FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GREGORY PETTERSON ZANELATO

BUSCA PREÇO – FERRAMENTA DE COMPARAÇÃO DE PREÇOS DE PRODUTOS DE SUPERMERCADOS

MARÍLIA
2005

GREGORY PETTERSON ZANELATO

BUSCA PREÇO – FERRAMENTA DE COMPARAÇÃO DE PREÇOS DE PRODUTOS DE SUPERMERCADOS

Monografia apresentada na Fundação de Ensino Eurípides Soares da Rocha, Mantenedora do Centro Universitário Eurípides de Marília-UNIVEM, como trabalho de conclusão de curso.

Orientador: Prof. Dr. Márcio Eduardo Delamaro.

MARÍLIA
2005

GREGORY PETTERSON ZANELATO

BUSCA PREÇO – FERRAMENTA DE COMPARAÇÃO DE PREÇOS DE PRODUTOS DE SUPERMERCADOS

Banca examinadora da monografia apresentada ao Programa de Graduação da UNIVEM/F.E.E.S.R., para obtenção do título de Bacharel em Ciência da Computação.

Resultado: _____

ORIENTADOR: Prof. Dr. Márcio Eduardo Delamaro

1º EXAMINADOR: Prof. Dr Márcio Eduardo Delamaro

2º EXAMINADOR: Prof. Dr. Ildeberto Aparecido Rodello

3º EXAMINADOR: Profa. Dra. Fátima de Lourdes dos Santos Nunes Marques

Marília, 06 de dezembro de 2005.

DEDICATÓRIA

Dedico esta monografia aos meus amigos e familiares e, em especial, em memória de minha mãe Cleusa Calegari Zanelato.

AGRADECIMENTOS

Agradeço a Deus por ter me ajudado a chegar até aqui e pela força que me dá todos os dias. Agradeço meu professor orientador Prof. Dr. Márcio Eduardo Delamaro.

Agradeço à minha família e amigos, por todo apoio e incentivo que me deram, por confiarem e acreditarem em mim.

Agradeço, igualmente, a todos que colaboraram para que este trabalho fosse concluído. Os graduandos e amigos Helton Hideharu Higute e Bruno Felipe Cerqueira Silva pela ajuda para a realização desse trabalho e a todos os meus amigos, colegas de turma, Carlos Eduardo Sanvido, Cristiano Seiti Takano, Dante Emanuel de Brito, Aparecido Nardo Junior, que estiveram comigo ao longo desses quatro anos de faculdade.

.

ZANELATO, Gregory Petterson. **Busca Preço: Ferramenta de Comparação de Preços de Produtos de Supermercados**. 2005. Trabalho de Conclusão de Curso - (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

.

RESUMO

Este trabalho consiste na implementação de um sistema baseado na WEB que permite ao usuário pesquisar e comparar os preços dos produtos de diversos supermercados de Marília, visando trazer, uma maneira simples e prática de economizar dinheiro e tempo. O objetivo é construir um website, que possa fornecer um comparativo de preços para produtos selecionados pelo usuário. O sistema deve também coletar dados como, por exemplo, os produtos mais consultados pelos consumidores e disponibilizar para os donos de supermercados, visando por exemplo, ajudar a aprimorar sua política de promoções.

ZANELATO, Gregory Petterson. **Busca Preço: Ferramenta de Comparação de Preços de Produtos de Supermercados**. 2005. Trabalho de Conclusão de Curso - (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

The project consists of a implementation of a system basing on the WEB that the user allows to search and to compare the products prices of diverse supermarkets of Marília, being aimed at to bring, a simple and practical way to save money and time. The objective is to construct a website, that can supply a comparative degree of selected products prices by the user. The system must also collect data that they will be return to the establishments with the more consulted products, aiming at to help the companies, for example, to improve its promotions politics.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – *Applications Programming Interface*

FTP– *File Transfer Protocol*

GML – *Generalized Markup Language*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Environment*

J2EE –*Java to Enterprise Edition*

J2SE –*Java to Standard Edition*

JCP – *Java Consortium Process*

JDBC - *Java Database Connectivity*

JNDI – *Java Naming Directory Interface*

JSP – *Java Server Pages*

JVM – *Java Virtual Machine*

SGBD – Sistema Gerenciador de Banco de Dados

SGML – *Standard Generalized Markup Language*

TelNet – Protocolo cliente-servidor de comunicações

XML – *Extensible Markup Language*

CAPÍTULO 1 - CONSIDERAÇÕES INICIAIS

1.1. Introdução

A Internet teve seu início em meados da década de 60, quando se desenrolou a Guerra Fria. Seu nome era ArphaNet e sua finalidade era manter comunicação entre as bases militares dos Estados Unidos. Após o término da guerra, a ArphaNet tornou-se praticamente inútil, assim, foi permitido o acesso aos pesquisadores que posteriormente cederam para as universidades as quais, sucessivamente, passaram para as universidades de outros países. Em 1970 já eram 15 universidades interligadas. Logo mais de 5 milhões de pessoas estavam conectadas com a rede (BOGO,2005).

Em 1990 surge a *World Wild Web* (WWW), criada no Laboratório Europeu de Física de Partículas, pelo físico Tim Berners-Lee, e até o ano de 1992 baseava-se na troca de informações por meio da utilização de e-mail ou por protocolos de transferência de arquivos, (*File Transfer Protocol* - FTP), via comandos, sem nenhuma interação com o usuário.

Com o surgimento do Mosaic em 1993, desenvolvido por Marc Andressen e outros estudantes do NCSA (*National Center for SuprerComputing Applications*), o usuário passou a contar com um programa de interfaces de interação com o usuário, sendo o marco da transformação e explosão da Internet. Logo, o Mosaic se tornou o NetScape.

Neste período a Microsoft Corporation lança o Internet Explorer e tem-se o início da "guerra" dos *browsers*. O crescimento da Web vinha sendo cada vez mais rápido, e repentinamente, o desafio passou a ser a criação de Web Sites maiores e melhores, até que então, a Web transformou-se em estratégia comercial.

Finalizado, a Web hoje é o palco multimídia para os negócios, é primordial que uma empresa tenha um website para desenvolver-se, além de não deixar de ser também uma estratégia de marketing, podendo assim, aumentar significativamente as vendas**.**

1.2. Objetivos

Atualmente, a Web é um dos principais meios que levam uma empresa ao sucesso. O mercado hoje está altamente competitivo, um website pode fazer a diferença entre uma empresa e outra. Procurou-se então desenvolver o estudo e em seguida o desenvolvimento de um website destinado não somente a uma empresa, mas a todas (no caso supermercados) que sentirem necessidade de associarem-se ao trabalho proposto. Além do estudo das tecnologias utilizadas para desenvolver o website, o grande foco deste trabalho é o usuário, pois ele poderá comparar preços de diversos produtos de vários supermercados, economizando dinheiro e tempo. As vantagens de um supermercado estar associado a este website, além de ser uma forma de marketing, é atrair clientes fazendo promoções ou reajustando os preços dos produtos. Evidentemente, o supermercado que oferecer o melhor preço ou a melhor qualidade de produtos atrairá mais clientes.

1.2.1. Geral

O principal objetivo deste trabalho é apresentar um estudo das linguagens Java, JSP, HTML XML, ou seja, buscar o domínio sobre as tecnologias utilizadas para o desenvolvimento de uma aplicação Web.

Com o desenvolvimento dessa aplicação Web, o usuário poderá consultar preços de diversos produtos e os donos dos supermercados terão acesso a um *feedback* da

quantidade de vezes que um produto foi procurado pelo consumidor. Esta aplicação facilitará a vida dos consumidores, pois os mesmos não precisarão sair de suas casas para pesquisarem preços de produtos. Os donos de supermercados poderão acompanhar qual produto está sendo mais bem aceito pelos consumidores, por meio do *feedback* que a aplicação disponibilizará.

1.3. Definição do Problema

O trabalho apresenta um estudo sobre as linguagens Java, JSP, HTML e XML, para o desenvolvimento de um sistema Web o qual o consumidor possa pesquisar e comparar os preços dos produtos diversos supermercados.

O principal problema apresentado será a padronização dos dados, ou seja, padronizar as informações contidas em diversos SGBD diferentes para que o sistema possa coletar essas informações sem maiores problemas.

Esse trabalho está embasado na linguagem Java e pretende trabalhar com suas características de forma a atender a necessidade da implementação de um sistema Web comercial voltado para os supermercados.

1.4. Justificativa

Com o avanço da utilização de sistemas via Web e com a alta taxa de desigualdade de preços dos produtos entre supermercados, surge a necessidade de desenvolver um sistema que proporcione ao consumidor uma maneira simples e eficiente de economizar dinheiro e tempo. A principal justificativa para o desenvolvimento deste sistema é a de que na cidade de Marília não existe um sistema de buscas de preços de produtos de supermercados, o qual será apresentado neste trabalho.

É de suma importância o estudo das tecnologias escolhidas para o desenvolvimento desse sistema Web, que é o grande objetivo deste trabalho.

Como justificativa para o tema proposto, pode-se destacar a importância do estudo da linguagem Java para implementar a funcionalidade do sistema e da linguagem HTML, fazendo com que a interface chegue ao usuário de uma maneira simples e amigável.

CAPÍTULO 2 - TECNOLOGIAS UTILIZADAS

2.1. Introdução

Foram utilizadas neste trabalho as linguagens, Java, JSP, HTML, XML. As linguagens Java e JSP foram utilizadas para darem funcionalidade a aplicação. A linguagem HTML foi utilizada para o desenvolvimento da interface do site e a linguagem XML foi utilizada para padronizar os registros de diversos bancos de dados para uma posterior leitura por meio de métodos desenvolvidos utilizando a linguagem Java.

2.2. Java

Java é uma linguagem de alto nível, com sintaxe extremamente similar à do C++, e com diversas características herdadas de outras linguagens, como *Smalltalk* e *Modula-3*. Adequada para o desenvolvimento de aplicações baseadas em Web, redes fechadas ou até mesmo programas *stand-alone*.

De acordo com Deitel (2002) os sistemas Java consistem em várias partes: um ambiente, a linguagem, a interface de programas aplicativos (API – *Applications Programming Interface*) Java e várias bibliotecas de classes.

Atualmente a linguagem Java é a força propulsora de alguns dos maiores avanços da computação mundial como:

- Comércio eletrônico no *World Wild Web*(WWW).
- Acesso remoto a banco de dados.
- Banco de Dados Distribuídos.

- Ensino à distância.
- Gerência de Documentos.
- Interatividade em páginas WWW.
- Integração entre dados e forma de visualização.

Algumas características da Linguagem Java de acordo com Deitel (2002).

- **SIMPLICIDADE:** Java é muito semelhante com C++, mas não possui sobrecarga de operadores, *unions*, *structs*, aritmética de ponteiros, e a memória alocada dinamicamente é gerenciada pela própria linguagem, com o *Garbage Collection* que é um algoritmo para deslocar regiões de memória que não estão mais em uso.

- **ORIENTAÇÃO A OBJETOS**: Java segue a linha iniciada por *Smalltalk*, ou seja, uma linguagem orientada a objetos, ao contrário de C++ que é uma linguagem híbrida. A maior parte do elementos de um programa em Java são objetos.

O código é organizado em classes, que podem ter relacionamentos de herança entre si. Ao contrário de C++ que possui herança múltipla, a linguagem Java só possui herança simples.

- **MULTITHREADING:** Atualmente os Sistemas Operacionais, como o Windows, OS/2, Linux, são multitarefa, ou seja, pode executar duas ou mais tarefas simultaneamente. Java fornece recursos para a programação concorrente, por exemplo, pode-se executar duas ou mais tarefas com o uso de *threads*, cada *thread* fica responsável por uma tarefa, como a programação concorrente em geral é muito complexa. Java também fornece recursos para sincronização de processos, o que torna a programação mais simples.

- **PROCESSAMENTO DISTRIBUÍDO:** Chamadas de funções de acesso remoto (*sockets*) e os protocolos Internet mais comuns (HTTP, FTP , TelNet, etc) são suportadas em Java, de forma que a elaboração de aplicativos baseados em cliente/servidor é facilmente obtida.

- **EXCEÇÕES:** O programador em C, C++, está acostumado com os "travamentos" da máquina. Por exemplo, na simples tentativa de abrir um arquivo inexistente, pode obrigar o programador a reiniciar a máquina. Em Java isso não acontece, já que a Maquina Virtual Java faz uma verificação em tempo de execução quanto aos acessos de memória, abertura de arquivos e uma série de eventos que possam desencadear um "travamento" na máquina. Com o tratamento de exceções, fica mais fácil para o programador detectar os erros e corrigi-los sem que a máquina trave.

- **GARBAGE COLLECTOR:** Em Java o programador não precisa se preocupar com o gerenciamento de memória. Todo bloco de memória é alocado dinamicamente. O *Garbage Collector* fica varrendo a memória de tempos em tempos liberando automaticamente os blocos que não estão sendo mais utilizados.

