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

VALDEMIR SILVEIRA

SISTEMA PARA SUPERMERCADOS MODULARIZADO DESENVOLVIDO EM JAVA

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2014

VALDEMIR SILVEIRA

SISTEMA PARA SUPERMERCADOS MODULARIZADO DESENVOLVIDO EM JAVA

Trabalho de conclusão de curso apresentado como requisito para a disciplina de TCC, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco.

Orientador: Prof. Msc. Robison Brito

PATO BRANCO
2014

ATA Nº: **241**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **VALDEMIR SILVEIRA.**

Às 16:10 hrs do dia 28 de fevereiro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Robison Cris Brito (Orientador), Fábio Favarim (Convidado) e Danilo Giacobo (Convidado), para avaliar o Trabalho de Diplomação do aluno Valdemir Silveira, matrícula 1066951, sob o título **Sistema para Gestão de Supermercados**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 16:40 hrs foi encerrada a sessão.

Prof. Robison Cris Brito, M.Sc.
Orientador

Prof. Fábio Favarim, Dr.
Convidado

Prof. Danilo Giacobo, M.Sc.
Convidado

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

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

AGRADECIMENTOS

Agradeço a Deus, por estar à frente de tudo.
À minha esposa Tânia pelo incentivo e paciência.
À professora Beatriz pelos ensinamentos fundamentais da programação.
Ao professor Robison que me ensinou o básico e o avançado do desenvolvimento Java, que possibilitou a construção desse projeto, pela paciência e orientação para esse trabalho de conclusão de curso.

RESUMO

SILVEIRA, Valdemir. **Sistema para supermercados modularizado desenvolvido em Java**. 2014. 66 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2014.

De forma geral, os sistemas informatizados para gerenciamento de supermercados podem ser basicamente divididos em frente de caixa e retaguarda. A frente de caixa faz a interface com o cliente, o que normalmente ocorre por meio de uma atendente de caixa. A retaguarda abrange o controle de estoque, financeiro e todas as funcionalidades gerenciais e administrativas, incluindo cadastro de produtos, contas a pagar e receber. Mesmo que o supermercado seja de pequeno porte são muitas as tabelas, os campos e os relacionamentos a serem controlados. Neste trabalho é registrado o desenvolvimento de um sistema para gerenciamento de supermercado e a estratégia de dividi-lo em partes para melhor gerenciar um *software* tão complexo. Os módulos possibilitam que o software seja modelado e implementado de forma customizada, reduzindo a complexidade do sistema, e facilitando o gerenciamento de erros e o atendimento de requisitos do usuário. Neste trabalho é apresentado uma visão geral do aplicativo e da análise utilizada para o seu desenvolvimento. Para o desenvolvimento foi utilizada a linguagem Java para *web* (sistema retaguarda) e *desktop* (frente de caixa) agregada por *frameworks* e bibliotecas.

Palavras-chave: Java para *web*. Aplicações *web*. Sistema para gerenciamento de supermercado.

ABSTRACT

Silveira, Valdemir. **System for supermarkets modularized in Java**. 2014. 66 f. Monograph (Work Completion of course) - Degree in Technology Analysis and Systems Development, Federal Technological University of Parana, Campus Pato Branco. Pato Branco, 2014.

Overall, the computerized systems for managing supermarkets can be basically divided into front and rear box. The front of box interfaces with the customer, which usually occurs by a box attendant. The rear covers inventory control, financial and all management and administrative functions, including product registration, account payable and receivable. Even the supermarket is small there are many tables, fields and relationships to be controlled. This work is recorded the development of a system for managing groceries and divide it into parts to better manage such a complex software strategy. The modules allow the system to be modeled and implemented customized, reducing system complexity and facilitating error management and meeting user requirements. In this paper an overview of the system and the analysis used in its development is presented. For the Java development language for web desktop system (rear) and (cashiers) aggregated by frameworks and libraries was used.

Keywords: Java web, web Applications, System Management Supermarket.

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWT	*Abstract Window Toolkit*
CGI	*Common Gateway Interface*
CNPJ	Cadastro Nacional de Pessoas Jurídicas
CPF	Cadastro de Pessoas Físicas
CRUD	*Criate, Read, Update, Delete*
CSS	*Cascading Style Sheet*
DANFE	*Documento Auxiliar de Nota Fiscal Eletrônica*
DBMS	*Data Base Management System*
DDL	*Data Definition Language*
DOM	*Document Object Model*
EJB	*Enterprise Java Beans*
FTP	*File Transfer Protocol*
GPL	*General Public License*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IIS	*Internet Information Services*
JDBC	*Java Database Connectivity*
JEE	*Java Enterprise Edition*
JNDI	*Java Naming and Directory Interface*
JSP	*JavaServer Pages*
JTA	*Java Transaction API*
LGPL	GNU *Lesser General Public License*
MD5	*Message-Digest algorithm 5*
MIT	*Massachusetts Institute of Technology*
MVC	*Model-View-Controler*
MVCC	*Multi Version Concurrency Control*
PAF	Programa Aplicativo Fiscal
PDF	*Portable Document Format*
PDV	Ponto de Venda
POJO	*Plain Old Java Object*
RIA	*Rich Internet Applications*
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
SSL	*Secure Sockets Layer*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TEF	Transferência Eletrônica Financeira
UML	*Unified Modeling Language*
URL	*Uniform Resouce Locator*
W3C	*World Wide Web Consortium*
WAL	*Write Ahead Logs*
XML	*Extensible Markup Language*

LISTA DE FIGURAS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais com uma visão geral do trabalho, os objetivos, a justificativa e a organização do texto.

1.1 Considerações Iniciais

Um supermercado, mesmo que não seja de grande porte, lida com diversos tipos de produtos e possui uma dinâmica de reposição e estoque bastante alta. Muitos desses produtos comercializados são perecíveis e muitos outros têm prazo de validade que deve ser adequadamente controlado. Os níveis de estoque e a emissão de pedidos de maneira a não gerar desabastecimentos também são fatores que devem ser devidamente realizados e monitorados.

A frente de caixa é o módulo do sistema por meio do qual os produtos vendidos para o cliente são registrados, gerando baixas de estoque, informando o setor administrativo dessas vendas e gerando movimentações financeiras e fiscais. A frente de caixa é uma parte muito relevante para o sistema. É por meio dela que há interação do usuário, normalmente por uma pessoa do próprio estabelecimento que registra as vendas e faz ou encaminha a cobrança. Contudo, o sistema de retaguarda que fornece suporte para essa frente de caixa é muito mais complexo e amplo.

Na retaguarda, além do cadastro detalhado dos produtos, está o sistema de pedidos, a forma de entrega das mercadorias, o controle de contas a pagar e a receber, dentre outros. Assim, é comum que um sistema para gerenciamento de supermercado seja segmentado em partes ou módulos. Os módulos se relacionam, mas podem ser implementados como aplicações independentes. Isso facilita a análise e o projeto que geram a modelagem e na codificação do sistema, resultando no aplicativo executável.

Como forma de aprendizado e de exemplificar o desenvolvimento de um sistema relativamente complexo, neste trabalho é reportada a implementação de um sistema para supermercado. Esse sistema tem finalidades comerciais. E, portanto, possui clientes reais, o que determinada que a análise e o projeto do mesmo sejam realizadas de maneira a efetivamente atender aos requisitos e interesses dos clientes. As tecnologias utilizadas para implementar o sistema foram estudadas, destacando-se a linguagem Java.

1.2 Objetivos

O objetivo geral se refere à finalidade principal deste trabalho e os objetivos específicos estão relacionados às demais atividades realizadas, complementando o objetivo geral.

1.2.1 Objetivo geral

Implementar um sistema para gerenciamento de supermercados.

1.2.2 Objetivos específicos

Como forma de atender o objetivo geral e complementá-lo os seguintes objetivos específicos são definidos:

- Segmentar o sistema em módulos e definir a interação e a composição dos mesmos, visando facilitar a sua implementação;
- Desenvolver um sistema utilizando os conceitos de Java web e Java desktop;
- Exemplificar a implementação do sistema, como resultado do estudo das tecnologias utilizadas, destacando-se a linguagem Java, o *framework* Vraptor, Hibernate e a biblioteca JQuery.

1.3 Justificativa

A implementação de um sistema para gerenciamento de supermercados se justifica pela grande quantidade desse tipo de estabelecimento. Além disso, um controle automatizado facilita o gerenciamento dos níveis de estoque, o prazo de validade dos produtos que devem ser especificamente controlados, o controle de fornecimento de mercadorias e o controle de contas a pagar e a receber.

O desenvolvimento do sistema, objeto deste trabalho, é realizada utilizando tecnologias de uso gratuito, especificamente a IDE Eclipse. Essa IDE possui recursos suficientes para implementar um sistema para supermercado utilizando orientação a objetos.

Para o desenvolvimento do sistema proposto foi escolhida a linguagem Java.

Java para *web* é utilizada para a implementação dos módulos relacionados à retaguarda, como administrativo, financeiro e estoque. Para a frente de caixa e utilizada a linguagem Java para *desktop*, visando, inclusive facilitar a comunicação com periféricos, como leitor de código de barras e de cartões eletrônicos. Os computadores que realizarão as atividades de frente de caixa possuirão banco de dados armazenados, pois essa é uma exigência do PAF (Programa Aplicativo Fiscal). E, também para gerar menos tráfego de rede, em comunicação com o servidor, o que poderia ocasionar baixo desempenho no sistema.

Juntamente com a linguagem Java são utilizados o *framework* Vraptor, Hibernate e a biblioteca JQuery, visando agilizar o desenvolvimento e facilitar a implementação do sistema.

1.4 Organização do texto

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a ideia do sistema, incluindo os objetivos e a justificativa.

No Capítulo 2 está o referencial teórico sobre aplicações Java para *web,* descrevendo um pouco do histórico das aplicações *web* e das funcionalidades e dos recursos da linguagem Java aplicados no desenvolvimento de aplicações *web*.

No Capítulo 3 está o método, que é a sequência geral de passos para o ciclo de vida do sistema, e os materiais utilizados. Os materiais se referem ao que é necessário para modelar e implementar o sistema, incluindo as tecnologias, as ferramentas e os ambientes de desenvolvimento utilizados.

O Capítulo 4 contém a modelagem do sistema e exemplifica a sua implementação.

No Capítulo 5 está a conclusão com as considerações finais.

2 APLICAÇÕES JAVA WEB

Este capítulo contém o referencial teórico do trabalho. Inicialmente é feita uma revisão histórica do desenvolvimento para *web* abrangendo arquiteturas *web* convencionais, que são baseadas em HTML (*Hypertext Markup Language*) e HTTP (*Hypertext Transfer Protocol*). Esta seção tem o objetivo de apresentar a trajetória das aplicações *web* que culmina com a necessidade que essas aplicações contenham outros recursos no sentido de funcionalidades oferecidas pelas aplicações *desktop* e que ao mesmo tempo minimizem o tráfego de rede.

A arquitetura de um sistema computacional é a estrutura do sistema, que é composta de elementos de software, as propriedades externamente visíveis desses elementos e os relacionamentos entre eles.

Contudo, ressalta-se que uma arquitetura não necessariamente está somente relacionada aos componentes de software. Esses elementos podem ser “partes” de *software*, como pacotes ou componentes, as tecnologias utilizadas para desenvolver e/ou executar o sistema, as atividades do processo de negócio que o sistema informatiza, as partes do sistema distribuídas nos dispositivos computacionais que serão utilizados para executar o sistema, dentre outras.

Assim, a definição de quais são os elementos que compõem a representação de uma arquitetura depende do interesse do modelo que representa a arquitetura. Interesses distintos definem visões, que são representações distintas de um mesmo sistema computacional de acordo com determinado ponto de vista. Uma visão de arquitetura pode representar os módulos do sistema e como eles interagem. Desta forma, os elementos podem ser componentes de código relacionados entre si. Outra visão de arquitetura pode representar as tecnologias utilizadas para implementar e representar o sistema. E, assim, os elementos são essas tecnologias relacionadas entre si e com as “partes” do sistema que elas implementam.

2.1 Arquiteturas web convencionais

De maneira informal, são denominadas arquiteturas *web* convencionais as que possuem a interface baseada em HTML, a comunicação entre cliente e servidor é realizada pelo protocolo HTTP e a camada de aplicação e de dados é desenvolvida com base em CGIs (*Common Getway Interface*) vinculadas a bases de dados

relacionais.

Para Oliveira, Pereira e Henriques (2005) aplicações *web* se referem ao conjunto de programas que implementa um sistema de informação de acordo com o paradigma cliente/servidor suportado pelo protocolo de comunicação HTTP e cuja camada interativa (de apresentação) está escrita em HTML. O uso de HTML faz com que a interface com o usuário seja assegurada pelos *browsers* tradicionalmente criados para navegação na rede de hiperdocumentos *web*.

As aplicações *web* podem variar desde páginas pessoais simples até sistemas complexos, como *e-commerce*, sites governamentais e sistemas *intranets*. Atualmente, outros conceitos e tecnologias se agregam aos hiperdocumentos baseados em HTML e HTTP. Desta forma, um conceito mais abrangente é considerado necessário. Assim, para este trabalho, conceitua-se aplicação *web* como a aplicação que usa a Internet como meio (rede de comunicação) e seus serviços, destacando-se a *web*. De acordo com esse conceito é indiferente quais são as tecnologias utilizadas para desenvolver a aplicação (interface, lógica de negócio ou banco de dados) e as auxiliares para executá-las, como a necessidade de *plugins* vinculados ao navegador da máquina cliente.

A Figura 1 apresenta uma representação ilustrativa da arquitetura de aplicações *web* mais simples. Essa pode ser considerada a primeira forma de arquitetura *web*, porque é definida a partir dos primeiros elementos de tecnologia utilizados para uma comunicação entre cliente e servidor utilizando ambiente Internet, ou seja, ela representa as tecnologias utilizadas para codificar as primeiras aplicações *web*, que são HTML e HTTP.

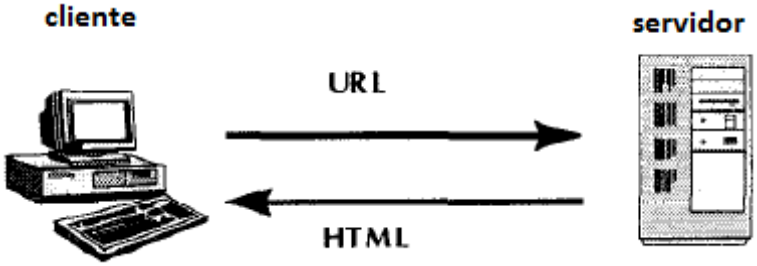

Figura 1 – Navegação em hipertexto usando HTML
Fonte: adaptado de CHO et al. (1997).

Uma evolução dessa navegação em hipertexto usando HTML que representa a forma mais simples de arquitetura *web* está apresentada na Figura 2. Essa

arquitetura é caracterizada pelas funcionalidades (pelo processamento) providas pelos programas CGI.

