2 descreve a biblioteca JFreeChart.

4.3.1 Implementação

Este módulo é responsável por gerar estatísticas relativas aos usuários pessoa física (candidatos) cadastrados no sistema. Este recurso foi inserido para tentar dar ao usuário pessoa jurídica um panorama geral do universo que está sendo pesquisado, e dessa forma auxiliar no processo de busca das informações. Vários tipos diferentes de informações estatísticas poderiam ser geradas, mas optou-se por implementar apenas algumas que podem ser mais importantes para os usuários pessoa jurídica no contexto de uma busca. As estatísticas geradas são: Porcentagens de candidatos por estado da federação, por escolaridade, por faixa de idade (nas faixas de 18 a 25 anos, 26 a 35 anos, 35 a 45 anos, 46 a 55 anos, 56 a 65 anos e maior que 65 anos) e por área profissional. Essas informações são exibidas para o usuário na forma de gráficos de *pizza*, escolhidos por facilitarem a visualização e compreensão dos dados pelo usuário, gerados com o auxílio da biblioteca *JFreeChart*. O *framework Struts* possui um *plugin* interno que é utilizado pela aplicação para a geração e exibição dos gráficos para o usuário a partir dos objetos da classe *JFreeChart*.

A FIGURA 18 a seguir mostra um exemplo de gráfico gerado pela aplicação. Este gráfico exibe as porcentagens de candidatos por estado da federação. Deve ser notado que neste caso o gráfico exibe apenas os estados que possuem algum candidato. Estados que exibem porcentagem igual a zero não são exibidos no gráfico.

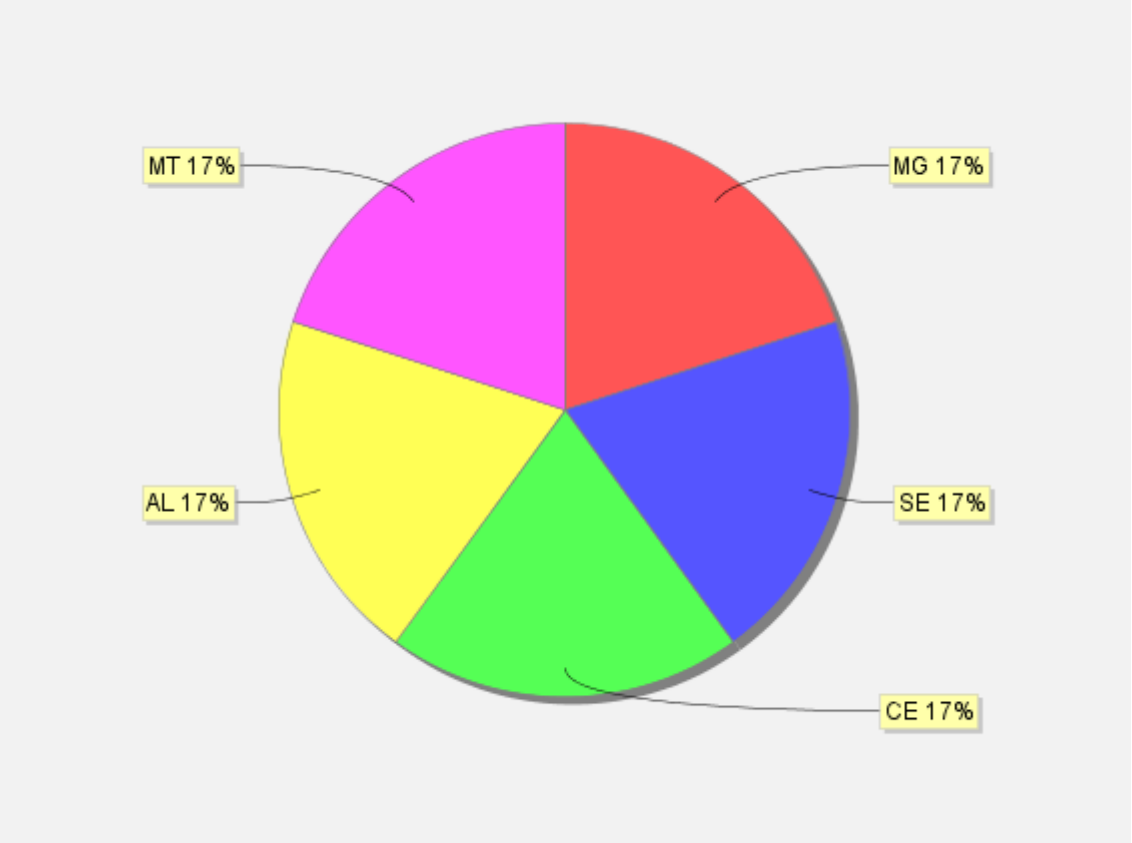

FIGURA 18: Exemplo mostrando um gráfico da aplicação gerado com o *JFreeChart*

Na TABELA 15 abaixo é exibido o código do método *gerarEstatisticas* da classe *GeradorDeEstatisticas*. Este método é responsável por gerar o objeto da classe JFreeChart de acordo com o tipo de estatística requerida. Este método principal utiliza os métodos auxiliares *porcentagemDeCandidatosPorIdade,porcentagemDeCandidatosPorEstado,porcentagemDeCandidatosPorEscolaridade* e *porcentagemDeCandidatosPorAreaProfissional* (que recebem como parâmetro um *array* de objetos PessoaFisica), para obter uma tabela com os dados estatísticos que serão utilizados, de acordo com o tipo de estatística requisitada pelo usuário. Por exemplo, o método *porcentagemDeCandidatosPorEstado* retorna uma tabela com os estados da federação que possuem candidatos (pessoas físicas) cadastrados e a porcentagem deles para cada estado. Esta tabela é implementada como um objeto da classe *HashMap* da linguagem java. O código do método *porcentagemDeCandidatosPorEstado* é exibido na TABELA 16.

```
public JFreeChart gerarEstatisticas(int tipo) {

    JFreeChart chart = null;
    HashMap<String, Float> dados = null;
    ArrayList<PessoaFisica> pfs = prep.listarPessoasFisicas();

    switch (tipo) {
       case porcentagemDeCandidatosPorIdade:
          dados = porcentagemDeCandidatosPorIdade(pfs);
          break;
       case porcentagemDeCandidatosPorEstado:
          dados = porcentagemDeCandidatosPorEstado(pfs);
          break;

       case porcentagemDeCandidatosPorEscolaridade:
          dados = porcentagemDeCandidatosPorEscolaridade(pfs);
          break;

       case porcentagemDeCandidatosPorAreaProfissional:
          dados = porcentagemDeCandidatosPorAreaProfissional(pfs);
          break;

       default:  break;
    }

    pfs = null;

    try {

       DefaultPieDataset pizza = new DefaultPieDataset();

       for (Iterator<String> i = dados.keySet().iterator(); i.hasNext();) {
          String key = i.next();
          if (key != null) {
             Double percent = new Double(dados.get(key));
             NumberFormat percentFormatter;
             String percentOut;
             percentFormatter = NumberFormat.getPercentInstance();
             percentOut = percentFormatter.format(percent);
             pizza.setValue(key + " " + percentOut, dados.get(key));
```

```
            }
        }
        chart = ChartFactory.createPieChart("", pizza, false, false, false);
        chart.setBackgroundPaint(Color.white);
        chart.setBorderVisible(false);
        chart.getTitle().setPaint(Color.lightGray);

        PiePlot plot = (PiePlot) chart.getPlot();
        plot.setLabelOutlinePaint(Color.LIGHT_GRAY);
        plot.setLabelBackgroundPaint(new Color(255, 255, 170));
        plot.setBackgroundPaint(new Color(242, 242, 242));
        plot.setLabelFont(new java.awt.Font("Arial", java.awt.Font.PLAIN, 12));



    } catch (Exception ex) {
        Logger.getLogger(GeradorDeEstatisticas.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
    }

    return chart;
}
```