2.3. JSP (JAVA SERVER PAGE)

A JSP é uma tecnologia baseada em Java com a exclusiva função de simplificar a produção de páginas dinâmicas. Com JSP, os designers e programadores da Web podem rapidamente incorporar elementos dinâmicos em páginas da Web usando Java embutido e algumas *tags* (formatação de código usada em documentos HTML) de marcação simples.

Inicialmente, o cliente faz uma requisição por meio de um *browser* de uma página JSP, que nada mais é do que um documento texto que combina *tags* HTML com *tags* JSP que então, será processada pelo servidor.

Se for a primeira vez que o cliente requisitou a página JSP, ela será transformada em um programa Java (denominado *Servlet*), sendo compilado e gerado um *bytecode* (denominado *.class*) e, a partir deste, é gerada uma página HTML que será enviada para o *browser* do cliente. Se for a segunda vez que a página foi requisitada, é verificado apenas se ocorreu alguma mudança, caso não ocorra, o *bytecode (.class)* é chamado para gerar a página HTML.

De acordo com Anselmo (2002), a JSP usa a linguagem Java como base para sua linguagem de *scripts*, utilizando todo o potencial desta, sendo por esse motivo que a JSP se apresenta muito mais flexível e muito mais robusta do que as outras plataformas.

O Quadro 2.1 representa um exemplo de código na linguagem JSP:

```
<% out.println(“Alô Mundo!”); %>
```

Quadro 2.1. Exemplo de Código em JSP

O código acima refere-se a uma instrução na linguagem JSP para exibir na tela o conteúdo passado entre as aspas. Toda instrução JSP deverá estar entre as *tags* especiais que são os simbolos menor(<) juntamente com o simbolo porcentagem(%), para delimitar quando uma instrução será iniciada e os simbolos porcentagem(%) juntamente com o simbolo maior(>), para especificar o término da instrução, conforme o Quadro 2.1.

```
<%
     instrução
%>
```

Quadro 2.2. Exemplo de tags especiais

2.3.1. JavaBeans

Os *JavaBeans* são componentes de software projetados para serem reutilizáveis. Esse modelo de componente pode ser definido como um conjunto de classes e interfaces na forma de pacotes Java, com o intuito de serem usados de uma maneira que permita isolar e encapsular um conjunto de funcionalidades. Os componentes *JavaBeans* são conhecidos também como *Beans* (Anselmo,2002).

JavaBean é um programa em Java com construtor vazio contendo métodos que podem ser invocados por meio de instruções na linguagem JSP.

Neste trabalho, os *JavaBeans* serão utilizados para implementar as principais funcionalidades do sistema. Foram desenvolvidos neste trabalho vários *Beans.* As classes desses *Beans* serão importadas por meio das páginas dinâmicas em JSP.

Algumas características do JavaBeans:

- **INTROSPECÇÃO E REFLEXIVIDADE:** um *JavaBean* usa um padrão de codificação permitindo que uma ferramenta de edição visual interaja com o componente e deduza/altere suas características,(eventos, propriedades e métodos) em *build-time* ou *run-time*.
- **PERSISTÊNCIA E EMPACOTAMENTO:** capacidade de armazenar, recuperar ou transmitir um componente por meio de uma mídia digital.

2.4. HTML

HTML é uma linguagem com a qual se definem as páginas web. Basicamente trata-se de um conjunto de etiquetas que servem para definir a forma na qual se apresentará o texto e outros elementos da página.

A HTML nada mais é do que um texto puro contendo algumas instruções de formatação, em forma de códigos marcação HTML ou *tags*, que informa ao *browser* como exibir e imprimir os documentos.

2.5. XML

O XML provém de uma linguagem que inventou IBM em meados da década de 70. A linguagem da IBM chama-se GML (*General Markup Language*) e surgiu pela necessidade que tinham na empresa de armazenar grandes quantidades de informação de temas diversos.

XML é uma tecnologia na verdade muito simples, que tem ao seu redor outras tecnologias que a complementam e a fazem muito maior e com possibilidades muito mais amplas.

XML, com todas as tecnologias relacionadas, representa uma maneira distinta de fazer as coisas mais avançadas, cuja principal novidade consiste em permitir compartir os dados com os quais se trabalha a todos os níveis, por todas as aplicações e suportes. Sendo assim, o XML tem um papel importantíssimo neste mundo atual, que tende à globalização e à compatibilidade entre os sistemas, já que é a tecnologia que permitirá compartir a informação de una maneira segura, confiável e fácil. Ademais, XML permite ao programador e aos suportes dedicar seus esforços às tarefas importantes quando trabalha com os dados, já que algumas tarefas trabalhosas como a validação destes ou o percorrido das estruturas corre a

cargo da linguagem e está especificado pelo padrão, de modo que o programador não tem que se preocupar por isso. XML é interessante no mundo da Internet e do *e-bussiness*, já que existem muitos sistemas distintos que necessitam comunicar-se entre si, porém, como se pode imaginar, interessa igualmente a todos os ramos da informática e o tratamento de dados, já que permite muitos avances na hora de trabalhar com eles.

2.5.1. Sintaxe da XML

Pode-se dizer que XML é parte do SGML, porque na realidade as normas que tem são muito simples. Escreve-se em um documento de texto ASCII, igual a HTML e no cabeçalho do documento é colocado o texto

```
<?xml versao="1.0"?>
```

Quadro 2.3 Cabeçalho de um documento XML

No resto do documento deve-se escrever etiquetas como as de HTML, por isso a linguagem se chama XML, linguagem de etiquetas espalhada. As etiquetas se escrevem aninhadas, uma dentro de outras conforme o exemplo no Quadro 2.4.

```
<ETIQ1>...<ETIQ2>...</ETIQ2>...</ETIQ1>
```

Quadro 2.4 Exemplo de Etiquetas

Qualquer etiqueta pode ter atributos. Pode-se colocar quaisquer atributos.

```
<ETIQ atributo1="valor1" atributo2="valor2"...>
```

Quadro 2.5 Exemplo de etiqueta com atributos

Os comentários de XML se escrevem igual aos de HTML, conforme o Quadro 2.6.

```
<!-- Comentário -->
```

Quadro 2.6 Exemplo de comentário em XML

A XML conta com muitas outras linguagens e tecnologias trabalhando ao seu redor, entretanto, não cabe a menor dúvida que a sintaxe XML é realmente reduzida e simples.

Por exemplo, desejando-se salvar a informação relacionada com um filme em um documento XML pode-se utilizar um esquema com as seguintes etiquetas.

```
<?xmlversao="1.0"?>
    <FILMEnome="OPadrinho"ano=1985>
        <ELENCO></DIRETORnome="GeorgieLucas">
           </INTÉRPRETEnome="MarlonBrando"interpreta-a="DonCorleone">
           </INTERPRETEnome="AlPacino"interpreta-a="MichaelCorleone">
        </ELENCO>
        </ROTEIROdescrição="Filme de máfias sicilianas nos EstadosUnidos">
    </FILME>
```

Quadro 2.7 Sintaxe XML

Como se pode ver no Quadro 2.7, foram criadas algumas etiquetas para colocar neste exemplo e foram ordenadas de forma que a etiqueta maior é O “FILME” e dentro dela contém o “ELENCO” e o “ROTEIRO”. Por sua vez, dentro de “ELENCO” contém tanto o “DIRETOR” como os atores (INTERPRETE).

2.6. Considerações Finais

Neste capítulo foram abordadas as histórias das linguagens que serão utilizadas neste trabalho para o desenvolvimento de um website e uma ferramenta de extração de dados. As

linguagens como Java e JSP, por exemplo foram escolhidas devido à uma grande influência no mercado atualmente e também por suas características.

CAPÍTULO 3 - DESENVOLVIMENTO DO PROJETO

3.1. Introdução

O objetivo deste trabalho é a construção de um website que colete a descrição e os preço de produtos de diversas bases de dados diferentes. Considerando três supermercados, cada um com um SGBD diferente (Oracle, Mysql, Postgre), seria um problema para efetuar as buscas. Resolveu-se então extrair os dados dos bancos e transformá-los em um arquivo em formato XML facilitando,assim, as buscas.

Primeiramente foi desenvolvida uma aplicação que extrai os dados dos bancos e gera um arquivo XML com os dados dos produtos. Feito isso, foi desenvolvido um website que faz as buscas diretamente sobre os arquivos XML gerados.

3.2. Arquitetura do Sistema

Primeiramente foi desenvolvido uma aplicação *desktop* que irá buscar os dados no banco do cliente (supermercado) e gerar um arquivo XML para ser acessado pelo website.

Para cada departamento do website, foram criados dois *JavaBeans*. Para o departamento “BEBIDAS” por exemplo, foram criados os *Beans*, “`buscaBebidas`” e `comboBebidas`, para o departamento “AVES” foram criados os *Beans* “`buscaAves`” e `comboAves` e assim por diante.

O *Bean* “buscaBebidas” é encarregado de buscar informações como a descrição e o preço do produto no arquivo XML dos supermercados. O *Bean* “comboBebidas” é encarregado de buscar a descrição e o código dos produtos, que também estão nos arquivos XML dos supermercados, para popular o “combo" da página.

Foi criado também um *Bean* que é usado por todos os departamentos. Este *Bean* foi denominado "`Conexao`" e sua única e principal funcionalidade é inserir no banco o código do departamento e a descrição do produto pesquisado pelo usuário.

No módulo "LOGIN", foi criado um *Bean* denominado "`login`" para controlar o acesso ao módulo relatório. Este *Bean* recupera do banco de dados o *login* e senha, que serão comparados com o *login* e senha digitada pelos usuários, no caso os donos de supermercados.

O *Bean* "`relatório`" foi o ultimo criado. Esse *Bean* interage com o módulo "RELATORIO" e sua função é resgatar a descrição e a quantidade de vezes que um produto foi pesquisado referente ao departamento selecionado pelo usuário. A Figura 3.2 ilustra a arquitetura do sistema.

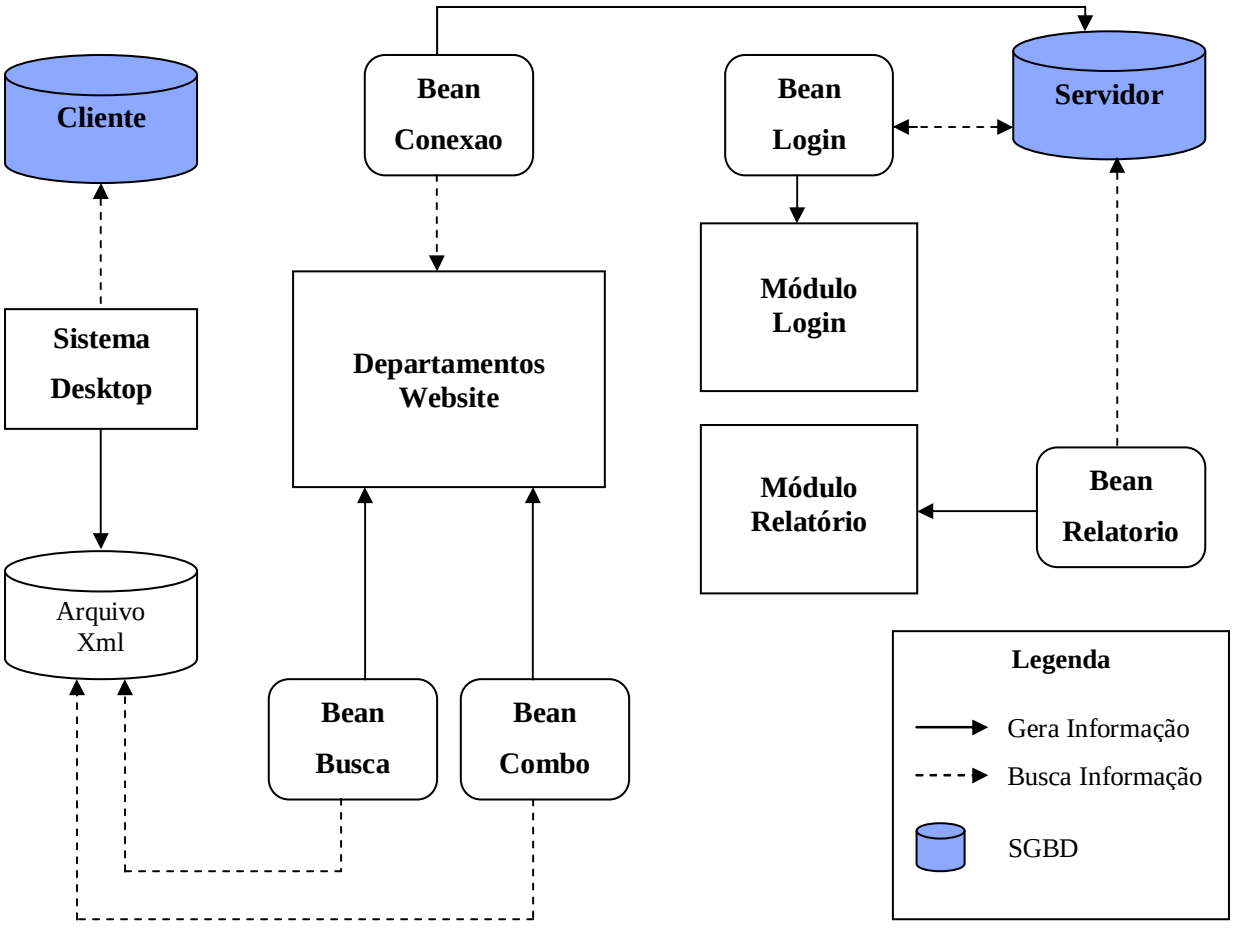

Figura 3.1: *Arquitetura do sistema*

No (Apêndice A) encontra-se o restante das fases da engenharia de software do sistema web e do sistema *desktop*

3.2.1 Diagrama de Classes

O diagrama de classe é um gráfico que mostra a estrutura do sistema : classes , tipos e seus conteúdos e relações.

