UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WALTER ARAÚJO MENDES JÚNIOR

A UTILIZAÇÃO DE FERRAMENTAS DE PROGRAMAÇÃO NO DESENVOLVIMENTO DE UM SISTEMA DE COMÉRCIO VIRTUAL ELETRÔNICO

São Luís
2012

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WALTER ARAÚJO MENDES JÚNIOR

A UTILIZAÇÃO DE FERRAMENTAS DE PROGRAMAÇÃO NO DESENVOLVIMENTO DE UM SISTEMA DE COMÉRCIO VIRTUAL ELETRÔNICO

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Prof.ª Maria Auxiliadora Freire.

São Luís
2012

Mendes Júnior, Walter Araújo

A utilização de ferramentas de programação no desenvolvimento de um sistema de comércio virtual eletrônico/ Walter Araújo Mendes Júnior . - 2012.

69 f.

Impresso por computador (fotocópia).

Orientadora: Maria Auxiliadora Freire.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2012.

1. Loja Virtual – Ferramenta de Segurança. 2. *Framework primefaces* 3.0. 3. *JSF* 2.0. 4. *Spring*. I. Titulo.

CDU 004.056:339.3/.5

WALTER ARAÚJO MENDES JÚNIOR

A UTILIZAÇÃO DE FERRAMENTAS DE PROGRAMAÇÃO NO DESENVOLVIMENTO DE UM SISTEMA DE COMÉRCIO VIRTUAL ELETRÔNICO

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em: 17 /07 /2012

BANCA EXAMINADORA

Profª Ms. Maria Auxiliadora Freire (Orientadora)
Mestre em Ciência de Engenharia - (PUC-RIO)
Universidade Federal do Maranhão

Profª Ms. Simara Vieira da Rocha
Mestre em Engenharia de Eletricidade (AC. Ciência da Computação) - (UFMA)
Universidade Federal do Maranhão

Prof Ms. Carlos Eduardo Portela Serra de Castro
Mestre em Informática - (PUC-RIO)
Universidade Federal do Maranhão

À minha mãe Joana, ao meu pai Walter.

AGRADECIMENTOS

À minha orientadora Profª Ms. Maria Auxiliadora Freire, pela orientação durante todas as etapas do processo de produção de monografia, incluindo projeto e implementação.

À minha família pelo apoio constante durante toda a graduação.

Aos professores do curso de Ciência da Computação que durante essa longa estrada me deram uma carga de conhecimento.

À todos os meus amigos que sempre me incentivaram a prosseguir no curso.

À empresa NTI e especialmente ao projeto SIGEL que colaborou muito com o meu conhecimento e possibilitou que eu fizesse um bom uso das tecnologias utilizadas no estágio.

E à todos contribuíram para a minha formação no curso de Ciência da Computação.

“A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.”

(Albert Einstein)

RESUMO

A existência de uma Loja Virtual se torna muito importante quando se pretende alcançar um público maior e deseja obter um lucro imediato. No entanto é necessário que se tome algumas medidas de seguranças para evitar que terceiros tenham acesso a dados pessoais. O uso de um *framework* como o *spring* tornará possível a implementação da segurança na Loja de Comércio Eletrônico, sem que se tenha conhecimento dos detalhes de sua implementação. É muito importante também, que o site tenha uma interface gráfica bem chamativa, para que o comprador se sinta uma comodidade maior ao navegar pelo sítio virtual da loja, para isso foi utilizado arquivos de folhas de estilo CSS, *frameworks* como JSF 2.0 e *primefaces* 3.0 que compunham toda essa parte visual.

Palavras-chave: Segurança, *framework primefaces 3.0, JSF 2.0, spring,* Loja Virtual.

ABSTRACT

The existence of a Shop becomes very important when trying to reach a wider audience and want to get an immediate profit. However it is necessary to take some security measures to prevent third parties have access to personal data. Using a framework like spring will allow the implementation of security in E-Commerce Store, without being aware of the implementation details. It is also very important that the site has a graphical interface and catchy, so that the purchaser feel greater comfort when navigating through the site's virtual store, it was used for files CSS style sheets, frameworks such as JSF 2.0 and 3.0 primefaces that made all the visual part.

Keywords: Security, framework primefaces 3.0, JSF 2.0, spring, Virtual Store.

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

C2C – Consumer to Consumer

CNPJ – Cadastro Nacional de Pessoa Jurídica

CPF – Cadastro de Pessoa Física

CRUD – Create, Read, Update e Delete

CSS - Cascading Style Sheets

DNS - Domain Name Service

DSL – Domain Specific Language

EDI - Eletronic Data Interchange

EFT - Eletronic Founds Transfer

HQL – Hibernate Query Language

HTML - Hyper Text Markup Language

HTTPS - HyperText Transfer Protocol Secure

IDE - Integrated Development Environment

JSF – Java Server Faces

JSP – Java Server Pages

MVC – Model, View, Controller

PDF – Portable Document Format

SGBD - Sistema Gerenciador de Banco de Dados

SQL - Structured Query Language

SSL - Secure Sockets Layer

XML – Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Nas duas últimas décadas, o comércio eletrônico tem se mostrado cada vez mais presente na vida cotidiana dos indivíduos. Hoje é muito comum as pessoas comprarem produtos, fazerem negócios, transferir dinheiro, tudo isso, sem sair de casa, gerando assim mais conforto e comodidade para essas transações. Isso ocorre em consequência do desenvolvimento da rede de computadores, avanço tecnológico da informação e o alcance em escala mundial dessas tecnologias nas mãos da população. O comércio eletrônico proporcionou mais comodidade aos clientes, o tirando do desconforto de se esperar por inúmeras horas em uma fila de um caixa de uma loja; para realizar uma consulta de saldo ou transferência de dinheiro em um banco. Mas apesar das inúmeras facilidades que essa nova forma de fazer comércio proporciona, é necessário que tanto os consumidores como os donos desse tipo de negócio, tomem algumas providências de segurança, tais como certificação digital, para evitar que os mesmos sejam alvo de crackers (pessoas com alto conhecimento de informática, que utilizam desse conhecimento para cometer atos ilegais). Hoje a distância já não é mais uma barreira que impeça o cliente de comprar um produto que acabou de ser lançado do outro lado do mundo, pois o comércio eletrônico com o uso da internet, tem a viabilidade de levar este produto até o consumidor.

O *e-commerce* (comércio eletrônico) surgiu no final dos anos 70. Primeiramente ele apareceu com o uso de tecnologias como *Eletronic Data Interchange* (EDI) e *Eletronic Founds Transfer* (EFT), permitindo que as empresas mandassem documentos comerciais como ordem de compra e contas de forma eletrônica. Mas o e-commerce teve seu auge por volta de 1995 com o surgimento da empresa Amazon nos Estados Unidos, que comercializava seus produtos pela internet gerando lucros para a empresa. Esse fato aconteceu primordialmente neste país devido ao fato de a internet ter chegado mais cedo e, também, pelo fato do desenvolvimento de uma tecnologia DSL e protocolos de segurança, que possibilitavam o tráfego mais seguro de informações pessoais tais como números de cartões de crédito, de identidade, CPF e outros documentos. No Brasil, este tipo de comércio chegou apenas a ser iniciado por volta de 5 anos mais tarde e com isso as vendas não pararam de crescer, aumentou exponencialmente.

O comércio eletrônico continua crescendo cada vez mais, isso se deve ao fato da popularização do computador e da velocidade de banda, sendo esta última, a responsável

por dá ao consumidor uma navegação mais agradável sobre as páginas do site de uma empresa.

1.1 Motivação

O desenvolvimento web de uma aplicação na área de comércio eletrônico é de suma importância, pois viabiliza que o dono do negócio abranja uma clientela maior e tenha gastos menores, o que não aconteceria se o mesmo continuasse utilizando os serviços de um comércio tradicional, onde o mesmo teria custos maiores com as instalações físicas e com uma grande quantidade de funcionários para atender os clientes que compram os produtos.

A utilização de tecnologias novas e ferramentas de desenvolvimento auxiliam na aceleração do desenvolvimento da aplicação, diminuindo prazos e possíveis custos. O uso de padrões no código, é fundamental para que o sistema se torne futuramente mais fácil de se fazer uma possível manutenção, levando em conta que essa manutenção nem sempre é feita por um mesmo desenvolvedor.

O emprego de uma interface gráfica é essencial para interação entre o usuário e o sistema, tornando viável a navegação no sistema.

Enfim, o emprego de um esquema de segurança no sistema é de alta prioridade, tornando o sistema mais confiável e, assim, evitando que dados pessoais dos clientes caiam nas mãos de pessoas não autorizadas.

1.2 Objetivos

1.2.1 Objetivo geral

O intuito desta monografia é mostrar como as tecnologias e as ferramentas de desenvolvimento de programação contribuem para o desenvolvimento de uma aplicação de comércio virtual eletrônico, facilitando a sua manutenção e acelerando o processo de desenvolvimento. Assim como um esquema de segurança da informação e uma interface de interação entre usuário e sistema tornam a navegação mais cômoda e segura.

1.2.2 Objetivos específicos

1. Descrever as principais tecnologias e ferramentas utilizadas durante o processo de desenvolvimento

2. Fazer uma análise de como essas ferramentas contribuem para o desenvolvimento do sistema.

3. Implementar um sistema voltado para o gerenciamento de venda de discos denominado *Loja Discoteka* utilizando o esquema de segurança da informação.

1.3 Organização do trabalho

Esta monografia encontra-se dividia em 5 (cinco) capítulos. No capítulo 1 fez-se uma introdução de forma generalizada a respeito do comércio eletrônico.

No capítulo 2, apresenta-se uma fundamentação teórica sobre o comércio eletrônico.

No capítulo 3, explica-se o mecanismo de segurança da informação utilizado no sistema de comércio eletrônico com ênfase na criptografia.

No capítulo 4, é demonstrado o estudo de caso do sistema criado voltado para o gerenciamento de venda de discos, onde são descritas as diversas tecnologias que foram utilizadas no sistema e contribuíram muito na implementação do sistema.

No capítulo 5, por fim, apresentam-se a conclusão da monografia e as sugestões de trabalhos futuros referentes a esta pesquisa.

2 O COMÉRCIO VIRTUAL ELETRÔNICO

O comércio eletrônico é qualquer transação comercial feita através de um aparelho eletrônico, seja para compra de produtos ou para a compra de serviços. Para compra de produtos físicos, a exemplo tem-se a loja virtual do submarino, na qual o consumidor compra um produto e a loja compunha de uma logística que engloba todos os processos que vão desde o pagamento até o serviço de entrega do produto. Para a compra de serviços tem-se a agência de turismo CVC, que oferece pacotes turísticos para o mundo todo. A transação pode ser feita também através de um telefone celular, que inclui compras de serviços imediatas tais como músicas em formato MP3, papéis de parede e compras de crédito; por um *tablet*, há uma grande empresa chamada *Apple* que através da *Apple Store* vende softwares que são carregados e instalados no aparelho instantaneamente; e, finalmente, através de *desktops* e *notebooks*. Foram mencionados dois tipos de redes nas quais são oferecidos estes serviços ou produtos: a rede telefônica e a rede mundial de computadores – a Internet, a qual dará-se-á ênfase. O comércio eletrônico voltado para Internet é um negócio lucrativo que cresce de maneira rápido e atinge uma clientela global. Esse crescimento está intimamente ligado ao desenvolvimento de tecnologias de segurança, pelo aumento da largura de banda – permitindo o tráfego mais agradável pela rede de Internet onde se encontra a loja virtual, e enfim, porque neste tipo de comércio, muitos encontram uma forma se desviar do caminho do desemprego muito abundante em nosso país.

2.1 Vantagens e desvantagens do comércio eletrônico

O comércio eletrônico voltado para web, apesar de oferecer inúmeras facilidades de compra de seus produtos pela internet, também pode gerar alguns inconvenientes que deixam em pauta o uso de seus serviços. Assim como há muitas pessoas que já se satisfizeram deste tipo de serviço, também já houve outras que de alguma forma tiveram algum tipo de inconveniente. A seguir, será visto as principais vantagens e desvantagens que o comércio eletrônico pode trazer para os usuários de seus serviços.

2.1.1 Vantagens

São inúmeras as vantagens que o comércio eletrônico pode proporcionar aos

consumidores de produtos e serviços pela internet, serão citadas aqui as principais:

1. É barato, o dono do negócio não precisa se preocupar com instalações físicas tais como instalações elétricas e decoração do ambiente;

2. É altamente lucrativo, através da internet muitos usuários poderão solicitar a compra de um produto concomitantemente, independentemente do local em que estejam, ou do horário em que o mesmo esteja comprando.

3. É cômodo, a pessoa já não precisa sair de casa para fazer a compra, pode simplesmente através de um clique, selecionar o produto requerido, colocar no carrinho de compras e, finalmente, mencionar a forma de pagamento e digitar os dados necessários para receber o produto em casa.

2.1.2 Desvantagens

Apesar das inúmeras facilidades que o comércio eletrônico proporciona, ainda encontram-se algumas desvantagens:

1. No caso de uma compra que esteja vinculada a roupas e acessórios, fica impossível a pessoa saber se aquele produto cairá bem nela ou não, pois geralmente esses produtos precisam ser provados antes da compra, para que se possa ter um controle de qualidade.

2. Como a realidade econômica do país ainda não é das melhores, muitas pessoas pertencem ainda ao grupo de exclusão digital, não tendo conhecimento na área de informática para poder participar dessa nova forma de fazer comércio.

3. Outra desvantagem desse tipo de comércio, é que deixa de existir um contato direto entre o cliente e o vendedor, sendo que agora o computador faz o papel de vendedor. O vendedor é fundamental para incentivar o cliente a levar aquele produto a sua casa, o que não acontece com o comércio eletrônico, sendo que o cliente, muitas das vezes por falta de incentivo, acaba desistindo da compra.

2.2 Tipos de E-Commerce

O *e-commerce* (comércio eletrônico)é caracterizado pelas relações entre os diferentes tipos de membros que participam de uma negociação. Há 3 tipos de comércio eletrônico que destacam a relação comercial que envolve estes membros: *business-to-consumer, consumer-to-consumer, business-to-business e business-to-government*.

2.2.1 Business to consumer

É a transação comercial que ocorre entre uma empresa e um consumidor. O usuário executa a compra no site de uma determinada empresa através da internet, ela avalia a compra para saber se a mesma foi aprovada ou não e, caso seja aprovada, o produto é encaminhado para a residência através dos correios ou pela própria transportadora. As lojas de comércio eletrônico virtual operam da seguinte forma:

1. O cliente faz seu cadastro no site da loja especificando principalmente seu endereço com nome de usuário e senha. O mesmo ainda tem a opção de escolher se deseja receber e-mails relativos a propagandas da respectiva loja.

2. O cliente escolhe um produto e clica na opção comprar para colocar o produto no carrinho de compras.

3. O cliente escolhe uma forma de pagamento: boleto bancário ou cartão de crédito. Caso tenha escolhido boleto bancário, a compra será liberada somente quando o comprador efetuar o pagamento da primeira parcela. Se for feita através do cartão de crédito, a compra será liberada após a autorização da operadora de crédito.

4. Tendo sido feitos os passos acima, o cliente através de seu nome de usuário e senha cadastrados no site da loja, pode acompanhar toda a logística da compra que vai desde a liberação do pedido até a entrega feita na residência do comprador.

2.2.2 Consumer to consumer

É a transação comercial que envolve a negociação de um produto entre dois consumidores - pessoa física, em um site da internet. Este tipo de comércio pode ser facilmente encontrado em muitos sites de leilões como é o caso do mercado livre.