TABELA 15: Código do método *gerarEstatisticas*

```
private HashMap<String, Float> porcentagemDeCandidatosPorEstado(ArrayList<PessoaFisica> pfs) {

    HashMap<String, Float> dados = new HashMap<String, Float>();
    int totais = pfs.size(); //quantidade total  de pessoas físicas

    for (int a = 0; a < pfs.size(); a++) {
        String valorUF = pfs.get(a).getUf(); //sigla do Estado da federacao

        if (dados.containsKey(valorUF)) {
            dados.put(valorUF, (float) (dados.get(valorUF) + 1 / (float) totais));
        } else {
            dados.put(valorUF, 1 / (float) totais);
        }

    }
    return dados;
}
```

TABELA 16: Código do método *porcentagemDeCandidatosPorEstado*

4.3.2 A biblioteca JFreeChart

O *JFreechart* é uma API escrita na linguagem java e que é utilizada para proporcionar a capacidade de geração de gráficos em aplicativos escritos nessa a linguagem. É uma API bastante flexível, sendo capaz de gerar vários tipos de gráficos (gráficos de barras, colunas, pizza, linha, XY, Gantt, linhas e colunas, etc.), com opções variadas de legendas, títulos, cores e fontes além de efeitos 3D. Tem a capacidade de exportar os gráficos gerados para os formatos de imagem mais comuns como o jpg, png, gif e mesmo o svg e o pdf. Como a linguagem java possui suporte para vários dispositivos gráficos como monitores, impressoras, *buffered images* via implementação da classe java.awt.Graphics2D, o *JFreeChart* através dessa abstração pode gerar gráficos para quaisquer desses dispositivos assim como outros implementados por terceiros. Possui ainda classes auxiliares para a sua utilização em applets e em servlets em ambiente Web.

4.4 Mecanismo de Busca

Esta seção descreve o mecanismo de busca da aplicação. Na seção 4.4.1 é apresentada a implementação e na seção 4.4.2 é apresentada a biblioteca Lucene.

4.4.1 Implementação

O mecanismo de busca foi implementado utilizando-se a biblioteca *Lucene*, versão 3.0, distribuída pela *Apache Software Foundation*. As classes deste módulo estão localizadas no diretório WEB-INF/classes/mecanismoDeBusca da aplicação e são: *Engine*, *ResultInfo*, *SbcAnalyserMaker* e *TermFreq*. A classe Engine é a principal, onde está a lógica responsável pela indexação e busca das informações dos currículos e vagas. A classe ResultInfo é uma classe auxiliar, utilizada para facilitar a exibição e manipulação das informações retornadas nas buscas feitas usando a API do *Lucene*. Os currículos e vagas são indexados e armazenados nos respectivos índices criados pelo Lucene e retornados em uma busca, usando a sua API, como documentos (objetos da classe *Document* do *Lucene*) com um número específico de campos. Dessa forma, utiliza-se cada objeto da classe ResultInfo para armazenar o conteúdo dos campos desejados de cada documento retornado pelo Lucene em uma busca, assim como os termos que aparecem com mais freqüência nos campos do documento e fragmentos de texto de determinados campos do documento que contém ocorrências das palavras-chave utilizadas na busca. Os atributos principais da classe ResultInfo são:

- id: Usado para armazenar o Id do objeto correspondente ao documento.

- pid: Usado para armazenar o Id do proprietário do objeto (pessoa física ou jurídica).
- nome: Usado para armazenar o nome da pessoa física ou título da vaga.
- highlight: Usado para armazenar fragmentos de texto de determinados campos do documento que contém as ocorrências das palavras-chave utilizadas na busca.
- termos: Usado para armazenar os termos que aparecem com mais freqüência nos campos do documento.

A classe TermFreq é uma classe auxiliar usada para a contagem de termos e respectivas frequências nos campos dos documentos, e a classe *SbcAnalyserMaker* é usada para retornar um objeto da classe *Analyzer* do Lucene para ser usado na indexação e, principalmente, para se definir os *stopwords* (palavras que serão removidas para a indexação, como preposições por exemplo) utilizados na filtragem do conteúdo a ser indexado.

Em relação à indexação, são utilizados dois índices diferentes, sendo um para armazenar os currículos e outro para armazenar as vagas. Os campos indexados para o índice de currículos são id, nome da pessoa, id da pessoa física, uf da pessoa física, área profissional, descrição da experiência, descrição dos conhecimentos, descrição dos idiomas, escolaridade, formação acadêmica e observações. Para o índice de vagas são indexados os campos id, uf da vaga, id da pessoa jurídica, área profissional, cargo, descrição e observações. Todas essas informações são obtidas da base de dados, processadas e indexadas. A indexação é disparada a intervalos de tempo pré-determinados e é executada em segundo plano através de um *thread*. O intervalo de tempo, assim como os diretórios onde estão localizados os índices, são configurado utilizando-se um arquivo de configurações de propriedades externo, no caso o arquivo MecanismoDeBusca.properties.

Outra funcionalidade utilizada no trabalho é o recurso de *highlight* (destaque) de palavras-chave, onde ocorrências das palavras-chave utilizadas na busca aparecem destacadas em fragmentos de texto de determinados campos, quando possível. Os fragmentos contextuais que aparecem no resultado de cada busca ajudam o usuário a julgar se os resultados são apropriados e se é necessário adicionar outras palavras-chave na busca para ajudar a estreitar o número de resultados retornados. Na busca de currículos os campos utilizados para o recurso de *highlight* são: descrição da experiência, descrição dos conhecimentos e observações. Na busca de vagas os campos utilizados são: descrição e observações. Esses campos foram escolhidos por possuírem uma grande quantidade de texto relevante para as buscas. Vale destacar que todos os campos indexados são pesquisados numa busca, mas

apenas os citados anteriormente são utilizados no recurso de *highlight*.

Adicionalmente são exibidos, para cada resultado de busca, os fragmentos de termos mais freqüentes naquele documento, para ajudar no julgamento dos resultados pelo usuário. Ainda em relação à exibição dos resultados de busca, destaca-se o uso do recurso de paginação, onde os resultados são divididos em páginas para melhor visualização do usuário, sendo que este pode escolher um número pré-determinado de resultados por página.

A FIGURA 19 exibe um exemplo do recurso de *highlight* no resultado de uma busca por currículos, mostrando a palavra-chave utilizada na busca no contexto do resultado exibido.