- **Diagrama de Classes Sistema Desktop**

A Figura 3.2 representa o diagrama de classe do sistema *desktop* desenvolvido para a extração de dados do SGBD do cliente e geração dos arquivos XML.

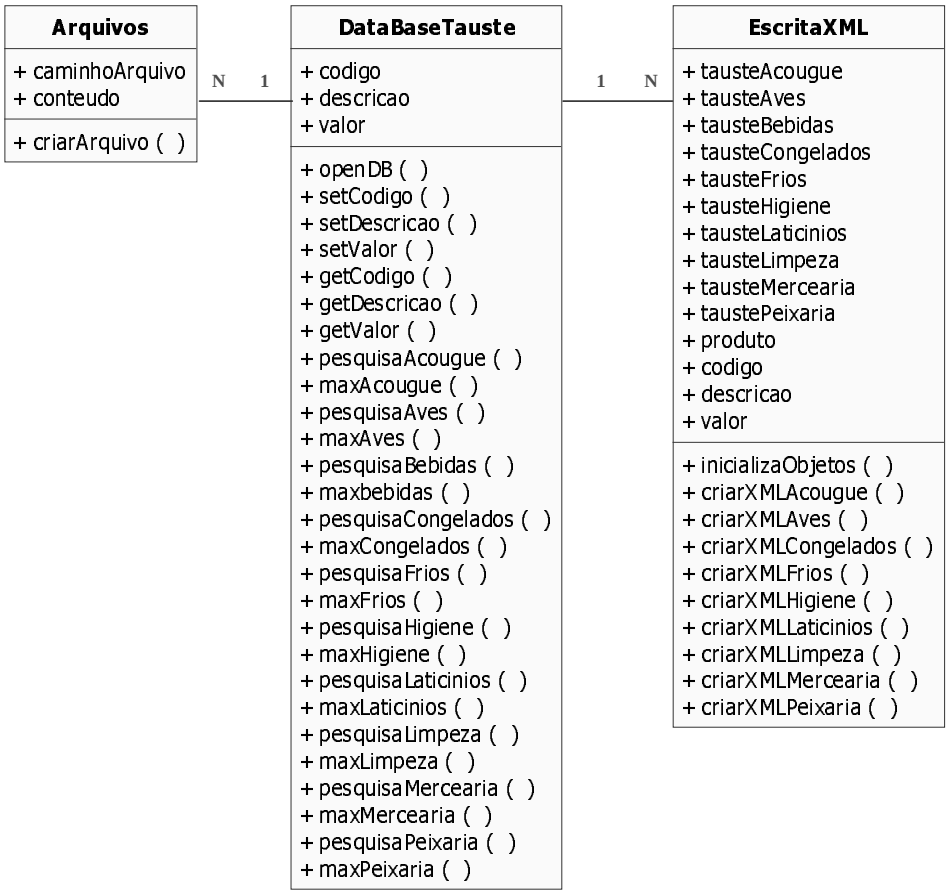

Figura 3.2: Diagrama de Classes do sistema desktop

- **Diagrama de Classes Sistema Web**

A Figura 3.3 esboça o diagrama de classe do sistema web. As classes “`Conexao`”, “`Relatorio`” e “`Login`” são as únicas que possuem relações, pois as outras classes são usadas somente para dar funcionalidade ao website e portanto, elas não possuem relações entre si.

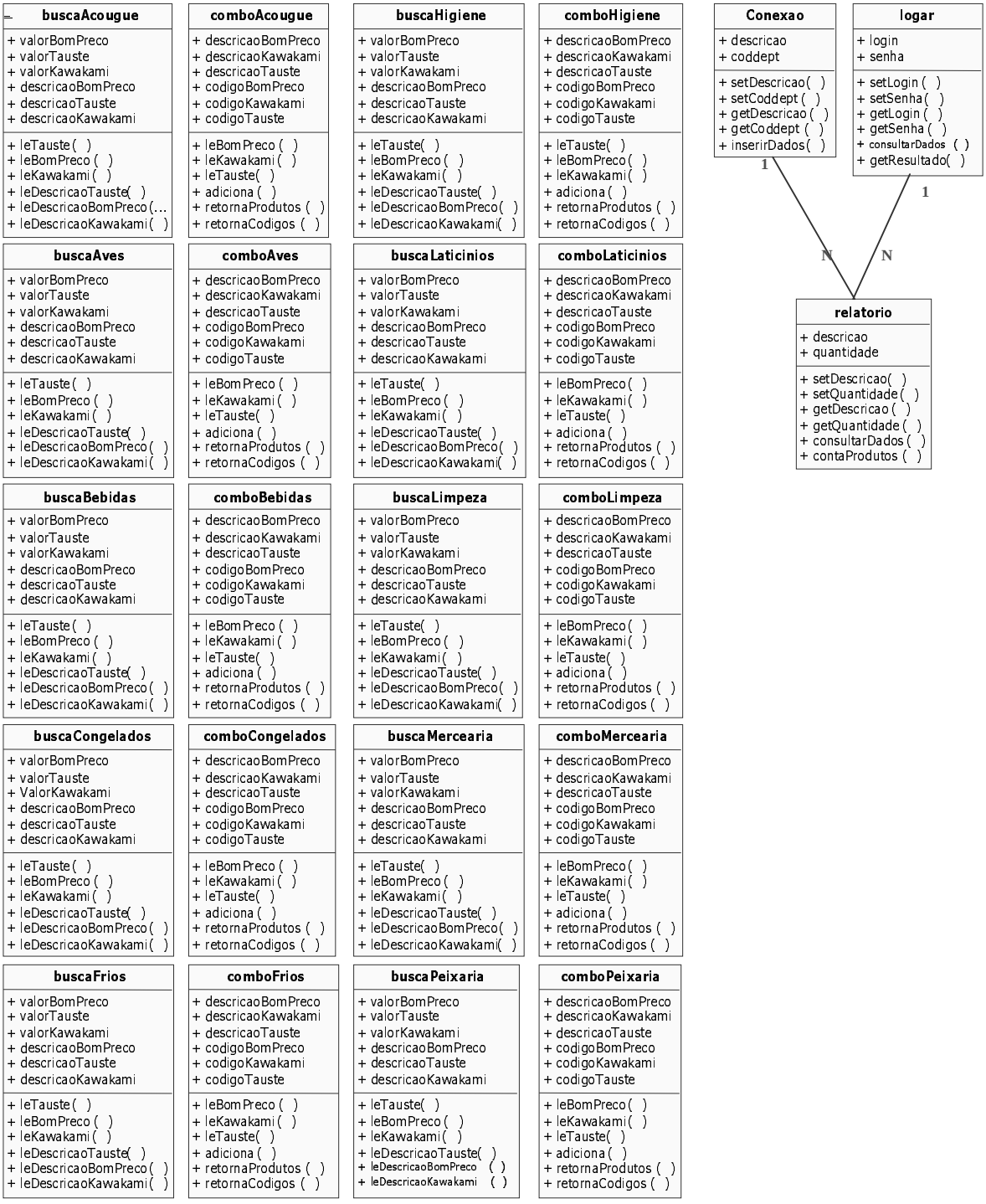

Figura 3.3: Diagrama de Classes do sistema web

3.2.2. Diagrama de Caso de Uso

Os diagramas de caso de uso exibem a visão externa do sistema e suas interações com o mundo exterior descrevendo seus requerimentos e suas responsabilidades e possuem três elementos: Ator (agente que interage com o sistema), Caso de Uso (o comportamento da classe) e Interação (envio e recebimento de mensagens da comunicação ator - sistema).

- **Diagrama de Caso de Uso Sistema Desktop**

Neste diagrama o ator que irá interagir com o sistema será os donos de supermercados. O ator executará em sua máquina, ou seja, na máquina do supermercado, a aplicação. Essa aplicação irá ler os registros do banco e criará um arquivo XML contendo os registros lidos.O arquivo é criado na máquina do cliente(supermercado), copiado para o servidor do cliente e é acessado posteriormente pela aplicação web desenvolvida. A Figura 3.4 representa o diagrama de caso de uso do sistema *desktop*.

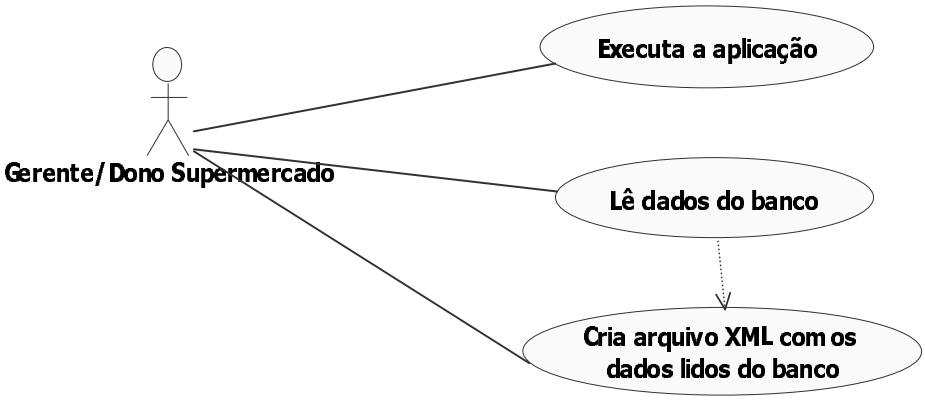

Figura 3.4: *Diagrama de Caso de Uso do sistema desktop*

- **Diagrama de Caso de Uso Sistema Web**

Neste diagrama os atores são os donos de supermercados e os usuários que efetuarão as buscas no site.

O ator cliente selecionará o departamento que desejar, logo selecionará um produto referente a este departamento. O cliente estando com o produto selecionado, o sistema fará uma busca e exibirá na tela dados do produto pesquisado e inserirá no SGBD do servidor a descrição do produto e o código do departamento.

O ator gerente irá efetuar o *login* no sistema e será direcionado para uma página de relatórios. Nesta página o gerente selecionará o departamento desejado e o sistema buscará na base de dados do servidor a descrição dos produtos do departamento selecionado e a quantidade de vezes que esses produtos foram pesquisados. A Figura 3.5 ilustra o diagrama de caso de uso do sistema web.

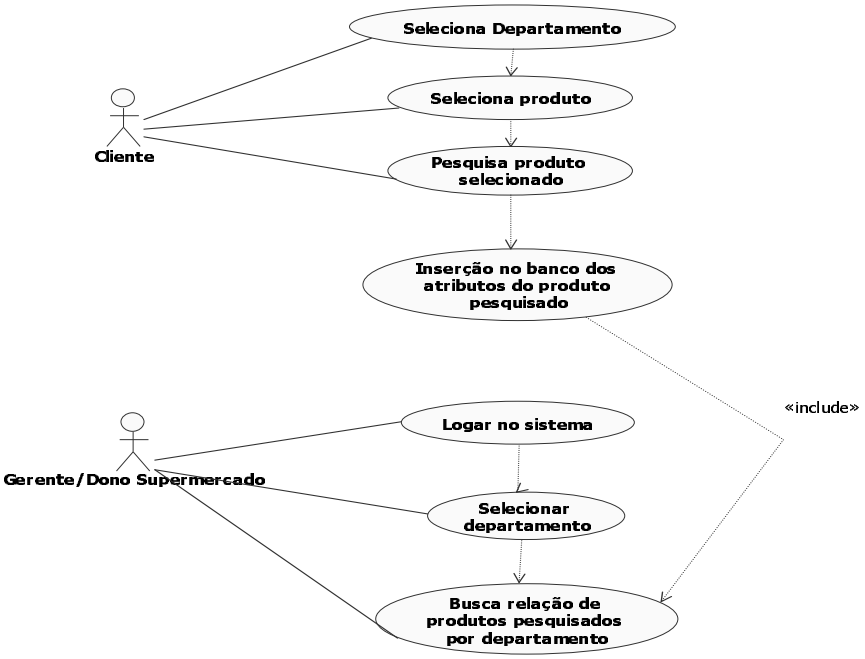

Figura 3.5: *Diagrama de Caso de Uso do sistema web*

3.3. Desenvolvimento da Aplicação *Desktop*

Foi desenvolvida uma aplicação *desktop* para extrair as informações do SGBD do cliente e criar um arquivo no formato XML contendo essas respectivas informações. A finalidade desta aplicação é a padronização de dados, ou seja, para que todas as informações dos SGBDs de diversos supermercados sejam convertidas para um arquivo no formato XML. Para manipular e processar os arquivos XML, foi utilizado a biblioteca JDom.