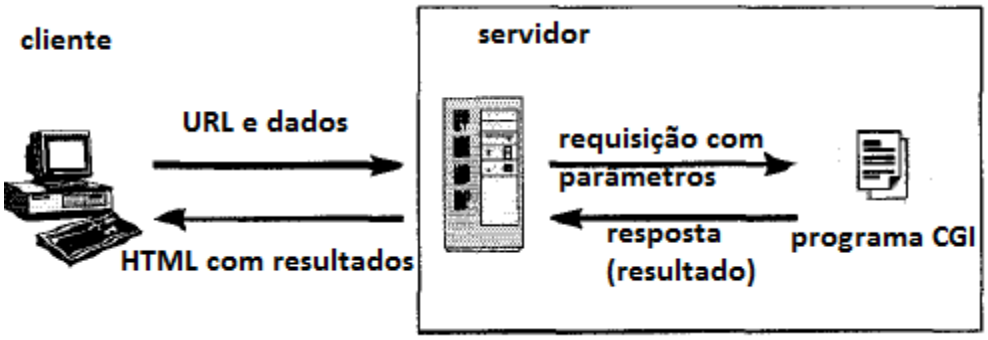

Figura 2 - Aplicação *web* usando CGI
Fonte: adaptado de CHO et al. (1997).

Na arquitetura representada na Figura 2, o cliente especifica, mesclado com *tags* HTML, o programa CGI e um conjunto de argumentos requeridos. Essa especificação é transmitida ao servidor. Após a recepção, o servidor decodifica os parâmetros e invoca o programa CGI especificado com os argumentos passados. Depois que o programa CGI é finalizado, o servidor empacota os resultados em formato HTML e os transmite para o cliente.

Uma evolução aos programas CGI é a interação com bases de dados. Contudo, da mesma forma, o cliente especifica o programa CGI com parâmetros que são transformados pelo programa em consultas SQL (*Structured Query Language*) para que o SGBD (*Sistema de Gerenciamento de Banco de Dados*) possa processar. Os resultados produzidos pelas operações com a base de dados serão empacotadas em documentos HTML que são transmitidos para o cliente. A representação da arquitetura de uma aplicação *web* com a inclusão de banco de dados no servidor é apresentada na Figura 3.

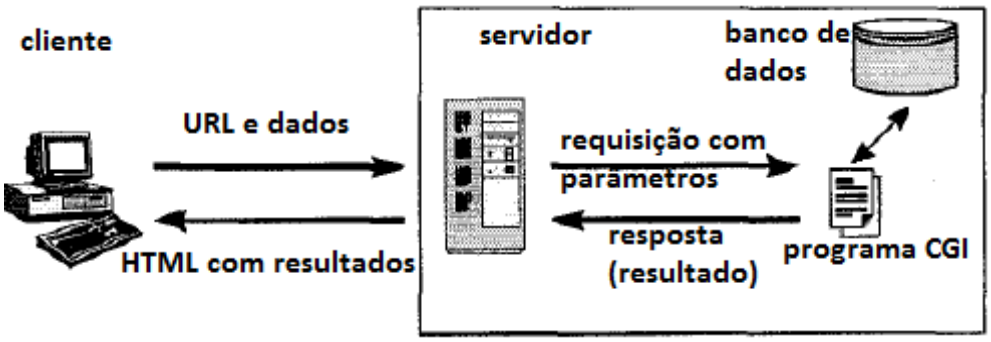

Figura 3 – Servidor *web* com interface para banco de dados
Fonte: adaptado de CHO et al. (1997).

A arquitetura, ou modelo, representado na Figura 3, apesar de apresentar uma evolução significativa ao CGI puro ainda apresenta demora de resposta devido ao fraco acoplamento entre o servidor e os programas CGI (CHO et al., 1997).

As características requeridas nas aplicações *web*, pelos usuários que utilizam sistemas *desktop* com finalidades profissionais estão acostumados a recursos como menus dinâmicos, efeitos de arrastar-e-soltar, botões diferenciados e outros. Esses recursos não são providos pelas aplicações *web* tradicionais, consideradas assim as que possuem interface baseada em HTML. Como forma de suprir essas necessidades surge as denominadas Aplicações Internet Ricas, as *Rich Internet Applications* (RIA). Duhl (2003) ressalta que as RIAs foram propostas com o objetivo de resolver problemas encontrados nas aplicações *web*. Elas provêm interface sofisticada para representar processos e dados complexos, ao mesmo tempo em que minimizam a transferência de dados entre cliente e servidor (FUKUDA, YAMAMOTO, 2008). Diversas tecnologias estão sendo propostas e outras adaptadas para que possam atender às novas exigências das aplicações *web*. Citam-se como exemplos dessas tecnologias estão Java fx, Flex para a camada de interface com Java para a camada de aplicação, por exemplo, e JavaServer Faces.

Ricca (2004) propõe um modelo de aplicação *web* capaz de gerenciar o dinamismo dessas páginas. Esse dinamismo se refere à geração dinâmica de código HTML no lado do servidor. A Figura 4 apresenta o modelo proposto por Ricca (2004). O modelo de uma determinada aplicação é uma instância do exposto na Figura 4.

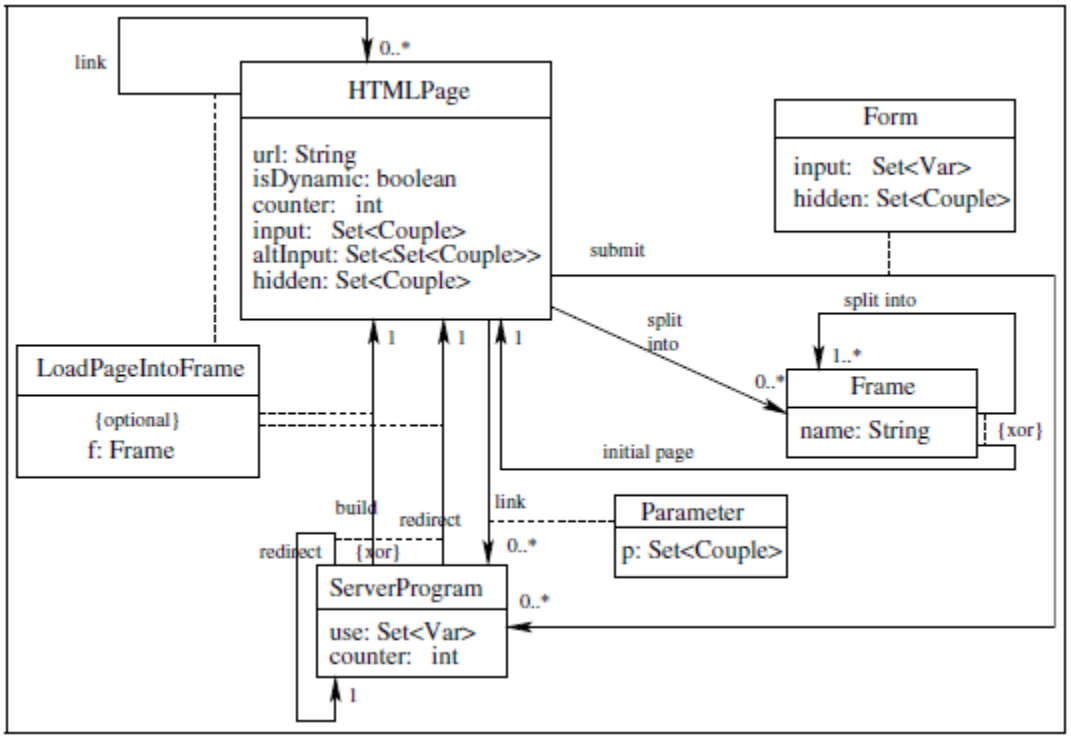

Figura 4 - Meta-modelo da estrutura de uma aplicação web genérica
Fonte: Ricca (2004), p. 13

Apesar de Ricca (2004) ter proposto um meta-modelo visando solucionar problemas relacionados à necessidade de dinamismo das páginas *web*, verifica-se que esse esquema pode ser utilizado para páginas *web* em geral. Com esse meta-modelo é possível organizar a forma de planejar e de implementar uma página *web*. É uma espécie de modelo de arquitetura para aplicações *web*.

Para complementar a ideia do meta-modelo proposto por Ricca (2004), pode-se utilizar a proposta de Fraternalli e Paolini (1998). Para esses autores, o projeto de uma aplicação *web* deve considerar particularidades de três dimensões:

a) Estrutural (conceitual) - define a organização das informações a ser tratada pela aplicação e os seus relacionamentos;

b) Navegacional - define como as informações serão acessadas através da aplicação;

c) Apresentação - define como as informações serão apresentadas e o acesso será realizado pelo usuário.

Cada uma dessas dimensões define visões diferentes do projeto, independentemente da arquitetura a ser adotada para o sistema.

2.2 Arquiteturas web baseada em Java

Uma solução aos problemas apresentados na arquitetura de aplicações *web* baseadas em CGI são as aplicações *web* em Java. Uma aplicação *web* em Java pode ser baseada em *applets* Java que são carregados no cliente a partir do servidor e executados no cliente.

A plataforma JEE define quatro tecnologias básicas para a construção de aplicações *web*: Java Servlets, JavaServer Pages, JavaServer Faces e Standard Tag Library (FIELDS, KOLB, 2000).

O código Java pode ser incorporado em páginas *web* que pode ser executado sem modificações, da mesma forma que é usado um site *web*. *Applets* Java são projetados para executar por meio de um navegador *web*. O código dos *Applets* está associado com *tags* HTML. Ressalta-se que a linguagem Java e JavaScript são diferentes em termos de sintaxe e de semântica. JavaScript é uma linguagem de *script* CGI. A linguagem Java é uma linguagem de programação e de aplicação de *applets*.

Utilizando uma linguagem como Java, a representação esquemática da arquitetura de uma aplicação *web* com banco de dados apresentada na Figura 3 é ampliada, possibilitando que tanto o cliente quanto o servidor possuam dados armazenados e realizem processamento. Essa nova arquitetura é representada na Figura 5.

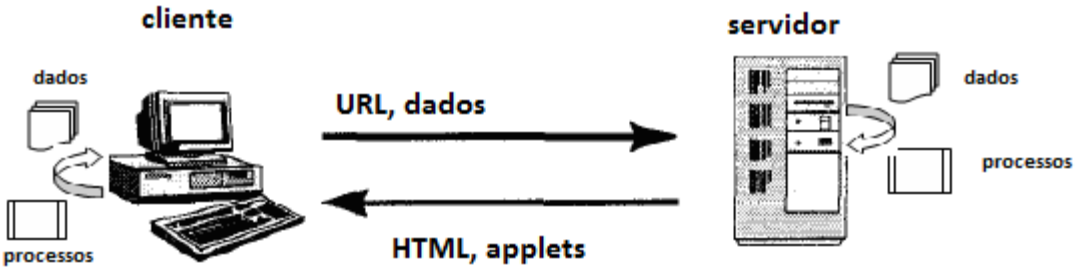

Figura 5 – Arquitetura web Java
Fonte: adaptado de CHO et al. (1997).

No modelo de arquitetura esquematicamente representado na Figura 5, tanto o cliente quanto o servidor podem armazenar dados e realizar processamento. Classes AWT (*Abstract Window Toolkit*), por exemplo, podem prover interfaces dinâmicas para o cliente, tanto *desktop* como *web*, mas nesse último deve ser feito

usando *applet*. Java Server Faces e Flex também são formas de promover dinamismo e recursos avançados para a interface de aplicações. Essa é a base das aplicações que possuem interface com recursos semelhantes às aplicações *desktop* e são denominadas RIA.

Uma arquitetura *web* baseada em Java oferece a possibilidade de:

a) Distribuição de objetos – os objetos definidos para um determinado domínio são distribuídos entre cliente e servidor. Objetos que executam no cliente podem ser implementados como *applets* e objetos que executam no servidor podem ser implementados como *servlets* ou aplicações Java. Uma distribuição adequada dos objetos melhorará o desempenho da aplicação.

b) Distribuição de processamento – o processamento é distribuído entre o cliente e o servidor. A carga de processamento que será alocada para o cliente ou para o servidor dependendo de diversos fatores como a capacidade de processamento do cliente e do servidor, a capacidade de armazenamento (se o volume de dados for considerável), a capacidade de tráfego da rede, a criticidade dos dados utilizados e os mecanismos de segurança utilizados, dentre outros.

c) Distribuição de dados – o particionamento dos dados também é dependente da capacidade do cliente e do servidor. Se o cliente possui recursos, dados multimídia, por exemplo, podem ficar armazenados no cliente em vez de em um servidor remoto. Isso pelo tráfego de rede que a transmissão desse tipo de dados gera.

Existem várias formas de implementar aplicações *web* utilizando Java. Dentre as formas possíveis de arquitetura para essas aplicações estão: aplicações baseadas em *servlets,* aplicações baseadas em páginas JSP (*JavaServer Pages*) e aplicações baseadas na arquitetura MVC (*Model-View-Controler* ou modelo-visão-controle).

2.2.1 Aplicações Java baseadas em servlets

De uma perspectiva conceitual, a *web* é uma aplicação distribuída com diversos servidores de conteúdo e clientes que acessam seus conteúdos. Cada par servidor *web* e navegador *web* corresponde a um par cliente/servidor de uma arquitetura duas camadas padrão (SHARMA, SHARMA, 2001).

De acordo com Sharma e Sharma (2001), *servlets* são peças de código Java que operam do lado do servidor. Para Bloch (2001), em certo sentido, enquanto *applets* operam do lado do cliente, *servlets* operam de forma complementar do lado

do servidor, mas, diferentemente dos *applets*, que não têm acesso aos recursos existentes na máquina do cliente. Os *servlets* têm acesso aos mesmos recursos disponíveis para as aplicações do lado do servidor, como acesso ao sistema de arquivos, bancos de dados locais ou outras conexões de rede.

Para Kurniawan (2002) a tecnologia *servlet* é a base do desenvolvimento de aplicativos *web* usando a linguagem de programação Java. Os *servlets* oferecem vantagens substanciais em termos de desempenho, porque eles podem permanecer ativos continuamente, eliminando o *overhead* que ocorre pela carga de programas. Outra vantagem é a portabilidade (SHARMA, SHARMA, 2001; BLOCH, 2001).

Para utilizar *servlets* ou JSP é preciso que o servidor os suporte. Isto pode ser uma característica nativa do servidor ou pode ser implementado por meio de módulos externos (*plugins*) que são disponíveis gratuitamente na Internet (BLOCH, 2001). Atualmente muitos servidores suportam *servlets* nativamente, dentre eles o Apache Tomcat, o O'Reilly's *Web*Site e o Sun Java*Web*Server.

Para executar um *servlet* é preciso instalá-lo em um contêiner, que é um servidor que executa código Java. Em seguida é necessário enviar uma requisição do cliente ao servidor para iniciar o *servlet*. Um *servlet* HTTP é um componente Java que pode ser usado como extensão de um servidor *web*, assim como um *applet* pode ser utilizado como extensão de um navegador. Com um *servlet*, o servidor HTTP pode oferecer serviços adicionais e personalizados como suporte a novos protocolos de comunicação, geração automática de páginas, acesso a banco de dados, controle remoto de aplicações e dispositivos, etc. (SHARMA, SHARMA, 2001).

A forma como o *servlet* é instalado determina se ele pode ser iniciado uma única vez e permanecer ativo até que o servidor seja desligado, ou se ele é iniciado cada vez que um cliente o requisita e é finalizado quando o cliente finalizar o serviço. O primeiro tipo é chamado de *servlet* permanente e o segundo, de *servlet* temporário.

JSP é semelhante aos *servlets*, pois é compilado para a forma de um *servlet* e ambos são processados no lado do servidor, retornando para o cliente apenas conteúdo HTML. A diferença entre eles é que JSPs incorporam código Java em código HTML estático, enquanto *servlets* são mais adequados para tarefas de controle de aplicação, permitindo incorporar código HTML incorporado a código Java.