Os negócios deste tipo de comércio são intermediados por uma empresa que oferece a infraestrutura tecnológica e administrativa. Tanto o comprador quanto o vendedor devem estar cadastrados no sistema e podem ser avaliados por todos os membros da comunidade de negócios pela quantidade de transações que já realizaram e pelas notas que receberam em cada transação, numa espécie de *ranking* dos bons negociadores. Outro mecanismo que oferece mais segurança aos usuários é o chamado “mercado pago”, um sistema com o qual o Mercado Livre recebe o pagamento do comprador e o transfere ao vendedor, após a entrega normal da mercadoria. Alguns dados sobre o *e-commerce* C2C, obtidos junto ao Mercado

Livre, parecem indicar fortes tendências: apesar de o negócio ter sido iniciado no formato de leilão, atualmente, cerca de 90% das transações no Mercado Livre são realizadas a um preço fixo estabelecido pelo vendedor; além disso, 80% dos produtos oferecidos são novos, diferentemente do que ocorria no início, onde a regra era a comercialização de produtos usados; e, por fim, nota-se, cada vez mais, a presença também de pequenas empresas oferecendo seus produtos. (Dailton Felipini, 2008)

Entre os fatores que contribuem para que este tipo de comércio cresça ainda mais são:

- o aumento do desemprego que ganha cada dia que passa dimensões maiores;

- a possibilidade de uma renda extra para pessoas que tenham um salário não muito favorável.

2.2.3 Business to business

É o tipo de transação comercial que ocorre entre duas empresas que oferecem seus produtos, serviços e troca de informações entre as mesmas, seja pela internet ou através de uma rede privada que possibilita a comunicação entre elas.

É de suma importância esse tipo de relacionamento para as empresas, pois torna o processo de troca de informação e negociações mais ágeis, além de poupar dinheiro, ou seja, se torna muito mais econômico, pois assim se poupa dinheiro com telefone e fax que eram muito utilizados antes no tipo de negociação tradicional.

Existem três portais em que se estabelecem as transações *Business to Business* entre as empresas:

- **portal para colaboradores (*intranet*)**: portal que promove o encontro e a comunicação interna entre os colaboradores de uma empresa. É uma área restrita para troca de informações e tomadas de decisão em locais remotos;

- **portal para parceiros (*extranet*)**: são portais em que a empresa estabelece relações com outras empresas parceiras oferecendo produtos, serviços e informações sobre os mesmos.

- **portal para terceiros**: plataforma de intermediação que promove a união entre várias empresas vendedoras e compradoras. A comunicação entre elas é feita por intermédio da internet.

2.2.4. Business to goverment

É o comércio eletrônico que reúne empresas fornecedoras de bens ou serviços e as administrações públicas (federal, estadual ou municipal). É a relação de negócios pela Internet entre governo e empresas. Por exemplo: as compras pelo Estado através da Internet por meio de pregões e licitações, tomada de preços, etc.

Nessa esteira, verifica-se no Brasil que os leilões eletrônicos envolvendo o setor público e as companhias estão ganhando relevo e fôlego. Contudo, demanda o cadastramento de empresas no sistema eletrônico utilizado pelo órgão governamental e a observância dos limites insculpidos na lei nº 8666, de 21 de junho de 1993 - que regulamenta o art. 37, inciso XXI, da Constituição Federal, institui normas para licitações e contratos da Administração Pública e dá outras providências - para dispensa de certame licitatório (____, **Presidência da República**, Lei N.º 8.666, de 21 de junho de 1993).

3 SEGURANÇA DA INFORMAÇÃO

A *internet* é constituída de uma variedade de recursos que podem oferecer ganhos de produtividade, além de oportunidades para geração de receita. Mas devido a inúmeros problemas que ocorrem na rede, seja ela interna ou externa, é necessário que muitos empresários de loja virtual tomem algumas providências para assegurar que estas informações não cheguem nas mãos de pessoas não autorizadas. As maiorias dos ataques são mais provenientes da rede interna do que a própria *internet*, então é necessário que o proprietário do negócio vise por um programa de segurança global para garantir uma proteção adequada.

No sistema *Loja Discoteka* utilizou-se o método de criptografia que consiste no processo de disfarçar uma mensagem de modo a ocultar seu conteúdo, ou seja, é um processo de criação de uma *escrita secreta*. A criptografia funciona da seguinte forma, o texto normal (original) é disfarçado em uma mensagem criptografada que se processa através de um conjunto de regras que governam a substituição de um caractere por outro, usando-se uma chave e, esta, pode ser uma *string* especial de caracteres que é combinada com a mensagem através de um algoritmo matemático (MD5). Ao processar a mensagem com a chave ela é totalmente disfarçada, de modo que uma pessoa não autorizada não consiga decifrar essa mensagem, a não ser que a mesma possua esta chave. O processo inverso da criptografia onde o texto criptografado é transformado no texto original é chamado de criptoanálise, praticada pelos criptoanalistas (Daniel C. Lync, 1996).

A figura 3.1 ilustra como é feita a codificação e decodificação no processo de criptografia. Um texto normal é transformado em um texto criptografado através de um algoritmo específico para isso, o processo inverso, feito pelo criptoanalista, consiste em decifrar o texto e transformá-lo em um texto original.

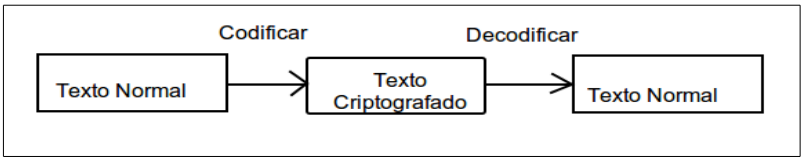


Figura 3.1 Codificação e decodificação no processo de criptografia

Os sistemas de criptografia são chamados de fortes ou fracos. Essa força pode ser expressa por meio de cinco princípios básicos:

- Identificação: é o processo que verifica se o remetente de uma mensagem é

realmente quem diz ser. Para isso no sistema de criptografia a identificação parte do princípio de que em alguma parte da informação;

- Autenticação: é o processo em que se verifica o verdadeiro remetente de um texto que faz o uso da criptografia, além de comprovar se o texto da mensagem em si foi alterado ou não;

- Verificação: relaciona-se com a capacidade de identificar e autenticar com segurança a comunicação criptografada específica;

- Impedimento da rejeição: é a qualidade de um sistema seguro que evita que qualquer pessoa possa negar ter enviado certos arquivos ou dados, quando de fato fez-se isso;

- Privacidade: capacidade de um sistema de criptografia de ocultar efetivamente as comunicações dos olhares curiosos (Daniel C. Lync, 1996).

3.1 Algoritmo de criptografia MD5

Há vários usos para a criptografia em nosso dia-a-dia: proteger documentos secretos, transmitir informações confidenciais pela *Internet* ou por uma rede local, etc.

A técnica usada em Criptografia envolve pura e simples matemática. O sistema de criptografia usado atualmente é extremamente seguro. Especialistas estimam que para alguém conseguir quebrar uma criptografia usando chaves de 64 bits na base da tentativa e erro, levaria cerca de 100.000 anos usando um PC comum.

O sistema de criptografia é bastante seguro, por exemplo, uma chave de 2 bits terá 4 combinações possíveis. Uma chave de 4 bits, terá 16 combinações possíveis e assim por diante. Agora, vejam a grande diferença em relação ao uso dessas chaves: uma chave de apenas 8 bits terá 65.356 combinações possíveis e, em uma chave de 32 bits existem mais de 4 bilhões de combinações, que para serem decifradas levariam mais de 2 meses (levando em conta 1000 tentativas por segundo, por parte do computador). O algoritmo MD5 é de 128 bits, então o número de combinações seria muito maior em relação ao de 32 bits, o que o torna mais seguro ainda.

Por ser um algoritmo unidirecional, um hash MD5 não pode ser transformado novamente na *password* (ou texto) que lhe deu origem. O método de verificação é, então, feito pela comparação das duas *hash*[1] (uma da base de dados, e a outra da tentativa de login). O

1Um *hash* é uma sequência de bits geradas por um algoritmo de dispersão, em geral representada em base hexadecimal, que permite a visualização em letras e números (0 a 9 e A a F).

MD5 também é usado para verificar a integridade de um arquivo através, por exemplo, do programa md5sum, que cria a hash de um arquivo. Isto pode-se tornar muito útil para *downloads* de arquivos grandes, para programas P2P que constroem o arquivo através de pedaços e estão sujeitos à corrupção de arquivos.

O MD5 é de domínio público para uso em geral. A partir de uma mensagem de um tamanho qualquer, ele gera um valor *hash* de 128 bits; com este algoritmo, é computacionalmente impraticável descobrir duas mensagens que gerem o mesmo valor, bem como reproduzir uma mensagem a partir do seu *digest*[2]. O algoritmo MD5 é utilizado como mecanismo de integridade em vários protocolos de padrão *Internet* (RFC1352, RFC1446, etc.), bem como pelo CERT e CIAC.

O código 3.1 mostra como é implementado o algoritmo de criptografia MD5 em linguagem java.

Código 3.1 – Código em java para implementação do algoritmo MD5

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Criptografia {
    private static MessageDigest md = null;
   static {
      try {
         md = MessageDigest.getInstance("MD5");
      } catch (NoSuchAlgorithmException ex) {
         ex.printStackTrace();
      }
   }
 private static char[] hexCodes(byte[] text) {
      char[] hexOutput = new char[text.length * 2];
      String hexString;

      for (int i = 0; i < text.length; i++) {
         hexString = "00" + Integer.toHexString(text[i]);
         hexString.toUpperCase().getChars(hexString.length() - 2,
         hexString.length(), hexOutput, i * 2);
      }
      return hexOutput;
   }
public static String criptografar(String pwd) {
      if (md != null) {
         return new String(hexCodes(md.digest(pwd.getBytes())));
      }
      return null;
   }
}
//Para fazer chamadas a esta classe, fazemos o seguinte:
public static void main(String[] args){
      String senha = " senhadeteste ";
      System.out.println(Criptografia.criptografar(senha));
}

//A saída para o código acima será:

//97D3B7B38E306F4A3AC787333A02E5DF
```

Como mostra o código, para que se possa gerar textos criptografados, é necessário seguir os seguintes passos:

2 *Digest* é um algoritmo de criptografia que obtém uma mensagem como entrada e produz um código de tamanho fixo como saída.

- Obter uma instância do algoritmo a ser usado: *MessageDigest md = MessageDigest.getInstance("MD5")*;

- Passar a informação que se deseja criptografar para o algoritmo: *System.out.println(Criptografia.criptografar(senha)*;

- Efetuar a criptografia;

Além do algoritmo MD5 de criptografia, os cientistas ainda estão estudando outros casos para aumentar a eficiência de um criptosistema, o que viabilizará ainda mais a segurança de uma loja virtual na rede de *Internet*.

3.2 Medidas de segurança tomadas pelo consumidor

Já o consumidor, que também pode-se chamar de E-consumidor, deve tomar algumas precauções para que não venha a ser objeto de fraude. Entre estas precauções destacam-se:

- Pesquisar sobre o produto a ser comprado em *sites* indicados por alguém, que já realizou diversas compras e nunca ocorreu algo de errado em suas transações.

- Procurar sempre saber a localização da empresa, onde ela se encontra instalada, se é registrada, se possui CNPJ, e se a mesma emite nota eletrônica. Esta nota eletrônica é garantia que a empresa é legalizada e não oferece riscos a compra.

- Deve-se manter o computador sempre protegido com antivirus.

- Observar se o site que se navega, possui o Certificado de Segurança (SSL), para fazer esta verificação, basta saber se o endereço começa com http:// ou https://, se estiver com este último ele está com o certificado digital, o que garantirá segurança na hora da compra.

No sistema web *Loja Discoteka*, será utilizado um *framework* chamado *spring*, ele é responsável por implementar toda as segurança do *site* sem que o usuário precise conhecer os detalhes de como ele foi implementado, ou seja, já vem com toda a implementação encapsulada.

4 ESTUDO DE CASO – UTILIZAÇÃO DE FERRAMENTAS DE PROGRAMAÇÃO VOLTADAS PARA A INTERNET

Este estudo de caso abordará os conceitos de programação voltada para web, onde se desenvolveu um sistema denominado *Loja Discoteka* voltado para a *internet*, com o intuito de explorar e descrever tecnologias como *hibernate* (tecnologia de mapeamento de classes), *primefaces* (*framework* utilizado na *view* de uma página), linguagem java (linguagem orientada a objetos), *JSF* (outro *framework* que está associado no desenvolvimento da *view* de uma página), *CSS* (arquivo de folhas de estilo que é responsável por dá uma estilização a página da web e, enfim o *framework spring* (tecnologia de segurança da informação, responsáveis por associar papéis ao usuário no sistema e possibilitar ou restringir o acesso deste a determinada página do site comercial.

Será mostrado as principais operações utilizadas com o *hibernate* assim como as consultas de sua linguagem – HQL, semelhante ao SQL só que orientada a objetos. Será visto também com detalhamento as anotações das classes de persistência e suas funcionalidades no sistema, as funcionalidades das tecnologias utilizadas na camada de visão, ou seja, os componentes utilizados e, como um arquivo de folha de estilo pode facilitar futura manutenção da interface com usuário.

4.1 Contextualização do problema

Devido a uma grande demanda de compra de discos, uma loja deseja automatizar suas vendas. Com isso, ficará mais fácil controlar as finanças da empresa, com a emissão de relatórios mensais para saber os lucros e gastos do negócio eletrônico. Para englobar o maior número de pessoas possível na realização de compras, é necessário que se hospede o portal em um servidor da *internet*, para que o produto seja comprado independentemente do espaço físico em que o comprador se encontre.

É necessário um esquema de segurança, já que o sistema terá diferentes tipos de usuário e cada um deles terá um papel diferenciado. Para isso o sistema terá que usar uma tecnologia que se encarregue de resolver essa questão.

Com isso foi desenvolvido um sistema *Loja Discoteka* voltado para o gerenciamento de venda de discos para simular uma loja de comércio virtual eletrônico real.

4.2 Descrição genérica do sistema

O sistema *web* chamado *Loja Discoteka* voltado para gerenciamento de discos desenvolvido na rede da *Internet*, tem como objetivo automatizar as transações comerciais para agilizar vendas e consequentemente gerar lucros imediatos.

Neste sistema o usuário poderá realizar as transações comerciais, desde que esteja antecipadamente cadastrado. Poderá visualizar as compras que foram feitas, assim como os produtos junto com a sua descrição geral.

Já o administrador, além de realizar todo o processo feito pelo cliente, poderá também cadastrar fornecedor, cadastrar produto e ter uma visão geral de todas as operações que foram feitas no sistema de compras.

No sistema, o usuário poderá ter um dos papéis:

- usuário comum: poderá se cadastrar e fazer transação comercial, além de poder visualizar suas compras feitas ao longo do tempo;
- administrador: além de realizar os mesmos papéis realizados pelo usuário comum, poderá fazer outras operações, ou seja, poderá fazer tudo no sistema, inclusive operações de cadastro para:
    - cliente;
    - produto;
    - fornecedor;
    - categoria.

Contudo, para que esses usuários desempenhem suas funções de maneira correta, será usado um *framework* de segurança, que será responsável pela autenticação do cliente e definição de seus papéis, assim, para cada perfil de usuário, a página ou parte dela terá visibilidade diferenciada.

4.3 Requisitos

Um requisito é tido como uma declaração abstrata, de alto nível, de uma função que o sistema deve fornecer ou de uma restrição do sistema. É considerado uma função detalhada, matematicamente formal, de uma função do sistema (Ian Sommerville, 2011). É uma forma de documentar, identificar e organizar essas funcionalidades para que o sistema final esteja correspondente com as regras que lhes foram passadas. No sistema gerenciador de vendas de

discos há dois tipos de requisitos: os funcionais e os não funcionais.

Os requisitos extraídos do sistema foram feitos através da reutilização de requisitos, ou seja, reaproveitando de padrões ou requisitos de outros sistemas na área de comércio virtual, já que a extração desses requisitos não foi feita com um cliente real.

4.3.1 Requisitos funcionais

Os requisitos funcionais são as especificações das funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como este deve se comportar diante de determinadas situações. Em alguns casos particulares, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer (Ian Sommerville, 2011).

No sistema gerenciador de vendas de discos, tem-se enumeradas os seguintes requisitos levantados e as soluções para resolvê-los:

Tabela 4.1 – Lista de Requisitos funcionais

Requisitos funcionais	Solução
O Administrador deverá cadastrar o fornecedor, cliente, categoria e produtos no Sistema de Gerenciamento de Discos. Ele realizará todas as operações padrões realizadas em uma página de cadastro.	Cria-se dois formulários um para listagem, que listará todos os dados armazenados para visualização onde serão feitas operações de atualização e remoção; no outro formulário, tem-se uma página de cadastro onde serão validados os dados, caso atendam os requisitos de validação.
O sistema será responsável por atribuir papéis ao usuário de forma que cada um tenha funções diferentes no sistema.	Será utilizado um software de segurança de modo que cada usuário opere no sistema com o papel que a ele foi atribuído.
O usuário deverá selecionar um produto, onde poderá obter todas as suas informações e, em seguida, poderá optar por efetuar a compra ou não.	Na página principal do sistema, serão listados todos os produtos, mas o usuário poderá utilizar um filtro para selecionar os produtos por categoria.
O usuário poderá ver todas as suas compras feitas durante todo o período como cliente.	Será criado uma página para visualização de compras efetuadas, ou seja, um histórico de compras.
O sistema deverá emitir um relatório com o contato de todos os fornecedores da loja virtual ordenados em ordem alfabética.	Será utilizado um software gerador de arquivo em formato pdf que emitirá esse relatório.

4.3.2 Requisitos não funcionais

Os requisitos não funcionais são as restrições sobre os serviços ou as funções oferecidos pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros (Ian Sommerville, 2011).

Os requisitos não funcionais mapeiam os aspectos qualitativos de um *software*, por exemplo: performance (tempo de resposta); segurança (restrições de acesso, privilégios); perspectiva do usuário (padrão das cores, disposição dos objetivos na tela); comunicabilidade (*e-mail*, *VoIP*, *Browser*); usabilidade e portabilidade (a aplicação deve rodar em vários tipos de aplicativos: móveis, *desktop*, *notebooks*). Este grupo é de suma importância e não deve ser desprezado durante o processo de produção de software. Como qualquer outro tipo de requisito, ele deve ser levantado, analisado, especificado e validado.

É importante salientar que a validação de um requisito não funcional requer a aplicação de atributos quantitativos. Porém, em alguns casos, esse tipo de atributo não é mapeado facilmente. Imagine que você tenha validar um requisito não funcional caracterizado pela perspectiva do usuário em uma aplicação que será acessada por milhares de pessoas. Neste caso um padrão de cores pode agradar alguns e desagradar outros.

É de fundamental importância encontrar um mecanismo que quantifique um requisito não funcional. Partimos do seguinte exemplo: Um requisito não funcional qualquer estabelece que o software tem que ter um bom tempo de resposta. Para quantificar esse requisito, pergunte ao usuário o que ele considera um bom tempo de resposta. Uma variação ente 0,5 e 2 segundos é aceitável?

Enfim, os requisitos não funcionais são de suma importância e podem delimitar o sucesso ou o fracasso de um projeto de software, dependendo do software que está sendo desenvolvido.

Os requisitos não funcionais também podem ser especificados de acordo com o seu tipo, no sistema web chamado *Loja Discoteka* têm-se os seguintes tipos para estes requisitos e suas especificações:

Tabela 4.2 – Lista de requisitos não funcionais

Tipo de requisito não funcional	**Especificação**
Requisito de segurança	Será utilizado o *framework spring*, responsável por controlar as operações processadas dentro de um sistema de acordo com o papel em que o usuário está inserido.

Requisito de implementação	Serão utilizados os seguintes softwares e/ou tecnologias no Sistema *Loja Discoteka*: - *hibernate*: *framework* de mapeamento que associa cada classe java com uma entidade no banco de dados. Ele é recomendado para aplicações em que as lógicas de negócios não são feitas diretamente no banco, como o sistema de gerenciamento de vendas de discos. - java: linguagem orientada a objetos, responsável por encapsular muitos métodos e facilitar o desenvolvimento do sistema. - *JSF 2.0*: *framework* utilizado na camada de visão do sistema, é uma extensão do html. - *primefaces*: outro *framework* utilizado no sistema, também é uma extensão do html, só que com uma aparência mais sofisticada. Também se encontra o *Jquery* embutido, evitando o trabalho de escrever arquivos *javascript* na página. Utiliza-se esse *framework* principalmente nas páginas de listagem, onde as buscas são feitas de forma dinâmica. - ireport: software que com bases em comados sql, é responsável por gerar relatórios em PDF no sistema de gerenciamento de vendas de discos. - *netbeans* 7.1: foi a IDE utilizada para a construção do sistema devido a sua facilidade de configuração e bibliotecas que já vem incluídas na mesma, sem que precise baixá-las para instalação. - *postgreSQL*: SGBD usado devido ao fato de muitas serviços de hospedagem darem suporte a este softwares. Ele também foi utilizado porque possui algumas características que o difere de muitos outros SGBS: - *schemas***:** recurso que permite cruzar informações em um mesmo banco de dados, mas em estruturas diferentes; - *SQL***:** sigla para *Structured Query Language*, é uma linguagem utilizada em bancos de dados relacionais; - *Views***:** os views consistem em um tipo de tabela virtual formada por campos extraídos de uma tabela "verdadeira", facilitando o controle sob os dados acessados. - *glassfish*: servidor de página web usado para rodar a aplicação.

4.4 Diagrama de Casos de Uso

O diagrama de casos de uso corresponde a uma visão externa do sistema e representa graficamente os atores[3], os casos de uso[4], e os relacionamentos entre estes elementos. Ele tem como objetivo ilustrar em um nível alto de abstração quais elementos externos interagem com

3 Atores especificam os usuários que interagem diretamente com o sistema.
4 Caso de uso é uma operação realizada pelo o ator.

que funcionalidades do sistema, ou seja, a finalidade de um diagrama de caso de uso é apresentar um tipo de diagrama de contexto que apresenta os elementos externos de um sistema e as maneiras segundo as quais eles as utilizam.

A figura 4.1 demonstra essas interações ocorridas no sistema de gerenciamento de venda de discos *Loja Discoteka* voltado para rede da *Internet*. Também demonstra que há dois tipos de atores principais, o usuário e o administrador do sistema, onde a relação entre eles circundam por uma generalização[5]. Cada um tem um papel diferente no sistema, e dependendo do mesmo, cada tipo será responsável por realizar operações diversas.

Para descrever cada um desses casos de uso, são utilizados os cenários, que consistem em descrever textualmente as operações feitas pelos atores no sistema. No sistema de gerenciamento de vendas de discos *Loja Discoteka* serão mostrados dois tipos de cenários, o principal, que consiste nas operações e comportamento normais do sistema levando em conta que todas as entradas são válidas e, o cenário variacional, que consiste no comportamento anormal do fluxo caso haja algum problema na realização de uma dada operação no sistema.

5 Generalização é um tipo de relação que há entre dois atores, onde um ator, além de realizar sua operações específicas realiza também as operações do outro através deste processo.

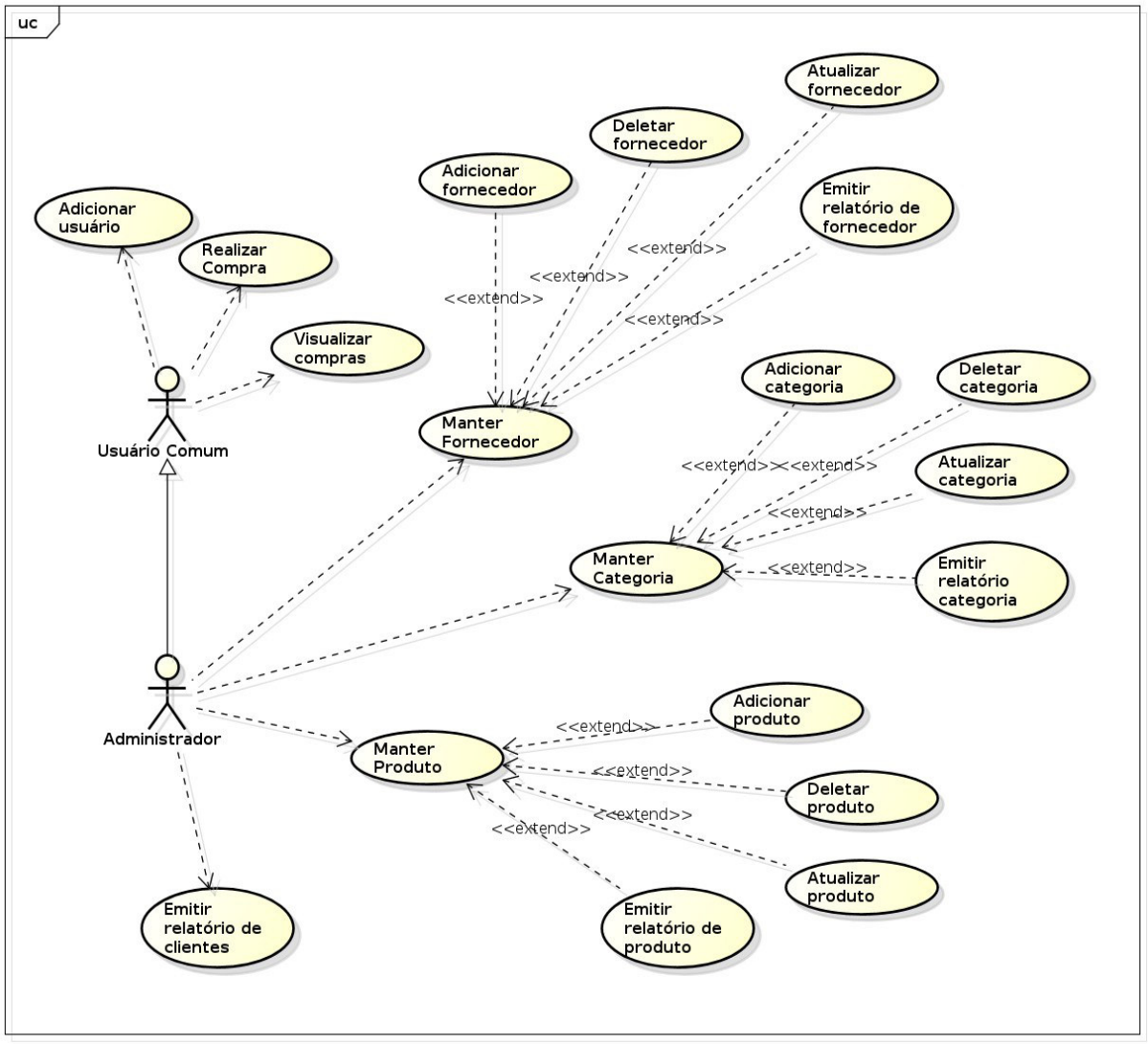


Figura 4.1 – Diagrama de caso do sistema *Loja Discoteka*

A tabela 4.3 ilustra o cenário para se fazer o cadastro de cliente para que o mesmo possa acessar o sistema. Esse cadastro poderá ser feito tanto pelo o usuário comum quanto pelo usuário administrador do sistema.

Tabela 4.3 – Cenário cadastrar cliente

Nome	Cadastrar cliente
Descrição	Cadastrar cliente para que o mesmo possa realizar a compra de um produto no site de gerenciamento de vendas de discos
Pré-condição:	O usuário deve ter em mãos os seus dados pessoais
Dados entrada e saída	Entrada: endereço, cidade, CEP, UF, pais, telefone, fax, e-mail, senha, nome Saída: Mensagem de sucesso da transação de cadastro de usuário cliente

Ações do ator	**Ações do sistema**
1. O usuário seleciona o comando para cadastrar um novo cliente. 2. O usuário informa o endereço, cidade, cep, uf, país, telefone, fax, e-mail, senha e nome válidos. 4. O usuário finaliza o cadastro clicando em cadastrar usuário.	3. O sistema verifica se os dados são válidos.
Cenário Variacional (Exceção)	
Ações do ator	**Ações do sistema**
	Caso os dados não atendam as regras de validação, o sistema emite uma mensagem de erro na hora do cadastro.
Pós-condição	Cadastro do usuário é realizado com sucesso

A tabela 4.4 ilustra o cenário para se fazer o cadastro de fornecedor de produtos para o sistema. Esse cadastro poderá ser feito somente pelo usuário administrador.