3.3.1. Introdução à Biblioteca JDOM

JDOM é uma biblioteca para processar XML que foi desenhada seguindo a estrutura do Java. Tanto DOM como SAX são independentes da linguagem por isso, algumas das formas de manejá-la podem parecer estranhas para um desenvolvedor Java.

No ano 2000 Janson Hunter iniciou este projeto, que pretendia criar uma API para processar o XML trabalhando ao estilo Java. Com o tempo a iniciativa levou ao *Java Consortium Process*, JCP (JSR 102), e hoje, a mais de 4 anos, surge a versão 1.0 do JDOM, distribuído sob licença Apache.

Neste trabalho, foi utilizada essa API para manipular os arquivos XML. É importante ressaltar que quando se deseja utilizar uma API, deve-se colocar o arquivo *.jar* correspondente no *classpath* do projeto, ou no seguinte diretório por exemplo: C:\Arquivos de programas\Java\jdk1.5.0_01\jre\lib\ext. Assim, quando importada uma classe desta API, não haverá problemas na compilação.

3.3.2. Análise do SGBD do Cliente

Para cada supermercado foi desenvolvida uma aplicação que extrai os dados do SGBD e gera um arquivo XML com esses respectivos dados.

Primeiramente foi feito fazer um levantamento dos nomes das tabelas, das colunas e do modelo do SGBD do cliente. Essa aplicação é executada no cliente, ou seja, na máquina do supermercado, que será acessada por meio da aplicação Web que foi desenvolvida. Usando como exemplo o SGBD Mysql com as tabelas, Açougue, Aves, Bebidas, Frios, Congelados, Higiene, Laticínios, Limpeza, Mercearia, Peixaria e os atributos, Código, Descrição e Valor, foi desenvolvida uma aplicação com as seguintes classes e métodos:

3.3.2.1. Classe DataBaseTauste

- **Conexão com o banco de dados**:

```
public void openDB(){

   try{
     Class.forName(driver do banco);
     Connection conn = DriverManager.getConnection(url, usuário, senha);
     Statement stm = conn.createStatement();
JOptionPane.showMessageDialog(null,"Conexão Estabelecida por Sucesso!");
   }catch(Exception e ){
 JOptionPane.showMessageDialog(null,"Impossível Estabelecer Conexão"+e);
              }

         }
```

Quadro 3.1 Método para conectar com o SGBD do cliente.

O método apresentado no Quadro 3.1 corresponde à efetuação da conexão com o SGBD do cliente. Este método acessa uma determinada base de dados, pode-se usar o mesmo

método para acessar bases de dados diferentes trocando apenas o driver do SGBD, url, usuário e senha.

- **Get e Set (métodos de acesso para um determinado atributo)**

```
      public void setDescricao(String desc)
            {
              if (desc != null){
                      if (desc.trim().length() == 0)
                          descricao = null;
                      else descricao = desc;
            }


            public String getDescricao()
            {
                  return descricao;  }
```

Quadro 3.2 Métodos get e set da classe DataBaseTauste.

O trecho de código ilustrado no Quadro 3.2 representa os métodos *get* e *set* O método "`setDescricao`" serve para atribuir um valor à variável "`descricao`" e o método "`getDescricao`" serve para ler esse valor atribuído.

- **Pesquisar os produtos no banco de dados:**

```
   public void pesquisaBebidas(int cd){
       String query =  "SELECT * FROM BEBIDAS WHERE CODIGO = "+"'"+cd+"'";
           try{
             res = stm.executeQuery(query);
                    if(res.next()) {
                       String cdb   = res.getString("Codigo");
                       String desc  = res.getString("Descricao");
                       String val    = res.getString("Valor");
                       setCodigo(cdb);
                       setDescricao(desc);
                       setValor(val);    }
           }catch(Exception e ){
                       setCodigo(null);
                       setDescricao(null);
                       setValor(null);
                       JOptionPane.showMessageDialog(null,"Erro"+e);}
    }
```

Quadro 3.3 Método pesquisaBebidas da classe DataBaseTauste

O método apresentado no Quadro 3.3 tem como finalidade buscar informações no SGBD do cliente. Neste método estão sendo buscados por meio dos métodos *get* e *set*, todos os atributos(Código, Descrição, Valor) do produto que apresentar o código passado como parâmetro (`cd`) da tabela “BEBIDAS”.

- **Método para retornar a quantidade de produtos de uma determinada tabela:**

```
public int MaxBebidas(int aux){
       String query = "SELECT MAX(CODIGO) FROM BEBIDAS";
              try{
                 res = stm.executeQuery(query);
                   if(res.next());
                   {
                    aux = res.getInt(1);
                   }
             }catch( Exception e){
                JOptionPane.showMessageDialog(null,"Erro!"+e);
             }
       return aux;
}
```

Quadro 3.4 Método maxBebidas da classe DataBaseTauste

O Quadro 3.4 representa o código do método “`maxBebidas`”, que tem como finalidade buscar a quantidade de produtos existentes em uma determinada tabela. Neste método está sendo recuperado o maior código da tabela “BEBIDAS” para identificar a quantidade de produtos existentes na mesma. Esse método foi usado para controlar um *loop* que irá inserir os nós no arquivo XML, ou seja, se a quantidade de produtos for cinco por exemplo, esse *loop* será executado cinco vezes, criando cinco nós com os nomes e atributos do produtos.

3.3.3. Análise das Classes em Java

Após a análise do SGBD do cliente, foram desenvolvidas algumas classes e métdos para que a aplicação esteja completa. A aplicação desenvolvida é executada na máquina do cliente, ou seja, na máquina de um determinado supermercado.

3.3.3.1. Classe Arquivos

A classe apresentada no Quadro 3.5 tem como finalidade criar um arquivo qualquer. Neste caso, é criado um arquivo XML.

A classe cria um arquivo passado por referência e preenche com seu respectivo conteúdo.

```
public class Arquivos {
   public void criarArquivo(String caminhoNomeArquivo, String conteudo){
      File arquivo = null;
      FileOutputStream arquivoPreenchido = null;
         try {
            //Apenas cria o arquivo passado como referência.
            arquivo = new File(caminhoNomeArquivo);
            //Preenche(escreve) o arquivo criado.
            arquivoPreenchido = new FileOutputStream(arquivo);
            arquivoPreenchido.write(conteudo.getBytes());
         } catch (FileNotFoundException exception) {
           System.out.println("O arquivo não pode ser criado: "
                 + exception.getMessage());

      }catch (IOException exception) {
            System.out.println("Erro ao preencher o arquivo: "
                           + exception.getMessage());
            }
      }
}
```

Quadro 3.5 Classe criaArquivo do Sistema Desktop

3.3.3.2. Classe EscritaXML

Para criar um arquivo XML deve-se seguir os seguintes passos: Inicializar os objetos, definir os atributos do nó pai, definir os atributos dos nós filhos, definir os atributos do produtos, definir o produto no supermercado. Foram criados os seguintes métodos para essa classe:

- **Método para inicializar os objetos**

```
private void inicializarObjetos() {

      this.produto = new Element("produto");
      this.codigo = new Element("codigo");
      this.descricao = new Element("descricao");
      this.valor = new Element("valor");
}
```

Quadro 3.6 Método inicializarObjetos da classe EscritaXML

O trecho de código do Quadro 3.6 representa o método para inicializar objetos. Esse método inicializa os objetos como produto, código descrição e valor. Esses objetos pertencem à classe "`Element`" da biblioteca JDom.

- **Construtor**

```
public EscritaXML() {

       this.tausteBebidas = new Element("supermercado");
       db.openDB();
}
```

Quadro 3.7 Construtor da classe EscritaXML

O Quadro 3.7 representa o construtor da classe “EscritaXML”. Esse construtor inicializa o objeto “tausteBebidas” da classe “Element” e invoca o método “openDB” da classe “DataBaseTauste” discutida na seção 3.3.2.1.

- **Método que irá criar o arquivo XML**

```
public void criarXMLBebidas() {
  int i = 1;
    do{
      inicializarObjetos();
      db.pesquisaBebidas(i);

      tausteBebidas.setAttribute("nome", "tauste");
      codigo.setText(db.getCodigo());
      descricao.setText(db.getDescricao());
      valor.setText(db.getValor());

      //"Setando" o atributos no produto
      produto.addContent(codigo);
      produto.addContent(descricao);
      produto.addContent(valor);

      tausteBebidas.addContent(produto);
      i++;
      }while(i <= db.MaxBebidas(0));

      //Criando o documento XML (montado)
      Document doc = new Document();
      doc.setRootElement(tausteBebidas);

      //Gerando o arquivo XML
      XMLOutputter xout = new XMLOutputter();
      Arquivos arq = new Arquivos();
      arq.criarArquivo("Bebidas.xml", xout.outputString(doc));
  }
```

Quadro 3.8 Método criarBebidas da classe EscritaXML

O código apresentado no Quadro 3.8 corresponde ao método “criarBebidas”. Neste método é inicializado os objetos, definidos os atributos do produto e criado o arquivo XML. Para a criação do arquivo XML foi passado como parâmetro para a classe “Arquivos” o nome do arquivo, que no caso é “Bebidas.xml” e o conteúdo do arquivo, que foi atribuído a um objeto do tipo “Document”(doc)da biblioteca JDom.

Quando o cliente executar essa aplicação, é criado um arquivo XML para cada departamento(Açougue, Aves, Bebidas...) contendo os produtos e seus atributos.

A Figura 3.6 e 3.7 ilustram, respectivamente, uma tela informando o *status* da conexão e a criação dos arquivos XML.

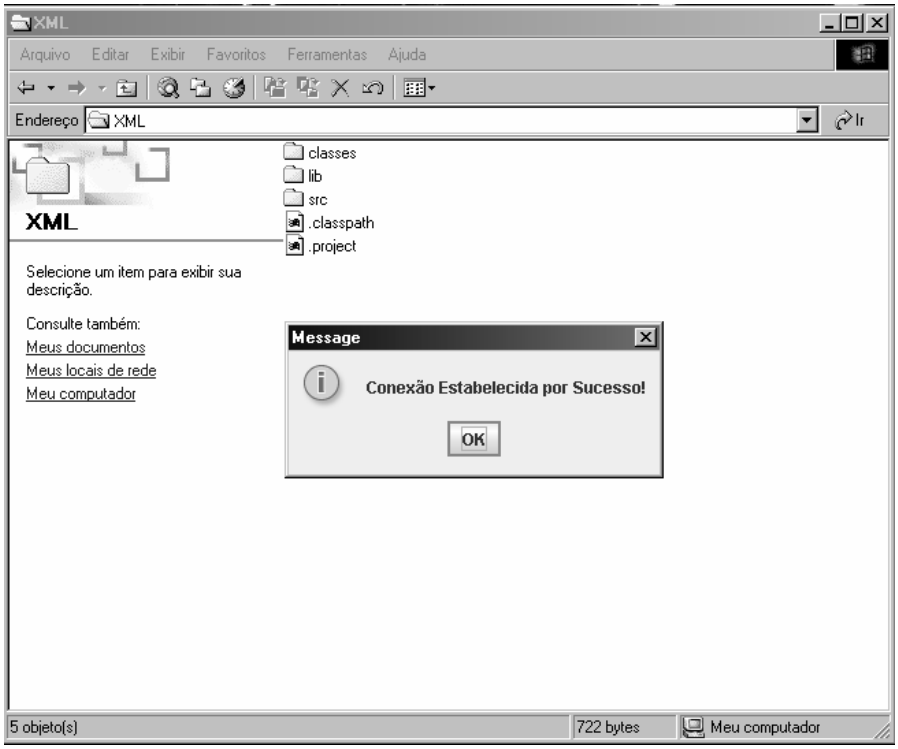

Figura 3.6: *Tela informando status da conexão*

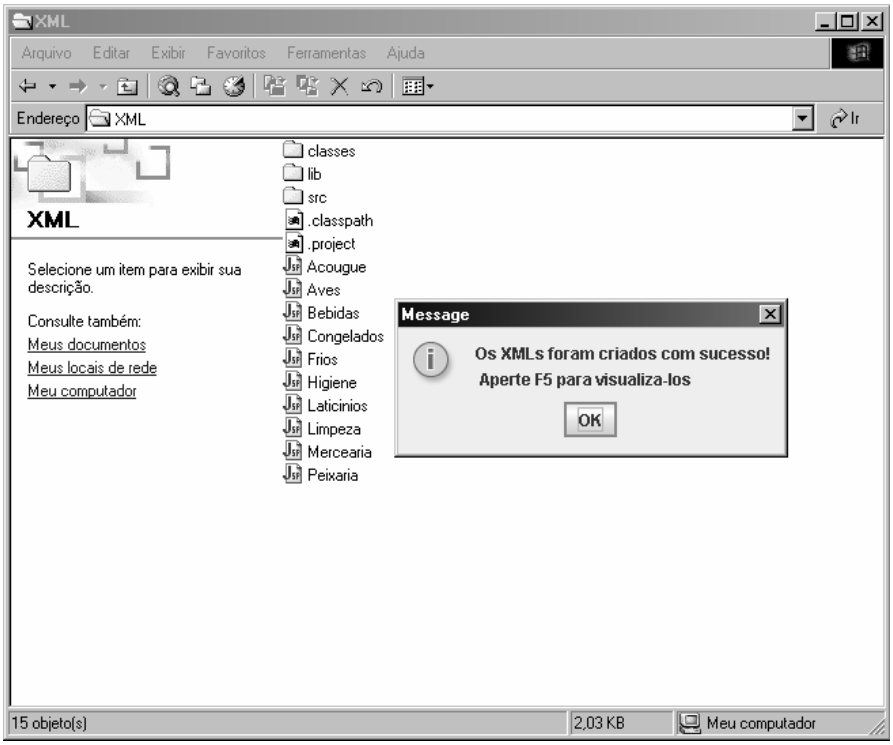

Figura 3.7 – Tela informando que os arquivos XML foram criados

3.4. Desenvolvimento do Sistema Web

Uma vez criada a aplicação para extrair os dados dos diferentes SGBD e gerar um arquivo XML, é necessário desenvolver uma aplicação à qual se possa acessar esse arquivo e fazer a leitura de seu conteúdo.