2.2.2 Aplicações Java baseadas em JavaServer Pages

JavaServer Pages é uma alternativa para criação de páginas *web* dinâmicas (FIELDS, KOLB, 2000). Uma página dinâmica permite a definição e a alteração de conteúdo em tempo de execução. Assim, a interface e a lógica da aplicação são tratadas separadamente (MARQUES, CAETANO, 2000).

Pipka (2003) propõem duas práticas como as melhores para desenvolver aplicações *web* utilizando Java: JSP e MVC. JSP representa uma arquitetura centrada na página. A principal diferença entre arquitetura centrada na página e a MVC é a localização do processamento da requisição.

Uma página JSP é simplesmente uma página *web* que contém porções de códigos em Java e porções de códigos em HTML, junto com outros elementos que proporcionam informação adicional ao computador no qual a página será visualizada. De uma forma mais técnica, uma página JSP é um documento baseado em texto que descreve como processar uma solicitação (*request*) para criar uma resposta (*response*) (BOMFIM JUNIOR, 2002).

JSP faz parte da plataforma JEE (*Java Enterprise Edition*) e, juntamente com os JavaServlets e JavaBeans, pode ser usada para desenvolver rapidamente aplicações *web* eficientes, escaláveis e seguras (HARTKE NETO, 2002).

A tecnologia JSP prevê o uso de *tags* equivalentes às do XML (*Extensible Markup Language*) e *scripts* escritos na linguagem de programação Java, para encapsular a lógica que gera o conteúdo da página.

Adicionalmente, a lógica da aplicação pode residir em recursos baseados em servidor, que as páginas têm acesso com essas *tags* e *scripts*. As *tags* HTML ou XML passam diretamente para a página de resposta. Por separar a lógica da página do seu *design*, a tecnologia JSP é, provavelmente, a forma mais fácil e mais rápida de construção de aplicações baseadas na *web* (MARQUES, CAETANO, 2000).

Em um cliente/servidor *web* utilizando tecnologia JSP, por meio de um navegador, o cliente envia uma solicitação para exibir o conteúdo dinâmico da página JSP. O servidor *web* verifica se a página JSP foi modificada, caso tenha sido modificada, o servidor de aplicação gera o código Java para torná-la um *servlet*. O código gerado é compilado e executado e, por fim, a resposta é enviada em forma de página HTML e é exibida pelo navegador Internet para o cliente.

Os principais benefícios da JSP são (HARTKE NETO, 2002): usar Java, ser

parte do pacote JEE e suportar programação em rede que é inerente à linguagem Java. Para esse mesmo autor, a finalidade de JSP é fornecer um método de desenvolvimento de *servlets* declarativos e centrados na apresentação.

Tipicamente, páginas JSP estão sujeitas a uma fase de tradução e outra de processamento da requisição. A fase de tradução é feita apenas uma vez, a menos que a página JSP seja alterada. Nesse caso ela precisa ser traduzida novamente. Supondo que não tenha havido erro de sintaxe na página, o resultado será uma página JSP que implementa a interface *servlet*.

A fase de tradução é tipicamente realizada pela máquina JSP, quando ela recebe uma requisição para a página JSP pela primeira vez. A especificação JSP 1.2 também permite que páginas JSP sejam pré-compiladas em arquivos *class*. Isto é útil para reduzir o tempo de carga da página JSP na primeira vez que ela é acessada.

2.2.3 Aplicações Java baseadas em Model-View-Controller

As aplicações centradas em páginas JSP possuem como características páginas JSP inter-relacionadas que são solicitadas diretamente pelos usuários e o cliente realiza o controle, a lógica de negócio e a apresentação. Essa forma de arquitetura apresenta como limitações a baixa reusabilidade de código, a falta de controle de fluxo porque cada JSP deve validar as solicitações provenientes do usuário e o código Java é definido junto com o HTML que compõem a página, ou seja, o HTML e o código Java estão mesclados em um mesmo arquivo.

Nas aplicações baseadas em *servlets*, este controla as solicitações dos clientes, executa a lógica da aplicação e atribui a apresentação às páginas JSPs. Aplicações baseadas em *servlets* apresentam como limitação a de ter o controle de solicitações e a implementação da lógica da aplicação mescladas. Essas aplicações provêm manutenibilidade, interoperabilidade e reusabilidade, ainda que reduzidas.

No desenvolvimento de aplicações *web* baseado na arquitetura MVC, a arquitetura em camadas é usada para dividir uma aplicação por responsabilidades.

Essa arquitetura é baseada em: modelo baseado em classes Java, visão composta pelas páginas JSP e controle realizado pelos servlets.

O modelo MVC apresenta um padrão arquitetural subdividido em modelo-visão-controle. A separação dos componentes de uma aplicação diminui o acoplamento

entre eles, tornando a manutenção mais fácil, de maneira que mudanças em um componente não tenham muito impacto em outro componente.

A camada *Model* (modelo) representa a parte de sua aplicação que implementa a lógica do negócio. Isto significa que ela é responsável por obter os dados convertendo-os em conceitos significativos para sua aplicação, assim como, processar, validar, associar e qualquer outra tarefa relativa ao tratamento dos dados.

Os objetos do tipo *Model* podem ser vistos como a primeira camada de interação com qualquer banco de dados, que o programador possa usar na sua aplicação. Mas em geral eles representam os principais conceitos em torno do qual o desenvolvedor implementa sua aplicação. No caso de uma rede social, a camada *Model* cuida de tarefas, como as de salvar os dados dos usuários e o relacionamento entre amigos, armazenar e recuperar as fotos dos usuários, encontrar novos amigos para sugestões e etc.

Uma *View* exibe uma representação dos dados modelados. Sendo separadas do objeto *Model*, é responsável por usar as informações disponibilizadas para produzir qualquer interface de apresentação que uma aplicação possa necessitar.

Por exemplo, como a camada *Model* retorna um conjunto de dados, a *View* pode usá-los para exibir uma página HTML ou retornar o resultado em um formato XML para que outros o consuma.

A camada *View* não está limitada à representações dos dados no formato HTML ou texto, podendo ser usada para entregar uma variedade de formatos diferentes, dependendo do que necessita, como vídeos, músicas, documentos e qualquer outro formato.

A camada *Controller* lida com as requisições dos usuários. É responsável por retornar uma resposta com a ajuda das camadas Model e View.

Os *Controllers* podem ser vistos como gerentes tomando os devidos cuidados para que todos os recursos necessários para completar uma tarefa sejam delegados para os trabalhadores corretos. Ele aguarda os pedidos dos clientes, verifica a validade de acordo com as regras de autenticação e autorização, delega dados para serem obtidos ou processados pelos *Models* e seleciona o tipo correto de apresentação dos dados para o qual o cliente está aceitando para finalmente delegar o trabalho de renderização para a camada de visualização.

3 MATERIAIS E MÉTODO

Este capítulo contém os materiais e o método utilizados para a modelagem e a exemplificação da implementação do sistema. Os materiais se referem às ferramentas e as tecnologias, incluindo linguagem de programação, banco de dados, interface de desenvolvimento e ferramenta para análise, projeto e exemplo de implementação. O método se refere aos procedimentos utilizados no ciclo de vida do sistema, abrangendo da definição dos requisitos à implantação do sistema.

3.1 Materiais

As seguintes ferramentas e tecnologias foram utilizadas para modelar e exemplificar a implementação do sistema:

a) Astah* Community - para a documentação da modelagem que é baseada na UML. A ferramenta Astah* foi escolhida por ser gratuita, possibilitar a modelagem utilizando UML e por ser a sucessora da Jude Community, que foi descontinuada.

b) Dezign for Database v6 – para o desenvolvimento das entidades relacionais. Essa ferramenta possibilita criar as entidades do banco de dados e os seus relacionamentos e gerar *scripts* para o PostgreSQL 9.1.

c) Linguagem Java – Java *web* para os módulos Estoque, Financeiro e Administrativo e Java *desktop* para o módulo Frente de caixa (PDV - Ponto De Venda).

d) Eclipse – IDE (*Integrated Development Environment*) de desenvolvimento para web.

e) PostgreSQL – como banco de dados, por ser gratuito, atende a maioria dos clientes e está a anos no mercado.

f) PGAdminIII - para administração de banco de dados, o PostgreSQL, uma ferramenta fácil de manipular e atende boa parte das nossas necessidades.

g) Hibernate - *framework* para o mapeamento objeto relacional, com a utilização desse *framework,* podemos ter uma alta produtividade ao realizar consultas no banco de dados.

h) Vraptor 3 – *framework* MVC, brasileiro, open source, para controle entre as classes Java e os JSPs.

i) JQuery – componentes para os *scripts* da interface, utilizado para animações de páginas e também proporcionar um visual mais sofisticado para o usuário.

j) Jboss 7 - servidor *web*, proporciona um bom gerenciamento de banco de dados e permite várias configurações para melhorar o desempenho e segurança da aplicação.

k) Apache maven – gerenciar dependências, realização de builds, estrutura padrão de diretórios, informações sobre o projeto e integração com a IDE eclipse.

3.1.1 Astah* Community

Astah* Commuity é o successor do Jude Community. É uma ferramenta gratuita para modelagem de sistemas com suporte para a UML 2.1 (ASTAH, 2014). Além da modelagem dos digramas, a ferramenta oferece ajustes de alinhamento e tamanho dos diagramas, impressão (com a marca d'água da ferramenta) e exportação das imagens dos diagramas (com a marca d'água da ferramenta).

A ferramenta Astah é apresentada em três versões:

a) Astah* Community como versão gratuita que suporta todos os diagramas UML básicos;

b) Astah* UML que suporta diagramas UML a mapas mentais. Mapas mentais são diagramas conceituais ou domínio para representar ideias e a visão geral de um sistema, por exemplo;

c) Astah* Professional que fornece suporte para UML, diagramas de entidade e relacionamento, diagrama de fluxo de dados, CRUD (*Criate, Read, Update, Delete,* criar, consultar, atualizar e excluir funcionalidades definidas por cada processo), fluxograma e mapas mentais.

O Quadro 1 apresenta um resumo dos diagramas suportados pelas respectivas versões da ferramenta Astah*.

Diagrama	Astah* community	Astah* UML	Astah* professional
UML 2.x classe, caso de uso, sequência, atividade, comunicação, máquina de estado, componentes, implantação, estrutura de composição, objetos e pacotes	X	X	X
Diagrama de processo Erikson-Penker (representado por diagrama de atividade)	Somente leitura	X	X
Mapa mental	Somente Leitura	X	X
Diagrama de entidade e relacionamento (IDEF1x)	Somente leitura	Somente leitura	X
CRUD	Somente leitura	Somente leitura	X
Diagrama de fluxo de dados (com notação DeMarco/Gane-Sarson)	Somente leitura	Somente leitura	X
Fluxograma	Somente leitura	Somente leitura	X
Tabela de requisitos (incluindo hierarquia de nomes, identificadores e texto, com exportação para o Excel).	Somente leitura	Somente leitura	X
Diagrama de requisitos	Somente leitura	Somente leitura	X

Fonte: http://astah.change-vision.com/en/feature/comparison-table.html

A Figura 6 apresenta a tela inicial da ferramenta Astah* com o objetivo de explicitar a sua interface principal e as partes que a compõe.

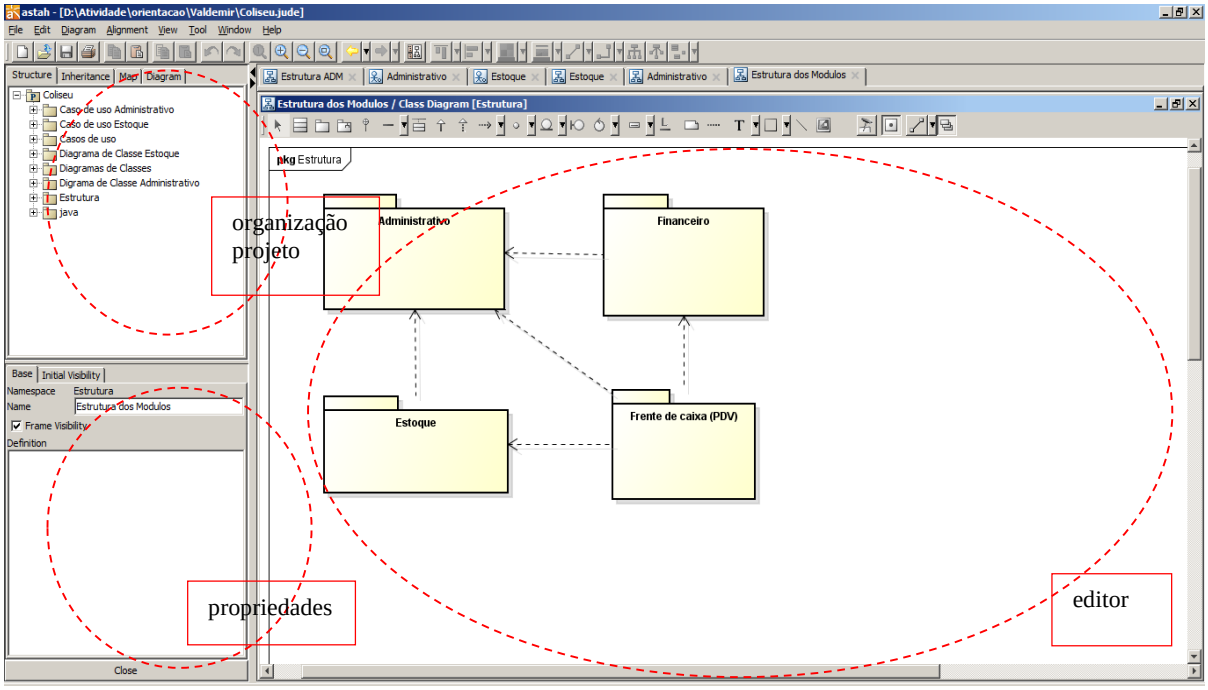

Figura 6 – Ferramenta de modelagem Astah* Community

A IDE da ferramenta Astah* Community está dividida em três partes principais, como pode ser visualizado na Figura 6:

a) Organização do projeto (área superior à esquerda da Figura 6) - é a área na qual estão as diferentes visões do projeto, são elas: *Structure* que é a árvore de estrutura do projeto, *Inheritance* que exibe as heranças identificadas, *Map* para mostrar todo o editor de diagramas e *Diagram* que mostra a lista de diagramas do projeto.

b) Visão das propriedades (área inferior à esquerda da Figura 6) - é a área na qual podem ser alteradas as propriedades dos elementos dos diagramas. As propriedades de um item selecionado são exibidas e podem ser editadas. Por exemplo, com um diagrama de casos de uso aberto e um ator selecionado são exibidas todas as propriedades definidas para esse ator, como nome e o tipo de visibilidade.

c) Editor do diagrama (área à direita da Figura 6) - é a área na qual são exibidos e construídos os diagramas. Ao ser selecionado um determinado diagrama, dos constantes na lista, o mesmo é carregado e todos os seus elementos gráficos são mostrados nessa área.

3.1.2 Dezign For Database v6

DeZign for Databases (DEZIGN, 2014) é uma ferramenta para projeto de banco de dados que auxilia a criar e manter bancos de dados. É um aplicativo pago na aquisição e também por upgrade. Essa ferramenta utiliza diagramas de entidade e relacionamentos para projetar e gerar bases de dados graficamente e sentenças SQLs.