Tabela 4.4 – Cenário cadastrar fornecedor

Nome	Cadastrar fornecedor
Descrição	Cadastrar fornecedor que irão fornecer os produtos utilizados na loja de discos virtual
Pré-condição:	O administrador deve estar logado no sistema
Dados entrada e saída	Entrada: cnpj, nome da empresa, nome do contado, cargo do contato, endereço, cidade, UF, cep, pais telefone, fax Saída: Mensagem de sucesso da transação de cadastro de fornecedor
Ações do ator	**Ações do sistema**
1. O administrador seleciona a opção cadastro de fornecedor. 2. O administrador fornece cnpj, nome da empresa, nome do contato, cargo do contato, endereço, cidade, UF, CEP, país, telefone e fax válidos. 4. O usuário finaliza o cadastro clicando em cadastrar fornecedor.	3. O sistema verifica se os dados são válidos.
Cenário Variacional (Exceção)	
Ações do ator	**Ações do sistema**
	Caso os dados não atendam as regras de validação, o sistema emite uma mensagem de erro na hora do cadastro.
Pós-condição	Cadastro do fornecedor é realizado com sucesso

A tabela 4.5 ilustra o cenário para se fazer o cadastro de produto. Esse cadastro poderá ser feito apenas pelo usuário administrador, para processar este cadastro o produto deverá ter uma categoria e fornecedor cadastrados no sistema.

Tabela 4.5 – Cenário cadastrar produto

Nome	Cadastrar produto
Descrição	Cadastrar produtos que serão utilizados para negociação na loja discoteka
Pré-condição:	O administrador deve estar logado no sistema. Devem está cadastrado fornecedor e categoria no sistema
Dados entrada e saída	Entrada: nome produto, quantidade por unidade, preço unitário, unidades em estoque, url_produto, fornecedor e categorias válidos Saída: Mensagem de sucesso da transação de cadastro de produto.
Ações do ator	**Ações do sistema**
1. O administrador seleciona a opção cadastro de produto. 2. O administrador fornece nome produto, quantidade, por unidade, preço unitário, unidades em estoque, url_produto, fornecedor e categorias válidos 4. O usuário finaliza o cadastro clicando em cadastrar produto.	3. O sistema verifica se os dados são válidos.
Cenário Variacional (Exceção)	
Ações do ator	**Ações do sistema**
	Caso os dados não atendam as regras de validação, o sistema emite uma mensagem de erro na hora do cadastro.
Pós-condição	Cadastro do produto é realizado com sucesso

A tabela 4.6 ilustra o cenário para se fazer cadastro de categoria de produtos. Esse cadastro poderá ser feito somente pelo usuário administrador.

Tabela 4.6 – Cenário cadastrar categoria

Nome	Cadastrar categoria
Descrição	Cadastrar categorias dos discos que serão vendidos na loja de disco virtual
Pré-condição:	O administrador deve estar logado no sistema.
Dados entrada e saída	Entrada: nome e descrição Saída: Mensagem de sucesso da transação de cadastro de categoria.
Ações do ator	**Ações do sistema**
1. O administrador seleciona a opção cadastro de categoria. 2. O administrador fornece nome nome e descrição da categoria válidos. 3. O administrador finaliza o cadastro clicando em cadastrar categoria.	3. O sistema verifica se os dados são válidos.
Cenário Variacional (Exceção)	
Ações do ator	**Ações do sistema**
	Caso os dados não atendam as regras de validação, o sistema emite uma mensagem de erro na hora do cadastro.
Pós-condição	Cadastro de categoria é realizado com sucesso.

4.5 Diagrama de Modelo do Banco de Dados

O diagrama de modelo da figura 4.2, tem como princípio básico representar a forma de como as entidades estão estruturadas no banco de dados e como estas estão relacionadas entre si. Neste diagrama está representado as entidades junto com seus atributos e as relações expressas por meio de suas cardinalidades.

Junto ao diagrama de modelo, há um documento que descreve os objetos e seus atributos. O seu objetivo é para que os analistas tenham uma descrição acerca desses objetos na hora de criar as classes que contêm as regras de negócios associadas a estes. Essa descrição do tipo textual serve para suprir as informações que não encontramos no diagrama.

A tabela 4.7 descreve as entidades encontrados no sistema de gerenciamento de venda de discos *Loja Discoteka* . O dicionário é constituído dos seguintes elementos: entidade[6], atributo[7], classe[8], domínio, tamanho[9] e descrição[10].

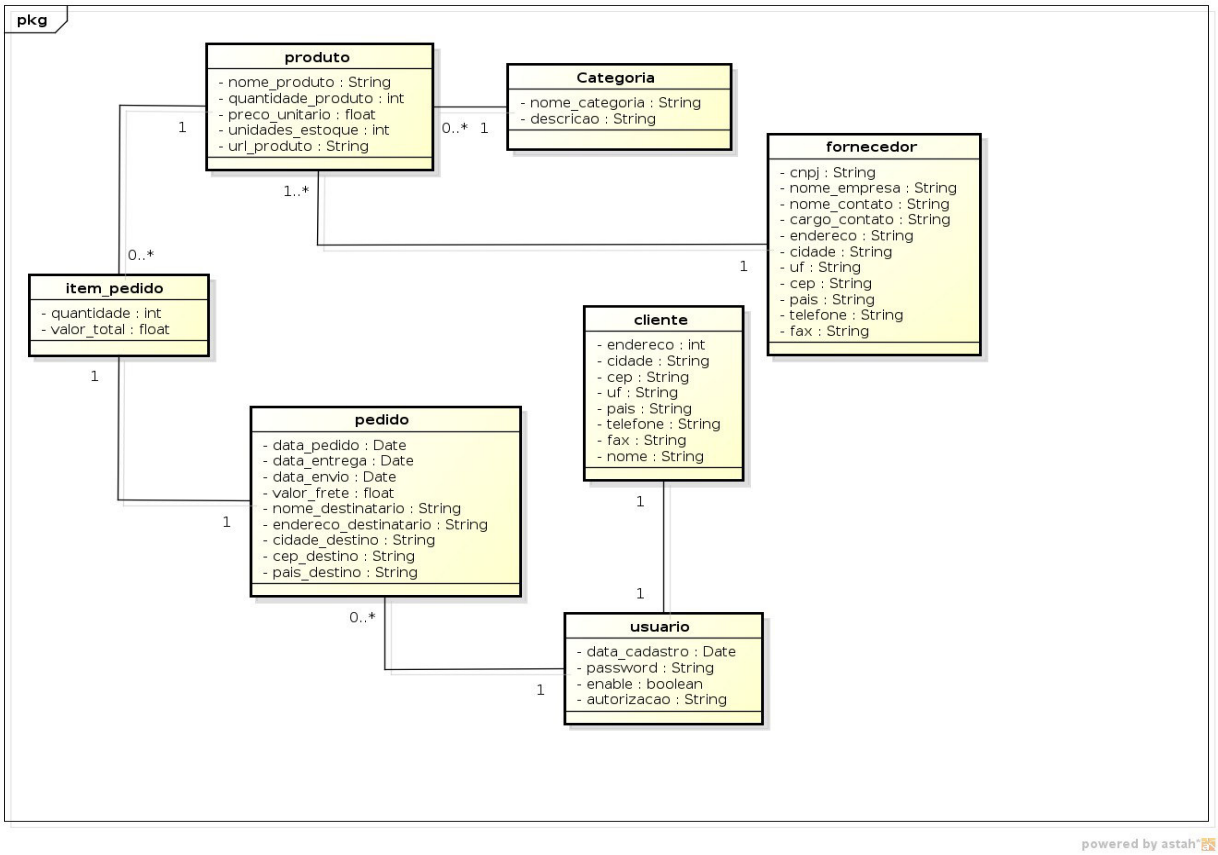


Figura 4.2 – Diagrama de modelo do sistema *Loja Discoteka*

6 Entidade é o nome da entidade definida no diagrama de classes. É o elemento do qual se quer guardar informações.
7 Atributo refere-se às características que a entidade possui.
8 A classe pode ser classificada em três tipos: simples, indicando um atributo normal; composto; multivalorado; e determinante, pode ser usado como chave.
9 Tamanho define a quantidade de caracteres necessários para armazenar um atributo do tipo texto.
10 A descrição é utilizada para dá a informação sobre um dado atributo.

Tabela 4.7 – Dicionário de dados da entidade cliente

Entidade: cliente				
Atributo	Classe	Domínio	Tamanho	Descrição
id_cliente	determinante	numérico		
endereco	composto	texto	100	Rua, Avenida, Bairro
cidade	simples	texto	20	
cep	simples	texto	08	Valor com máscara de entrada
uf	simples	texto	2	Estado
pais	simples	texto	15	
telefone	multivalorado	texto	16	
fax	multivalorado	texto	16	
nome	simples	texto	100	Nome do cliente
Entidade: usuario				
Atributo	Classe	Domínio	Tamanho	Descrição
username	determinante	texto	20	Nome de usuário
Password	simples	texto	20	Senha do usuário
autorizacao	simples	texto	20	Papel que o usuário representa
data_cadastro	simples	data		Data que o usuário se cadastrou
Entidade: categoria				
Atributo	Classe	Domínio	Tamanho	Descrição
id_categoria	determinante	numérico		
nome_categoria	simples	texto	20	
descricao	simples	texto	100	
Entidade: fornecedor				
Atributo	Classe	Domínio	Tamanho	Descrição
id_fornecedor	determinante	numérico		
cnpj	simples	texto	18	
nome_empresa	simples	texto	20	
nome_contato	simples	texto	50	
cargo_contato	simples	texto	20	
endereco	composto	texto	50	
cidade	simples	texto	20	
uf	simples	texto	2	
cep	simples	texto	9	
pais	simples	texto	20	
telefone	multivalorado	texto	17	
fax	multivalorado	texto	17	

Entidade: produto				
Atributo	Classe	Domínio	Tamanho	Descrição
id_produto	determinante	numérico		
nome_produto	simples	texto	80	
quantidade_unidade	simples	texto		
preco_unitario	simples	numérico		
unidades_estoque	simples	numérico		
url_produto	simples	texto	50	Referência da figura do produto
Entidade: item_pedido				
Atributo	Classe	Domínio	Tamanho	Descrição
id_item_pedido	determinante	numérico		
valor_total	simples	numérico		Valor total das compras
quantidade	simples	numérico		
Entidade: item_pedido				
Atributo	Classe	Domínio	Tamanho	Descrição
id_pedido	determinante	numérico		
data_pedido	simples	data		
data_entrega	simples	data		
data_envio	simples	data		
valor_frete	simples	numérico		
nome_destinatário	simples	texto	100	
endereco_destinatario	composto	texto	100	
cidade_destino	simples	texto	20	
cep_destino	simples	texto	9	
pais_destino	simples	texto	15	

4.6 Técnicas de implementação

No sistema web *Loja Discoteka* voltado para gerenciamento de discos, foram utilizadas tecnologias que auxiliaram e, ao mesmo tempo, diminuíram o tempo de implementação do sistema. Para a persistência dos dados foi utilizado o SGBD (Sistema de Gerenciamento de Banco de Dados) ***PostgreSQL***. O *framework* ***hibernate*** foi extremamente importante, pois o mesmo foi responsável por encapsular muitas operações de persistência, pois se essas operações fossem feitas diretamente utilizando cláusulas SQL para o banco, haveria um retrabalho muito maior. Uma interface indutiva, facilitaria muito a interação humano máquina para a entrada e saída de dados, no entanto utilizou-se dois *frameworks*: **JSF (*Java Server Faces*) 2.0** e ***Primefaces* 3.0**, ambos possuem um conjunto de bibliotecas que

possuem componentes que dão funcionalidade mais dinâmicas ao sistema que, se fossem utilizados apenas códigos de formatação HTML, as páginas não teriam essas funcionalidades, então pode-se dizer, que estes dois *frameworks* funcionam como uma extensão para o HTML. Quando se desenvolve um sistema web na rede de Internet, e nesta rede são trafegados dados pessoais, é de fundamental importância que se implante um sistema de segurança, por isso foi utilizada uma tecnologia chamada ***Spring Security 3.0***, um *framework* que restringe acesso a determinadas páginas dependendo do usuário que estiver logado no sistema. Por fim, foi utilizado um programa chamado ***Ireport*** - versão 2.0.5, para gerar relatórios como forma de controlar as transações ocorridas dentro da loja virtual.

4.6.1 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional (SGBDOR). O POSTGRES foi pioneiro em vários conceitos que somente se tornaram disponíveis muito mais tarde em alguns sistemas de banco de dados comerciais. O PostgreSQL é um SGBD de código fonte aberto, que suporta grande parte do padrão SQL e oferece muitas funcionalidades modernas. O PostgreSQL tem inúmeras características que o torna bastante funcional para inúmeras aplicações que faz o seu uso. Entre essas características pode-se citar: MVCC (controle de concorrência); múltiplos tipos de índice; *constraints/foreign key;* sistema de *backup* online; suporte a *schemas;* neste SGBD pode-se também criar as tabelas através de duas formas: *scripts* ou por intermédio de interface gráfica, no sistema *Loja Discoteka* voltado para gerenciamento de vendas de discos, utilizou-se a criação das tabelas por intermédio de interface gráfica, para agilizar no processo de implementação; possui também como subconjunto do *schema* um espaço para se criar as sequências que incrementam os id's de cada chave primária das entidades, com exceção da tabela usuário que não possui id; etc.

- No *MVCC* (controle de concorrência de multi-versão), os processos de leitura não bloqueiam processos de escrita e vice-versa, reduzindo drasticamente (às vezes, eliminando) a contenção entre transações concorrentes e paralisação parcial ou completa (*deadlock*).

- Em relação aos *múltiplos tipos de Índice,* o PostgreSQL suporta índices *B-Tree*, *rTree* e *Hash*, permitindo a escolha do índice mais eficiente para cada aplicação. Estes índices são utilizados para melhorar o desempenho da busca na aplicação.

- *Constraints* se referem a um conjunto de chaves primárias[11] e estrangeiras[12] que possuem uma determinada tabela de um banco. A figura 4.3 mostra essas duas chaves criadas para a tabela produto, indicando também a referência de uma outra tabela para a qual as chaves estrangeiras presentes na tabela produto referenciam. Como chave primária criou-se a coluna id_produto, onde toda vez em que se cadastrar um produto, essa coluna terá um incremento automático e um valor único. O nome da *constraint* associada a essa coluna é pk_produto. Quando se cadastrar um produto é também necessário informar a categoria daquele produto e os dados do fornecedor que se encontram em outra tabela, para isso utilizou-se um conjunto de chaves estrangeiras que contêm essas referências nomeadas como fk_id_categoria e fk_id_fornecedor.

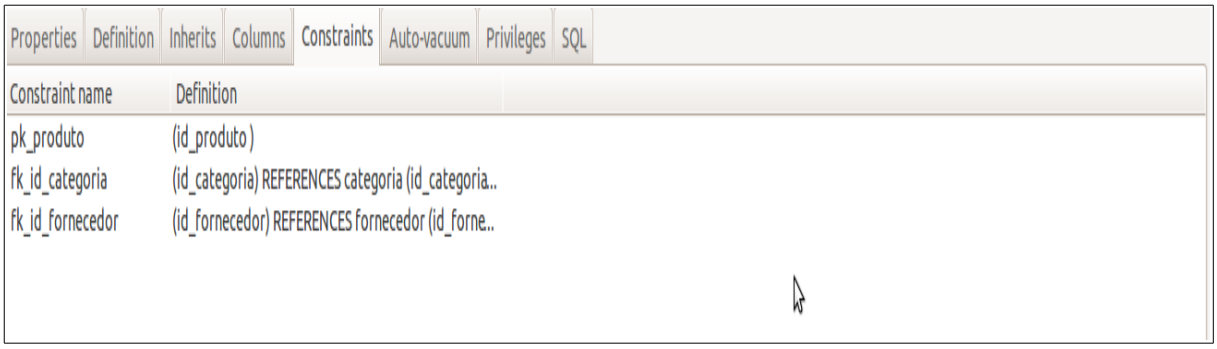


Figura 4.3 – Chave primária e chaves estrangeiras para tabela produto

- Backup On-line é o processo em que é feita uma cópia do banco ou parte dele para se fazer a transferência dos dados para uma outra máquina, ou até mesmo por questão de segurança, caso haja alguma falha no servidor em que o banco esteja hospedado.

- Além disso, o PostgreSQL permite cruzar informações de um mesmo banco em estruturas diferentes, denominadas *schemas*. Também conta com uma interface de fácil manipulação, onde o usuário pode criar *schemas* para estruturar dados. As figuras 4.4 e 4.5 mostram os *schemas* e a interface de manipulação de dados respectivamente.

Como se pode ver na figura 4.5, a interface do usuário para criação de tabelas e colunas é bastante simples, ou seja, o desenvolvedor não precisa criar *scripts* de forma manual, acelerando assim o processo de desenvolvimento do sistema. Nesta interface ele além de criar tabelas e colunas, pode definir privilégios para o usuário do banco, chaves primárias e

11 Chaves primárias se referem a uma dada coluna da tabela que possui um valor único capaz de identificar a tupla da instância.
12 Chaves estrangeiras se referem a uma dada coluna da tabela que tem a referência de outra.

chaves estrangeiras.

Figura 4.4 – Estrutura de um banco no SGBD comercioeletronico

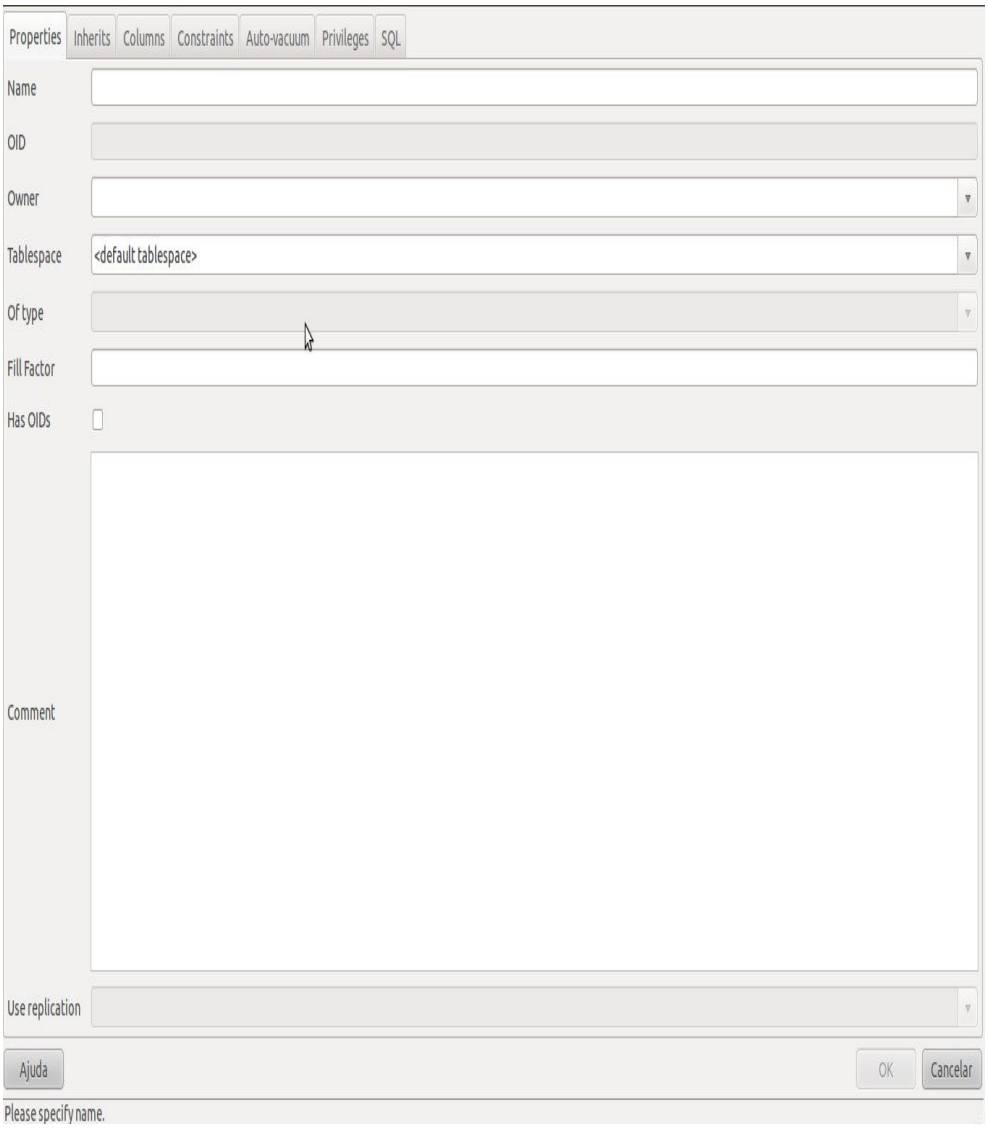


Figura 4.5 – Estrutura de criação de uma tabela para um banco

Como se pode ver na figura 4.5, a interface do usuário para criação de tabelas e colunas é bastante simples, ou seja, o desenvolvedor não precisa criar *scripts* de forma manual, acelerando assim o processo de desenvolvimento do sistema. Nesta interface ele além de criar tabelas e colunas, pode definir privilégios para o usuário do banco, chaves primárias e chaves estrangeiras.

- Dentro de cada *schema* possui uma região que serve para criar as sequências que tem como propósito incrementar as chaves primárias automaticamente sem que se precise dá uma valor a ela manualmente como é feito com outras colunas. Como esse valor é incrementado demos um tipo para ele denominado *integer* (inteiro). A figura 4.6 mostra a

estrutura dessas sequências no banco para a tabela fornecedor. Pode-se observar que se tem as duas visões que se pode visualizar como foi criada essa sequência: a gráfica – a forma utilizada na implementação do sistema, e a de *scripts*. É importante ressaltar algumas propriedades importantes para a sequencia criada: *minimum,* contém o valor mínimo para o id da tabela, no caso 1; *increment*, representa o número pelo qual será incrementado o id, em cada cadastro na tabela fornecedor; *name*, o nome dado para a sequência.

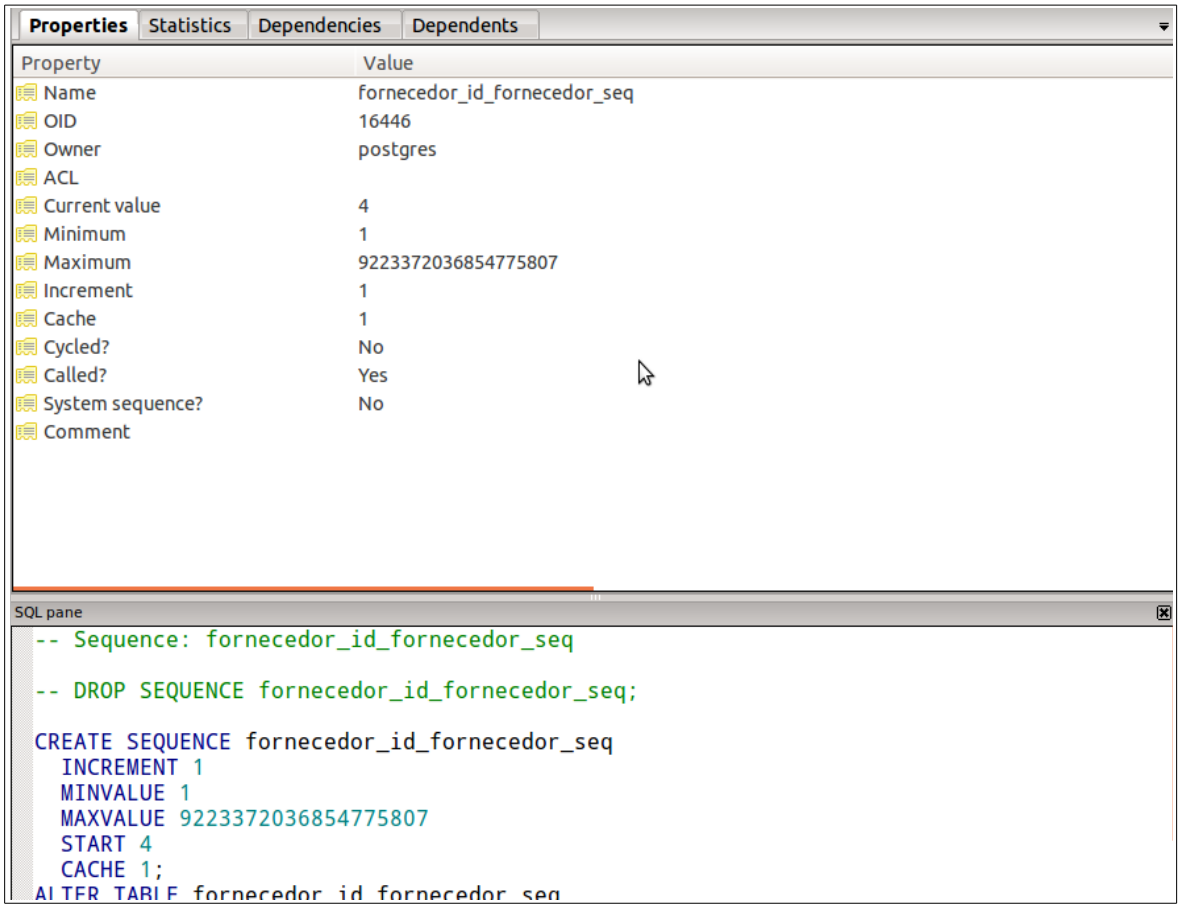


Figura 4.6 – Estrutura de uma sequência criada para a tabela fornecedor

O PostgreSQL foi de suma importância para a implementação da *Loja Discoteka* devido a sua fácil manipulação que se tem com os dados através de um meio gráfico e, também e também pela estrutura rica em utilidades que não se encontram em alguns SGBD's.

4.6.2 Hibernate

É um *framework* de mapeamento objeto/relacional para linguagem java. Ele associa os

dados tabulares de um banco de dados a um grafo de objetos definido pelo desenvolvedor. Com esta ferramenta o desenvolvedor não gasta muito tempo escrevendo muito código de acesso a banco de dados e *SQL*, trabalho que ele teria se não utilizasse esta ferramenta. Assim, as operações de *CRUD* (Criação, Atualização, Deleção, Cadastro) já são definidas, porém estão encapsuladas, ou seja, o desenvolvedor não precisa saber os detalhes de sua implementação, apenas utilizá-las.

Este *framework* não é indicado para sistemas em que toda lógica de regras de negócios se encontra no banco de dados e que tenha um modelo de objetos pobres. Mas no caso do Sistema de Gerenciamento de Discos, onde as operações das transações se resumem ao padrão CRUD e é rica em modelo de objetos, o uso do *hibernate* é essencial.

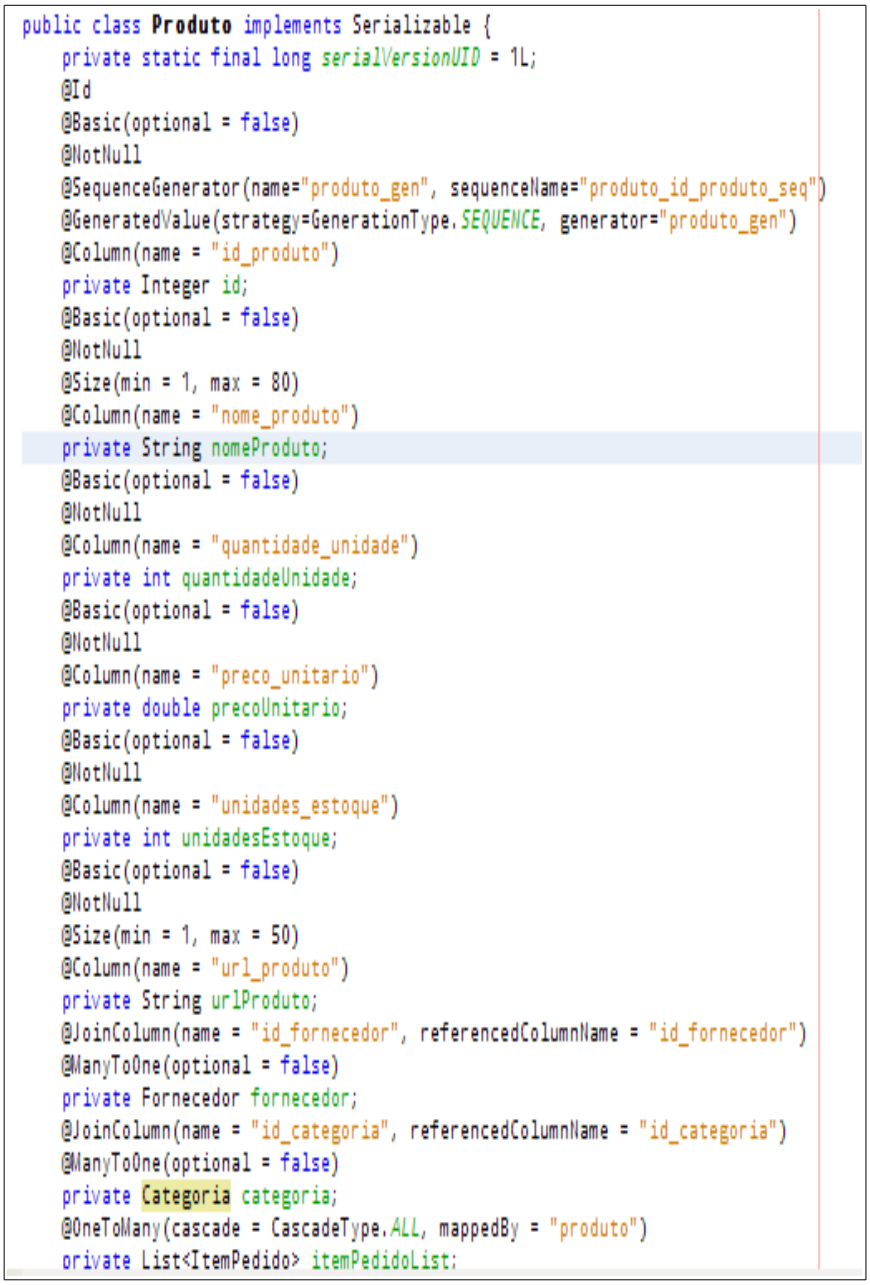
4.6.2.1 Mapeamento das classes para as tabelas

No *hibernate* existem dois tipos de mapeamento, via XML ou anotações. No sistema de gerenciamento de discos foi utilizado o mapeamento por anotação.

As anotações podem ser definidas como metadados que aparecem no código fonte e são ignorados pelo compilador. Qualquer símbolo em um código Java que comece com uma @ (arroba) é uma anotação. Este recurso foi introduzido na linguagem Java a partir da versão Java SE 5.0. Em outras palavras, as anotações marcam partes de objetos de forma que tenham algum significado especial.

O código 4.1 mostra essas anotações.

Código 4.1 – Classe produto com mapeamento do tipo anotações