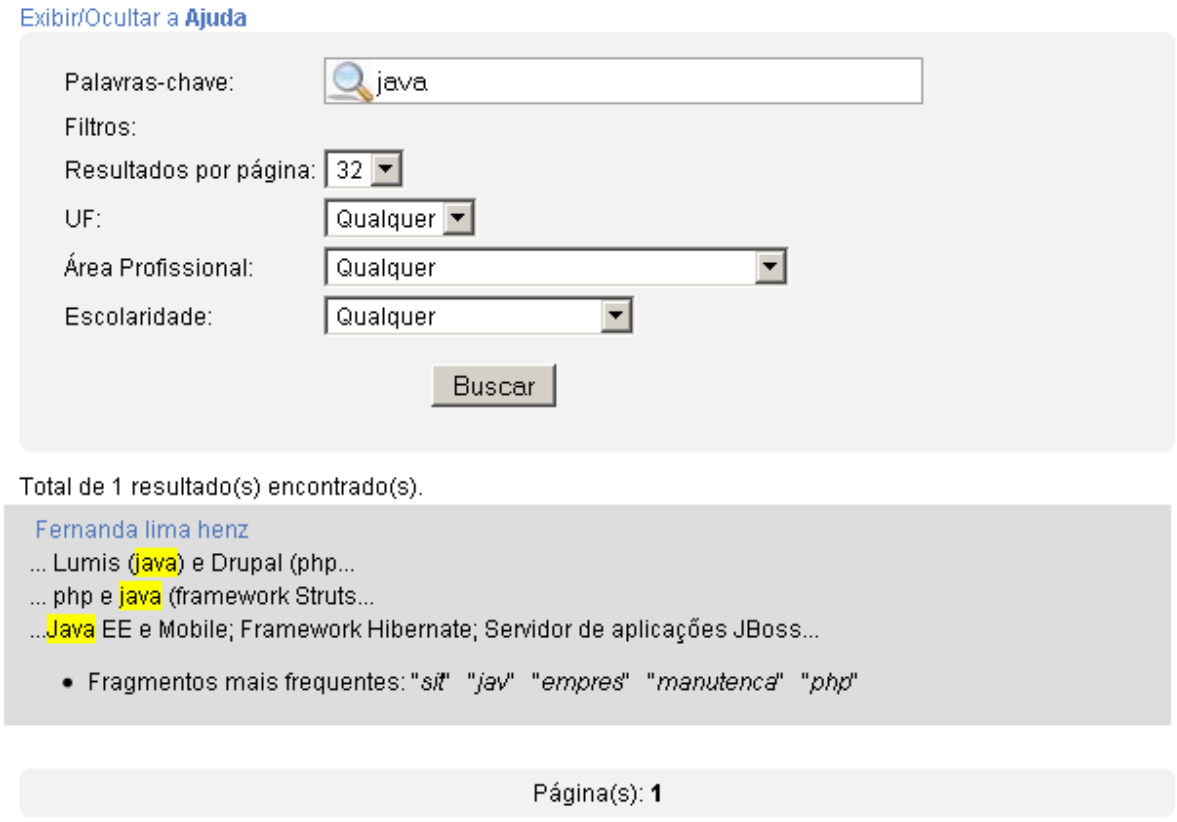

FIGURA 19: Exibição de resultados de uma busca

O fragmento de código do método *indexarTudo* da classe *Engine*, que contém a lógica responsável pela indexação dos currículos, é exibido a seguir na TABELA 17 como exemplo:

```
...
for (Iterator<PessoaFisica> it  = prep.listarPessoasFisicas().iterator();it.hasNext();) {
        Document doc = new Document();
        PessoaFisica pf = it.next();
      doc.add(new Field("id", String.valueOf(pf.getCurriculo().getId()), Field.Store.YES,
Field.Index.NO));
      doc.add(new Field("pid", String.valueOf(pf.getId()), Field.Store.YES, Field.Index.NO));
      doc.add(new Field("nome", filtrar(pf.getNome()), Field.Store.YES, Field.Index.NO));
      doc.add(new Field("UF", filtrar(pf.getUf()), Field.Store.NO, Field.Index.ANALYZED));
      doc.add(new Field("areaProfissional", filtrar(pf.getCurriculo().getAreaProfissional()),
Field.Store.NO, Field.Index.ANALYZED));
      doc.add(new Field("descExperiencia", filtrar(pf.getCurriculo().getDescExperiencia()),
Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.YES));
      doc.add(new Field("descConhecimentos", filtrar(pf.getCurriculo().getDescConhecimentos()),
Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.YES));
      doc.add(new Field("descIdiomas", filtrar(pf.getCurriculo().getDescIdiomas()), Field.Store.NO,
Field.Index.ANALYZED, Field.TermVector.YES));
      doc.add(new Field("escolaridade", filtrar(pf.getCurriculo().getEscolaridade()), Field.Store.NO,
Field.Index.ANALYZED));
      doc.add(new Field("descFormacaoAcademica",
filtrar(pf.getCurriculo().getDescFormacaoAcademica()), Field.Store.NO, Field.Index.ANALYZED,
Field.TermVector.YES));
      doc.add(new Field("observacoes", filtrar(pf.getCurriculo().getObservacoes()), Field.Store.YES,
Field.Index.ANALYZED, Field.TermVector.YES));
      indexWriterCurriculo.addDocument(doc);
   }
 ...
```

TABELA 17: Fragmento do código para a indexação dos currículos

Para cada currículo, são obtidos os campos relevantes usados na indexação. O método filtrar é utilizado para evitar a indexação de valores que eventualmente sejam nulos. O currículo é adicionado ao índice através do método doc.add(...) com os respectivos parâmetros que podem ser:

- *Field.Store.YES*: O valor do campo é armazenado inteiro no índice para posterior leitura.
- *Field.Store.NO*: O valor do campo não é armazenado no índice.
- *Field.Index.NO*: O campo não é indexado, logo não pode ser utilizado em buscas.
- *Field.Index.ANALYZED*: O campo é indexado e tem seu conteúdo filtrado/analisado

(stopwords removidas, por exemplo).

- *Field.TermVector.YES*: É criado um vetor com termos e freqüências das palavras contidas no campo.

O fragmento de código do método *fndCurriculos* da classe *Engine*, que contém a lógica responsável por permitir a busca no índice de currículos, é exibido a seguir como exemplo na TABELA 18. O código executa uma busca, com recurso de paginação, no índice de currículos e retorna os resultados num *array* de objetos *ResultInfo*, usado para a auxiliar na exibição dos resultados para o usuário. Destaca-se o uso dos métodos auxiliares *getHtmlHighligh*, para obter os fragmentos de texto contendo o destaque das palavras-chave da busca, e *obterTermosMaisFrequentes*, para obter a lista com os termos mais freqüentes no documento. A lógica responsável pela busca de vagas (implementada no método *fndVagas*) é semelhante a deste método.

```
...
String[] userQueryArray = new String[camposCurriculo.length];
Arrays.fill(userQueryArray, userQuery);
ArrayList<ResultInfo> resultados =  new ArrayList<ResultInfo>();
IndexReader reader = null;
TopDocs hits = null;

try {
   reader = IndexReader.open(FSDirectory.open(indexCurriculo), true);
   Searcher searcher = new IndexSearcher(reader);
   Query query = MultiFieldQueryParser.parse(Version.LUCENE_CURRENT, userQueryArray,
camposCurriculo, analyzer);

   hits = searcher.search(query, reader.numDocs());
   int totais = hits.totalHits;

//verificacao da range do numero da pagina enviada
   int numPgs = (totais % quantidade) == 0 ? (totais / quantidade) : ((totais / quantidade) + 1);
   if (pagina > numPgs) {
      pagina = 1;
   }

   int indiceInferior = (pagina - 1) * quantidade;
   int indiceSuperior = pagina * quantidade;

   for (int i = indiceInferior; i < Math.min(indiceSuperior, totais); i++) {

      Document doc = searcher.doc(hits.scoreDocs[i].doc);
      ResultInfo res = new ResultInfo();

      res.setId(Integer.parseInt(doc.get("id")));
      res.setNome(doc.get("nome"));

      String[] campos = {"descExperiencia", "descConhecimentos", "observacoes"};
      ArrayList<String> listaFinalHihglight = new ArrayList<String>();

      for (String campo : campos) {
         for (String s : getHtmlHighlight(analyzer, query, campo, doc.get(campo), 3, 30)) {
            listaFinalHihglight.add(s);
         }
```

```
        }

        res.setHighlight(listaFinalHihglight);
        ArrayList<String> listaFinalTermos = new ArrayList<String>();

        for (TermFreq termo : obterTermosMaisFrequentes(hits.scoreDocs[i].doc, indexCurriculo,
camposCurriculo, 5)) {
            listaFinalTermos.add(termo.getTerm());
        }

        res.setTermos(listaFinalTermos);
        resultados.add(res);
    }
...
return resultados;
```

TABELA 18: Fragmento de código responsável pela busca no índice de currículos

4.4.2 A biblioteca Lucene

O *Lucene* é uma biblioteca para indexação e busca de texto (proveniente de páginas Web em servidores remotos, de arquivos locais, arquivos de texto simples, arquivos xml, arquivos pdf ou qualquer outra fonte de dados da qual se possa extrair informação textual), de código fonte aberto e mantida pela *Apache Software Foundation*. Sua principal função é proporcionar a implementação de mecanismos de busca em aplicações de propósito geral. Sua principal característica é a de liberar a aplicação que a utiliza para lidar apenas com as regras de negócio do seu domínio principal, sem ter que se preocupar com os detalhes relativos à indexação e busca [14]. O seu sistema de busca permite o uso de *queries* sofisticadas, incluindo caracteres curinga (* e ?), uso do operador "~" para buscas usando lógica *fuzzy*, operadores NOT, OR e AND e suas combinações. Embora originalmente escrito apenas para a linguagem java, o *Lucene* hoje tem implementações para as linguagens Perl, Python, Ruby, C++, and C# (.NET). Existem também diversas bibliotecas adicionais escritas por terceiros para uso junto com o *Lucene*, para a expansão e adição de funcionalidades.