Para desenvolver esse website, foram utilizadas as linguagens HTML e JSP. A escolha da linguagem JSP se deu pela interação com a linguagem Java, pois se pode criar uma classe em Java que leia arquivos em formato XML, utilizando a biblioteca JDOM, e importá-la para o website utilizando a linguagem JSP.

3.4.1. Desenvolvimento do *layout do* Site

Para o desenvolvimento do *layout* do website foram utilizadas as ferramentas *DreamWeave*r e *Photoshop*. Foi visado desenvolver uma interface simples e auto-instrutiva, para que qualquer usuário, independentemente de seu conhecimento em informática, possa utilizar o website sem maiores problemas.

O website foi dividido em setores, assim organizando os produtos e facilitando para o usuário no momento em que desejar pesquisar um determinado produto. Em cada setor haverá um "`combo`" contendo a descrição dos produtos dos supermercados associados. O "`combo`" é preenchido com a descrição dos produtos extraídas diretamente dos arquivos XML existentes.

Haverá somente um botão no qual o usuário ira buscar o produto desejado. Efetuando-se a busca, é mostrado na tela o nome do supermercado, a descrição do produto e seu respectivo preço. Será também exibida na tela uma foto meramente ilustrativa do produto pesquisado. Haverá também um link denominado *login*, onde os donos dos supermercados

irão efetuar o *login*. Feito o *login*, abrirá uma página que conterá um "`combo`" com os departamentos. Selecionando um determinado departamento, uma lista de produtos é exibida na tela com a descrição e a quantidade de vezes que o produto foi pesquisado. Sendo assim, os donos dos supermercados poderão acompanhar, sempre que desejarem, quais produtos estão sendo mais pesquisados pelos consumidores.

3.4.2. Criação de JavaBeans

Para a manipulação dos arquivos XML e para uma posterior conexão com um SGBD do servidor, foram utilizados *Javabeans* os quais terão seus métodos invocados por meio da linguagem JSP. Os *Bean*s foram criados por departamento, ou seja, cada departamento terá seu *Bean*. Existem dez módulos ou departamentos. Para cada módulo ou departamento foram criados *Bean*s que são semelhantes entre si, diferindo apenas nos nomes. Foi desenvolvido um *Bean* que é usado por todos os departamentos que foi denominado "`Conexao`". Para o módulo "BEBIDAS" por exemplo, foram criados os *Bean*s "`comboBebidas`" e "`buscaBebidas`".

Apenas para os módulos "LOGIN" e "RELATORIO" foram desenvolvidos um *Bean* específico diferindo dos beans dos outros departamentos. Será abordado nesta monografia apenas o módulo "BEBIDAS".

3.4.2.1. Módulo Bebidas

No módulo "BEBIDAS", já abordado anteriormente, foram criados dois *Beans*, sendo o primeiro denominado "`comboBebidas`" e o segundo "`buscaBebidas`".O *Bean* "`comboBebidas`" terá como finalidade popular o "`combo`" da página com a

descrição dos produtos. A função de *Bean* “buscaBebeidas” é buscar os produtos selecionados pelo consumidor. Analisando os *Bean*s e seus métodos:

3.4.2.2. *Bean* comboBebidas

- **Método de leitura**

```
public void leBomPreco() {
 InputSource f = new
 InputSource("http://localhost:8080/BuscaPreco/XML/BomPreco/Bebidas.xml");

           SAXBuilder sb = new SAXBuilder();

           Document d = null;
                      try {
                      d = sb.build(f);
                      } catch (JDOMException e) {

                      } catch (IOException e) {
                }

                Element supermercado = d.getRootElement();

                List elements = supermercado.getChildren();
                Iterator i = elements.iterator();

                      while(i.hasNext()) {

                      Element element = (Element) i.next();

                descricaoBomPreco[a] = element.getChildText("descricao");
                      codigoBomPreco[a] = element.getChildText("codigo");
                      a++;
           }
                      }
```

Quadro 3.9 Método leBomPreco do bean comboBebidas

O método apresentado no Quadro 3.8 irá acessar o arquivo “Bebidas.xml”. No vetor “descricaoBomPreco”, é inserido todas as descrições dos produtos existentes e no vetor “codigoBomPreco”, é inserido todos os códigos do produtos existentes. O mesmo processo foi feito para os demais supermercados.

Primeiramente foi criado um documento nulo. Foi atribuído a esse documento toda a estrutura de um arquivo XML. Recupera-se o elemento *root*, que é o principal, ou seja, como o próprio nome já diz, a "raiz" do nó (`d.getRootElement()`). Logo foi recuperado os elementos filhos (`supermercado.getChildren()`). Por fim, serão iterados os elementos filhos e os elementos filhos dos filhos. Feito isso, foi criado um método para eliminar duplicidade de produtos, ou seja, para evitar que no "`combo`" tenha listado dois produtos com a mesma descrição.

O método mostrado no Quadro 3.9 é denominado "`adiciona`". Sua finalidade é varrer todos os vetores de descrição e de código dos produtos de todos os supermercados e juntar todas as informações em um só vetor.

- **Eliminando duplicidade.**

```
public  String[] adiciona(String[] array1, String[] array2)
    {
    int tem, count = 0;
          while(array2[count] != null) {
                count = count + 1;
          }
    for (int i = 0; i<array1.length; i++){
          tem = 0;
               if(array1[i] != null) {
                for(int j=0; j<array2.length; j++){
                      if (array1[i].equals(array2[j])==true)
                      tem = 1;
                      }
                      if(tem == 0) {
                      array2[count]=array1[i];
                      count += 1;
                      }
                }
          }
          return array2;
    }
```

Quadro 3.10 Método adiciona do bean comboBebidas

Pode-se passar como parâmetro neste método o vetor que contém a descrição ou o código dos produtos. Se passado por exemplo como parâmetro o vetor que contém a descrição dos produtos, o método retornará um vetor contendo as descrições dos produtos de todos os supermercados.

- **Método para retornar um vetor sem duplicidade de descrições:**

```
public String[] retornaProdutos(){

     descricaoBomPreco = adiciona(descricaoTauste,descricaoBomPreco);
     descricaoKawakami = adiciona (descricaoBomPreco,descricaoKawakami);

     return descricaoKawakami;
 }
```

Quadro 3.11 Método retornaProdutos do bean comboBebidas

No Quadro 3.11 foi mostrado o método "`retornaProdutos`". Neste método será buscado três vetores contendo os atributos dos produtos supermercados, ou seja, um vetor para cada supermercado, e comparado entre eles os produtos existentes. Feito isso é retornado um vetor sem a duplicidade de produtos. O mesmo foi feito no método representado no Quadro 3.12, entretanto, serão retornados os códigos dos produtos.

- **Método para retornar um vetor sem duplicidade de códigos:**

```
public String[] retornaCodigos(){

        codigoBomPreco = adiciona(codigoTauste,codigoBomPreco);
        codigoKawakami = adiciona (codigoBomPreco,codigoKawakami);

        return codigoKawakami;
}
```

Quadro 3.12 Método retornaCodigos do bean comboBebidas

Os métodos “retornaCodigos” e “retornaProdutos”, serão usados respectivamente para posteriormente fazer uma busca por meio do código e para popular o “combo” do departamento “BEBIDAS”.

3.4.2.3. *Bean* buscaBebidas

- **Método de leitura, retornando o valor do produto**

```
public String leBomPreco(String search){
  InputSource f = new
  InputSource("http://localhost:8080/BuscaPreco/XML/BomPreco/Bebidas.xml");

            SAXBuilder sb = new SAXBuilder();

            Document d = null;
            try {
                  d = sb.build(f);
            } catch (JDOMException e) {
                  System.out.println(e.getMessage());

            } catch (IOException e) {
                  System.out.println(e.getMessage());

            }
                  Element supermercado = d.getRootElement();

            List elements = supermercado.getChildren();
            Iterator i = elements.iterator();

            while(i.hasNext()) {

                  Element element = (Element) i.next();
                  if(element.getChildText("codigo").equals(search)){

               valorBomPreco = element.getChildText("valor");

                        }
             }
            return valorBomPreco;

}
```

Quadro 3.13 Método leBomPreco do bean buscaBebidas

O método apresentado no Quadro 3.13 foi denominado “`leBomPreco`”. Consiste em um método de leitura semelhante ao abordado no *Bean* “`comboBebidas`”. O que difere entre eles é que neste método é buscado apenas o produto que tiver seu código passado como parâmetro. Neste método será retornado o valor do produto. O método do Quadro 3.14 é semelhante ao método do Quadro 3.13 porém, este método retornará a descrição do produto.

- **Método de leitura, retornando a descrição do produto**

```
public String leDescricaoBomPreco(String search){
 InputSource f = new
 InputSource("http://localhost:8080/BuscaPreco/XML/BomPreco/Bebidas.xml");

                  SAXBuilder sb = new SAXBuilder();
                  Document d = null;
                        try {
                              d = sb.build(f);
                        } catch (JDOMException e) {
                              System.out.println(e.getMessage());
                        } catch (IOException e) {
                              System.out.println(e.getMessage());
                        }

                        Element supermercado = d.getRootElement();

                        List elements = supermercado.getChildren();
                        Iterator i = elements.iterator();

                        while(i.hasNext()) {

                              Element element = (Element) i.next();

      if(element.getChildText("codigo").equals(search)){

            descricaoBomPreco = element.getChildText("descricao");

                                    }
                               }
                  return descricaoBomPreco;
                  }
```

Quadro 3.14 Método leDescricaoBomPreco do bean buscaBebidas

É importante ressaltar que na linha onde é informado o nome do arquivo, o caminho passado é do servidor local (localhost:8080/BuscaPreco/XML/BomPreco/Bebidas .xml). Se

for substituido esse endereço por qualquer endereço remoto, a aplicação funcionará corretamente. Acessando diretamente o arquivo “Bebidas.xml” do servidor do supermercado Bom Preço por exemplo, pode-se modificar da seguinte maneira: (http://www.bompreco.com.br/BuscaPreco/XML/bebidas.xml).

Os métodos “`leDescricaoBomPreco`” e “`leBomPreco`” serão invocados posteriormente para mostrar na tela a descrição e o preço dos produtos selecionados pelo usuário. Para todos os outros departamentos foi utilizada a mesma metodologia.

3.4.2.4. *Bean* Conexao

O *Bean* “`Conexao`” foi desenvolvido para registrar no SGBD do servidor todos os produtos pesquisados pelos consumidores. Primeiramente foi criado uma tabela denominada “PRODUTOS” contendo os seguintes atributos: “`descricao`”, referente ao nome do produto e “`coddept`”, referente ao código do departamento. O código do departamento é um valor estático atribuído para diferenciar os departamentos. O código do departamento “AÇOUGUE” por exemplo, é 1 (um), o do “AVES” é 2 (dois), o de “BEBIDAS” é 3 (três) e assim por diante. O código deste *Bean* será discutido superficialmente por ser extremamente trivial. Existe o método “`inserirDados`” que uma vez invocado, irá inserir no SGBD do servidor os valores atribuídos nos métodos “`setDescricao`” e “`setCoddept`”. Os métodos “`setDescricao`” e “`setCoddept`” serão invocados diretamente na página por meio de instruções em JSP. O *Bean* “`Conexao`” é o mesmo para todos os departamentos.

- **Método para inserção dos dados.**

```
public void inserirDados(){

try {
String query = "insert into produtos(descricao,coddept)
          values(\""+descricao+"\","+coddept+")";
stm.executeUpdate(query);

}catch (SQLException e){
System.out.println("Erro na inserção:" + e.getMessage());}
}
```

Quadro 3.15 Método inserirDados do Bean Conexao

O método descrito no Quadro 3.15 executa uma *query* a qual irá inserir no banco a descrição e o código do produto.