A ferramenta DeZign for Databases oferece um ambiente visual para modelagem do banco de dados de aplicação. Essa ferramenta, com a tela principal apresentada na Figura 6, possibilita visualizar a estrutura de base de dados, criar base de dados, modificá-las e fazer engenharia reversa de bases de dados existentes.

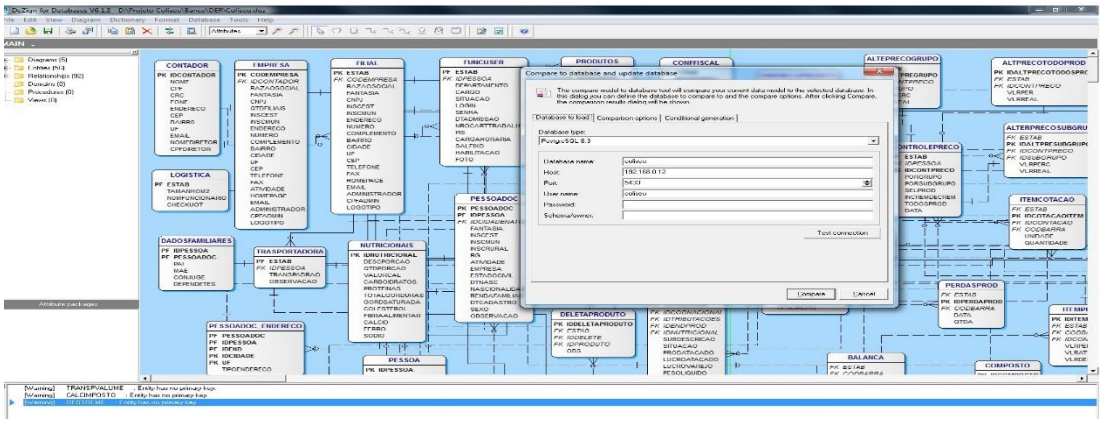

Figura 7 – IDE da ferramenta Dezign For Database

Na barra superior da Figura 7 estão os recursos comuns de aplicativos como novo, abrir, salvar e excluir. Na metade direita desta barra estão os elementos para a modelagem do banco de dados.

Na parte superior da barra lateral esquerda (Figura 7), estão as pastas com os diagramas, as entidades, os relacionamentos, domínios, procedimentos e visões do diagrama atual. Na parte inferior dessa barra lateral são apresentadas as propriedades do elemento selecionado.

Na parte central da Figura 7 está a área de desenho. A tela em primeiro plano sendo mostrada permite acessar a funcionalidade de comparação entre o diagrama e o banco de dados. Facilitando a identificação de inconsistências entre a modelagem e o banco.

Os principais recursos da ferramenta DeZign for Databases são (DEZIGN, 2014):

a) Modelagem visual do banco de dados – a modelagem é feita graficamente por meio de diagramas de entidades e relacionamentos. O projeto pode ser apresentado em níveis de detalhes distintos.

b) Gerar base de dados – gerar *scripts* DDL (*Data Definition Language*) completos para criar a base de dados ou gerar a base de dados diretamente.

c) Importar base de dados – derivar um modelo gráfico da base de dados a partir de base de dados existentes. A engenharia reversa é feita diretamente a partir do banco de dados ou importada de *scripts* SQL.

d) Sincronização de modelo ou base de dados – funcionalidades de sincronização e comparação para todos os casos de uso: do modelo para a base de dados, do modelo para *script*, da base de dados para o modelo, do *script* para o

modelo e do modelo para o modelo.

A funcionalidade de comparação e junção de modelos é importante quando duas ou mais pessoas trabalham juntas em um mesmo modelo. A versão 6 do DeZign oferece um recurso para comparação e junção de projetos permitindo comparar e juntar duas versões de um mesmo modelo de dados.

e) Relatórios – geração de relatórios com diferentes níveis de detalhe. Descrições e outras informações relevantes podem ser exportadas para HTML, Microsoft Word ou PDF (*Portable Document Format*).

f) Verificação de erros – os modelos podem ser verificados quanto à existência de erros durante a fase de projeto. O Dezign verifica se os elementos do modelo estão corretos e completos. Objetos reusáveis (pacotes de atributos e domínios) e nome de padrões possibilitam a definição de objetos consistentes na base de dados.

g) Modelos físicos ou modelos independentes de banco de dados – Dezign suporta a modelagem para um DBMS (*Data Base Management System*) físico ou ser independente de banco de dados, gerando um modelo portável de tipos de dados lógicos.

3.1.3 Linguagem Java

A plataforma ou o ambiente de programação Java permite desenvolver aplicativos utilizando qualquer uma das linguagens criadas para a plataforma Java. Uma grande vantagem dessa plataforma é a de não estar vinculada a um único sistema operacional ou *hardware*, pois seu código gerado executa através de uma máquina virtual que pode ser emulada em qualquer sistema operacional que suporte à linguagem C++.

Java é uma linguagem orientada a objetos, sendo assim, a maior parte dos elementos de um programa Java são objetos (DEITEL, DEITEL, 2005). Como exceção cita-se os tipos básicos, como o *int* e o *float*. O código é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si.

Chamadas a funções de acesso remoto (*sockets*) e os protocolos Internet mais comuns (HTTP, FTP (*File Transfer Protocol*), Telnet, etc.) são suportadas em Java, de forma que a implementação de programas baseados em arquiteturas cliente/servidor é facilitada.

Java provê o gerenciamento de memória por meio de *garbage collection* (coleta

de lixo). Sua função é a de verificar a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Esse procedimento pode deixar o sistema de *software* com execução mais demorada por manter uma *thread* paralela à execução do programa. Porém, esse procedimento evita problemas como referências perdidas e avisos de falta de memória quando ainda há memória disponível na máquina.

3.1.4 IDE Eclipse

A IDE Eclipse Juno (ECLIPSE, 2014) é um ambiente de desenvolvimento utilizado para produzir código fonte de acordo com a sintaxe e a semântica da linguagem Java. É um ambiente de desenvolvimento multiplataforma, uma ferramenta para escrever, compilar, depurar (*debug*) e gerar instaladores. A IDE Eclipse facilita o trabalho de desenvolvimento de aplicativos por possuir um grande conjunto de bibliotecas, módulos e APIs (*Application Programming Interface*) que são basicamente um conjunto de rotinas, protocolos e ferramentas para a construção de *software*.

Como a ferramenta Eclipse é escrita em Java, ela é independente de plataforma, executa em qualquer sistema operacional que suporte a máquina virtual Java. E está disponível para vários sistemas operacionais, como Linux, Windows e Mac OS.

A Figura 8 apresenta a tela principal da IDE Eclipse.

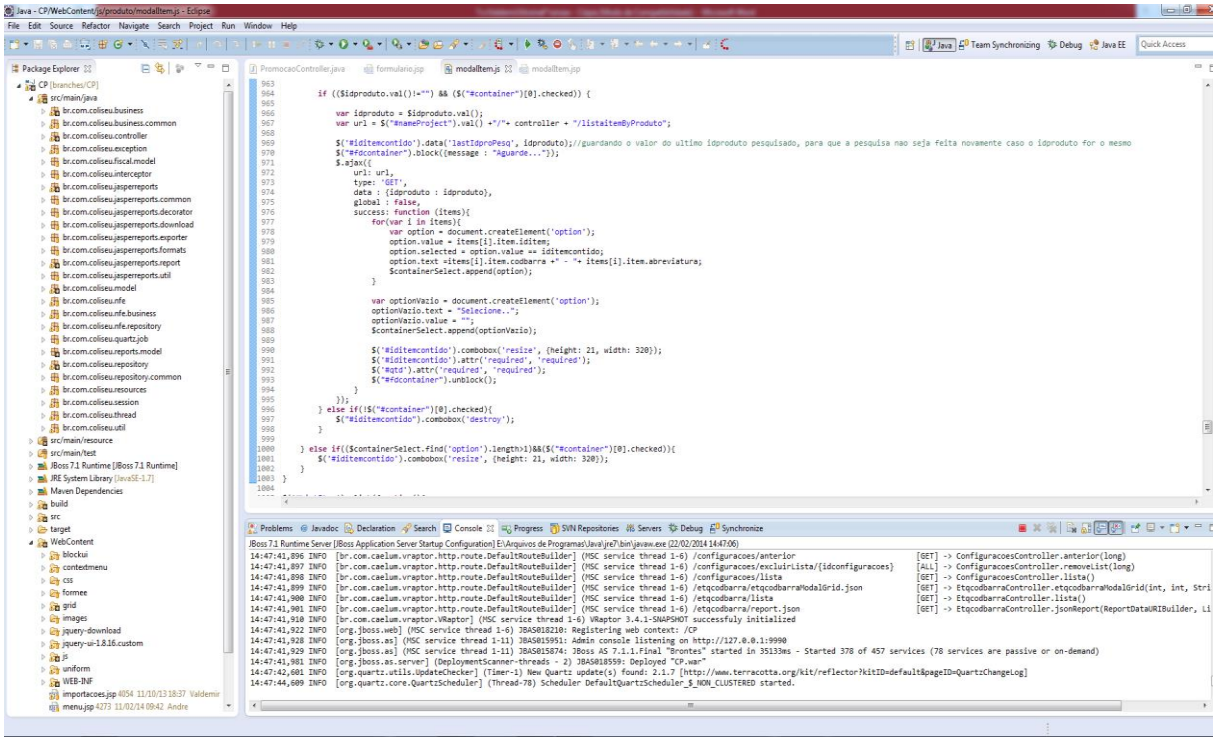

Figura 8 – IDE Eclipse

Na Figura 8, na área a esquerda estão os elementos que são compostos com a ferramenta. Na área superior à direita está o editore de código. E na parte inferior direita, estão as abas de saída (resultados) do editor.

3.1.5 PostgreSQL 9.1

PostgreSQL (POSTGRESQL, 2014) é um sistema de banco de dados objeto-relacional de código aberto. Pode ser executado na maioria dos sistemas operacionais incluindo Linux, Mac OS X, Solaris e Windows. Possui suporte total a chaves estrangeiras, *joins, views, triggers* e *stored procedures*. As suas principais características são (SQL MAGAZINE, 2009):

a) Recuperação automática após *crash* de sistema (WAL - *Write Ahead Logs*);

b) MVCC (*Multi Version Concurrency Control*) ou controle de concorrência de multi versão. Nesse mecanismo, os processos de leitura não bloqueiam processos de escrita e vice-versa, reduzindo drasticamente a contenção entre transações concorrentes e paralisação parcial ou completa (*deadlock*);

c) Commit, *rollback* e *checkpoints* de transações;

d) Triggers e *stored procedures;*

e) Foreign keys;

f) Backup on-line;

f) Tamanho ilimitado de registro. Não há limite para o tamanho dos tipos de dados;

g) Índices em *cluster*: cada tabela pode suportar um índice em *cluster* (*clustered index*). Esse índice classifica fisicamente os dados na mesma sequência como especificada pelo índice. Um índice de *cluster* permite a maior velocidade possível na recuperação de dados melhorando o desempenho geral do banco de dados.

3.1.6 PgAdmin

A ferramenta pgAdmin é um gerenciador de banco de dados PostgreSQL gratuito (PGADMIN, 2014) que permite escrever sentenças SQL simples ou complexas e desenvolver bancos de dados com a ajuda das funções PostgreSQL. pgAdmin é desenvolvido pela comunidade do PostgreSQL e está disponível em diversos idiomas. É um software gratuito lançado sob a licença Artistic License.

O aplicativo pgAdmin é utilizado para gerenciar o PostgreSQL 7.3 e versões superiores em qualquer plataforma, bem como versões comerciais do PostgreSQL como EnterpriseDB e Mammoth PostgreSQL. A sua interface gráfica oferece suporte a todos os recursos do PostgreSQL e facilita a sua administração. A Figura 9 apresenta a tela principal desse aplicativo.

Figura 9 – IDE da ferramenta pgAdmin

Na Figura 9, as partes marcadas indicam:

1) Barra de tarefas, por meio da qual são feitas as conexões com banco de dados, atualização, edição, exclusão, acesso ao visualizador de dados e ao editor de

SQL.

2) É a janela que permite explorar os bancos de dados conectados, oferecendo acesso às suas tabelas, esquemas, funções, usuários e gatilhos.

3) Essa área oferece visualização de tabelas e outros elementos que estão sendo manipulados nos bancos de dados conectados.

4) Ao explorar as tabelas e outras propriedades, são mostradas nessa janela as instruções SQL da tabela ou propriedade em uso.

O pgAdmin também inclui um editor com destaque de sintaxe, um editor de código do lado do servidor, um agendador de tarefas SQL/*batch/shell*, suporte para o mecanismo de replicação Slony-I, dentre outros. A conexão com o servidor pode ser feita usando TCP/IP (*Transmission Control Protocol/Internet Protocol*) ou Unix Domain Sockets (em plataformas *nix) e pode ser criptografado via SSL (*Secure Sockets Layer)* para segurança. Nenhum *driver* adicional é exigido para comunicação com o servidor de banco de dados.

3.1.7 Hibernate

O Hibernate (HIBERNATE, 2014) é um *framework* para o mapeamento objeto-relacional escrito na linguagem Java, mas também é disponibilizado em .Net como o nome NHibernate. Esse *framework* facilita o mapeamento dos atributos entre uma base de dados relacional e o modelo de objetos de uma aplicação, mediante o uso de arquivos XML.

O Hibernate é um *framework software* livre de código aberto distribuído com a licença LGPL (GNU *Lesser General Public License)*. O objetivo do Hibernate é diminuir a complexidade dos programas Java, baseados no modelo orientado a objeto, cujo banco de dados é no modelo relacional, especialmente no desenvolvimento de consultas e atualizações dos dados.

O Hibernate transforma os dados tabulares de um banco de dados em um grafo de objetos definido pelo desenvolvedor. Sua principal característica é a transformação das classes Java para tabelas de dados e dos tipos de dados Java para tipos SQL. O Hibernate gera as sentenças SQL, mantendo o programa portável para qualquer banco de dados padrão SQL.

Apesar de existirem APIs no Hibernate para operações de controle transacional, ele delegará essas funções para a infraestrutura na qual foi instalado.

Assim, o gerenciamento de transações e a tecnologia de acesso à base de dados são de responsabilidade de outros componentes do programa.

Em aplicações construídas para serem executadas em servidores, o gerenciamento das transações é realizado segundo o padrão JTA (*Java Transaction API*). Nas aplicações *desktop*, o programa delega o tratamento transacional ao *driver* JDBC (*Java Database Connectivity*). Hibernate pode ser utilizado em aplicações Java *desktop* ou em aplicações JEE (*Java Enterprise Edition*), utilizando *servlet* ou sessões EJB (*Enterprise Java Beans*).

O Hibernate possui algumas interfaces importantes (INFOBLOG, 2014):

a) **Interface de sessão (*session*)** - *session* é a interface primária do Hibernate e é a mais fácil de criar e de destruir.

b) **Interface SessionFactory** - a interface SessionFactory é compartilhada entre diversas *threads*. Para a SessionFactory há um único aplicativo criado na inicialização. Se o aplicativo acessar múltiplos bancos de dados usando Hibernate, é necessário uma SessionFactory para cada banco. Essa interface controla dados armazenados e que foram lidos em uma unidade de trabalho, podendo ser reutilizados em uma unidade futura, se o mapeamento de classe especificar o armazenamento do segundo nível. Geralmente a SessionFactory armazena instruções SQL e outros metadados que são executados em tempo de execução.

c) **Interface de configuração** - a interface de configuração é a primeira a ser encontrada quando o Hibernate é usado. Após essa interface ser configurada, o Hibernate criará o SessionFactory.

d) **Interface de transação** – a interface de transação é uma API opcional do Hibernate. Uma transação permite controlar os limites das transações usando uma API consistente. Isso permite tornar portáveis os aplicativos do Hibernate com ambientes de execução diferentes.

e) **Interface e critérios de consulta** - as consultas são escritas em SQL para conectar parâmetros e limitar o número de resultados obtidos por elas.