No *Lucene*, conceitualmente, um índice é representado como sendo um conjunto de documentos. Um documento é uma coleção de campos e um campo é uma seqüência de termos (Strings). Quando se adiciona um campo a um documento pode-se escolher se o

conteúdo dele vai ser armazenado no índice ou não, e se o conteúdo vai ser pré-processado ou não (para remoção de *stpowords*, que são palavras como preposições, conjunções). O índice também armazena estatísticas diversas sobre os termos (como freqüência e posição no documento) de forma que as buscas possam ser mais eficientes. [14].

Capítulo 5

Conclusões

Na Seção 5.1 deste capítulo, é feita uma análise retrospectiva e são abordadas as experiências adquiridas com o desenvolvimento deste projeto, além das facilidades e dificuldades encontradas ao longo do desenvolvimento. Na Seção 5.2, são descritos os principais trabalhos que podem ser desenvolvidos futuramente em cima deste projeto.

5.1 Análise Retrospectiva

Ao desenvolver este trabalho de conclusão de curso, pude colocar em prática e aprimorar os ensinamentos adquiridos nas disciplinas cursadas, principalmente os relativos a banco de dados e modelagem de sistemas usando *UML*, e também desenvolver o aprendizado e a aplicação de tecnologias novas e padrões de mercado, como o mapeamento objeto-relacional usando o *Hibernate*, a aplicação do padrão *MVC* para desenvolvimento e a utilização do *Lucene* para a construção do mecanismo de indexação e busca.

O aprendizado do modelo *MVC* para programação em componentes/camadas juntamente com o padrão *DAO* foi de muita valia, pois são tecnologias cada vez mais usadas e que facilitam e racionalizam o desenvolvimento de sistemas. Outro aprendizado interessante foi a implementação do mecanismo de busca utilizando o *Lucene*, que permite buscas com um alto grau de refinamento das *queries* e requer um esforço comparativamente pequeno para a sua implementação. A utilização do *Hibernate* foi muito importante, pois foi uma oportunidade de implementar a persistência de dados de uma maneira mais enxuta, não requerendo o esforço de se utilizar código JDBC puro. Também vale destacar o contato que tive com vários artigos técnicos, relativos à busca e indexação de informações, durante o desenvolvimento deste trabalho, os quais poderão ser muito úteis em outros projetos futuros.

Através deste trabalho pôde ser mostrada a importância de mecanismos de recuperação de informações (mecanismos de busca) em sistemas de informação de uma maneira geral, mas principalmente, para aqueles que trabalham com grandes volumes de informação. A utilização e o dimensionamento correto destes mecanismos tem um impacto grande no sentido de diminuir o tempo de recuperação das informações requeridas pelo usuário e aumentar a sua precisão.

5.1.1 Facilidades e Dificuldades Encontradas

O uso do *framework Struts* se provou de extrema valia, pois eliminou esforços repetitivos de programação (validação de campos, exibição de dados para o usuário) e passíveis facilmente de ocorrência de erros (muitas vezes difíceis de encontrar). A divisão do código entre componentes (*MVC, DAO*) se mostrou um pouco trabalhosa no início da implementação, mas se mostrou extremamente útil quando algumas partes do código foram modificadas para manutenção. Sem esta divisão em componentes a manutenção teria se tornado muito mais trabalhosa e complexa. O uso do *Hibernate* foi muito importante para tornar menor o esforço do mapeamento objeto-relacional, mas a sua configuração (mapeamento das classes de domínio e tabelas dos banco de dados) foi um pouco difícil, devido ao número de itens a serem configurados e dos tipos de relacionamentos entre as classes de domínio.

A construção do mecanismo de busca utilizando o *Lucene* foi relativamente simples, pois esta biblioteca possui uma *API* bem organizada e bem documentada, além de existirem várias comunidades de desenvolvedores que compartilham informações e exemplos de implementação sobre ela.

5.1.2 Prós e Contras das Tecnologias Usadas

O *framework Struts* tem como ponto forte a qualidade e a agilidade no desenvolvimento de aplicações Web, provendo meios para diminuir em muito a programação repetitiva e ajudando na legibilidade do código através da separação em camadas especializadas. Contudo, parte do código HTML gerado não segue alguns dos padrões Web adotados hoje (*tableless*, no caso específico de formulários com validação) e os códigos javascript gerados são relativamente grandes (código criado automaticamente para a validação de formulários, por exemplo).

O modelo *MVC* tem como grande atrativo a separação e organização dos diferentes tipos de código usados no desenvolvimento do sistema. Isso permite que, por exemplo, manutenção e mudanças no sistema sejam feitas facilmente, e que cada desenvolvedor trabalhe numa parte do sistema (interface com o usuário, gerenciamento de dados, etc.) em paralelo. Mas o desenvolvimento em camadas é um pouco mais demorado (devido à sua complexidade) e pode vir a se tornar confuso (pela quantidade de arquivos de código existente a mais em relação ao desenvolvimento sem camadas) se não for bem gerenciado.

O *Hibernate* simplifica o mapeamento objeto-relacional, tornando menor o trabalho de

implementação e facilitando a manutenção do código em caso de mudanças relativas ao banco de dados. Porém, o *Hibernate* apresenta um conjunto complexo de configurações e consome uma considerável quantidade de recursos do sistema, além de ter uma curva de aprendizado grande, o que torna a sua assimilação um pouco lenta.

5.2 Trabalhos Futuros

Seria interessante seguir na direção de fazer do sistema um serviço mais ágil e com mais recursos para os usuários, para isso novas características poderiam ser desenvolvidas e implementadas como:

- Implementar *feeds* por RSS ou Atom: Os usuários do sistema poderiam receber boletins com vagas recentes cadastradas, diretamente em seus leitores de *feed*, sem precisar acessar o sistema para saber se há novidades. Da mesma forma poderia ser implementado um sistema semelhante via e-mail.
- Desenvolver interfaces alternativas para acesso por dispositivos móveis (*mobile*): O sistema poderia ser acessado, além de por computadores, também por interfaces de celulares e *smartphones*, permitindo que os usuários possam acessar o sistema de qualquer lugar.
- Melhorar o sistema de buscas visando organizar os resultados da busca de currículos e de vagas por semelhança/proximidade de um determinado documento.
- Desenvolver um serviço Web (*Webservice*) para disponibilizar os currículos e vagas para outros sistemas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] PERSONA, Mario. A Internet e o Rh. Disponível em <http://www.mariopersona.com.br>. Acessado em 5 de Novembro de 2009.

[2] LYMAN, Peter; HAL R. Varian. How Much Info Project. Disponível em: <http://www.sims.berkeley.edu/how-much-info>. Acessado em 10 de Junho de 2010.

[3] SINGHAL, Amit (2001). Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering Nº 24.

[4] RIJSBERGEN, Van. Information Retrieval. Butterworth, 1980. ISBN 0-408-70929-4.

[5] CHO ET AL. Efficient Crawling Through URL Ordering. Disponível em: <http://dbpubs.stanford.edu:8090/pub/1998-51>. Acessado em 15 de Junho de 2010.

[6] FOWLER, Martin. UML Essencial. São Paulo: Bookman, 2004. ISBN 978-85-7307-610-3.

[7] ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de Banco de dados. Pearson, 2005. ISBN 478-45-7037-710-3.

[8] MANZANO, Jose Augusto. Mysql 5 - Interativo - Guia Básico de Orientação e Desenvolvimento. Érica, 2007. ISBN 9788536501451.

[9] CRAWFORD, William; KAPLAN, Jonathan. J2EE Design Patterns. O'Reilly, 2001. ISBN 0-596-00427-3.

[10] APACHE SOFTWARE FOUNDATION. Apache Struts. Disponível em: <http://struts.apache.org/> Acessado em 8 de Julho de 2010.