3.4.2.5. *Bean* Relatorio

Esse *Bean* é utilizado para selecionar todos os produtos de um determinado departamento e contar todas as suas ocorrências. Esses produtos foram inseridos no SGBD do servidor por meio do *Bean* "`Conexao`".Analisando os métodos:

- **Método para selecionar os produtos de um determinado departamento.**

O trecho de código do Quadro 3.16 representa o método "`consultarDados`". Neste método é passado o código do departamento como parâmetro e por meio de uma *query*, será selecionado todos os produtos deste mesmo departamento. A variável "`aux`" está recebendo a descrição de todos os produtos existentes no departamento.

```
     public String consultarDados(String qnt){

        try {
          String query = "select * from produtos where coddept='"+ qnt+"'";
                  res = stm.executeQuery(query);

                   while (res.next() ){

                     aux += "\n"+(String)(res.getString("descricao"));
                    }

        }catch (SQLException e){System.out.println("Erro na inserção:" +
e.getMessage());}

                  return aux;
                  }
```

Quadro 316 Método consultarDados do Bean Relatorio

- **Método para retornar a quantidade de um determinado produto.**

```
 public String contaProdutos(String desc){

         String quant = null;

            try {
String query = "select count(coddept) from produtos where descricao=
'"+desc+"'";
                  res = stm.executeQuery(query);
                  if (res.next() ){

                  quant = res.getString(1);
                  }
            }catch (SQLException e){
            System.out.println("Erro na inserção:" + e.getMessage());
            }
            return quant;
}
```

Quadro 3.17 Método contaProdutos do Bean Relatorio

O código descrito no Quadro 3.17 refere-se ao método "ContaProdutos".Este método, por meio de uma *query*, retornará todas as ocorrências de um produto referente a um

determinado departamento. A descrição do produto a ser pesquisado é passado como parâmetro.

3.4.2.6. *Bean* Logar

Esse *Bean* foi desenvolvido para controlar o acesso na área restrita do website, ou seja, no módulo “RELATORIO”, onde somente os donos de supermercados poderão acessar.

Previamente foi cadastrado um *login* e uma senha para cada gerente/dono de supermercado. Esses dados serão guardados no SGBD do servidor. Analisando os método do *Bean* “`Logar`”.

- **Método consultar dados**

```
   public boolean consultarDados(String log){

      boolean testa = false;

         try {
           String query = "select senha from admin where login='"+ log+"'";
               res = stm.executeQuery(query);

                 if (res.next()){testa = true;}
                 else{testa = false;}

          }catch (SQLException e){System.out.println("Erro na inserção:" +
e.getMessage());
          }
           return testa;
   }
```

Quadro 3.18 Método consultarDados do Bean Logar

O método apresentado no Quadro 3.18 tem com finalidade selecionar a senha do respectivo *login*(usuário) passado como parâmetro. O resultado deste método é recuperado pelo método mostrado no Quadro 3.19.

- **Método getResultado**

```
public ResultSet getResultado() {
return res;
}
```

Quadro 3.19 Método getResultado do Bean Logar

Se for invocado o método "`getResultado`" ele retornará algo parecido com: "com.mysql.jdbc.ResultSet@8b819f". O ocorrido foi que o resultado atribuído à variável "`res`", do tipo `ResultSet`, seria a execução do *select* sem ser tratada corretamente. Para que o método "`getResultado`" retorne corretamente a senha do respectivo *login*, é necessário usar o método "`getString`" da interface "`ResultSet`". Este método foi aplicado diretamente na página para a verificação de *login* e senha.

3.5. Implementação da Página JSP

A implementação da página que será discutida nas próximas seções é referente ao módulo "BEBIDAS", pois como abordado anteriormente, nesta monografia será discutido apenas um módulo.

3.5.1. Sintaxe da JSP

A utilização da linguagem JSP em conjunto com HTML se dá por meio de *tags* especiais, pode-se também utilizar entre essas *tags* a linguagem Java pura. Exemplo:

```
<%
      Classe novaClasse = new Classe();
      out.println(“Centro Universitário Eurípides de Marília”);

            %>
```

Quadro 3.20 Exemplo de utilização da linguagem java entre tags JSP.

Para importar uma classe Java em JSP, utiliza-se a seguinte diretiva:

```
            <%@ page import=”nomeClasse”%>
```

Quadro 3.21 Exemplo de importação de classes I

Existem também outras maneiras de se importar uma classe em JSP, porém a maneira mostrada no Quadro 3.21 é mais transparente. Se a classe importada estiver dentro de um pacote, deve-se proceder conforme mostrado no Quadro 3.22.

```
            <%@ page import=”nomePacote.nomeClasse”%>
```

Quadro 3.22 Exemplo de importação de classes II

Em uma página, pode-se utilizar códigos em JSP dentro de um função Javascript, mas não se pode utilizar códigos em Javascript dentro de um método JSP.

3.5.2. Análise do Código em JSP

A maioria das funcionalidades do sistema web serão feitas pela importação das classes dos *Bean*s que foram criadas. Analisando a implementação da página e a utilização das classes dos *Beans* criados anteriormente.

3.5.2.1. Preencher o Combo

Primeiramente deve-se preencher o “combo” com a descrição dos produtos dos supermercados. Os produtos e seus atributos estão gravados em um arquivo XML, como já visto anteriormente. Para popular o “combo” foi chamada a classe “`comboBebidas`”, e invocados todos seus métodos.

```
<%
        String aux = null;
        comboBebidas combo = new comboBebidas();
        combo.leBomPreco();
        combo.leTauste();
        combo.leKawakami();
        combo.retornaProdutos();
        combo.retornaCodigos();
%>
```

Quadro 3.23 Invocando os métodos do Bean comboBebidas

O Quadro 3.23 representa um trecho de código que invoca os métodos do *Bean* “`comboBebidas`”. Os métodos “`retornaProdutos`” e “`retornaCodigos`” invocados do *Bean* “`comboBebidas`” retornam respectivamente um vetor de *string* contendo a descrição e o código dos produtos. Esses vetores foram denominados respectivamente como, “`descricaoKawakami`” e “`codigoKawakami`”. O Quadro 3.24 representa um trecho de código que irá preencher o “combo” e seu “`<option value>`” (valores que serão passados por parâmetro ao selecionar um determinado produto) com a descrição dos produtos.

```
<select name="selectBebidas" id="selectBebidas" >
    <%
        for(int i = 0; i < combo.descricaoKawakami.length; i++)
          if(combo.descricaoKawakami[i] != null)
    {%>
       <option value="<%=combo.descricaoKawakami[i]%>">
           <%out.print(combo.descricaoKawakami[i]);%>
        </option>
    <%}%>
</select>
```

Quadro 3.24 Populando o combo do módulo bebidas

Foi inserido entre as tags HTML instruções JSP, que consistem em uma estrutura de repetição (`for`), variando de 0 (zero) até o número de produtos existentes no vetor "`descricaoKawakami`", que varrerá e irá resgatar todos os valores contidos neste vetor.

Note que na *tag* "`<option value>`" foi passado também a descrição dos produtos contidas no vetor.

3.5.2.2. Efetuação de Buscas

As buscas foram manipuladas por uma função em JavaScript. Quando um produto for selecionado no "combo", é passado como parâmetro a descrição do mesmo para a função JavaScript. O Quadro 3.25 representa o trecho de código referente as buscas do site.

A descrição dos produtos serão passadas no "`switch`"(`switch/case`) e haverá somente um "`case`" que funcionará da seguinte maneira: Foi inserido entre o código JavaScript, mais precisamente dentro do "`switch`", instruções em JSP, que consiste em uma estrutura de repetição (`for`) que varrerá os vetores de descrição e de código. O vetor de descrição será o valor de um único "`case`" existente, pois qualquer produto que seja selecionado pelo usuário, entrará neste "`case`". Dentro deste "`case`", haverá uma variável

auxiliar “aux”, que receberá o código referente ao produto selecionado. Esse código é passado como parâmetro nos métodos “leDescricaoBomPreco” e “leBomPreco” para retornar respectivamente a descrição e o valor do produto pesquisado. Esse procedimento é feito para cada supermercado, entretanto, está sendo citado apenas para o supermercado Bom Preço.

```
switch (valor){
  <% for(int l = 0; l < combo.descricaoKawakami.length; l++){
            buscaBebidas busca = new buscaBebidas();
    %>
            case '<%out.print(combo.descricaoKawakami[l]);
                  aux = combo.codigoKawakami[l];
   %>':
           <%if(busca.leBomPreco(aux) != null)
               {%>
                        precoBomPreco+= '<%=busca.leBomPreco(aux)%>';
                        nomeBomPreco+=  'SUPERMERCADOS BOM PREÇO';
        <%}else {%>
                        precoBomPreco+= 'R$0,00';
                        nomeBomPreco+=  'SUPERMERCADOS BOM PREÇO';
               <%}%>
       <%if(busca.leDescricaoBomPreco(aux) != null) {%>
            descricaoBomPreco+= '<%=busca.leDescricaoBomPreco(aux)%>';
       <%}else {%>
            descricaoBomPreco+= 'Não temos esse produto em nosso estoque!';
        <%}%>
```

Quadro 3.25 Função JavaScript em conjunto com instruções em JSP

3.5.2.3. Inserção de Produtos Pesquisados no SGBD do Servidor

Para gerar o relatório dos produtos pesquisados, foi desenvolvido o *Bean* “Conexao”, abordado anteriormente. Para inserir os produtos pesquisados no SGBD do cliente, como mostrado no Quadro 3.26, foi implementado da seguinte forma: Primeiro foi criado duas *strings*, uma recebendo a descrição do produto selecionado no “combo” e uma outra recebendo o valor da ação do botão. Se a *string* ação for igual a parâmetro “Buscar”, a

classe “Conexao” é instanciada e é invocado os métodos “SetDescricao”, “setCoddept” e “inserirDados”.

O parâmetro passado no método “setDescricao”, é a descrição do produto selecionado no “combo”. Já o parâmetro passado no método “setCoddept”, é o código do departamento, que é 3 (três) neste caso. O método “inserirDados”, irá inserir esses valores no banco.

```
<%

String valorCombo = request.getParameter("selectBebidas");
      String acao = request.getParameter("bebidasBuscar");

              if(acao != null){

      if(acao.equals("Buscar")){
            Conexao con = new Conexao();
            con.setDescricao(valorCombo);
            con.setCoddept(3);
            con.inserirDados();
                              }
}
      %>
```

Quadro 3.26 Inserção de dados no banco do servidor

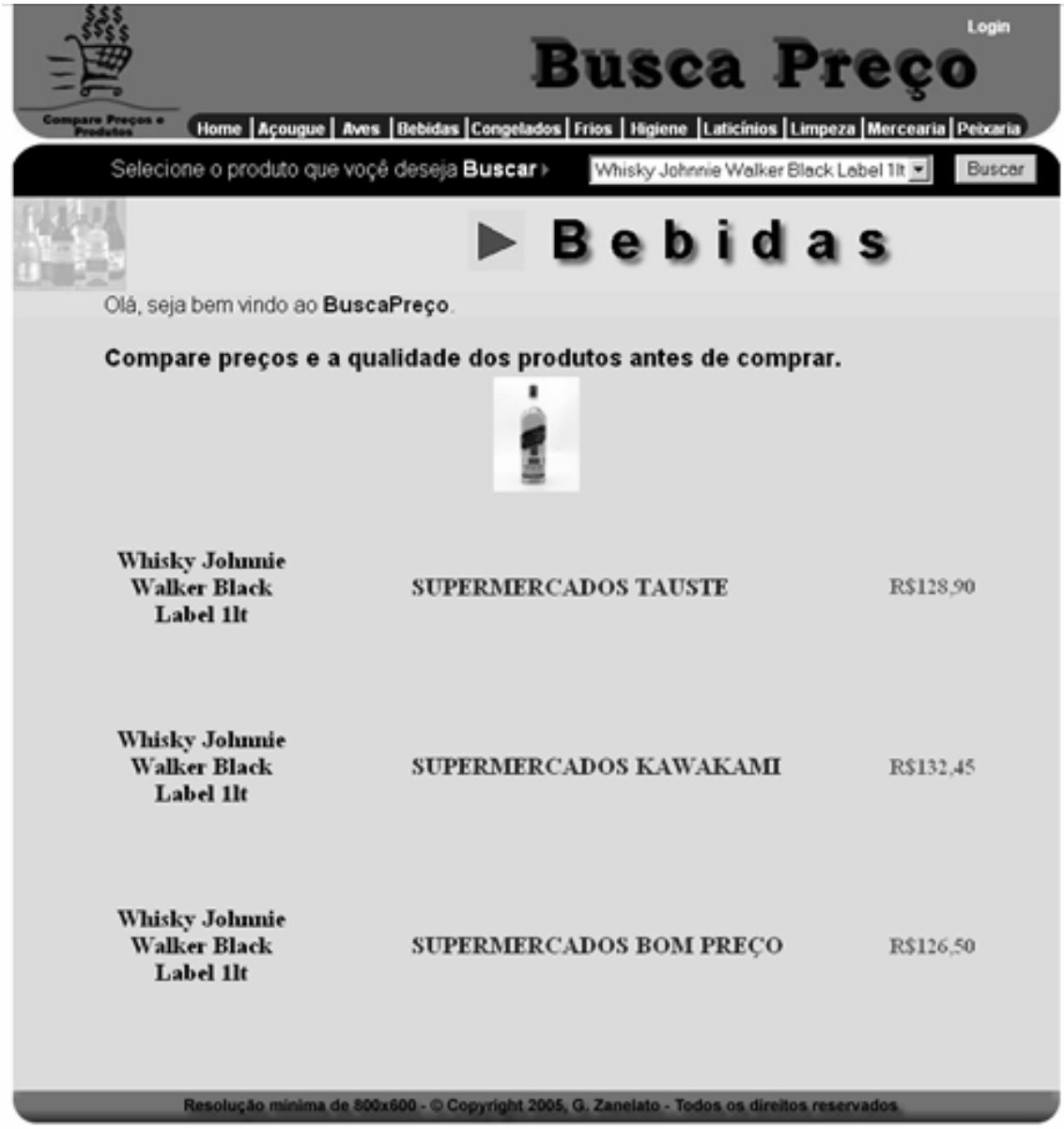

Figura 3.8: *Tela do Departamento Bebidas*

3.5.2.4. Módulo Login

Para efetuar o *login*, foi desenvolvida uma página simples com os campos, “LOGIN” e “SENHA” e os botões, “LOGAR” e “LIMPAR”. Para tratar a funcionalidade desta página foram invocados os métodos do *Bean* “`logar`”, abordado anteriormente.