3.1.8 VRaptor 3

O VRaptor 3 é um *framework* MVC em Java focado no desenvolvimento rápido e simples e na manutenção do código fácil (CAVALCANTI, 2014). VRaptor é uma iniciativa brasileira, nascida dentro da Universidade de São Paulo.

O VRaptor 3 foca em simplicidade e, portanto, todas as funcionalidades têm como meta resolver o problema do programador da maneira menos intrusiva possível em seu código. Tanto para salvar, remover, buscar e atualizar como funcionalidades que costumam ser mais complexas como *upload* e *download* de arquivos e resultados de pesquisas em formatos diferentes (xml, json, xhtml etc.) são realizadas por meio de funcionalidades simples do VRaptor 3, que sempre procuram encapsular HttpServletRequest, Response, Session e toda a API do Javax Servlet (BEVILAQUA, LACERDA e LIMA, 2014).

O VRaptor trabalha com a ideia de POJOs (*Plain Old Java Objects*) como componentes para executar as lógicas de negócio. A utilização desse *framework* é baseada na criação de objetos muito simples, com código Java comum.

3.1.9 JQuery

jQuery (JQUERY, 2014) é uma biblioteca JavaScript que simplifica a identificação de *tags* em documentos HTML, a manipulação de eventos, o desenvolvimento de animação e de interações Ajax facilitando o desenvolvimento *web*.

jQuery é uma biblioteca JavaScript criada por John Resig e disponibilizada como software livre e aberto, ou seja, de emprego e uso regido segundo licença conforme as regras estabelecidas pelo MIT (*Massachusetts Institute of Technology*) ou pelo GPL (*General Public License*). jQuery é uma biblioteca JavaScript que possui as seguintes características (SILVA, 2010):

a) uso de seletores CSS (*Cascading Style Sheet*) para localizar elementos componentes da estrutura de marcação HTML da página;

b) arquitetura compatível com instalação de *plugins* e extensões em geral;

c) indiferença às inconsistências de renderização entre navegadores;

d) oferece interação implícita, isto é, não há necessidade de construção de estruturas de repetição para localização de elementos no documento;

e) admite programação encadeada, ou seja, cada método retorna um objeto.

f) é extensível, pois admite criação e inserção de novas funcionalidades em bibliotecas existentes.

jQuery se destina a adicionar interatividade e dinamismo às páginas *web*, incrementando de forma progressiva e não obstrutiva a usabilidade, a acessibilidade

e o projeto, enriquecendo a experiência do usuário, facilitando:

a) adicionar efeitos visuais e animações;

b) acessar e manipular o DOM (*Document Object Model*;

c) buscar informações no servidor sem necessidade de recarregar a página;

d) prover interatividade;

e) alterar conteúdos;

f) modificar apresentação e estilo;

g) simplificar tarefas específicas de JavaScript;

jQuery foi criada com a preocupação de ser uma biblioteca em conformidade com os padrões *web*, ou seja, compatível com qualquer sistema operacional, navegador e com suporte total para as CSS3. Evidentemente isso não significa que todo código escrito com uso de jQuery resulta em um documento válido segundo a ótica dos padrões *web*. A biblioteca foi criada e está em acordo com as diretrizes do W3C (*World Wide Web Consortium*), mas cabe ao desenvolvedor escrever *scripts* em conformidade.

3.1.10 JBoss 7

JBoss Aplication Server 7 (JBOSS 7, 2014). Em 2006 o JBoss foi adquirido pela a Red Hat, a partir de então houve um grande avanço relação a plataformas e projetos existentes na comunidade.

O JBoss Application Server 7 ou simplesmente AS 7 é a primeira plataforma desta nova etapa, que traz inúmeras inovações, como rapidez na inicialização, baixíssimo consumo de recursos, serviços iniciando on-demand, entre tantas outras, pensando no que estamos vivendo hoje e o que poderá vir para amanhã, adaptando-se a ambientes nas nuvens com clusters elásticos ou até mesmo a dispositivos que cabem em nossos bolsos.

O AS 7, é a mais nova versão do Servidor de Aplicação Java de código aberto mais utilizado por desenvolvedores de aplicações corporativas baseadas na plataforma Java EE. A nova versão teve seu núcleo totalmente remodelado para fornecer um servidor leve, modular, de fácil administração e melhor performance. Sem sombra de dúvidas, uma das características que mais se destaca nessa nova versão é a sua velocidade de inicialização.

Outras melhorias incluem o novo modo de operação baseado em domínios de servidores de aplicação e as novas ferramentas de administração. Com um núcleo totalmente modular e otimizado para ambientes multiprocessados, o AS 7 leva aproximadamente três segundos para iniciar em um hardware comum encontrado em qualquer laptop pessoal utilizado nos dias de hoje. Essa versão não possui um instalador automático. Para instalar o AS 7 basta fazer o download do pacote distribuído em formato compactado (ZIP ou TAR GZ) e extrai-lo em um diretório qualquer.

O único pré-requisito para instalação é ter o JDK 6 ou JDK7 com a variável de ambiente JAVA_HOME configurada.

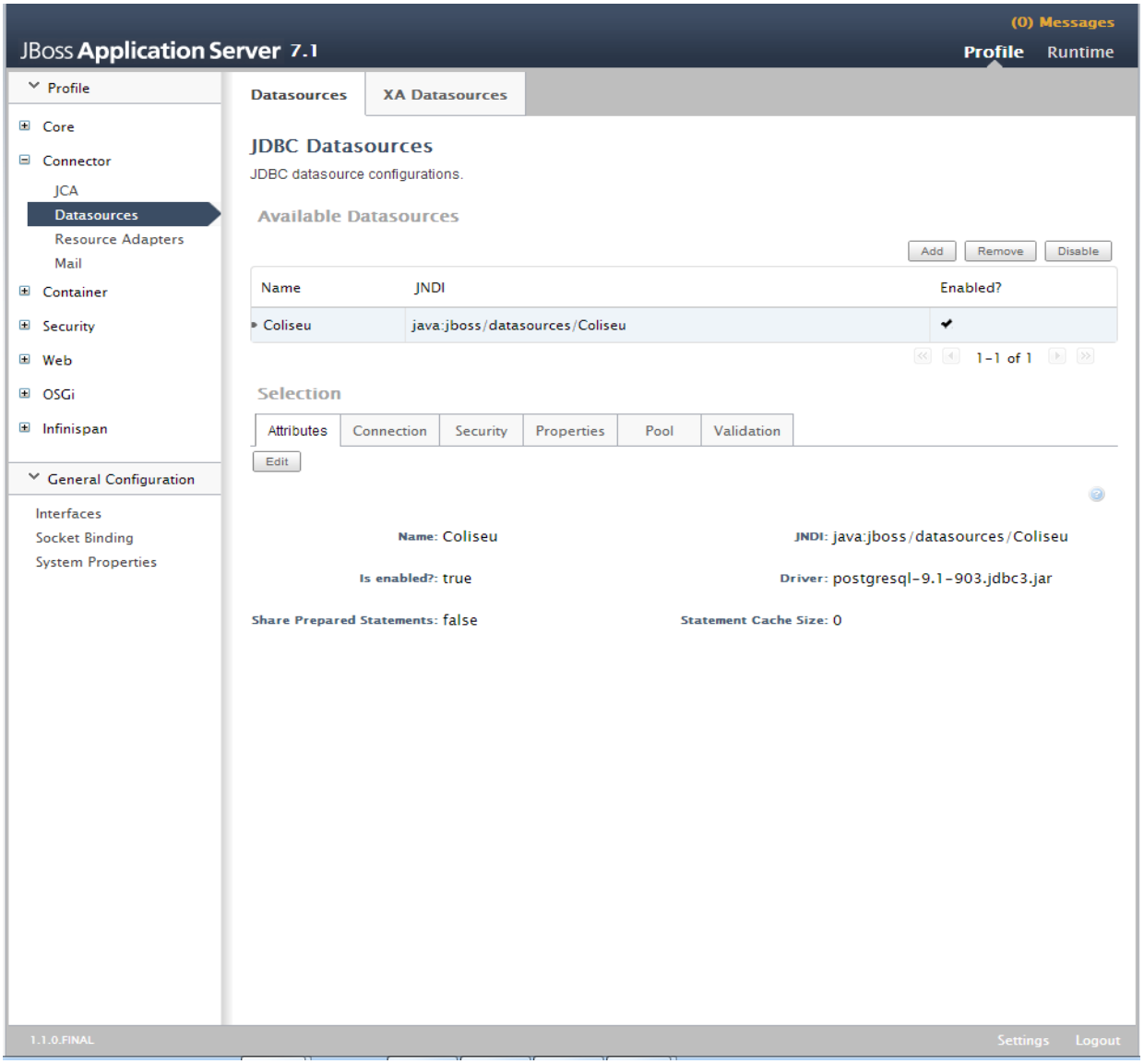

Figura 10 – JBoss Aplication Server

Na Figura 10, o lado esquerdo da tela está o menu que dá acesso as configurações, no centro da tela é aonde entra com os dados de configurações: banco de dados, segurança, web services, entre outras. Através dessa tela também pode-se realizar deploy da aplicação.

3.1.11 Apache Maven

O processo de criação de um projeto Java EE, em geral, envolve a criação de um diretório principal com vários subdiretórios, a configuração de diversos arquivos XML, a obtenção de bibliotecas para o projeto e, posteriormente, a execução dos testes unitários, a criação dos pacotes de publicação, a geração de documentação *javadoc*, entre outras etapas.

Normalmente, até algum tempo atrás, cada projeto tinha sua própria estrutura, seu próprio jeito de gerar pacotes, de efetuar cada um destes passos. Projetos complexos, com vários módulos, ainda podem precisar que estes sejam compilados em determinada ordem, para que o pacote final seja criado. Nesse ponto que entra o Maven.

O projeto é hospedado pela Apache Software Foundation, que fazia parte do antigo projeto Jakarta. Para a configuração utiliza-se um arquivo XML (POM) para descrever o projeto de software sendo construído, suas dependências sobre módulos e componentes externos, a ordem de compilação, diretórios e *plug-ins* necessários. Ele vem com objetivos pré-definidos para realizar certas tarefas bem definidas como compilação de código e seu empacotamento.

O Maven (MAVEN, 2014) faz *download* de bibliotecas Java e seus plug-ins dinamicamente de um ou mais repositórios, e armazena-os em uma área de cache local. Este cache local de artefatos baixados podem também ser atualizados com artefatos criados por projetos locais.

O Maven é construído utilizando uma arquitetura baseada em *plugin*, que permite que ele faça uso de qualquer aplicação controlável através da entrada padrão. Teoricamente, isto permitiria qualquer um escrever *plug-ins* para fazer interface com ferramentas de construção (compiladores, ferramentas de teste de unidade, etc.) para qualquer outra linguagem.

Alguns pontos vantajosos no uso o Apache Maven:

a) Com o Maven o programador atinge rapidamente um nível de automatização muito alto e com pouquíssima configuração. Essa automatização se reflete no processo de release de um versão do seu software que fica muito, mais simples com o maven;

b) O Maven incentiva que se criem testes unitários. Dessa maneira, quando o programador faz o *build* ou gera um *deploy* em um servidor, nesse momento o Maven executa todos os testes unitários. Caso algum teste unitário falhar ele aborta o *build/deploy*, essa é uma excelente política para forçar os testes e a qualidade de software;

c) Esta ferramenta tem todo um ciclo de vida do desenvolvimento definido. Assim, a medida que algo vai acontecendo e ele vai mudando de estados no ciclo de vida, determinados *plug-ins* vão ser executados. O programador pode parametrizar o que quer que aconteça em que momento do ciclo de vida do maven;

d) No Maven tudo é igual, o desenvolvedor faz todas as parametrizações através do arquivo pom.xml e tudo da mesma forma. O que o programador faz em um projeto pode ser aproveitado em outro projeto;

e) Esta ferramenta prove a gerencia de *jar*, bem como *plug-ins*. Esses artefatos não ficam em controle de versão. O programador não cria uma pasta **lib** e coloca os *jars* lá dentro, eles ficam no repositório local do Maven;

3.2 Método

No método utilizado para o desenvolvimento do sistema são consideradas as três fases análise, projeto e implementação propostas por (RUMBAUGH, et al., 1997). Contudo, são acrescentadas outras denominações e fases são acrescentadas visando explicitar com mais detalhes o que foi realizado durante o ciclo de vida do sistema.

Considerando que o sistema a ser modelado é amplo, houve necessidade de definir uma primeira visão geral do sistema, mas os seus requisitos serão complementados e revistos à medida do andamento do projeto. Apesar de em uma primeira iteração terem sido levantados e definidos todos os requisitos considerados essenciais, várias revisões ocorrerão. Desta forma, as fases apresentadas a seguir estão em ordem sequencial, mas a sua execução não foi única para cada fase e também não foram sequenciais e lineares. Os modelos da arquitetura, por exemplo,

não foram todos definidos após a definição dos casos de uso, como está colocado no Capítulo 4. O modelo que representa a visão tecnologia da arquitetura foi elaborado antes dos casos de uso. Contudo, todos os modelos da arquitetura estão colocados juntos para facilitar o seu entendimento.

As principais fases para o desenvolvimento do projeto foram:

a) Planejamento do sistema

Inicialmente foi elaborada uma visão geral do sistema com os seus módulos principais. Foram definidos os módulos Administrativo, Financeiro, Estoque e Frente de Caixa como os principais agrupamentos de requisitos. Esses módulos compõem a visão da arquitetura de alto nível.

Nesta fase decidiu-se que o sistema seria modelado com base nos requisitos inicialmente levantados, mas à medida que os módulos seriam implementados os requisitos seriam revistos.

Com base nessas decisões foi estabelecido um cronograma visando que tanto documentação do trabalho (o relatório de estágio), quando a modelagem e uso das tecnologias pudessem ser realizados simultaneamente.

Em uma primeira etapa do trabalho ocorreu o levantamento bibliográfico, incluindo os conceitos constantes no Capítulo 2 e sobre as tecnologias utilizadas, compondo o texto da Seção 3.1 do Capítulo 3. Conceitos relacionados à arquitetura auxiliaram a entender a divisão do sistema em módulos e em camadas e a organizar as tecnologias que seriam utilizadas, no sentido de relacioná-las às camadas do sistema.

b) Análise, projeto e construção de UML

Nesta etapa foi realizada a análise do sistema por módulos: Estoque, Administrativo, Financeiro e Frente de caixa. Foi necessário consultar a Receita Federal (RECEITA, 2014) para definir o leiaute da nota fiscal e do PAF para a TEF (Transferência Eletrônica Financeira).

Para definir os modelos da análise foi utilizado como base outro sistema existente na própria empresa em que o autor deste trabalho exerce atividade profissional, para enriquecer, foi realizado a análise de outros sistemas existentes, além de entrevistas com um dos possíveis clientes do sistema. As entrevistas basearam-se no levantamento de interesses e necessidades do cliente e foram acompanhadas de visitas ao supermercado, visando, assim, que os requisitos do sistema pudessem mais efetivamente representar a realidade, os interesses e as

necessidades dos clientes e usuários.