[11] BODOFF, Stephanie; HAASE, Kim; GREEN, Dale. The J2EE Tutorial. Addison-Wesley, 2008. ISBN 0-201-79168-4.

[12] BUSH, Vannevar. As We May Think. Atlantic Monthly, July, 1945. Disponível em: <http://www.theatlantic.com/doc/194507/bush>. Acessado em: 20 de Junho de 2010.

[13] SALTON, Gerard. The SMART System - Experiments in Automatic Document Processing. Prentice-Hall, 1971.

[14] HATCHER, Erik; GOSPODNETIĆ, Otis; MCCANDLESS, Michael. Lucene

in Action. Manning, 2008. ISBN 1933988177.

[15] GONÇALVES, Edson. Desenvolvendo Aplicações Web com Jsp, Servlets, Javaserver Faces, Hibernate, Ejb 3. Ciência Moderna. ISBN 9788573935721 .

APÊNDICE I: *Script* de criação das tabelas do banco de dados

```
CREATE TABLE Candidatura (
  id INTEGER UNSIGNED NOT NULL,
  Pessoafisica_Pessoa_id INTEGER UNSIGNED NOT NULL,
  Vaga_id INTEGER UNSIGNED NOT NULL,
  dataResposta DATETIME NULL,
  mensagem TEXT NULL,
  PRIMARY KEY(id, Pessoafisica_Pessoa_id, Vaga_id),
  INDEX Candidatura_FKIndex1(Pessoafisica_Pessoa_id),
  INDEX Candidatura_FKIndex2(Vaga_id)
);

CREATE TABLE Curriculo (
  Pessoafisica_Pessoa_id INTEGER UNSIGNED NOT NULL,
  areaProfissional VARCHAR(255)  NULL,
  descExperiencia TEXT NULL,
  descConhecimentos TEXT NULL,
  descIdiomas TEXT NULL,
  escolaridade TEXT NULL,
  dataUltimaAlteracao DATETIME NULL,
  descFormacaoAcademica TEXT NULL,
  observacoes TEXT NULL,
  PRIMARY KEY(Pessoafisica_Pessoa_id),
  INDEX Curriculo_FKIndex2(Pessoafisica_Pessoa_id)
);

CREATE TABLE Pessoa (
  id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nome VARCHAR(255)  NOT NULL,
  endereco VARCHAR(255) NULL,
  cep VARCHAR(8) NULL,
```

```
  telefone VARCHAR(8) NULL,
  uf VARCHAR(2) NULL,
  cidade VARCHAR(255) NULL,
  email VARCHAR(50) NULL,
  PRIMARY KEY(id)
);

CREATE TABLE Pessoafisica (
  Pessoa_id INTEGER UNSIGNED NOT NULL,
  cpf VARCHAR(11) NOT NULL,
  sexo CHAR NULL,
  dataDeNascimento DATE NULL,
  PRIMARY KEY(Pessoa_id),
  INDEX pessoafisica_FKIndex1(Pessoa_id)
);

CREATE TABLE Pessoajuridica (
  Pessoa_id INTEGER UNSIGNED NOT NULL,
  cnpj VARCHAR(15) NOT NULL,
  siteCorporativo VARCHAR(255) NULL,
  PRIMARY KEY(Pessoa_id),
  INDEX pessoajuridica_FKIndex1(Pessoa_id)
);

CREATE TABLE Usuario (
  Pessoa_id INTEGER UNSIGNED NOT NULL,
  login VARCHAR(30) NOT NULL,
  senha VARCHAR(30) NOT NULL,
  tipo INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(Pessoa_id),
  INDEX Usuario_FKIndex1(Pessoa_id)
);
```

```
CREATE TABLE Vaga (
  id INTEGER UNSIGNED NOT NULL,
  Pessoajuridica_pessoa_id INTEGER UNSIGNED NOT NULL,
  cargo VARCHAR(255) NULL,
  descricao TEXT NULL,
  dataDePublicacao DATETIME NULL,
  observacoes TEXT NULL,
  uf VARCHAR(2) NULL,
  areaProfissional VARCHAR(255) NULL,
  PRIMARY KEY(id),
  INDEX vaga_FKIndex1(Pessoajuridica_pessoa_id)
);
```

APÊNDICE II: Manual do usuário

***Login* do Sistema**

Para acessar o sistema são necessários um nome de usuário e senha cadastrados anteriormente.

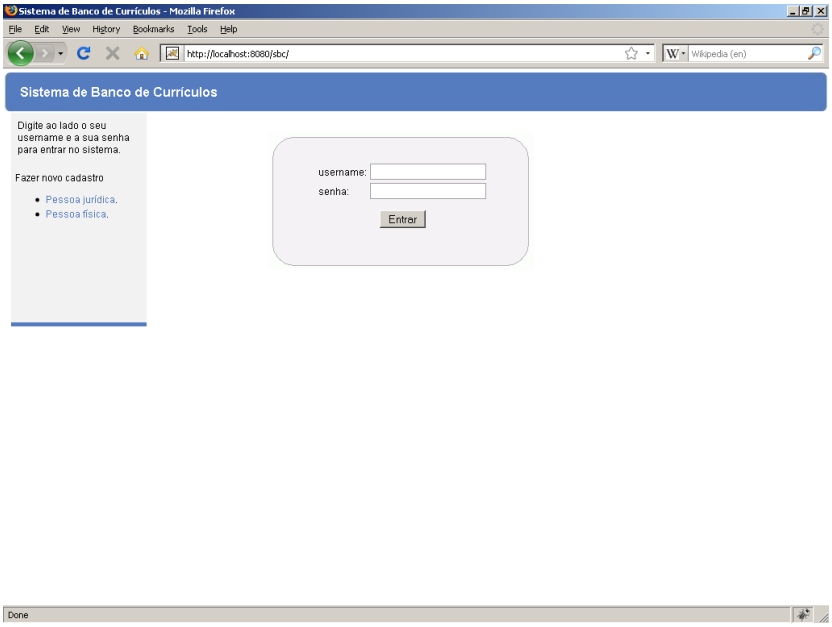

FIGURA 20: Tela de *Login* do Sistema

Cadastro de novos usuários

Para se cadastrar no sistema é necessário acessar uma das opções na tela de *login*: fazer novo cadastro de pessoa física ou fazer novo cadastro de pessoa jurídica.