```
public class Produto implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @SequenceGenerator(name="produto_gen", sequenceName="produto_id_produto_seq")
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="produto_gen")
    @Column(name = "id_produto")
    private Integer id;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 80)
    @Column(name = "nome_produto")
    private String nomeProduto;
    @Basic(optional = false)
    @NotNull
    @Column(name = "quantidade_unidade")
    private int quantidadeUnidade;
    @Basic(optional = false)
    @NotNull
    @Column(name = "preco_unitario")
    private double precoUnitario;
    @Basic(optional = false)
    @NotNull
    @Column(name = "unidades_estoque")
    private int unidadesEstoque;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 50)
    @Column(name = "url_produto")
    private String urlProduto;
    @JoinColumn(name = "id_fornecedor", referencedColumnName = "id_fornecedor")
    @ManyToOne(optional = false)
    private Fornecedor fornecedor;
    @JoinColumn(name = "id_categoria", referencedColumnName = "id_categoria")
    @ManyToOne(optional = false)
    private Categoria categoria;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "produto")
    private List<ItemPedido> itemPedidoList;
```

A tabela 4.8 mostrará cada uma dessas anotações junto a sua descrição:

Tabela 4.8 – Anotações utilizadas no mapeamento da entidade produto

ANOTAÇÃO	SIGNIFICADO
@Id	O campo é uma chave primária
@NotNull	O campo não pode ser nulo
@SequenceGenerator	Sequencia gerada para o campo que contem a chave primária. Há dois tipos de atributos: 1. name: nome dado para sequencia; 2. sequenceName: identificador da referencia no SGBD PostgreSQL.

@GeneratedValue	Define a forma em que os valores da chave primária será gerada. Possui dois atributos: 1. *strategy*: é utilizado para indicar o tipo de estratégia que será utilizado para gerar os dados da chave primária. Neste caso foi utilizado o valor *GenerationType.SEQUENCE* para indicar que o valor processado para chave primária é uma sequência. 2. *generator*: possui o nome da referência para gerar os valores daquela chave primária.
@Column	Atributo relaciona com a coluna referenciada no banco, nela podemos especificar os seguintes atributos: 1. *name*: indica qual o nome que está sendo utilizado lá no banco; 2. *nullable*: serve para determinar se o campo é obrigatório ou não. Caso possua o valor *true*, significa que aquele campo é obrigatório. Caso contrário não.
@Basic	Está relacionada com a obrigatoriedade ou não de um campo, ela possui um atributo chamado *optional*, caso seja setado com o valor false, significa que o campo é obrigatório, caso esteja setado com o valor *true*, significa que aquele campo não é obrigatório.
@OneToMany	Especifica o tipo de relacionamento que uma entidade possui em relação a outra. Significa que um objeto dessa entidade possui uma lista de objetos da outra. Nela podemos encontrar os seguintes atributos: 1. *cascade*: significa que se você fizer alguma operação com o objeto desta entidade, a outra relaciona a rela poderá ou não sofrer ou não determinada operação dependendo do valor que que é passado para este atributo, vejamos alguns dele: 1.1. *CascadeType.All*: executa todas operações que se processadas no objeto pai, será executada no objeto filho seja elas: *merge, save, update* ou *delete*. 1.2. *CascadeType.Delete*: executa a operação de *delete* nos objetos pais e filhos. 1.3. *CascadeType.Merge*: executa a operação de *merge* nos objetos pais e filhos. 1.4. *CascadeType.Save*: executa a operação de *save* nos objetos pais e filhos.
@ManyToOne	Especifica um tipo de relacionamento de muito para um, ou seja, a referida entidade que possui esse mapeamento possui a referencia de um objeto de uma outra entidade.
@JoinColumn	Especifica um junção que ocorre quando duas entidade está relacionada a outra pelo relacionamento muitos-para-um. Essa anotação podemos definir dois tipos de atributos: 1. *name*: nome dado a junção. 2. *referencedColumnName*: nome da coluna responsável por referenciar uma entidade com a outra.
@Size	Anotação utilizada pra validar o tamanho de uma *String*. É composta de dois atributos: 1. *min*: valida o tamanho mínimo que a *String* pode ter. 2. *max*: valida o tamanho máximo que a *String* pode ter.

4.6.2.2 Objetos transientes, persistentes e *detached*

Nas diversas aplicações existentes, sempre que for necessário propagar o estado de um objeto que está em memória para o banco de dados ou vice-versa, há a necessidade de que a aplicação interaja com uma camada de persistência. Isto é feito, invocando o gerenciador de persistência e as interfaces de consultas do *hibernate*. Quando interagindo com o mecanismo

de persistência, é necessário para a aplicação ter conhecimento sobre os estados do ciclo de vida da persistência.

Em aplicações orientadas a objetos, a persistência permite que um objeto continue a existir mesmo após a destruição do processo que o criou. Na verdade, o que continua a existir é seu estado, já que pode ser armazenado em disco e então, no futuro, ser recriado em um novo objeto. Em uma aplicação não há somente objetos persistentes, pode haver também objetos transientes. Objetos transientes são aqueles que possuem um ciclo de vida limitado ao tempo de vida do processo que o instanciou. Em relação às classes persistentes, nem todas as suas instâncias possuem necessariamente um estado persistente. Elas também podem ter um estado transiente ou *detached*.

O Hibernate define estes três tipos de estados: persistentes, transientes e detached. Objetos com esses estados são definidos como a seguir:

- Objetos Transientes: são objetos que suas instâncias não estão nem estiveram associados a algum contexto persistente. Eles são instanciados, utilizados e após a sua destruição não podem ser reconstruídos automaticamente;
- Objetos Persistentes: são objetos que suas instâncias estão associadas a um contexto persistente, ou seja, tem uma identidade de banco de dados.
- Objetos detached: são objetos que tiveram suas instâncias associadas a um contexto persistente, mas que por algum motivo deixaram de ser associadas, por exemplo, por fechamento de sessão, finalização de sessão. São objetos em um estado intermediário, nem são transientes nem persistentes.

O ciclo de vida de um objeto persistente pode ser resumido a partir da figura 4.7.

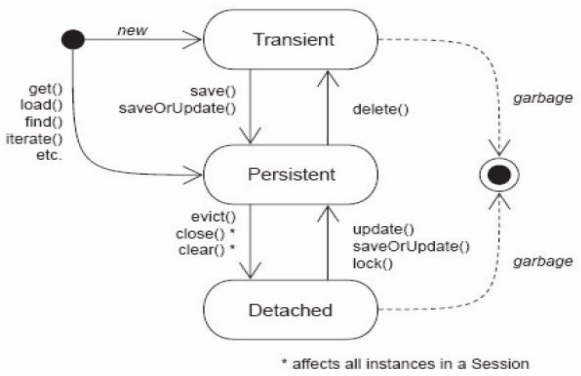


Figura 4.7 Estados do objeto

De acordo com a figura 4.7, inicialmente, o objeto pode ser criado e ter o estado

transiente ou persistente. Um objeto em estado transiente se torna persistente se for criado ou atualizado no banco de dados. Já um objeto em estado persistente, pode retornar ao estado transiente se for apagado do banco de dados.

Também pode passar ao estado *detached*, se, por exemplo, a sessão com o banco de dados por fechada. Um objeto no estado *detached* pode voltar ao estado persistente se, por exemplo, for atualizado no banco de dados. Tanto do estado *detached* quanto do estado transiente o objeto pode ser coletado para destruição.

4.6.2.3 Linguagem de consulta do *hibernate* (HQL)

O *Hibernate* vem com uma linguagem de consulta (HQL) que é muito parecida com o SQL. No entanto, comparado com o SQL o HQL é totalmente orientado à objetos, e compreende noções de herança, polimorfismo e associações.

As Consultas não diferenciam maiúscula de minúscula, exceto pelo nomes das classes e propriedades Java. Portanto, *SeLeCT* é o mesmo que *sELEct* que é o mesmo que SELECT, mas *br.ufma.discoteka.modelo.Produto* não é *br.ufma.discoteka.modelo.produto*.

A figura 4.2 mostra o exemplo de uma consulta HQL para buscar um produto por categoria.

Código 4.2 – Consulta HQL para busca de produto por categoria