As principais tarefas desta fase, tendo como base os módulos definidos foram:

- Definir os casos de uso para o sistema;
- Definir a arquitetura do sistema;
- Definir as classes com seus atributos, operações e relacionamentos.
- Desenvolver o projeto do banco de dados, com a definição das tabelas, campos, chaves e os relacionamentos.
- Gerar os *scripts* para criar o banco para PostgreSQL 9.1.

c) Preparação do ambiente e da infraestrutura para implementação do sistema

A definição das tecnologias que seriam utilizadas foi realizada na fase de planejamento. Nesta fase essas tecnologias foram preparadas, instaladas e configuradas para que a implementação pudesse ocorrer.

d) Implementação dos módulos

A implementação do sistema é realizada por módulos, inicialmente os módulos de estoque, administrativo e financeiro foram implementados com Java para *web*. Em seguida foi implementado o módulo frente de caixa com Java para *desktop*.

e) Realização de testes

Os testes foram realizado manualmente pelo do programador visando verificar detalhes e a correções de defeitos encontrados nos testes.

4 MODELAGEM DO SISTEMA E IMPLEMENTAÇÃO

Este capítulo apresenta a modelagem e algumas implementações do sistema para gerenciamento de um supermercado, incluindo frente de caixa e retaguarda com controle de estoque, gestão administrativa e financeira.

4.1 Visão geral do sistema

O sistema para supermercado que é objeto deste trabalho é composto, basicamente, por frente de caixa e retaguarda. Na frente de caixa são realizadas as vendas e terá comunicação com leitor de código de barras, impressora de cupom fiscal e máquina de cartão de crédito e de débito.

Os usuários da frente de caixa tem acesso somente às funcionalidades relacionadas à venda de produtos. O módulo frete de caixa contém um cadastro prévio de clientes que necessitam de entrega das compras, caso o mesmo não esteja no banco de dados. Esse cadastro é composto por nome, telefone e endereço.

Cada terminal (computador) frente de caixa possui um banco de dados local. Esse banco de dados permite que vendas possam ser realizadas normalmente se não houver comunicação por rede com o servidor.

Esse banco é sincronizado a cada cupom fiscal encerrado. Conforme o caixa for registrando os produtos as suas respectivas quantidades são baixadas do estoque. O valor limite estabelecido de estoque somente pode ser atualizado com permissão do supervisor. Esse valor indica a quantidade mínima de estoque que o produto deve ter segurança de não desabastecimento.

Cada produto possui um valor de quantidade mínima em estoque, dependendo da procura do produto. Esse valor, definido como grau crítico, é estipulado automaticamente ou manualmente, se o usuário preferir.

O sistema deve gerenciar a quantidade de saída do produto diariamente e com isso gerar um relatório diário da quantidade de produtos no estoque e a necessidade de efetuar pedido dos mesmos. Esse relatório será gerado para os responsáveis pelos pedidos dos produtos. O sistema também proverá o controle de vencimento de produtos em estoque para que possam ser realizadas promoções dos mesmos.

O controle financeiro abrange as funcionalidades do sistema relacionadas a contas a pagar e a receber, controle de bancos, controle de cheques emitidos e

recebidos, além do controle de pagamentos para fornecedores, de inadimplência dos clientes, de pagamento de funcionários e despesas do supermercado.

No módulo administrativo do sistema são realizados os cadastros de pessoas que abrangem clientes, fornecedores e funcionários. Nesse módulo também é feita a emissão de nota fiscal eletrônica e nota de entrada. Ainda, disponibilizará controles de gastos no supermercado, com manutenção da estrutura, cadastro de promoções, análise de saídas de produtos, emissão de relatório para reposições de produtos na prateleira, seguindo ordem de vencimento de produto por lote.

O sistema ainda possibilita o gerenciamento de empresas com matriz e filiais. O controle de estoque é realizado pela matriz, os dados são separados por um código de estabelecimento. A empresa poderá fazer o pedido de grande quantidade junto aos seus fornecedores visando obter preço menor e distribuir os produtos para as suas filiais.

4.2 Modelagem do sistema

A modelagem do sistema não está por completo registrada neste relatório. Não são colocados todos os modelos gerados em decorrência da quantidade, que são muitos devido à amplitude do sistema, contudo, os tipos de diagramas utilizados e a forma de modelagem do sistema estão representados. Assim, nas subseções a seguir estão os diagramas desenvolvidos para alguns dos quatro módulos do sistema.

O projeto atualmente conta com mais de 700 classes, aproximadamente 200 interfaces, 172 entidades e 180 tabelas. Devido a esta complexidade, bem como por se tratar de um projeto comercial, a apresentação da análise e do aplicativo desenvolvido será simplificada neste documento.

4.2.1 Casos de uso

As Figuras 11, 12 e 13 apresentam os casos de uso dos módulos Administrativo, Estoque e Financeiro.

A Figura 11 apresenta os casos de uso do módulo Administrativo.

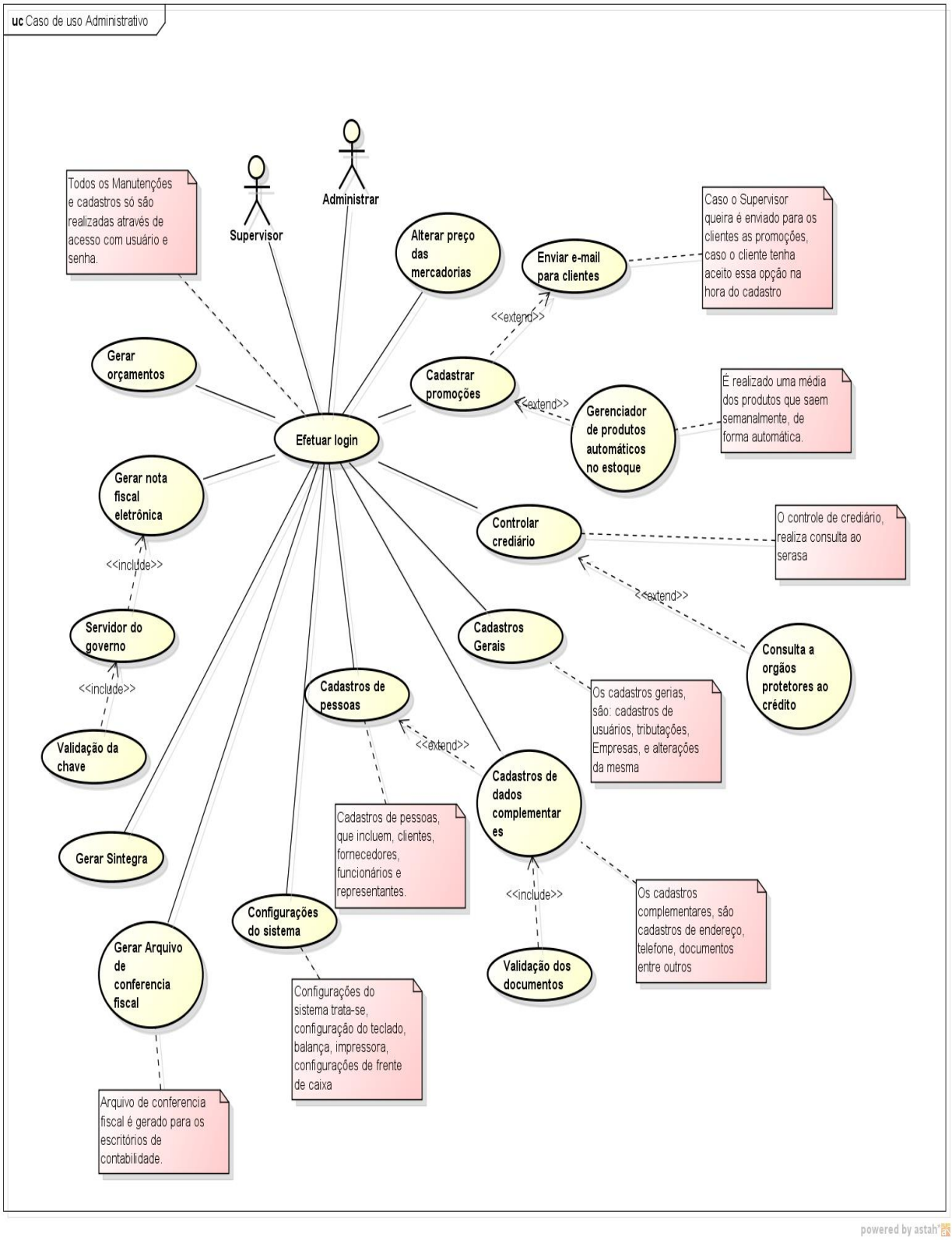

Figura 11 – Casos de uso do módulo administrativo

Os casos de uso do módulo administrativo são realizados pelos atores Supervisor e Administrador. E todas as funcionalidades do sistema são realizadas a partir de um *login*. Isso significa que o usuário para acessar o sistema deve estar

cadastrado e somente usuários nessa condição e com *login* e senha informados corretamente acessam o sistema.

A Figura 12 apresenta os casos de uso do módulo estoque.

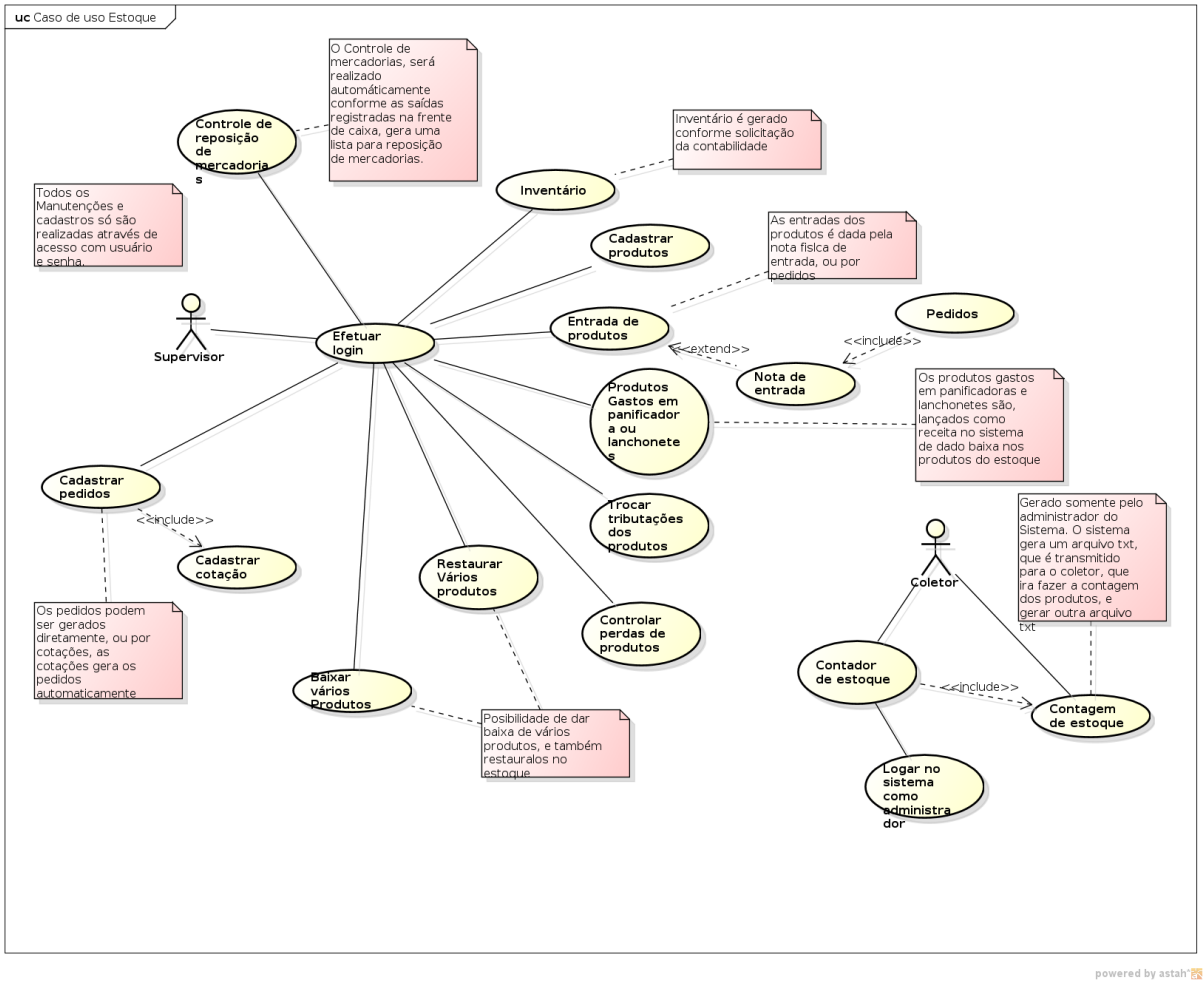

Figura 12 – Casos de uso do módulo estoque

Os casos de uso do módulo de estoque são realizados pelos atores Supervisor e um equipamento chamado Coletor. A contagem de estoque é gerada pelo administrador do sistema. Os resultados da contagem são disponibilizados ao coletor por meio de um arquivo texto. O coletor apanha os dados necessários e outro arquivo texto é gerado e enviado ao sistema.

A Figura 13 apresenta os casos de uso do módulo financeiro. Os casos de uso do módulo financeiro possuem como atores o Administrador e o Supervisor.

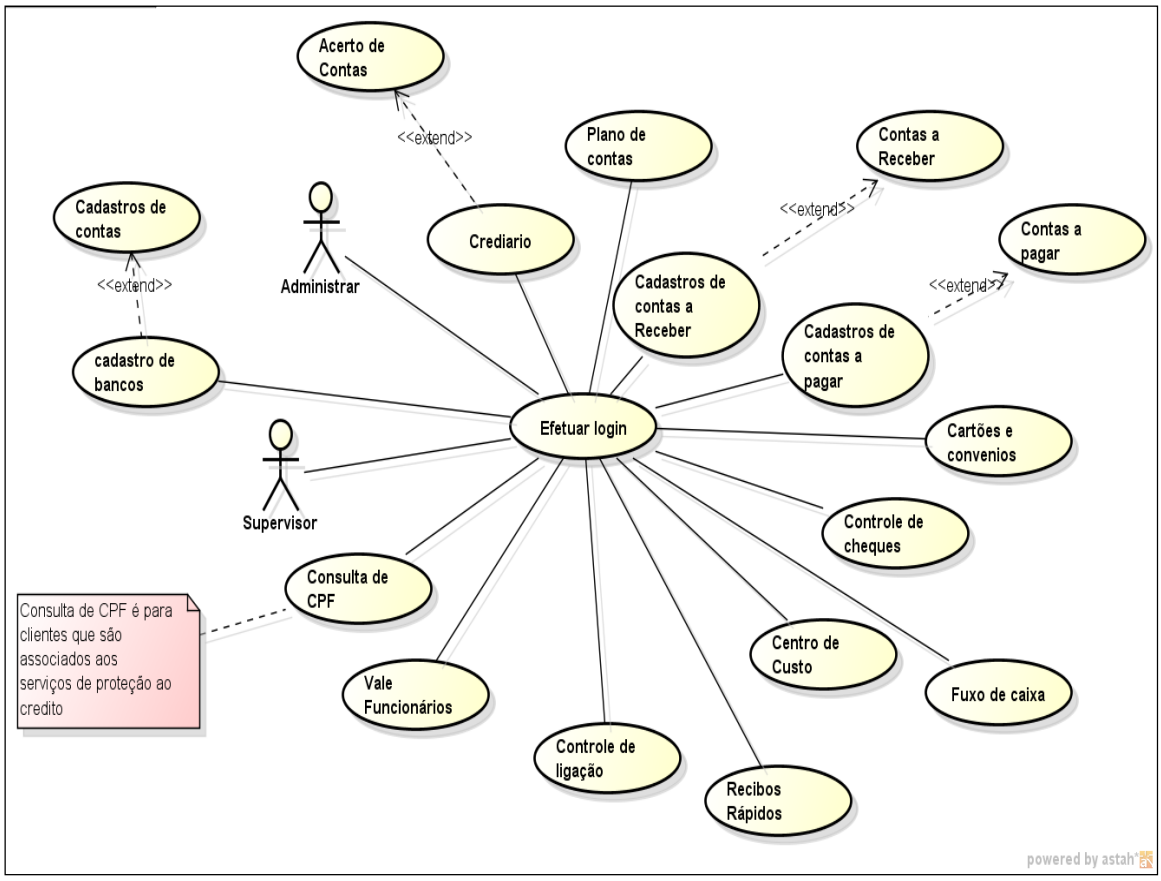

Figura 13 - Casos de uso do módulo financeiro

Com os casos de uso foi possível ter uma visão dos requisitos do projeto, assim facilitando quais seriam os passos a ser tomado, para cada módulo a partir do login do usuário. Através dele também foi desenvolvido os diagramas de classes.