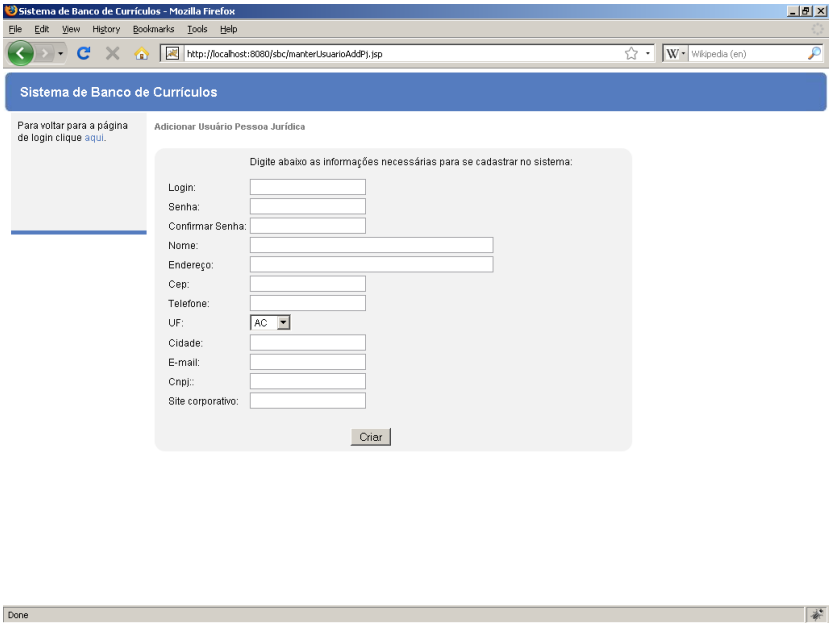

FIGURA 21: Tela de cadastro de pessoa jurídica

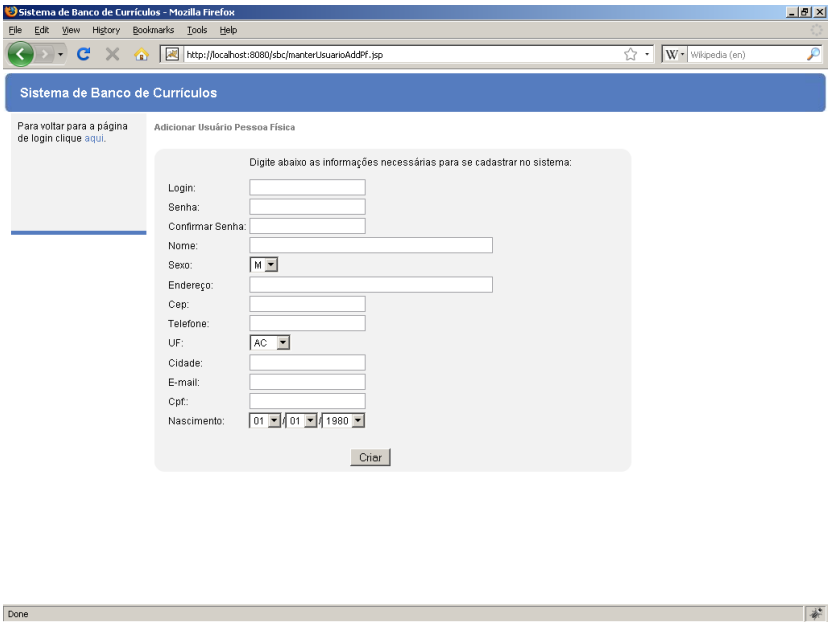

FIGURA 22: Tela de cadastro de pessoa física

Manutenção de vagas (pessoa jurídica)

Depois de logado o usuário do tipo pessoa jurídica pode acessar no menu a opção de manutenção de vagas, onde poderá cadastrar novas vagas, excluir e editar vagas previamente cadastradas e exibir as candidaturas para cada vaga, bastando para isso utilizar as opções disponíveis para cada vaga que aparece na lista. Clicando no nome da vaga na lista exibida, será aberta uma janela exibindo as propriedades da vaga. Para cadastrar uma nova vaga deve-

se clicar no botão “Adicionar nova vaga”. Uma janela de cadastro será aberta para permitir o cadastro da nova vaga.

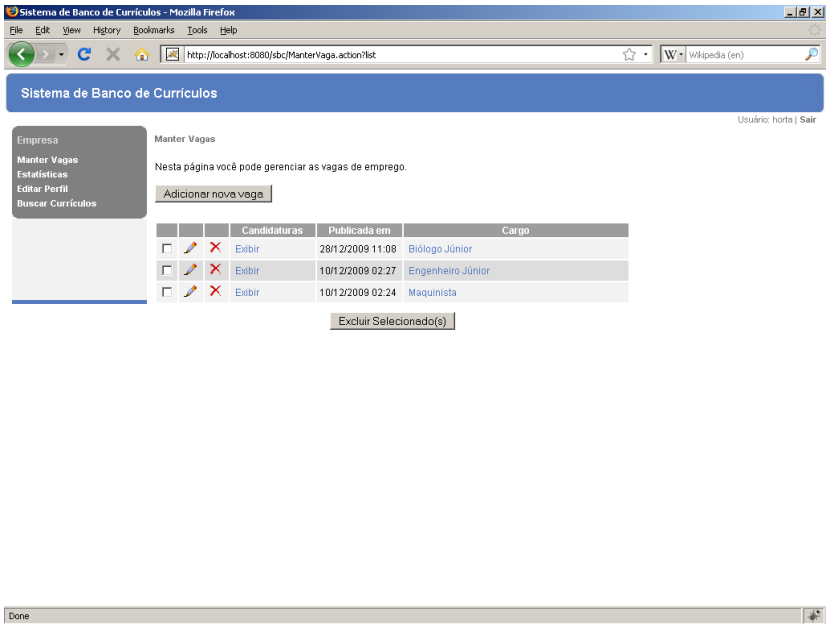

FIGURA 23: Tela de manutenção de vagas – lista de vagas e opções

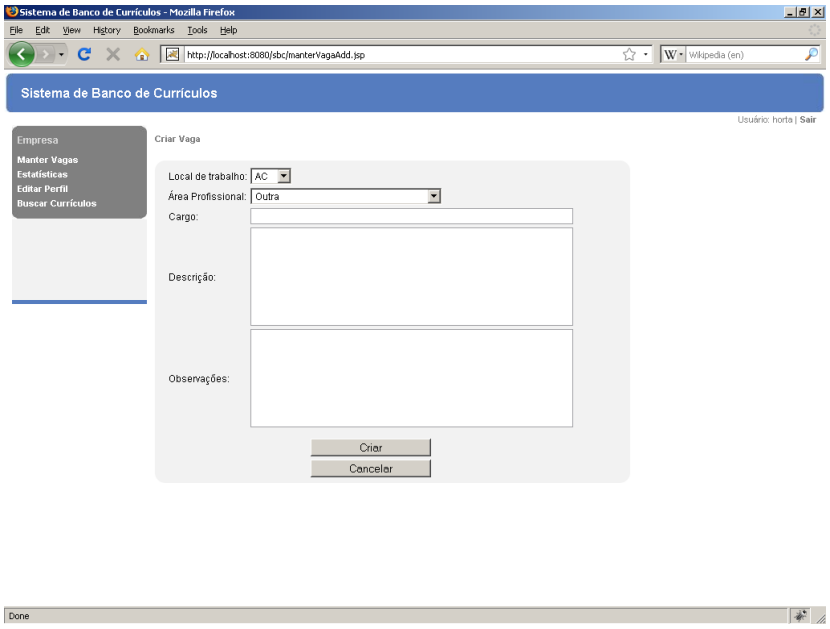

FIGURA 24: Tela de cadastro de nova vaga

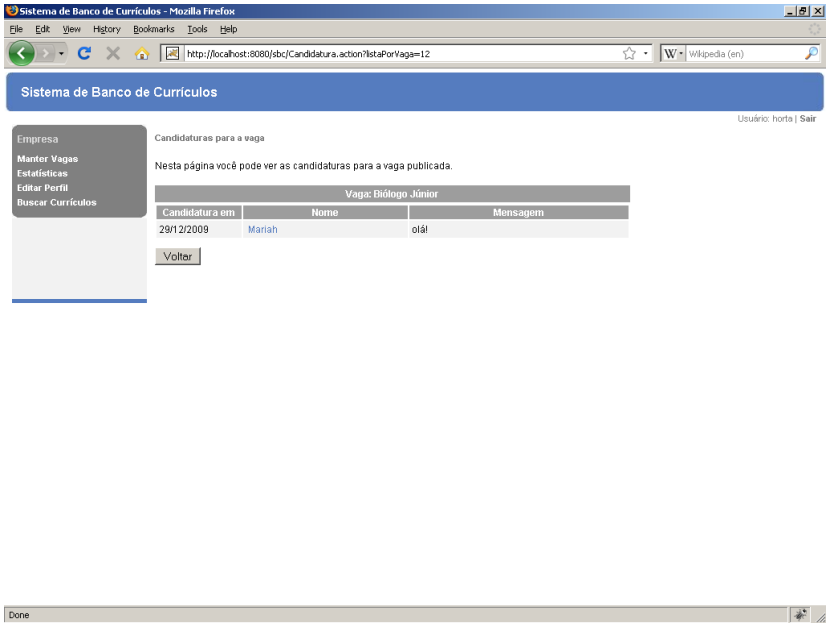

FIGURA 25: Tela de exibição de candidaturas para vaga

Busca de currículos (pessoa jurídica)

Depois de logado o usuário do tipo pessoa jurídica pode acessar no menu a opção de busca de currículos, onde poderá efetuar buscas na base de currículos cadastrados. Esta página contém um pequeno texto de ajuda para auxiliar o usuário na busca.