Como mostrado no Quadro 3.27, primeiramente recupera-se o valor do *login* e o valor da senha postada pelo usuário, esses valores são atribuidos à uma *string*. O método “`consultarDados`” é invocado, e seu parâmetro será o *login* digitado pelo usuário. Se o *login* existir no banco e a senha digitada pelo usuário for igual a senha cadastrada no banco referente ao respectivo *login*, o usuário é direcionado para a página de relatório, caso contrário, aparecerá uma mensagem de erro na tela, conforme mostrado na Figura 3.10.

```
            <%
            String login = request.getParameter("loginfield");
            String senha = request.getParameter("senhafield");
            String acao = request.getParameter("btLogar");
            String loginInvalido = null;
            String senhaInvalida = null;

                  if(acao != null){

                  if(acao.equals("Logar"))
                        {
                        logar log = new logar();

            if(log.consultarDados(login) == true ){
               if(log.getResultado().getString(1).equals(senha)){
response.sendRedirect("http://localhost:8080/BuscaPreco/relatorio.JSP");

            }else{ senhaInvalida = "Senha Inválida";}

            }else { loginInvalido = "Usuário Inválido";}

                                    }
                              }
              %>
```

Quadro 3.27 Efetuando o login

A Figura 3.9 ilustra a tela de login onde o dono do supermercado entrará com o login e senha.

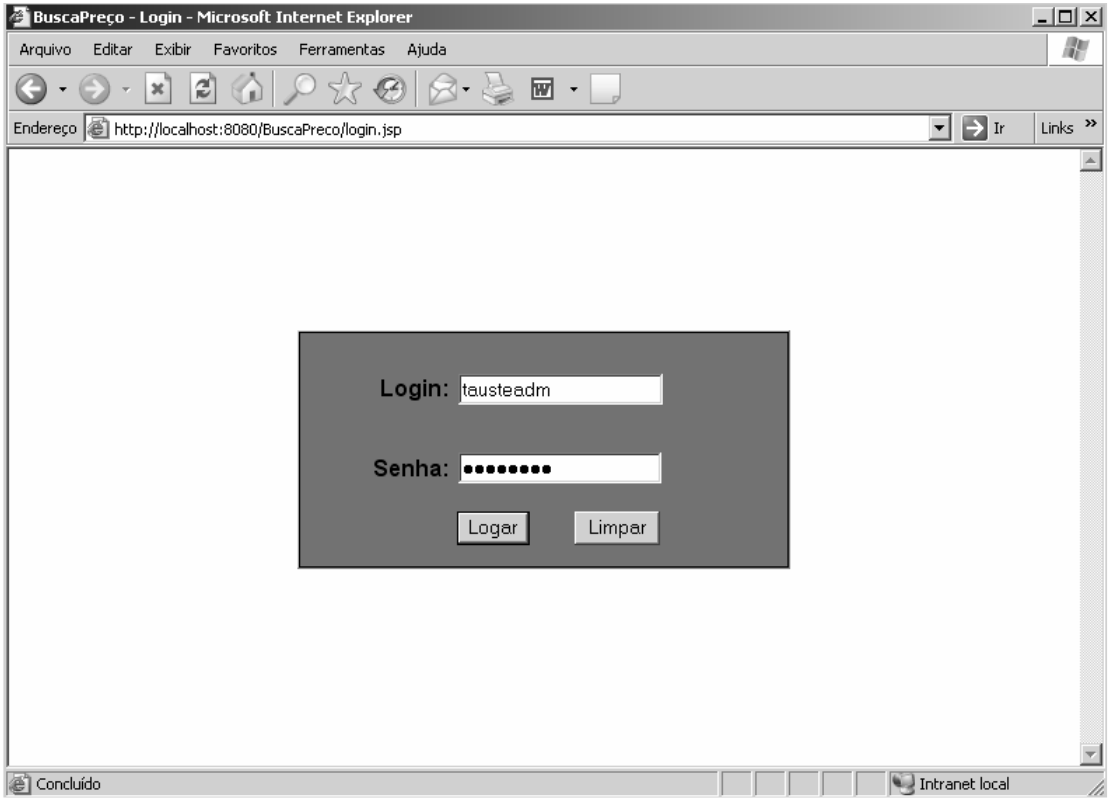

Figura 3.9: Tela de login

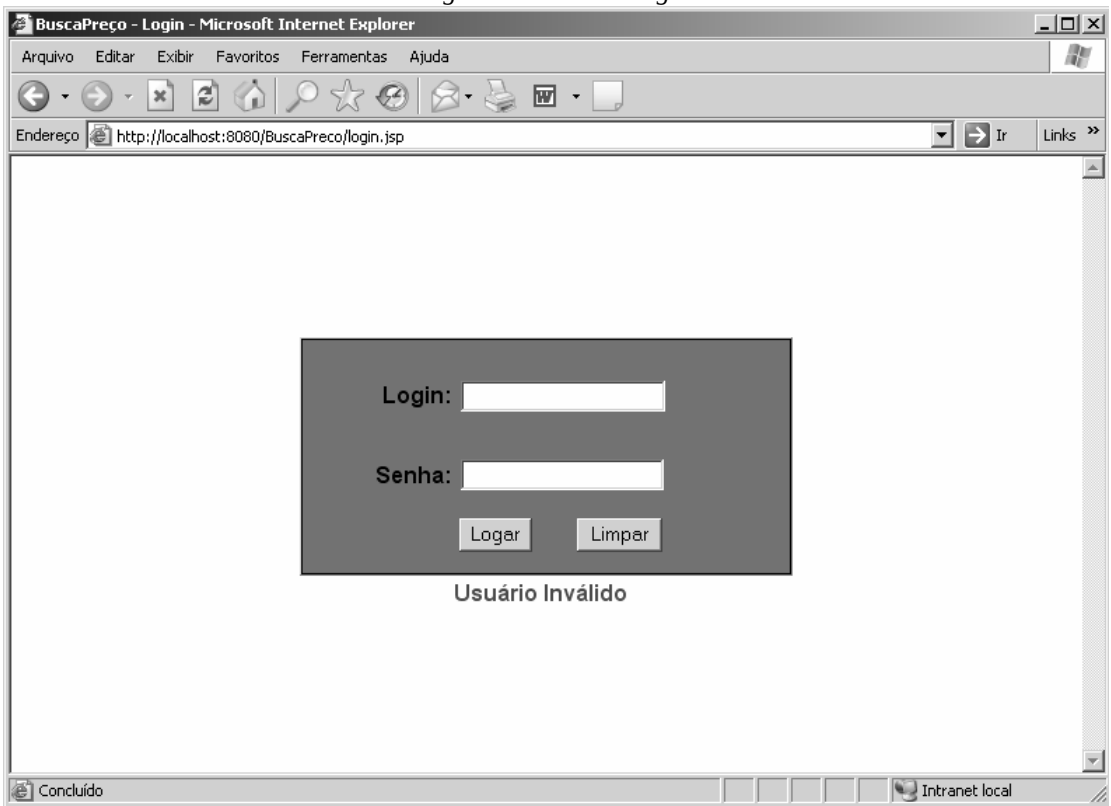

Figura 3.10: *Tela de login alertando o usuário*

3.5.2.5. Módulo Relatório

O módulo relatório conterá as informações dos produtos pesquisados. Para a implementação deste módulo, foi invocado o método "`consultarDados`" do *Bean* "`Relatorio`", abordado anteriormente. A tela exibirá todos os produtos do departamento selecionado e a quantidade de vezes que o mesmo foi pesquisado. Os produtos estão sendo buscados diretamente do arquivo XML. Foi verificado que seria interessante mostrar todos os produtos do departamento, mesmo os que não tenham sido pesquisados nenhuma vez.

O trecho de código do Quadro 3.28 tem como finalidade buscar os produtos referentes ao departamento selecionado pelo usuário.

Primeiramente foi invocado o método "`consultarDados`", o parâmetro é o código do departamento selecionado pelo usuário. Esse método retornará os produtos do departamento selecionado.

```
<%
String coddept = request.getParameter("selectDepartamento");
String acao = request.getParameter("departamentoBuscar");

      relatorio relat = new relatorio();
      if(acao != null)
            {
            if(acao.equals("Buscar"))
                  {
                        try{

                  relat.consultarDados(coddept);

                   }catch(Exception e){}
                  }
            }
 %>
```

Quadro 3.28 Inserindo no banco produtos pesquisdos

O código do departamento foi declarado estaticamente no "`<option value>`" do "combo", conforme mostrado no trecho de código do Quadro 3.29.

```
            <select name="selectDepartamento">
                    <option value="1">A&ccedil;ougue</option>
                    <option value="2">Aves</option>
                    <option value="3">Bebidas</option>
                    <option value="4">Congelados</option>
                    <option value="5">Frios</option>
                    <option value="6">Higiene</option>
                    <option value="7">Latic&iacute;nios</option>
                    <option value="8">Limpeza</option>
                    <option value="9">Mercearia</option>
                    <option value="10">Peixaria</option>
            </select>
```

Quadro 3.29 Código do combo do módulo relatório.

Para exibir a descrição dos produtos na tela, foi feito o mesmo procedimento usado para popular o "combo" na seção 3.5.2.1. Foi invocado os métodos do *Bean* "`comboBebidas`", e posteriormente varreu-se o vetor "`descricaoKawakami`" para resgatar a descrição dos produtos de todos os supermercados.

Para exibir na tela a quantidade de vezes que o produto foi pesquisado, foi criado uma estrutura de repetição `(for)` que varrerá o vetor "`descricaoKawakami`". Os valores deste vetor serão passados como parâmetro para o método "`contaProdutos`" do *Bean* "`Relatorio`" para retornar a quatidade de produtos, conforme mostrado no trecho de código do Quadro 3.30.

```
     <%
       for(int i = 0; i < combo2.descricaoKawakami.length; i++)
            {
    if(combo2.descricaoKawakami[i] != null)
                {
            out.print(relat.contaProdutos(combo2.descricaoKawakami[i]));
     }else break;
     %> <br> <%}%>';
```

Quadro 3.30 Código que exibe os produtos na tela do módulo relatório.

A Figura 3.11 mostra a tela de relatórios do departamento “BEBIDAS”. O gerente/dono de supermercado irá selecionar o departamento desejado. É exibido na tela uma tabela contendo a descrição e a quantidade de vezes que um produto foi pesquisado pelos consumidores sendo assim, os donos de supermercados poderão fazer um comparativo de quais produtos estão sendo mais pesquisados pelos consumidores e, por exemplo, fazer promoções destes produtos.

Figura 3.11: *Tela do módulo Relatório*

3.6. Considerações Finais

Neste Capítulo foram analisados as classes e os *Bean*s criados e seus principais métodos para o funcionamento do sistema *desktop* e do sistema web. O sistema *desktop* é executado na máquina do cliente(supermercado) e suas principais funcionalidades são a extração de dados do SGBD do cliente e a criação do arquivo XML. O sistema Web é executado por meio de um servidor, no caso foi utilizado o servidor TomCat, que é um container Web que abrange a tecnologia JSP.

As principais funcionalidades do sistema Web são a leitura dos arquivos XML, inserção de produtos pesquisado e a leitura dos dados do SGBD do servidor. A leitura dos arquivos XML tem como finalidade popular o “combo” com a descrição dos produtos e retornar a descrição e o preço do produto selecionado. A inserção dos produtos pesquisados no SGBD do servidor tem como finalidade, registrar todas as ocorrências de pesquisas de um determinado produto. A leitura dos dados do SGBD do servidor tem como finalidade, retornar um *feedback* aos donos dos supermercados, trazendo a descrição e a quantidade de vezes que um produto foi pesquisado pelos consumidores.