4.2.2 Arquitetura

Uma arquitetura para o sistema foi definida como forma de facilitar o entendimento e a visualização do mesmo. Foram definidas as visões de componentes e de tecnologia. A visão componentes apresenta as partes do sistema, os seus módulos e os componentes que implementam cada um dos módulos. A visão tecnologias apresenta as tecnologias utilizadas na implementação e/ou execução do sistema em um modelo de arquitetura em três camadas: apresentação (interface com o usuário), aplicação (lógica de negócio) e dados (persistência).

A Figura 14 apresenta a visão tecnologia da arquitetura contendo as principais tecnologias utilizadas para implementar o sistema, considerando três camadas: interface, lógica de negócio e persistência. Essas tecnologias foram planejadas

visando uma possível expansão do sistema. Assim, o banco de dados Oracle poderá ser utilizado se uma versão para estabelecimentos de grande porte for implementada.

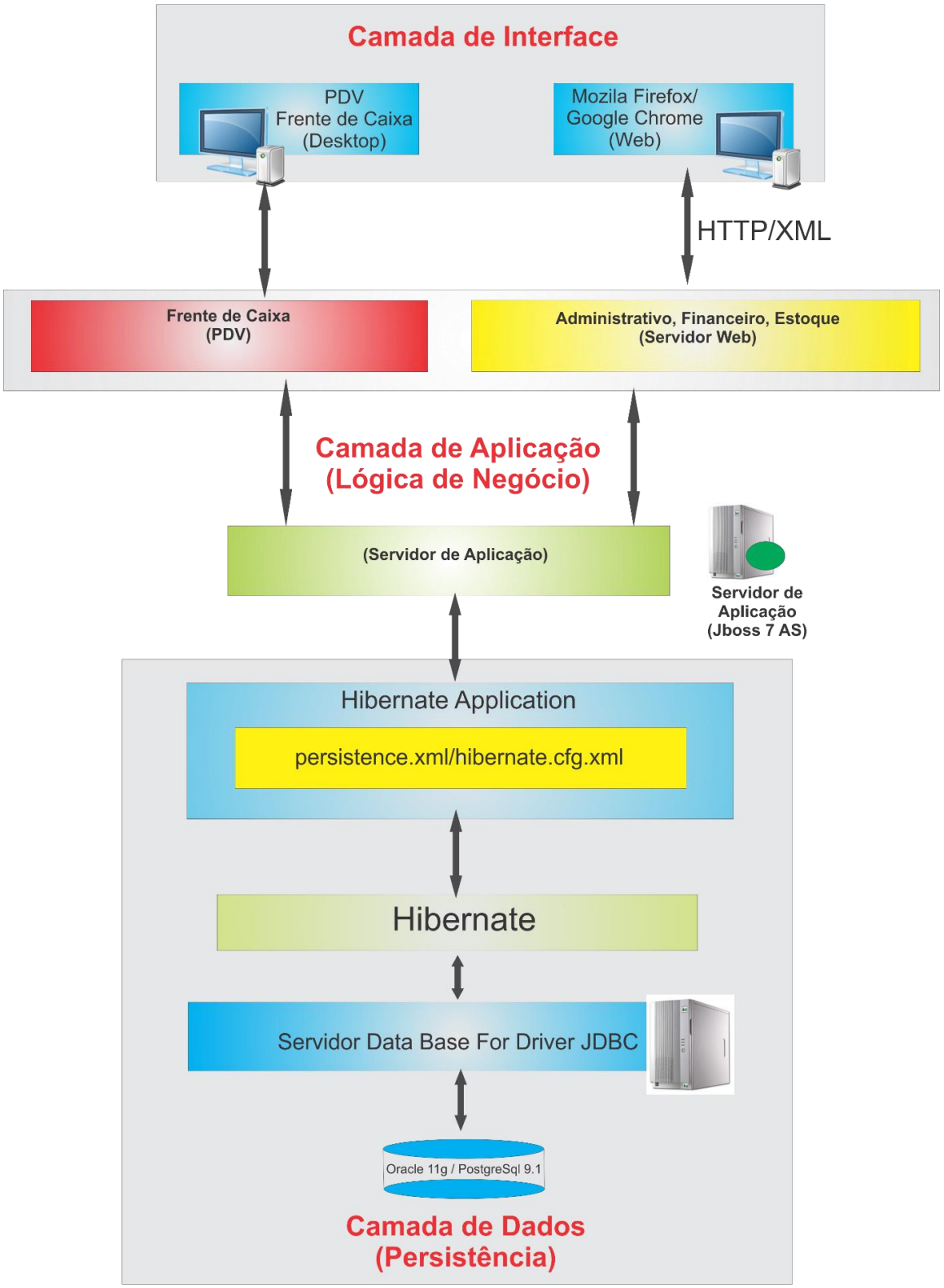

Figura 14 – Visão tecnologias da arquitetura

A organização da arquitetura tem como base um modelo de arquitetura de três camadas:

a) Camada de interface – contém a interface de interação do sistema tanto para a frente de caixa como para a retaguarda. A frente de caixa fará o acesso ao sistema por meio de uma aplicação *desktop*. Também fazem parte da interface frente de caixa, o acesso ao sistema por meio do leitor de código de barras e das máquinas de cartão de crédito e de débito. O acesso a retaguarda (módulos Estoque, Financeiro e Administrativos) é realizado por meio de uma aplicação *web*. Assim, tanto frente de caixa (representada por um PDV) como a retaguarda (composta pelos módulos Estoque, Financeiro e Administrativos) acessam a camada de aplicação. Frente de Caixa por meio da rede interna e a retaguarda por meio de HTTP e XML. Também são utilizadas as tecnologias Java e um navegador *web* (como o Mozilla Firefox ou Google Chrome).

b) Camada de aplicação – contém as regras de negócio implementadas tanto para a retaguarda como para frente de caixa (haverá processamento no módulo Frente de Caixa). Além da aplicação em si, essa camada contém o servidor de aplicação que é o JBoss 7. O desenvolvimento e a execução da camada de aplicação são baseados na linguagem Java.

c) Camada de dados – realiza a persistência dos dados. Essa camada é composta pelo Hibernate, pelo *driver* JDBC e pelo banco de dados PostegreSQL 9.1. Existe a possibilidade de utilizar outros bancos de dados, como por exemplo o Oracle.

Para a visão de componentes da arquitetura foram identificados os módulos Frente de Caixa (PDV), Estoque, Financeiro e Administrativo. Esses três últimos compõem a retaguarda. A Figura 14 apresenta esses módulos e os relacionamentos entre eles. Esses módulos definem a arquitetura de componentes de mais alto nível do sistema.

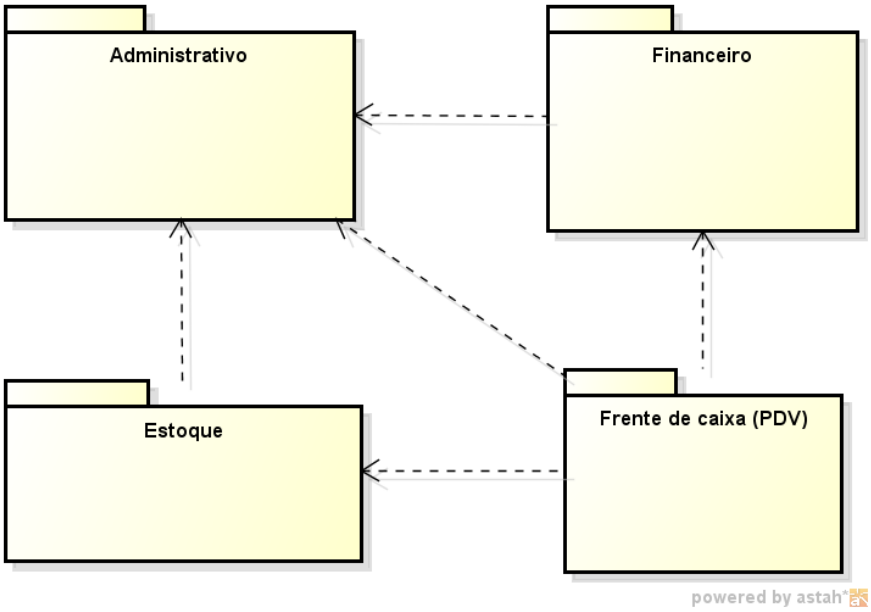

Figura 15 – Visão de componentes da arquitetura

A seguir estão listadas as características ou as funcionalidades de cada um dos módulos que compõe a visão componentes da arquitetura exposta na Figura 15.

a) Módulo Frente de Caixa PDV:

Banco de dados PostgreSql 9.1: em cada terminal que envia arquivos do cupom fiscal para a receita (PAF (Programa Aplicativo Fiscal) – ECF (Emissor de Cupom Fiscal));

Vendas de produtos: Por meio de um terminal e um atendente, os clientes fazem as compras e os pagamentos;

Impressão de nota fiscal: A nota fiscal é emitida para compras acima de R$ 2.000,00 e somente para órgãos públicos, para as demais compras e clientes é emitido o cupom fiscal ou nota fiscal com cupom vinculado;

Sincronizar entre banco terminal e central: a cada venda os terminais se comunicam com o servidor, para realizar baixa de estoque e atualizar o banco de dados local;

Controle de limite do cliente: cada cliente tem um limite de crédito, que é cadastrado no momento de abertura do crediário, esse limite é atualizado a cada compra e pagamento do cliente;

Controle de vasilhame: alguns produtos têm valores diferenciados, caso seja comprado com a vasilha, sem a devolução, por esse motivo deve haver um

controle de entradas e saídas desses produtos;

Consulta de cheques: nas compras com cheque é realizada uma consulta aos órgãos protetores de crédito, para verificar o estado do cliente;

Cancelar cupom fiscal: caso aconteça a compra seja cancelada, ou haver problemas no crediário do cliente é necessário cancelar o cupom fiscal, se esse já foi emitido;

Cancelar produtos: para os produtos que o cliente queira devolver após a venda é necessário permitir o cancelamento da compra e devolução do produto no estoque;

Troca de mercadorias (verificar código do consumidor): o sistema deve permitir a troca de mercadorias, conforme legislação;

Câmbio de moedas: utilizado para clientes que moram em divisas de país, por exemplo, Barracão e Foz do Iguaçu e cidades turísticas;

Consulta de segunda via do cupom fiscal no sistema é armazenado um cupom fiscal e por meio desse cupom é possível ao usuário fazer uma consulta da segunda via do cupom fiscal;

Consulta de clientes: verifica se há inadimplência de clientes do crediário;

Recebimento de contas: por meio dos terminais permitir fazer pagamento de contas como luz água e outros;

TEF (Transferência Eletrônica de Fundos): com esse aplicativo é possível fazer vendas por cartões de crédito e registrar automaticamente no sistema a venda;

Configurações de impressora de cupom fiscal: o sistema através de *dll's* deve comunicar-se com impressoras fiscais no momento da venda;

Configuração do leitor de código de barra: um leitor de código de barra é um dispositivo de entrada de dados, que lerá o código do produto e passará o respectivo valor para o campo em foco no sistema.

b) Módulo Estoque

Cadastros de produtos: todos os produtos para venda devem estar cadastrados no sistema;

Importação das vendas: realizado pelo PAF, que sincronizará com o banco de dados na rede, em cada tempo estimado;

Controle estoque pela nota fiscal: todos os produtos comprados possuem a

nota fiscal, o abastecimento do estoque é feito por ela;

Controle de baixa de produtos: produtos que já venceram devem ser baixados do sistema e retirados do mercado;

Gerar lista de produtos que estão em falta e/ou em baixa no estoque: no final do dia, ou por solicitações do usuário do sistema, deve ser gerada uma lista de produtos que atendem a essas condições;

Gerenciar procura de produtos de forma automática, gerar gráficos, para auxiliar nas decisões: conforme a procura pela compra do produto, o sistema deve gerenciar de forma automática e emitir um relatório, dos produtos que saiam com mais frequência e dos que tem baixa procura;

Controle de perdas de produtos: os produtos que vencem e produtos hortifrutigranjeiros que são perdidos, o sistema deve gerar um relatório;

Troca de tributações: muitos produtos possuem a mesma tributação, o sistema deve gerenciar isso que a partir do momento da troca de tributações, seja persistido em todos os produtos com a mesma tributação;

Baixa de vários produtos: o sistema deve permitir baixa de vários produtos ao mesmo tempo;

Contagem de estoque: por meio de um aparelho é feita a contagem de estoque e esses valores são enviados para o sistema;

Derivados: carcaça bovina, por exemplo;

Transformar pedido em nota fiscal: vincular pedido a nota fiscal;

Composto dos produtos: cesta básica.

c) Módulo Financeiro

Integrar com todo o sistema, separando por matriz e filial;

Impressão de recibos e promissórias avulsas: pagamentos de valor parcial do crediário do cliente emitir um recibo;

Fluxo de caixa: será controlado por plano de contas;

Vale de funcionários: fornecer vales para funcionários e ter um devido controle separando tudo no plano de contas;

Cancelamento de pagamento: para pagamentos que já tenham sido agendados no sistema, mas o pagamento não será realizado;

Acompanhamento da conta na hora do cadastro: à medida que despesas e receitas são cadastradas mostrar o valor de saídas, entradas e o saldo;

Controle de contas a pagar a receber: o controle de saídas e entradas organizado em um plano de contas;

Cadastros separados de contas e de pagamento de contas;

Fechamento bancário: controlar os lançamentos de despesas e de crédito em conta bancária;

Cadastros de bancos: contas de banco cadastradas para emitir boleto entre outras funções;

Recebimento de títulos: receber títulos de empresas conveniadas;

Fluxo de caixa: controle de cada caixa, diariamente junto com entradas de outras pagamentos (Crediário, luz, telefone);

Duplicatas: cadastro de duplicatas a receber e a pagar;

Controle de cheques: cadastro de cheques que foram emitidos e recebidos;

Acerto financeiro: pagamento dos clientes.

d) Módulo Administrativa

Cadastros de promoções de mercadorias: controle do tempo que uma ou mais mercadorias permaneceram em promoção;

Alteração de preço de mercadorias: agendamento de troca de preço, para disparar em horário sem movimento;

Controle de crediário: com consulta no Serasa;

Cadastros: Clientes, fornecedores, funcionários e transportadoras;

Cadastros gerais: endereços, entre outros;

Administração dos usuários do sistema: administrador, supervisor e usuário;

Tributações: cadastros de tributações conforme legislação;

Nota fiscal eletrônica: emissão de nota fiscal eletrônica;

Configurações do sistema: configurações do sistema incluindo acessos de usuários;

Configurações da frente de caixa: tipo de impressora, leitor, atendente;

Controle de produtos: para reposição de mercadorias;

Cotações de produtos: realizar cotações junto aos fornecedores;

Orçamentos: fazer orçamento com vários fornecedores e analisar qual o melhor preço de cada produto;

Coletor de dados: recebe os dados do coletor e persiste no banco de dados;

Pedido de compra: realizar pedido e encaminhar para os fornecedores;

Etiquetas - gerar etiquetas para as prateleiras, com valores e descrição dos produtos;

Configurações de juros: configurar os juros para clientes inadimplentes;

Fiscal gerar Sped: gerar sped para encaminhar a receita;

Gerar arquivos para importação de escritório de contabilidade.