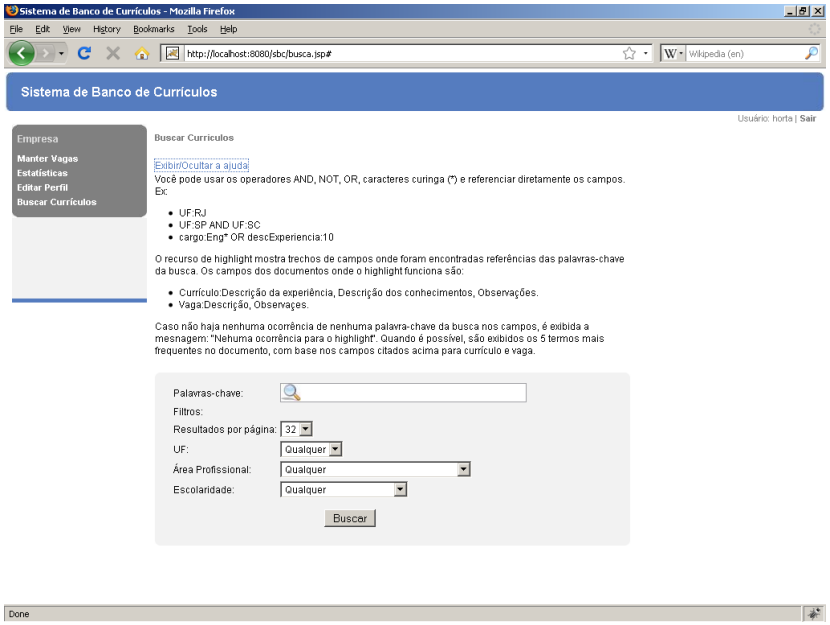

FIGURA 26: Tela de busca de currículos

Exibição de estatísticas (pessoa jurídica)

Depois de logado o usuário do tipo pessoa jurídica pode acessar no menu a opção de exibição de estatísticas dos usuários do tipo pessoa física cadastrados no sistema. As opções disponíveis são: porcentagens de candidatos por estado, por escolaridade, por faixa de idade e por área profissional.

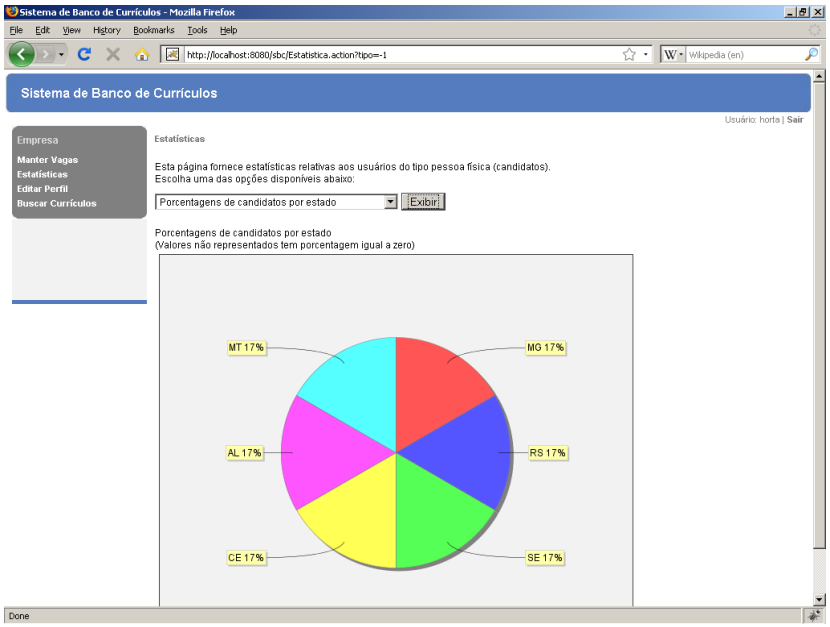

FIGURA 27: Tela de exibição de estatísticas

Editar perfil (pessoa jurídica)

Depois de logado o usuário do tipo pessoa jurídica pode acessar no menu a opção de editar perfil, onde poderá alterar dados relativos ao seu cadastro.

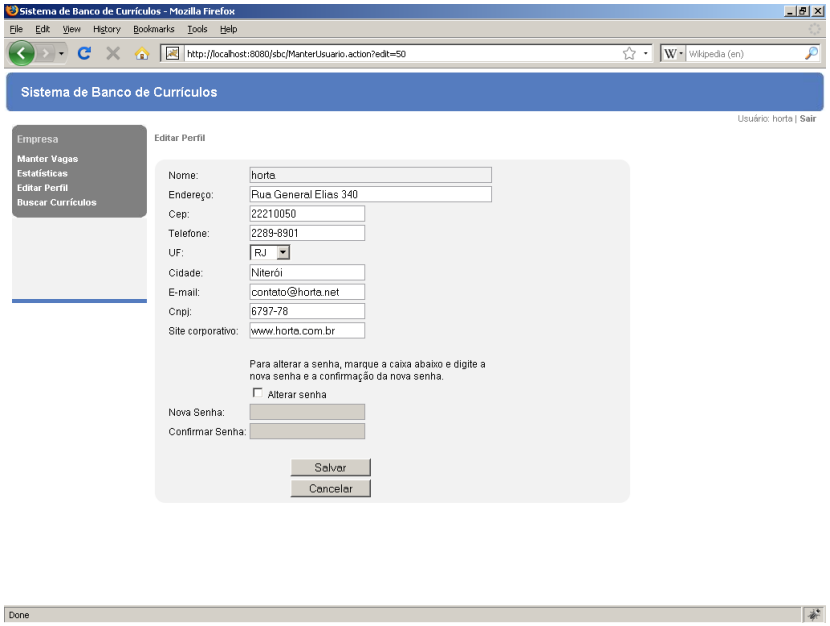

FIGURA 28: Tela de edição de perfil de pessoa jurídica

Buscar vagas (pessoa física)

Depois de logado o usuário do tipo pessoa física pode acessar no menu a opção de busca de vagas, onde poderá efetuar buscas na base de vagas cadastradas. Esta página contém um pequeno texto de ajuda para auxiliar o usuário na busca.

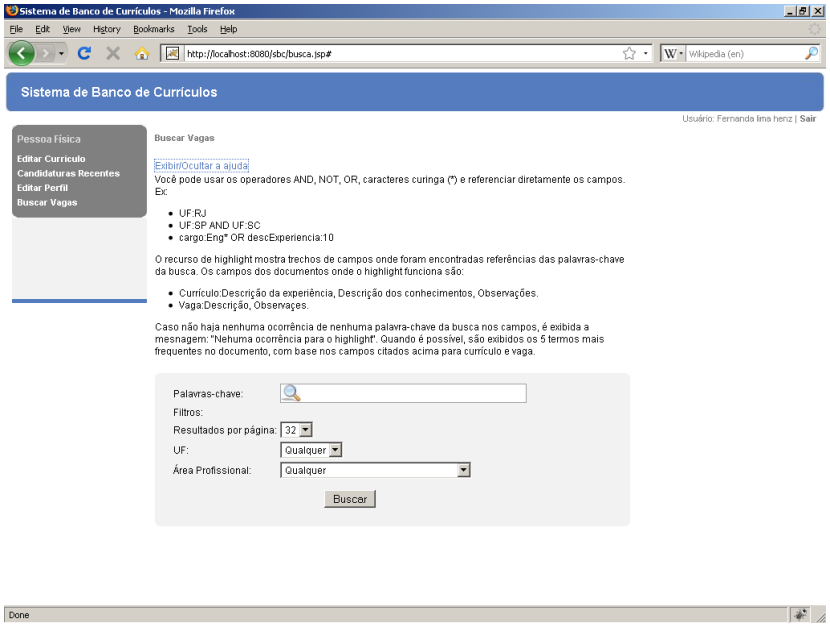

FIGURA 29: Tela de busca de vagas

Exibir candidaturas recentes (pessoa física)

Depois de logado o usuário do tipo pessoa física pode acessar no menu a opção de exibir as suas candidaturas recentes. Clicando no nome da vaga na lista exibida, terá acesso às propriedades da vaga.