```
public List<Produto> buscaPorCategoria (String nomeCategoria){

    Query q;
    List lista = null;
    if (nomeCategoria!=null){
        Transaction t = session.beginTransaction();
        String hql = "SELECT p FROM Produto p, Categoria c "
                + "WHERE p.categoria.id = c.id  AND "
                + "c.nomeCategoria = :nomeCategoria";

        q = session.createQuery(hql);
        q.setParameter("nomeCategoria", nomeCategoria);
        lista = q.list();
        t.commit();
    }else{
        lista = super.findAll();
    }
    return lista;
}
```

Analisando o código pode-se observar o seguinte:

– é passada uma *string* como parâmetro do nome da categoria pela qual se quer

buscar um determinado produto da *Loja Discoteka*.

- Cria-se uma variável genérica do tipo *List* que receberá o valor da consulta, ou seja, uma lista de produtos, caso encontre alguma ocorrência.
- A variável HQL do tipo *String*, contém as cláusulas que irão compor a busca e, nesta, é feita a junção de duas entidades – produto e categoria, para se fazer a filtragem da categoria a qual pertence aquele produto. Dentro da *String* temos o parâmetro "*:nomeCategoria*" onde tem o seu valor setado através do método "*setParameter"* do objeto *Query*.
- Depois de feita a consulta, a lista é retornada.
- Caso seja passado o valor *null*, será retornado todos os produtos cadastrados no banco.

4.6.3 CSS

CSS significa *Cascading Style Sheetes* (Folhas de Estilo em Cascata). É uma linguagem de estilo utilizada com o intuito de separar um documento html ou xhtml da sua formatação. Para isso cria-se um link do documento com o arquivo .css que contém os atributos de formatação dos componentes do documento. Isso evita que o desenvolvedor num mesmo documento ou documentos diferentes, reescrevam a mesma formatação para um determinado componente, fazendo com que a manutenção visual do sistema seja feita com mais facilidade.

No código 4.3 mostra como é feito a importação do arquivo .css para atribuir nossa página xhtml as propriedades de formatação presentes nesse arquivo.

Código 4.3 – Importação do arquivo *CSS*

```
<h:head>
    <title>DISKOTECA - Cadastro de Fornecedor</title>
    <link type="text/css" rel="stylesheet" href="css/estilo.css"/>
</h:head>
```

Para importar o arquivo css à página xhtml foi utilizada uma *tag* do *html* chamada link, que é responsável por fazer a ligação da página com o arquivo que contém as propriedades dos componentes utilizados. Para isso bastou-se acrescentar o atributo *href* que indica o caminho onde está presente o arquivo de estilo.

Para utilizar as propriedades deste arquivo é necessário que dentro de um componente

da nossa página, seja utilizado o atributo chamado *styleClass*, que terá como valor o nome dado dentro do arquivo com extensão .css. O código 4.4 mostra que é feito esse processo.

Código 4.4 – Atributo *styleClass* para importação de propriedade

```
<h:body styleClass="cor_fundo_padrao">
    <f:view>
        <h:form>
            <div>
            <ui:insert id="cabecalhoFornecedor" name="header">
                <ui:include src="template/cabecalho.xhtml"/>
            </ui:insert>
            </div>
            <fieldset>
                <legend>CADASTRO DE FORNECEDOR</legend>
                    <h:panelGrid>
                        <h:panelGrid>
                            <h:messages styleClass="mensagem_validacao"/>
                        </h:panelGrid>

                        <h:panelGrid columns="2">

                            <h:panelGrid>
                                <h:graphicImage url="imagens/fundo/fundo.png"/>
                            </h:panelGrid>
```

O código 4.5 mostra algumas propriedades aplicada a página *fornecedorForm.xhtml* da aplicação da *Loja Discoteka*.

Código 4.5 – Propriedades CSS dos componentes

```
.mensagem_validacao{
    color: red;
}

.cor_fundo_padrao{
    background-color: #e1e1e1;
}


fieldset{
    margin-left: 5%;
    margin-right: 5%;
}
```

A propriedade color define a cor da mensagem de validação que aparecerá na tela de cadastro de fornecedor, caso o usuário tenha dado como entrada, um dado inválido.

A propriedade *background-color* define a cor do fundo de todas as páginas utilizadas no sistema.

As propriedades *margin-left* e *margin-right* indicam respectivamente margem a esquerda e margem a direita do campo de texto utilizado para delimitar a área de cadastro do fornecedor.

A figura 4.8 da tela de cadastro de fornecedor do sistema *Loja Discoteka,* mostra como fica visualmente a página com a aplicação dessas propriedades. As mensagens em vermelho aparecem apenas no contexto em que o usuário do sistema não digita nada em um dos campos de entrada. As propriedades das mensagens de validação estão representadas no código 4.5 através da classe *mensagem_validacao*.

Figura 4.8 – Tela de cadastro fornecedor com propriedades do arquivo estilo.css

Se não fosse utilizado CSS em cada uma dessas páginas, e todas elas utilizassem o mesmo padrão e, algum dia fosse preciso trocar esses padrões de propriedades, o retrabalho seria maior, pois teria que mudar página por página essas propriedades sendo que, com o uso do CSS bastaria apenas fazer a mudança no arquivo que contém essas propriedades. Então pode-se dizer que o CSS é de suma importância para uma aplicação que se processa na rede da Internet, pois facilita a sua manutenção.

4.6.4 JSF 2.0 (*Java Server Faces* 2.0)

É uma tecnologia de desenvolvimento web que utiliza um modelo de interfaces gráficas baseado em eventos. Ela está associada a camada de visão do sistema *Loja Discoteka*, onde é feita a interação direta com o usuário. Esta tecnologia foi definida pelo JCP (*Java Community Process*), o que a torna um padrão de desenvolvimento e facilita o trabalho dos fornecedores de ferramentas ao criarem produtos que valorizem a produtividade no desenvolvimento de interfaces visuais (_____, Algaworks – softwares e treinamentos, 2010).

O sistema Loja Discoteka é baseado no padrão MVC, criado com intuito de separar a camada de visão das regras de negócio. Neste padrão temos três camadas essenciais utilizadas no sistema:

- *Model* (modelo): é formado pelas classes de domínio do sistema, onde se encontram os objetos de negócio. Elas são encontradas no pacote *br.ufma.diskoteka.modelo.*
- *View* (visão): é a camada de visão, ela contém a interface gráfica que fará a interação direta com o usuário, ou seja, as telas onde serão inseridas ou recuperados os dados de acordo com a requisição que será feita.
- *Controller* (controlador): é uma camada intermediária, que faz a comunicação entre a camada de visão e a camada que contém os objetos de negócio. Além disso ela também é responsável por processar as ações de negócios e direcionar o usuário para uma outra página.

Em JSF, o controle é feito através de um *servlet* chamado *Faces Servlet*, por vários manipuladores de ações e observadores de eventos. O *Faces Servlets* recebe as requisições dos usuários web, redireciona para o modelo e envia uma resposta. Os manipuladores de eventos são responsáveis por receber os dados da camada de visualização, acessar o modelo e devolver o resultado para o usuário através do *Faces Servlet* (_____, Algaworks – softwares e treinamentos, 2010).

A visualização é composta por uma hierarquia de componentes (*component tree)*, o que torna possível unir componentes para construir interfaces mais ricas e complexas.

A figura 4.9 mostra a arquitetura do JSF baseada no padrão MVC (_____, Algaworks – softwares e treinamentos, 2010).

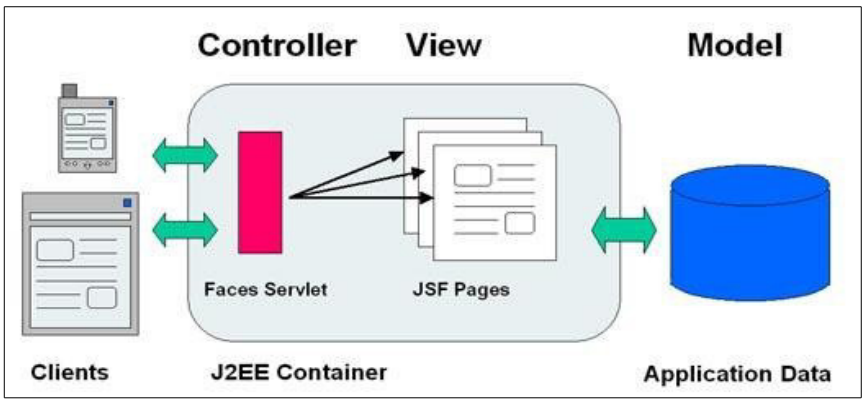


Figura 4.9 – Arquitetura do Java Server Faces baseada no MVC

4.6.4.1 *ManagedBeans*

O *ManagedBeans* é formado por um conjunto de classes que se comunicam com as páginas que constituem a interface gráfica do sistema. É uma forma de separar as ações que regem as regras de negócio do layout da página.

O código 4.6 mostra um exemple de um *ManagedBeans* chamado *FornecedorController*, aqui foi ocultado os métodos *getters* e *setters*. Ele é usado com o objetivo de acessar os objetos de negócios e processar ações que compunham as regras de negócio.

Código 4.6 – Classe *ManagedBean FornecedorController*

```
@SessionScoped
@ManagedBean
public class FornecedorController {
    private DataModel dadosBusca;
    private Fornecedor fornecedor;
    private boolean cadastra;
    private boolean edita;

    public FornecedorController(){

    } (...)
```

No código podemos observar a anotação *@SessionScoped*, assim que uma página que contém este *bean* é renderizada, ela terá o estado daquele objeto durante toda a sessão em que o sistema está sendo processado. A anotação *@ManagedBean* é utilizada para que a página que contém o código jsf possa ter acesso aos atributos e aos métodos do *ManagedBean*. Para acessar essa classe utiliza-se um delimitador assim:

#{fornecedorController.nomeAtributo/nomeMétodo}

onde *fornecedorController* é o mesmo nome da classe só que com as iniciais minúsculas que é gerado por padrão caso não se dê um outro nome para esse *controller* e depois vem seguido pelo nome do atributo, considerando que este atributo terá que ter um método *getter* para acessá-lo, caso isso não acontece, será gerado uma exceção. Caso se queira chamar uma ação, em vez do atributo, coloca-se o nome do método.

4.6.4.2 Hierarquia de Componentes

Cada tag jsf possui uma classe java que a representa, chamada *tag handler*, executada assim que a página é processada. Essas classes trabalham em conjunto e constroem uma hierarquia de componentes na memória do servidor, seguindo o que foi programado no arquivo JSF.

A hierarquia de componentes representa os elementos da interface visual presentes na página. Esses componentes são responsáveis por gerenciar seus estados e comportamentos. São responsáveis por gerenciar os seus estados e os seus comportamentos.

A figura 4.10 representa um exemplo de hierarquia de componentes JSF.

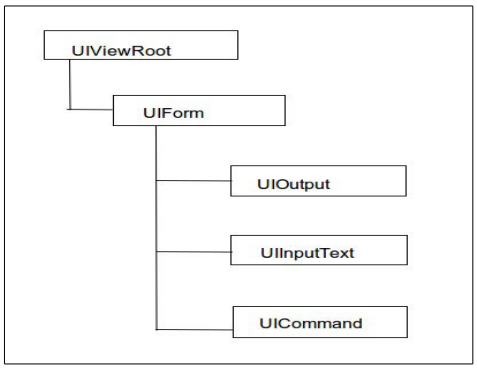


Figura 4.10 – Hierarquia de componentes JSF

O componente *UIViewRoot* representa a raiz dos componentes que são gerenciados pelo JSF, que é especificado através da tag <*f:view*>. Caso o desenvolvedor queira utilizar outros componentes JSF, ele precisa inserir outros componentes entre essa tag, o *UIForm* representa o formulário (<*h:form*>), o *UIInput* representa o label de escrita de textos e, enfim o UIInput representa os campos de entrada de texto.

Durante o processamento do arquivo JSP, uma página HTML é renderizada e enviada ao *browser*. Os componentes JSF são convertidos para código HTML.

4.6.4.3 Recursos JSF 2.0

O *framework* do JSF 2.0 oferece diversos recursos para o desenvolvimento da camada de visão, entre eles temos:

- **conversão de dados:** no jsf existe diversos conversores, pois quando um dado é capturado da tela, o servidor o recebe como um texto, que deve ser convertido em

algum tipo de dado específicos. Por exemplo, se quisermos entrar com um tipo data para fazer cadastro da data de nascimento de uma pessoa, deverá ser usado uma tag do *JSF* 2.0 tipo "*f:convertDateTime*", assim o dado será convertido para o tipo a que ele está associado na classe de domínio.

- **Validação:** ela deve ser feita antes que os dados sejam comitados no banco, ela tem a ver com a obrigatoriedade ou não do campo, se o campo aceita somente números ou letras. Ela pode ser feita de 2 formas:
  - no camada de visão;
  - ou na camada de domínio.
- **Renderizadores**: o JSF gera código HTML por padrão, porém o framework suporta a inclusão de outros renderizadores plugáveis que podem produzir códigos em outras linguagens para visualizadores de diferentes tipos, como celulares, Telnet, Flash, etc.

4.6.4.4 Componentes

A seguir será mostrado alguns componentes utilizados no sistema *Loja Discoteka,* tanto jsf 2.0 quanto html, assim como sua descrição. Para isso toma-se como base a página *produtoForm.xhtml,* tela em que é feita o cadastro de produtos.

No código 4.7 tem-se a *tag* h:head, ela tem como objetivo definir o cabeçalho para a página que será exibida no *browser*. Dentro da árvore de componentes temos a tag title, ela exibirá o título da página no *browser*. A tag link terá o caminho do arquivo de configuração das propriedades CSS que conterá a página.

Código 4.7 – Tag HTML *h:head*