CONCLUSÕES

O trabalho desenvolvido serviu para propor uma idéia simples, porém ainda não implementada, de economizar dinheiro, tempo e trazer para o usuário toda comodidade de poder fazer pesquisas de sua própria casa. O maior problema deste projeto é convencer os supermercados a se associarem, pois foi pesquisado que nem todos os donos de supermercados gostariam de ter o preço de seus produtos expostos para a concorrência porém, acredita-se que com o tempo esse problema deixe de ser um empecilho e todos associem-se a esse sistema**.** O desenvolvimento deste trabalho serviu para ampliar meus conhecimentos em relação às linguagens Java, JSP e também a configurar corretamente as ferramentas utilizadas para o desenvolvimento do mesmo. Com o conhecimento adquirido no transcorrer do trabalho acredito estar melhor preparado para o mercado de trabalho.

O conteúdo abordado durante os quatro anos do curso foi de suma importância para o cumprimento do cronograma e o desenvolvimento do trabalho, como por exemplo, as disciplinas de Banco de Dados, Laboratório de Programação, Lógica de Programação, entre outras. Evidentemente, para uma abordagem mais complexa do trabalho apresentado, tomaria mais tempo e estudo, a idéia é que isso seja apenas “uma porta que se abrirá mostrando novos horizontes”.

REFERÊNCIAS BIBLIOGRÁFICAS

ANSELMO, F. – *"Tudo o que Você Queria Saber Sobre JSP Qunado Utiliza o Servidor TomCat com o Banco Mysql"*, VisualBooks, 2002.

BOMFIM Jr, T.F. – *"JSP – Java Server Pages: Atecnologia Java na Internet",* Érica , 2002.

DEITEL, H. M.; DEITEL, P.J. – *"Java - Como Programar",* Bookman, 4 Edição 2002.

YOSHIDA, S.P. – *"Universidade Java"* Digerati , 2004.

Informações sobre a linguagem HTML. Disponível em:
http://www.htmlstaff.org Acesso em Agosto. 2005.

Informações sobre a linguagem XML. Disponível em:
http://www.xml.com. Acesso em Setembro. 2005.

Informações sobre a linguagem XML. Disponível em:
http://www.xml.org. Acesso em Setembro. 2005.

Informações sobre a linguagem JAVA. Disponível em:
http://www.sun.com. Acesso em Outubro. 2005.

Informações sobre a linguagem JAVA. Disponível em:
http://www.javafree.com. Acesso em Outubro. 2005.

BOGO, Kellen Cristina. A Historia da Internet – "Onde tudo Começou.." Disponível em:
http://www.kplus.com.br. Acesso em ago. 2005

APÊNDICE A

Lista de Problemas e Alterações

Requisito 1.1

- **Check 1**

Foi verificado que para facilitar as buscas, o sistema disponibilizará os produtos divididos por setores, como por exemplo, frios, bebidas, limpeza, etc.

- **Check 2**

Foi verificado que é interessante que o sistema retorne um *feedback* dos produtos procurados separados por departamento, para os donos de supermercados.

- **Check 3**

Foi verificado que seria interessante inserir os nomes dos produtos dentro de um "combobox", assim evitando erros de digitação em buscas.

Requisito 1.2

- **Check 1**

Não poderão serem listados os produtos que estiverem em falta no estoque, o sistema exibirá uma mensagem alertando que não há o produto no estoque.

- **Check 2**

Incluir um botão de busca.

Engenharia de Sistema

✓ Objetivo do Sistema

O aplicativo a ser desenvolvido tem por definição oferecer a comodidade aos clientes para fazerem buscas de produtos e compararem os supermercados que oferecem os preços mais baixos. O cliente sai de sua casa direto ao supermercado em que foram encontrados os produtos mais baratos sendo assim, economizando tempo e dinheiro. O projeto desenvolvido é constituído por dois sistemas: sistema *desktop* e sistema Web, que serão descritos a seguir.

SISTEMA DESKTOP

Esse aplicativo é utilizado para buscar os produtos de uma base de dados e converte-los em um arquivo no formato XML. Seu principal módulo é o de escrita de XML.

- **Módulo de Escrita de XML**

Esse módulo irá criar para cada setor do supermercado um arquivo XML contendo os produtos correspondentes.

SISTEMA WEB

O sistema é baseado em WEB, e seus principais módulos, serão o de busca, relatório, login.

- **Módulo de Buscas**

  - Consulta dos preços dos produtos de diversos supermercados;
  - Feedback para o usuário;

- **Módulo de Relatório**

  - Feedback para os donos de supermercados, trazendo os produtos mais procurados, divididos por setores, pelos usuários;

- **Módulo de login**

  - Tela onde os donos/gerentes de supermercados irão logar para entrar no módulo relatório.

Análise de Requisito

SISTEMA DESKTOP

- **Módulo de Produtos**

  - ✓ **Cadastro de Produtos**

    - Dados dos Produtos

Código	INTEGER	NOT NULL
DescriçãoProduto	STRING	NOT NULL
Valor	STRING	NOT NULL

O sistema possuirá apenas o módulo de produtos, pois não serão cadastrados usuários e nem dados dos supermercados. Serão cadastrados apenas os atributos dos produtos que serão pesquisados pelos usuários. Esses dados estarão armazenados em um arquivo em formato XML. Haverá uma aplicação *desktop* na qual o dono do supermercado irá executar sempre que achar necessário, para a atualização desse arquivo XML que conterá os dados dos produtos que estão inseridos no banco de dados do supermercado.

Havendo vários bancos de dados distintos, para cada supermercado, será desenvolvida uma aplicação específica.

SISTEMA WEB

- **Módulo de Buscas**

  - ✓ **Busca de Produtos**

No módulo busca de produtos, os produtos serão buscados diretamente dos arquivos XML criados previamente. Esse módulo também irá inserir no banco a descrição e o código

do departamento do produto pesquisado pelo usuário para uma posterior consulta dos donos de supermercados.

- Dados dos Produtos a ser Inserido

Coddept	INTEGER	NOT NULL
Descricao	STRING	NOT NULL

- **Módulo de login**

No módulo “LOGIN”, os gerentes/donos de supermercados irão efetuar o *login* no sistema para efetuar uma pesquisa de produtos procurados por setor. O *login* e a senha do usuário são previamente cadastrados pelo administrador do sistema.

- Dados do *login*

login	STRING	NOT NULL
senha	STRING	NOT NULL

- **Módulo de relatorio**

Será neste módulo que os donos/gerentes de supermercados terão um *feedback* dos produtos pesquisados pelos usuários.

APÊNDICE B

INSTALAÇÃO E CONFIGURAÇÃO DE FERRAMENTAS

Apache TomCat

O Tomcat é um servidor de aplicações Java para web. É *open source* (código aberto) desenvolvido dentro do conceituado projeto *Apache Jakarta* e oficialmente endossado pela *Sun* como a Implementação de Referência (RI) para as tecnologias *Java Servlet* e *JavaServer Pages* (JSP). O Tomcat é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção.

Tecnicamente, o Tomcat é um *Container* Web, parte da plataforma J2EE que abrange as tecnologias *Servlet* e JSP, incluindo tecnologias de apoio relacionadas como *Realms* e segurança, JNDI *Resources* e JDBC *DataSources*. O Tomcat tem a capacidade de atuar também como servidor web/HTTP, ou pode funcionar integrado a um servidor web dedicado como o Apache httpd ou o Microsoft IIS.

Para instalar este servidor, primeiro deve-se fazer o download no link http://jakarta.apache.org/tomcat/. Após o download deverá ser executado o programa instalador, o assistente de instalação será iniciado. Siga os passos normalmente, deve-se apenas ter atenção na fase em que é solicitado o caminho da JVM (*Java Virtual Machine*) como mostrado na Figura 1 O caminho a ser preenchido deverá ser o diretório do j2sdk e não do j2re.

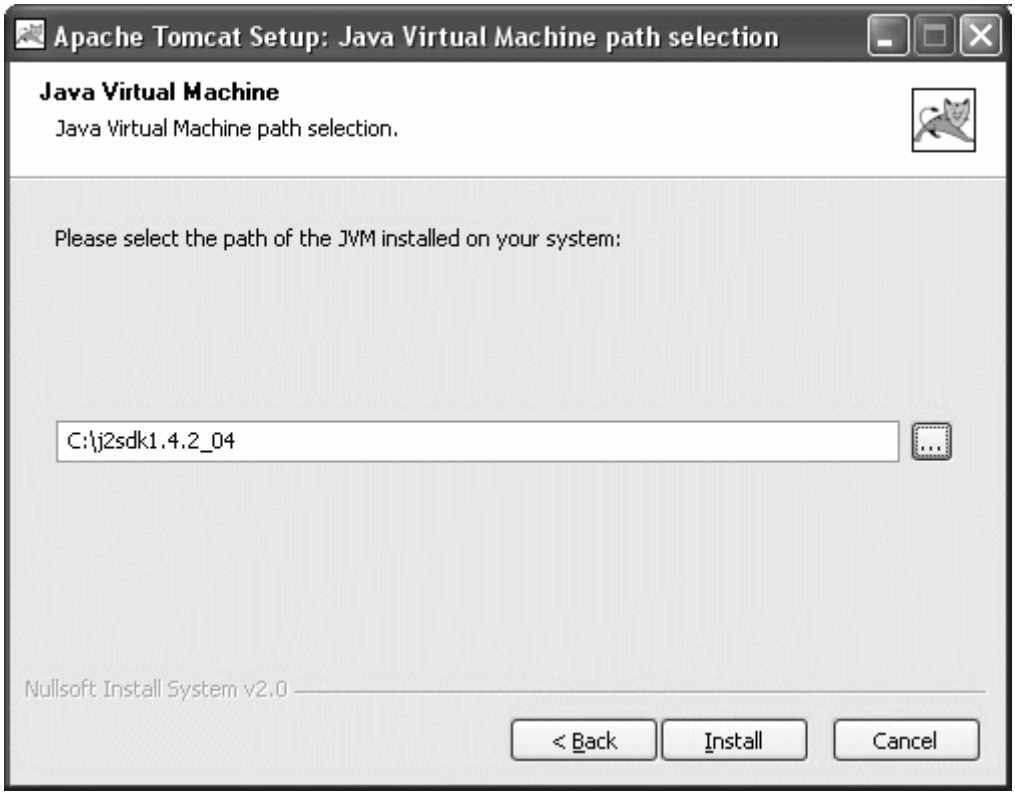

Figura 1 – Tela onde será definido o caminho da JVM

Após o término da instalação, deve-se configurar as variáveis de ambiente para que o Tomcat funcione perfeitamente com as aplicações JSP.

- **JAVA_HOME:** Local de instalação do kit de desenvolvimento Java J2SE (JDK).
- **CATALINA_HOME:** Local de instalação do Tomcat.
- **CLASSPATH:** Caminhos (pacotes e diretórios) de localização de classes Java, o *classpath* deve incluir o(s) *jar*(s) dos pacotes *Servlet* e JSP do Tomcat.
- **PATH:** Caminhos (diretórios) de localização de executáveis no sistema operacional, deve incluir o diretório *bin* das ferramentas do Java SDK.

Para configurar as variáveis de ambiente deve-se abrir, Painel de Controle, Sistema, Avançado, Variáveis de Ambiente. Configure as variáveis de acordo os caminhos de sua máquina, exemplo:

- JAVA_HOME = `C:\Arquivos de programas\Java\jdk1.5.0_04`

- CATALINA_HOME = *C:\Arquiv~1\Apache~1\Tomcat 5.0*
- CLASSPATH = %CATALINA_HOME%\common\lib\servlet-api.jar;.;%CLASSPATH%
- CLASSPATH = %CATALINA_HOME%\common\lib\JSP-api.jar;%CLASSPATH%
- PATH = %JAVA_HOME%\bin;%PATH%

Eclipse

Eclipse é uma IDE aberta para a construção de softwares. O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. O gasto inicial da IBM no produto foi de mais de 40 milhões de dólares.

Hoje o Eclipse é a IDE Java mais utilizada no mundo. Possui como características marcantes o uso da *SWT* e não do *Swing* como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em *plug-ins* e o amplo suporte ao desenvolvedor com centenas de *plug-ins* que procuram atender as diferentes necessidades de diferentes programadores.

Para instalar o eclipse deve ser feito o download no link http://www.eclipse.org/. Após o download do mesmo, basta descompactá-lo em qualquer diretório e executar o arquivo eclipse.exe.

Configuração de Bibliotecas

Neste trabalho foram utilizadas duas bibliotecas além das padrões do Java. As bibliotecas utilizadas foram a de manipulação de arquivos Xml (JDOM) e a de *drivers* jdbc para o mysql. Para configurar corretamente as bibliotecas deve-se inserir no caminho: C:\Arquivos de programas\Java\jdk1.5.0_01\jre\lib\ext, os arquivos *.jar* correspondentes às bibliotecas. JAR significa *Java Archive*, e é simplesmente um arquivo *.zip* normal, com uma extensão diferente apenas. Estes arquivos são usados para facilitar a distribuição de pacotes e

manter a simplicidade na hora de configurar os ambientes de desnvolvimento, uma vez que é possível colocar todos seus pacotes (conjunto de classes) dentro de um único arquivo JAR.