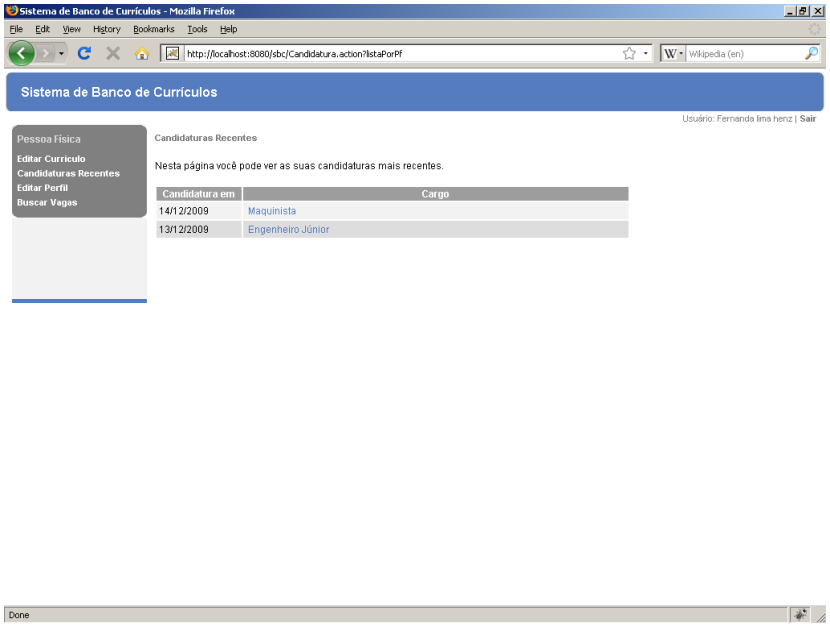

FIGURA 30: Tela de exibição de candidaturas recentes

Editar currículo (pessoa física)

Depois de logado o usuário do tipo pessoa física pode acessar no menu a opção de editar o seu currículo, alterando os dados que quiser.

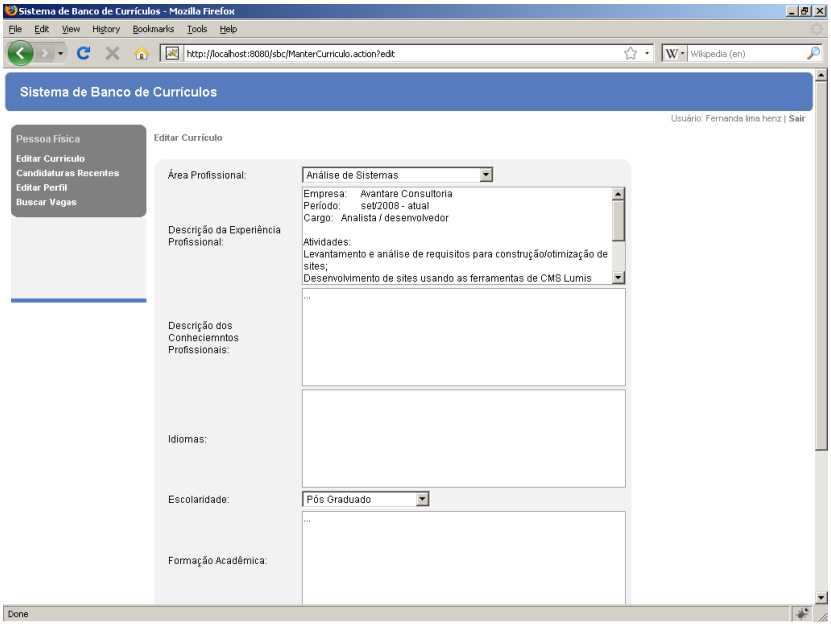

FIGURA 31: Tela de edição de currículo

Editar perfil (pessoa física)

Depois de logado o usuário do tipo pessoa física pode acessar no menu a opção de editar perfil, onde poderá alterar dados relativos ao seu cadastro.

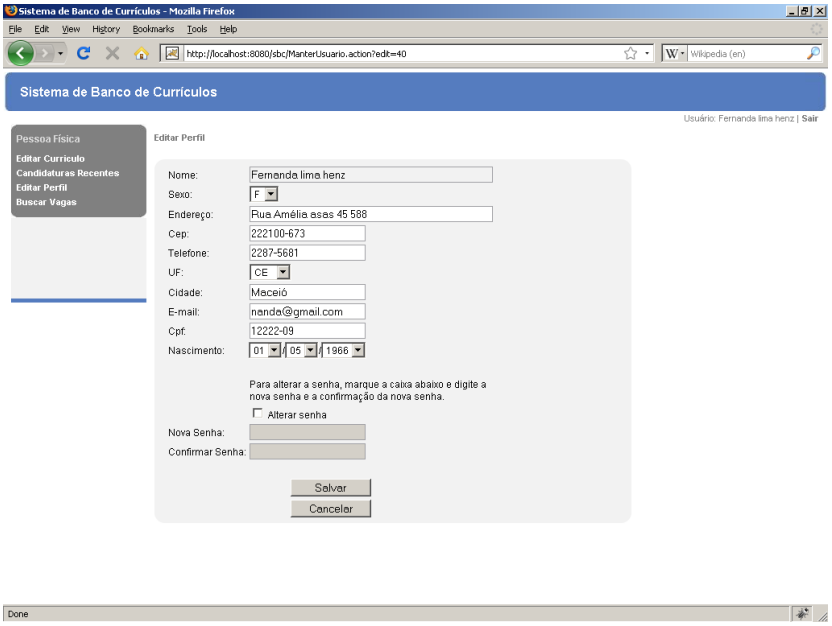

FIGURA 32: Tela de edição de perfil de pessoa física

Gerenciar usuários (Administrativo)

Depois de fazer o *login*, o administrador terá acesso à tela de gerenciamento de usuários. Para exibir um ou mais usuários cadastrados, deve-se digitar o nome ou parte dele na caixa de busca e clicar no botão “Localizar”. O sistema retornará uma lista com os usuários encontrados. Nesta lista há opções para gerenciamento dos usuários (editar, excluir, visualizar). Para se exibir todos os usuários cadastrados, deve-se deixar a caixa de busca em branco e clicar no botão “Localizar”.

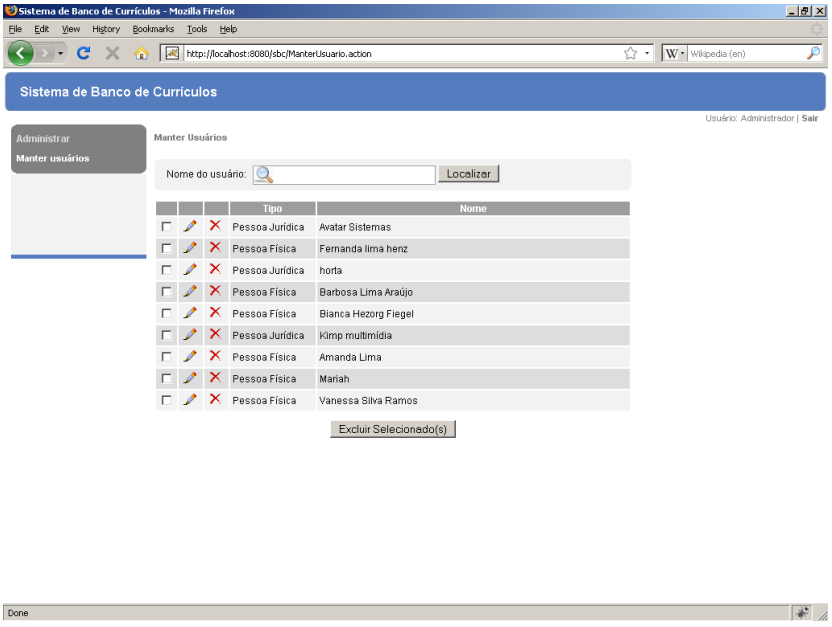

FIGURA 33: Tela de gerenciamento de usuários