```
<h:head>
    <title>DISKOTECA - Cadastro de Produto</title>
    <link type="text/css" rel="stylesheet" href="css/estilo.css"/>
</h:head>
```

O código 4.8 encontram-se as seguintes tags:

– *f:view*: é um container para todos os seus componentes *Java Server Faces* usadas em uma página. Ou seja, se forem utilizadas tags exclusivas do JSF 2.0 dentro da página, tem-se que delimitar essas tags dentro da *view*.

- *h:form*: Esta marca representa a classe *javax.faces.component.UIForm* da árvore de componentes subjacente. Um componente UIForm é uma forma de entrada, com componentes filhos que representam os dados que são apresentados, quer para o utilizador ou submetidos com o formulário. A tag de formulário inclui todos os controles que exibem ou coletar dados do usuário. A tag de formulário também pode incluir marcação HTML para o layout dos controles na página. A tag de formulário próprio não executa qualquer layout; seu objetivo é coletar dados e declarar atributos que podem ser usados por outros componentes na página form. Pode incluir tags de forma múltipla, mas apenas os valores do formulário que o usuário envia serão incluídos no *postback*.
- *ui.insert*: é um componente do facelets que no sistema foi utilizado para reaproveitamento de código. Foi utilizado em todas as páginas para definir template de cabeçalho e rodapé também. No código tem-se como componente da árvore a tag ui:include que define o caminho onde se encontra a figura do cabeçalho através do atributo *src*.
- *fieldset*: é um delimitador de campo de texto. No sistema ele foi utilizado para delimitar a área de componentes de entrada para cadastro de um produto.
- *legend*: é a legenda para o delimitador de campo de texto.
- *h:panelGrid*: é um gerenciador de layout que funciona para organizar os componentes dentro de uma página de forma tabular.
- *h:messages*: serve para exibir as mensagens de validação na tela. O atributo *styleClass* importa a classe onde se encontram as propriedades dentro do arquivo estilo.css.
- *h:graphicImage*: responsável por importar uma imagem dentro da página.

Código 4.8 – *Tags f:view, h:form, ui:insert, fieldset, h:panelGrid, h:messages e h:graficImage*

```
<h:body styleClass="cor_fundo_padrao">
    <f:view>
        <h:form>
            <div>
            <ui:insert id="cabecalhoProduto" name="header">
                <ui:include src="template/cabecalho.xhtml"/>
            </ui:insert>
            </div>
            <fieldset>
                <legend>CADASTRO DE PRODUTO</legend>
                    <h:panelGrid>
                        <h:panelGrid>
                            <h:messages styleClass="mensagem_validacao"/>
                        </h:panelGrid>

                        <h:panelGrid columns="2">

                            <h:panelGrid>
                                <h:graphicImage url="imagens/fundo/fundo.png"/>
                            </h:panelGrid>
```

O código 4.9 pode-se visualizar a *tag* para combobox chamada *h:selectOneMenu*. Ela contém uma lista de fornecedores, no qual apenas um será selecionado para que se possa cadastrar um produto. O atributo converter é um conversor utilizado para converter a *String* em um objeto fornecedor, pois como já foi falado, em jsf o servidor captura uma *String* e precisa-se de um conversor para transformar aquele dado em outro tipo. O atributo *required* com o valor *true* indica que aquela entrada para o campo fornecedor é obrigatória, caso o usuário não selecionar nenhum fornecedor na tela, será disparada a mensagem redigida no atributo *requiredMessage*.

Código 4.9 – Tag para *combobox*

```
<h:selectOneMenu value="#{produtoController.produto.fornecedor}"
                 styleClass="label_cadastro"
                 converter="simpleConverter"
                 required="true" requiredMessage="Campo fornecedor é obrigatório">
    <f:selectItem value="#{null}" itemLabel="------- selecione um fornecedor -------"/>
    <f:selectItems var="forn" value="#{produtoController.listaFornecedores}"
                   itemValue="#{forn}"
                   itemLabel="#{forn.nomeEmpresa}"/>
</h:selectOneMenu>
```

Em JSF 2.0 outro componente muito importante são os botões de ações, responsáveis por, após serem clicados, setarem os valores dos seus atributos e chamar uma ação do *bean*.

O código 4.10 mostra como esses botões são utilizados e suas principais *tags*. O

atributo *action* tem como função chamar um método de ação do bean, mas antes de ser chamado, os valores de entradas são armazenados através de métodos *setters*. O *image* indica o caminho da imagem que será inserida no botão. *Title* é a legenda para aquele botão. Enfim, o *rendered* serve para renderizar ou não um botão de acordo com o valor da variável a a qual foi associada.

Código 4.10 – Botão de ação

```
<h:commandButton action="#{produtoController.adicionaProduto()}"
                 image="imagens/botoes/processar.png"
                 title="Salva Produto"
                 rendered="#{produtoController.cadastra}"/>
```

A figura 4.11 mostra como ficou a página de cadastramento de produto após o uso dos componentes JSF 2.0 e a aplicação de folhas de estilo CSS.

Figura 4.11 – Tela de cadastramento de produto com CSS

4.6.5 *Primefaces* 3.0

Trata-se de um complemento para o *framework* JSF 2.0, só que com recurso visual melhorado. Também é uma tecnologia voltada para linguagem java.

No sistema *Loja Discoteka*, foram utilizados alguns de seus componentes. Será visto

com detalhamento cada um deles.

O código 4.11 mostra alguns componentes que foram utilizados no sistema *Loja Discoteka*, veremos com detalhes cada um deles.

- *p:dataTabe:* é um componente utilizado para construir tabelas, tanto dinâmicas quanto estáticas. No sistema foi utilizado tabelas dinâmicas com o objetivo de fazer buscas eficientes através de substrings. Neste componente pode-se encontrar os seguintes atributos:

- var: é o atributo que contém a variável que vai preencher as linhas da tabela;
- *value*: consiste ao atributo que devolverá uma lista de fornecedores;
- widgetVar: conterá o valor da variável para o filtro das colunas;
- *emptyMessage*: contém a mensagem que será exibida caso não seja encontrado algum fornecedora na busca feita na própria tabela;
- *rows*: contém o valor do número de linhas que terá a tabela;
- *paginator*: caso possua o valor true, a tabela terá paginação, onde em cada página haverá tantas linhas de acordo com o valor passado para o atributo rows.

– *p:outputPanel*: corresponde a um painel delimitador, é constituído dos seguintes atributos:

– *p:inputText*: é o campo de entrada utilizado para entrada. É formado dos seguintes atributos:

- *id*: identificador utilizado caso queira se fazer alguma referência a este componente na página JSF ou em alguma classe java;
- *onkeyup*: tratador de eventos, ou seja, a partir do momento em que digitado um valor no campo, o framework é responsável por buscar um fornecedor por qualquer um dos atributos correspondente a este objeto através da substring do campo.

– p:column: corresponde a coluna da tabela. Seus atributos são:

- *filterBy*: corresponde ao valor do campo pelo qual será feito a busca;
- *headerText*: corresponde ao texto título que ficará na parte superior da tabela;
- *footerText*: corresponde ao texto título que ficará na parte inferior da tabela;
- *filterMatchMode*: forma de como será feita a busca. Existem três formas:
  - *startsWith*: busca por um valor que começa com aquela substring;
  - *endsWith*: busca por um valor que termina com aquela substring;
  - *contains*: busca por um valor que contenha aquela substring. Foi o utilizado no

sistema.

Pôde-se observar que, apesar das buscas terem sido dinâmicas, não foi necessário o uso de *javascript* no código, pois o componente *dataTable* do primefaces já contém o *JQuery* embutido nele.

Código 4.11 – Componentes primefaces 2.0

```
<p:dataTable var="fornecedor" value="#{fornecedorController.dadosBusca}"
             widgetVar="fornecedoresTable"
             emptyMessage="Fornecedor não encontrado!"
             rows="5" paginator="true"
             styleClass="margem_tabela_lista_dados">
    <f:facet name="header">
        <p:outputPanel>
            <h:outputText value="BUSCA GERAL:" styleClass="texto_busca_geral"/>
            <p:inputText id="globalFilter" onkeyup="fornecedoresTable.filter()"
                         styleClass="texto_ent_busca_geral"/>
        </p:outputPanel>
    </f:facet>

     <p:column filterBy="#{fornecedor.cnpj}" headerText="CNPJ"
               footerText="BUSCA POR CNPJ" filterMatchMode="contains"
               styleClass="texto_coluna_tabela">
        <h:outputText value="#{fornecedor.cnpj}"
                      styleClass="texto_saida_linha_tabela"/>
    </p:column>
```

A figura 4.12 mostra a visão da tela de listagem de fornecedor com a utilização dos componentes *primefaces*.

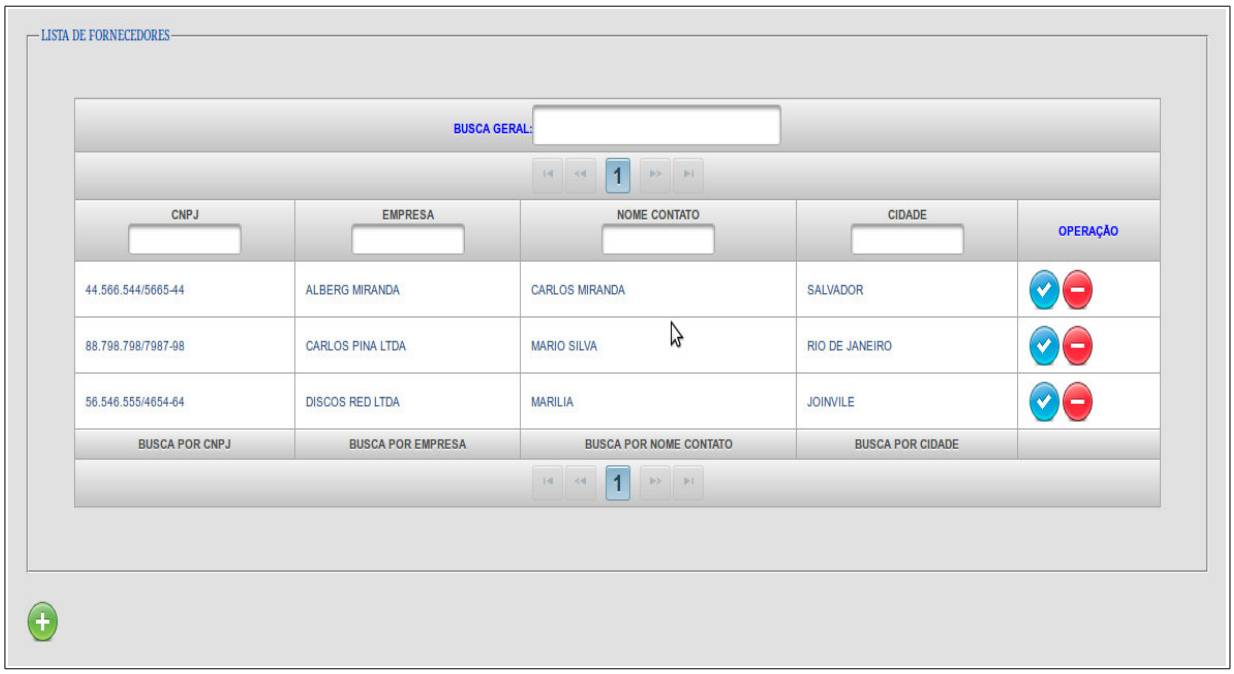


Figura 4.12 – Componente *dataTable* do *primefaces*

4.6.6 *Ireport*

É muito comum é várias aplicações fazer a impressão de dados de uma referida empresa com o objetivo de controlar as transações feitas na mesma e fazer consultas. O *iReport* é um software do tipo *OPEN SOURCE*, utilizado para criar relatórios dos mais simples aos mais complexos no formato da biblioteca *JasperReports*. Ele é escrito em 100% java, ou seja, o que significa que é multiplataforma, executa em qualquer sistema operacional. O desenvolvimento do relatório é feito em linguagem XML, mas se o usuário é iniciante, este programa gera automaticamente o arquivo XML por meio de suas ferramentas. (Gonçalves, Edson; 2008)

No sistema *Loja Discoteka* este software foi utilizado para imprimir relatórios de fornecedores, produtos e clientes.

A figura 4.13 demonstra visualmente a IDE onde é feita os relatórios do sistema, será relatado a função de cada uma das regiões que corresponde estes relatórios:

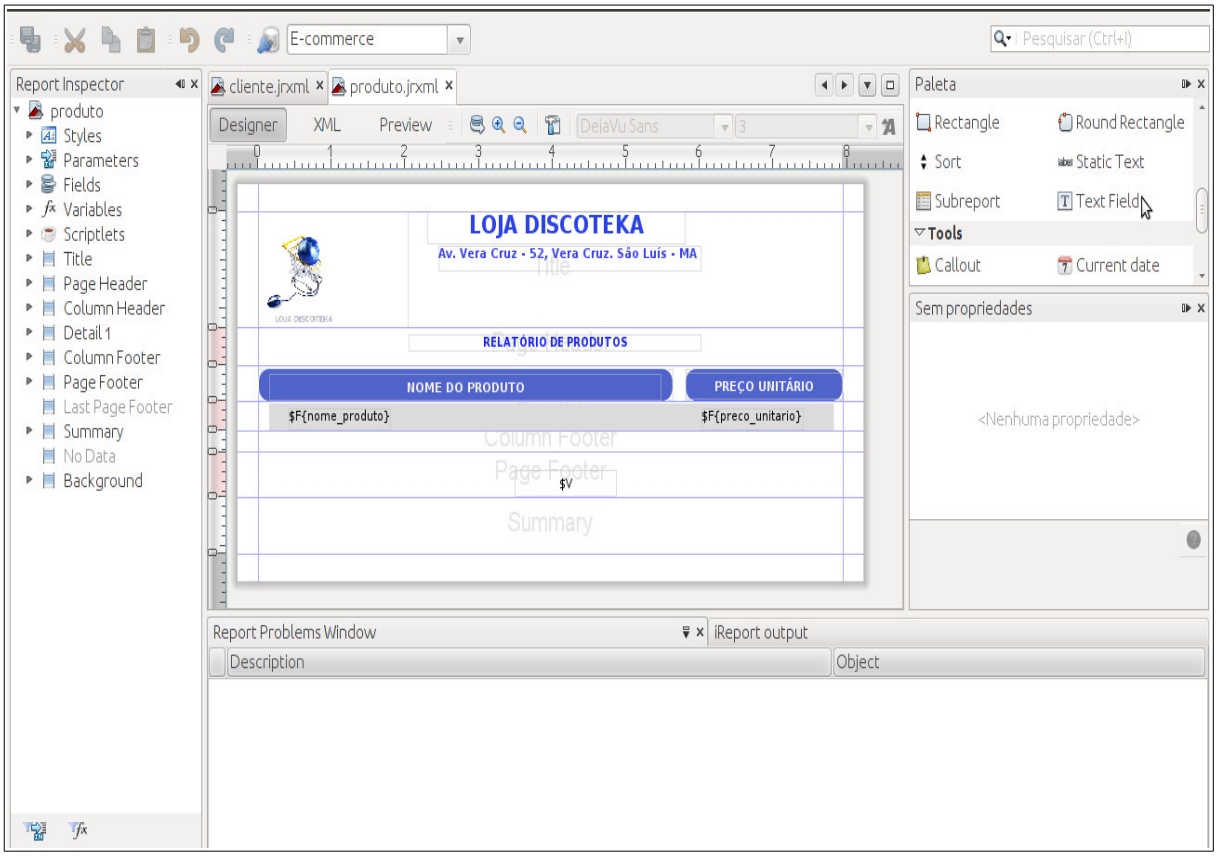


Figura 4.13 – IDE do Ireport para construção de relatório

- *title*: corresponde a região onde é inserida o timbre do relatório. No sistema o timbre é o nome da loja virtual junto ao seu endereço;
- *page header*: corresponde ao título que será dado ao relatório;
- *column header*: corresponde ao nome dado as colunas da tabela que conterá os dados do banco;
- *details*: contêm os campos dos dados provenientes do banco;
- *column footer*: região onde é inserida um título na parte inferior da tabela para cada coluna;
- *page footer*: corresponde ao rodapé do relatório que no sistema foi utilizado para inserir o número da página.

A IDE possui três tipos de visões relacionados desenvolvimento do relatório:

- designer: visão em que o relatório é construído com as ferramentas do Ireport;
- XML: visão onde o relatório é desenvolvido através da linguagem XML, mas para isso o desenvolvedor terá que ter conhecimentos mais profundos em relação a linguagem;
- *preview*: nesta, será visualizado o documento no formato PDF;

Também podemos encontrar no software:

- paleta: contém os componentes que são utilizados no relatórios. No relatório do sistema *Loja Discoteka* utilizou se: *static text*, para inserir um texto na tela; *round rectangle*; para inserir um retângulo com cantos arredondados para servir de título para cada coluna da tabela; *image*, para inserir uma figura;
- lista de propriedades: características atribuídas aos componentes presentes na paleta;
- report datasources: região onde é definida a conexão com o SGBD utilizado.

A figura 4.14 mostra a imagem de um relatório de produtos do sistema *Loja Discoteka* gerado pelo IReport.

Figura 4.14 – Relatório de produtos gerado pelo IReport

4.6.7 *Spring Security* 3.0

Quando se projeta uma aplicação na rede de Internet em que informações são trocadas a todo momento, principalmente informações do tipo pessoal, logo se pensa em como esses dados devem ser mantidos seguros, para que determinadas informações não possam chegar nas mãos de terceiros. Então faz-se necessário que o sistema tenha uma tecnologia que seja responsável por essa segurança.

No sistema *Loja Discoteka*, utilizou-se o *framework Spring Security* 3.0, que implementa toda essa parte de segurança sem que o desenvolvedor se preocupe com os detalhes de implementação. Toda essa configuração é feita através de um arquivo XML chamado *ApplicationContext*, onde o desenvolvedor precisa apenas conhecer as suas *tags* e atribuir valores a ela.

O código 4.12 demonstra como essa configuração é feita. O atributo *access-dinied-page* da *tag* http, desvia para uma página de negação, caso o usuário tente entrar no página que não é do seu papel no sistema. Os atributos *pattern* e *access* da tag *intercept-url* indicam respectivamente a página e os tipos de usuários que podem ter acesso a ela. *Autentication-failure-url* é responsável por atribuir um valor true para uma variável erro caso não tenha

conseguido fazer a autenticação do usuário no banco, isso acontece quando o usuário tenta logar no sistema e o nome de usuário e/ou senha estão incorretos. *Default-target-url* representa o atributo que contém o valor da página padrão aonde o usuário será direcionado depois de ter feito o login. O *login-page* representa a página de login, ou seja, caso usuário tente através da URL entrar direto em uma das páginas do sistema, este atributo é responsável. Na *tag authentication-manager* compreende a parte do código onde é feita uma consulta no banco para saber se o usuário está cadastrado ou não no banco de dados, ou seja se a senha de usuário está correta ou não. Caso ele esteja cadastro no sistema, há uma classe chamada *UsuarioController*, como mostra o código 4.13, responsável por autenticar esse usuário no sistema e guardar os seus dados durante toda a sessão, até que o usuário faça *logout*. Tem-se também uma *tag* muito importante utilizado para a criptografia de senha de usuário, chamado *password-encoder,* onde há um atributo *hash* que contém o valor do algoritmo de criptografia a ser utilizado no sistema, no caso da *Loja Discoteka*, utilizou-se o algoritmo md5.

Código 4.12 – Arquivo XML de configuração de segurança do sistema

```
    <http auto-config="true" use-expressions="true" access-denied-page="/faces/negado.xhtml">
        <intercept-url pattern="/faces/index.xhtml" access="hasAnyRole('ROLE_ADMIN', 'ROLE_USUARIO')"/>
        <intercept-url pattern="/faces/categoriaForm.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <intercept-url pattern="/faces/categoriaList.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <intercept-url pattern="/faces/produtoForm.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <intercept-url pattern="/faces/produtoList.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <intercept-url pattern="/faces/fornecedorForm.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <intercept-url pattern="/faces/fornecedorList.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <intercept-url pattern="/faces/relatorio.xhtml" access="hasRole('ROLE_ADMIN')"/>
        <form-login login-page="/faces/login.xhtml" authentication-failure-url="/faces/login.xhtml?erro=true"
                    default-target-url="/faces/index.xhtml"/>
    </http>
    <authentication-manager>
        <authentication-provider>
            <password-encoder hash="md5"/>
            <jdbc-user-service data-source-ref="dataSource"
              users-by-username-query="SELECT username, password, 'true' as enable FROM usuario WHERE username=?"
              authorities-by-username-query="SELECT username, autorizacao FROM usuario WHERE username=?"/>
        </authentication-provider>
    </authentication-manager>
    <b:bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
        <b:property name="url" value="jdbc:postgresql://localhost:5432/comercioeletronico" />
        <b:property name="driverClassName" value="org.postgresql.Driver" />
        <b:property name="username" value="postgres" />
        <b:property name="password" value="w47312191j" />
    </b:bean>
</b:beans>
```

Código 4.13 – Classe responsável por autenticar usuário no sistema

```
@ManagedBean
@SessionScoped
public class UsuarioController implements Serializable {

    private Usuario usuario = null;

    public UsuarioController() {

        usuario = new Usuario();
        SecurityContext context = SecurityContextHolder.getContext();
        if (context instanceof SecurityContext){
            Authentication authentication = context.getAuthentication();
            if (authentication instanceof Authentication){
                User user =  ((User)authentication.getPrincipal());
                usuario = new UsuarioDAO().buscaPorUsername(user.getUsername());
            }
        }
    }

    (...) // Metodos getters and setters
```

A figura 4.15 mostra a tela de login que o usuário deverá logar para entrar no sistema. Pode-se observar que nela possui dois campos: o de usuário, onde ele deverá colocar o nome de usuário em que se cadastrou e a sua senha. Caso o usuário digite os dados incorretamente, o sistema emitirá uma mensagem de erro. Se o usuário não tiver uma conta no banco, poderá se cadastrar entrando clicando no link *cadastre-se aqui*.

Figura 4.15 – Tela de *login* do sistema *Loja Discoteka*

Para sair do sistema basta utilizar um *link* do próprio *spring security*, ele próprio é responsável por encerrar a sessão do usuário. Na URL do link basta colocar o seguinte valor *#{facesContext.externalContext.requestContextPath}/j_spring_security_logout"*.

A figura 4.16 ilustra a página que é renderizada, quando o usuário que não tem o papel para ela, tenta acessá-la. Esta renderização é configurada no arquivo que contém as regras de segurança – o *ApplicationContext.xml*. Neste exemplo, o usuário com o papel "ROLE_USUARIO" tentou acessar uma página *categoriaList que* contém a lista de categorias de produtos. Como esta página só pode ser acessada pelo usuário que tenha o papel "ROLE_ADMIN", é renderizada esta página de restrição de acesso.

O *spring security* 3.0, assim como os outros *frameworks* contribuíram muito para o desenvolvimento do sistema, diminuindo também o tempo que se levou para implementá-lo.

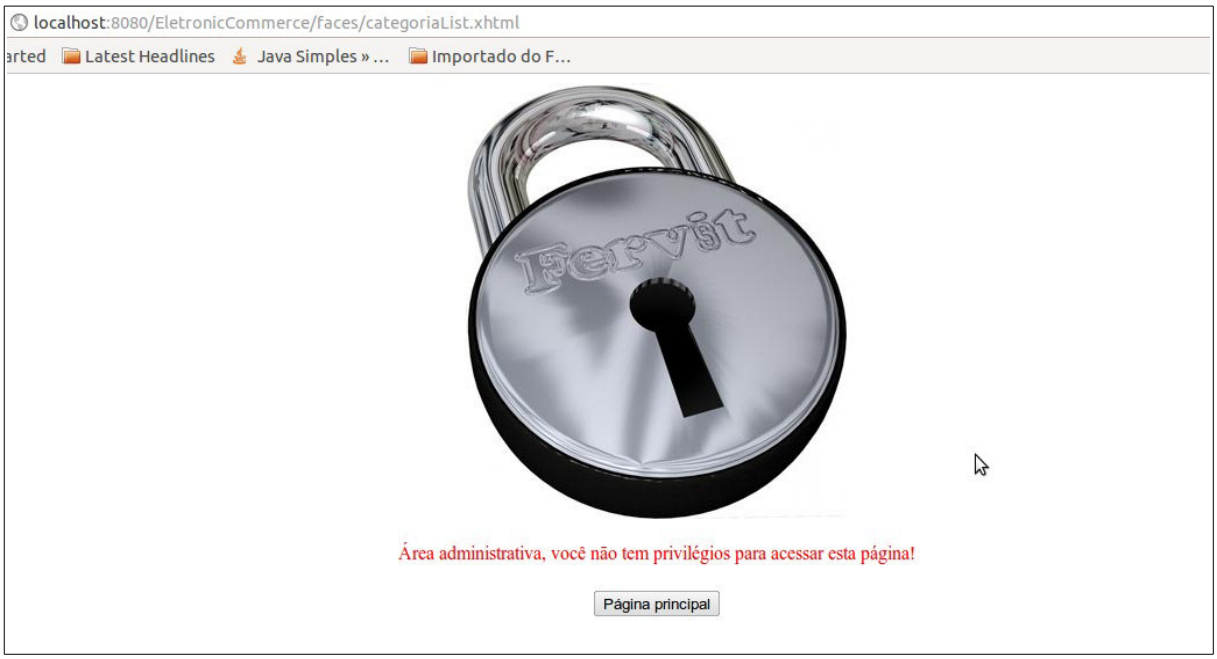


Figura 4.16 – Página renderizada quando o usuário não possui um papel para acessá-la

Pode-se dizer que o *framework spring security 3.0* é uma tecnologia muito poderosa e de fácil implementação, uma tecnologia indispensável para o desenvolvimento de sistemas da rede de *Internet*.

5 CONCLUSÃO

O uso da tecnologia de segurança da informação é muito importante quando se produz um sistema de comércio virtual eletrônico. Pois além de garantir que os dados pessoais que trafegam na rede de *Internet* não sejam utilizados por usuários não autorizados, também facilita a implementação deste sistema, acelerando o processo de desenvolvimento. Para isso foi utilizado o *framework spring security 3.0* que atribui papéis a determinado usuário do sistema e permite ou não que o mesmo acesse determinada página dependendo do papel que a ele foi concedido.

Um planejamento do sistema feito através de uma documentação de requisitos levantados, facilita muito no tempo de implementação e diminui a quantidade de erros presentes neste sistema. Utilizou-se uma ferramenta de modelagem *astah community* que possibilitou o desenvolvimento gráfico de casos de uso para melhor entendimento das regras de negócios que circundam o sistema.

Para estruturar o diagrama de classes, o *astah community* foi uma ferramenta muito importante para processar a construção dessas classes e construir seus relacionamentos.

O uso de tecnologias utilizadas no âmbito de implementação de um sistema de comércio eletrônico, utilizou-se um conjunto de bibliotecas já implementadas, também diminuindo o custo em relação ao tempo de desenvolvimento, ou seja, com código encapsulado, o desenvolvedor do sistema não precisa conhecer os detalhes de sua implementação. Utilizou-se também, o *framework hibernate,* que possibilita as operações CRUD genéricas sem que se precise utilizar *querys* para realizar tais operações. Para interface gráfica utilizou-se o JSF 2.0 e *primefaces* 3.0 que permite estruturar as páginas com uma interface mais rica e dinâmica do que se fosse feita somente com *tags* HTML's.

No início do desenvolvimento do sistema houve muita dificuldade em desenvolvê-lo devido a falta de conhecimento que se teve em relação as tecnologias utilizadas. Mas a partir do momento em que se foi tomando conhecimento dessas tecnologias, o desenvolvimento ficou facilitado, e chegou-se a meta esperada, que foi o término do sistema.

Em suma, o planejamento rigoroso e uso de tecnologias foram essenciais para o desenvolvimento do sistema *Loja Discoteka* utilizado para o gerenciamento de venda de discos.

5.1 Trabalhos futuros

Ultimamente foi desenvolvido uma linguagem de formatação para rede de *Internet* denominada HTML5 que possui recursos mais amplos do que as linguagens de formatação anteriores.

Uma sugestão é desenvolver um site de comércio eletrônico com HTML5 e com associação com uma entidade bancária, para que se possa vender produtos novos ou usados, sendo que estes produtos serão vendidos em escala menor e com muitas limitações em relação as grandes redes comerciais eletrônicas, já que será controlada individualmente.

REFERÊNCIAS

FELIPINI, Dailton. Artigo: O comércio eletrônico. Disponível em <http://www.e-commerce.org.br/>.

____, **Presidência da República**, Lei N.º 8.666, de 21 de junho de 1993.

LYNCH, Daniel C.; LUNDQUIST, Leslie. **Dinheiro digital: o comércio na internet**. Rio de Janeiro: Campus: 1996.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Prentice Hall, 2011.

Pressman, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 7ª Edição.

BERNSTEIN, Terry; BHIMANI, Anish B.; SIEGEL, Carol A.; SCHULTZ, Eugene; SIEGEL. **Segurança na Internet**. Rio de Janeiro: Campus, 1997.

GEARY, David; HORSTMANN, Cay. **Core JavaServerFaces**.

_____. Algaworks – softwares e treinamentos. **Desenvolvimento Web com JavaServer Faces.** 2ª Edição: 2010. Disponível em <http://www.algaworks.com/treinamentos/apostilas/>.

SAAB, Felipe**. Blog: Deixando sua aplicação Web segura.** Disponível em: <http:www.javasimples.com.br/spring-2/spring-security-3-deixando-sua-aplicacao-web-segura>.

____. **Primefaces: Ultimate JSF Componente Suite.** Disponível em: <http://www.primefaces.org>

____. **Documentação: The PostgreSQL Global Development Group**. Disponível em: <http://www.postgresql.org/docs/>.