A Figura 16 apresenta a estrutura de componentes do módulo administrativo. Essa figura é um detalhamento do módulo Administrativo que compõe a arquitetura de componentes de mais alto nível apresentada na Figura 17. Cada um dos módulos é composto por componentes de software.

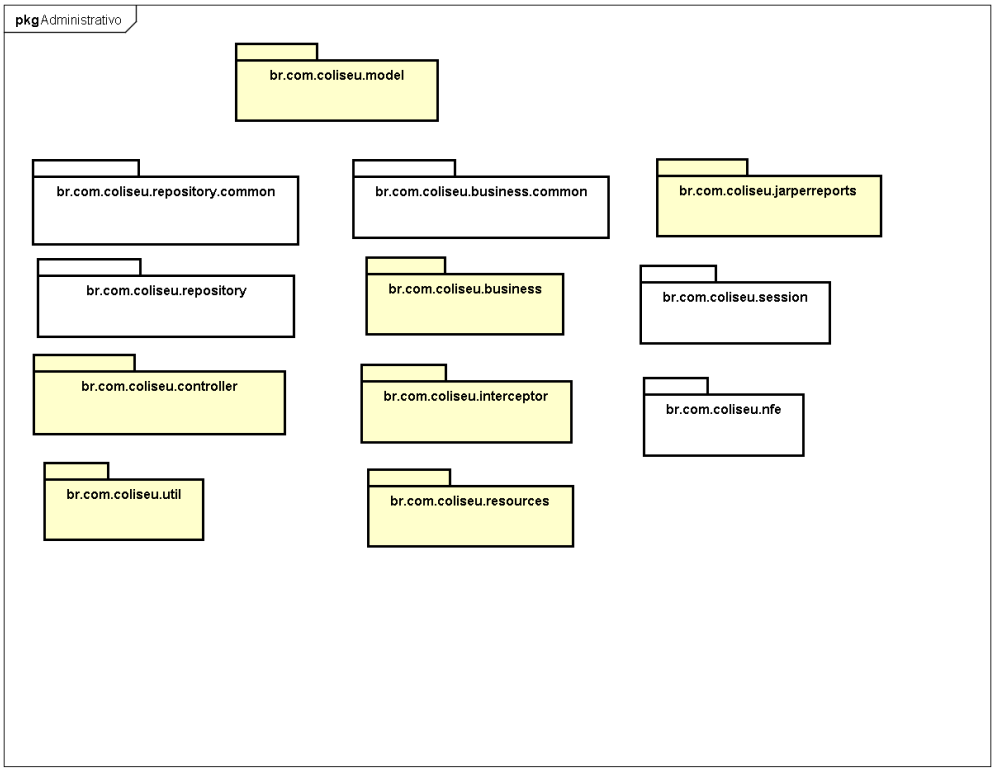

Figura 16 – Visão de componentes do módulo Administrativo

4.2.3 Classes e tabelas

São muitas as classes que compõem o sistema. Desta forma, optou-se por colocar neste documento apenas as classes do módulo Administrativo. A Figura 17 apresenta o diagrama de classes do módulo Administrativo.

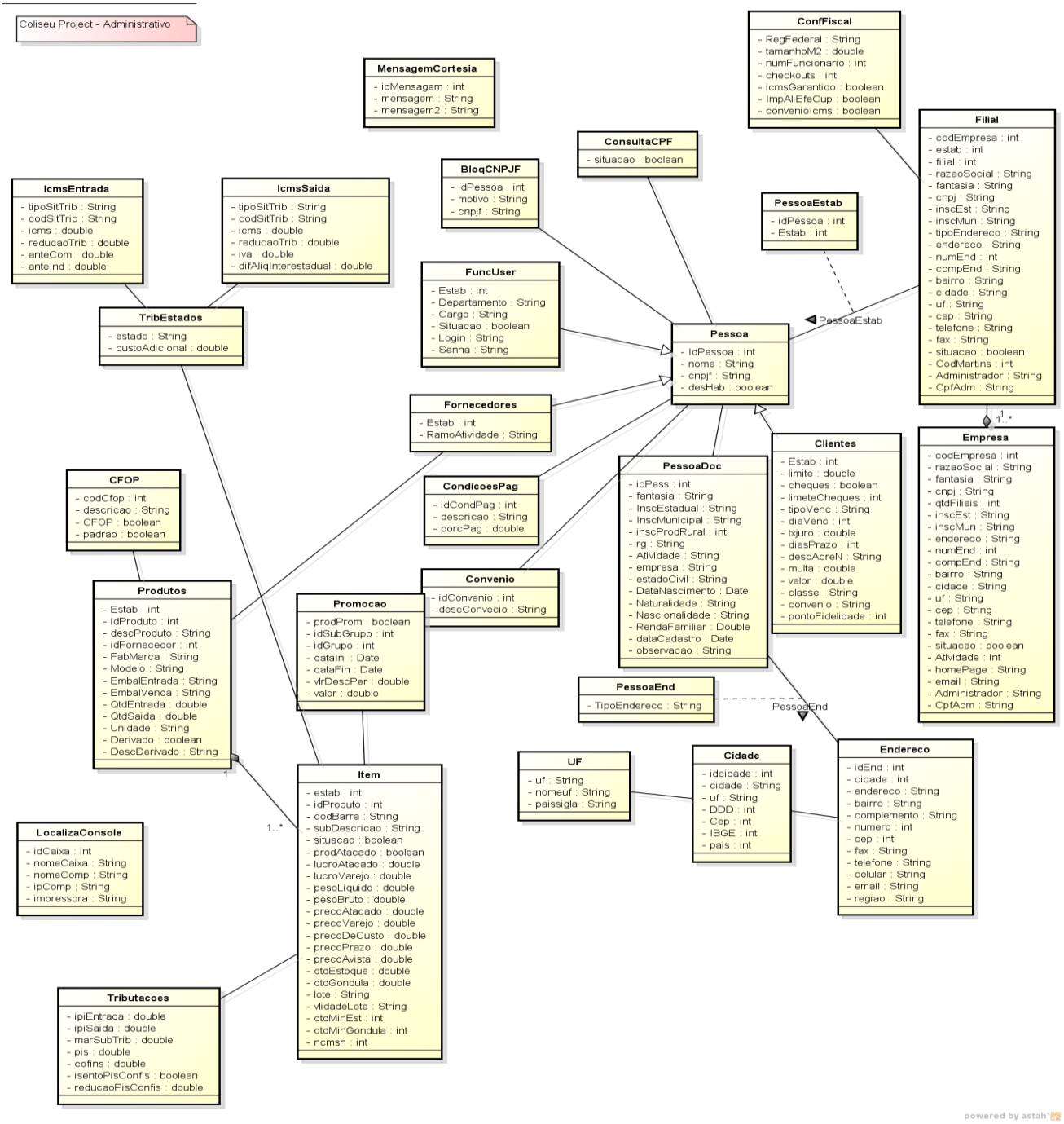

Figura 17 - Diagrama de classes do módulo administrativo

A Figura 18 apresenta o diagrama de entidades e relacionamentos do banco de dados. Apenas as entidades do módulo Administrativo estão representadas nessa figura. Esse diagrama mostra somente os campos chaves de cada tabela.

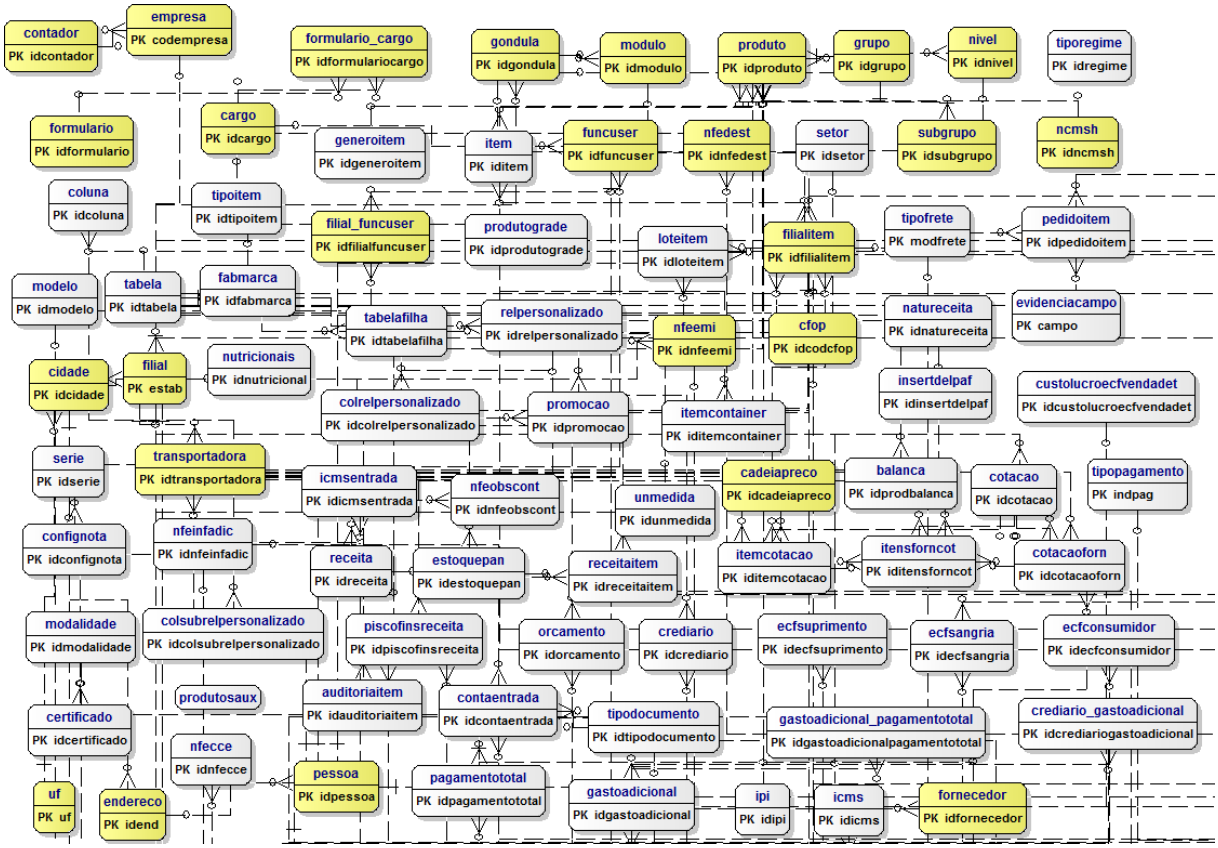

Figura 18 – Diagrama de entidades e relacionamentos – módulo administrativo

4.3 Descrição do sistema

A primeira tela que usuário visualiza ao acessar o sistema é a apresentada na Figura 19. Nessa tela são solicitados os dados de estabelecimento, nome do usuário e senha.

Figura 19 – Tela de login do sistema

Se os dados informados estiverem corretos o usuário será direcionado para a tela principal do sistema - Figura 20.

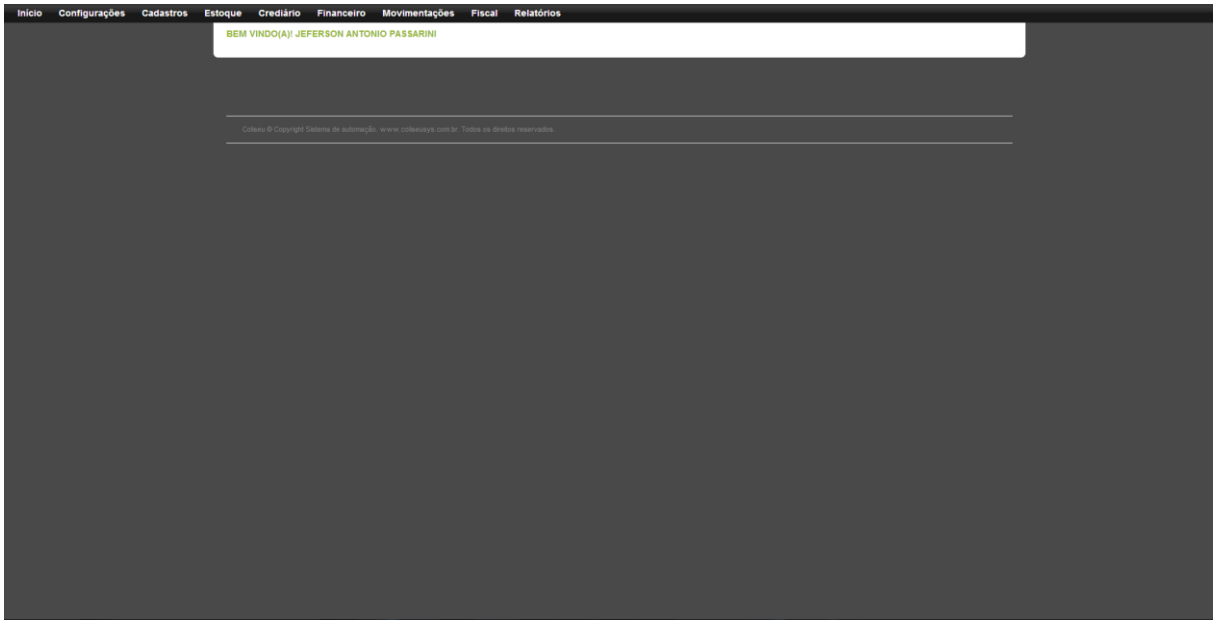

Figura 20 – Tela principal do sistema

Nas Figuras 21, 22, 23 e 24 são apresentadas algumas telas do módulo administrativo, que envolve cadastros de clientes, fornecedores, produtos, lançamento de notas de saída e de entrada.

Na Figura 20 está a tela principal do sistema. Nessa tela o usuário visualizará na parte superior da tela o menu por meio da qual terá acesso as telas de cadastros,

consultas e relatórios, dentre outras.

A Figura 21 apresenta o menu cadastro, composto dos itens Contador, Empresas, Filial, Pessoas, Localidades e Convênios, Promoções, dentre outros. Pessoas e Localidades são itens para acesso a outros cadastros.

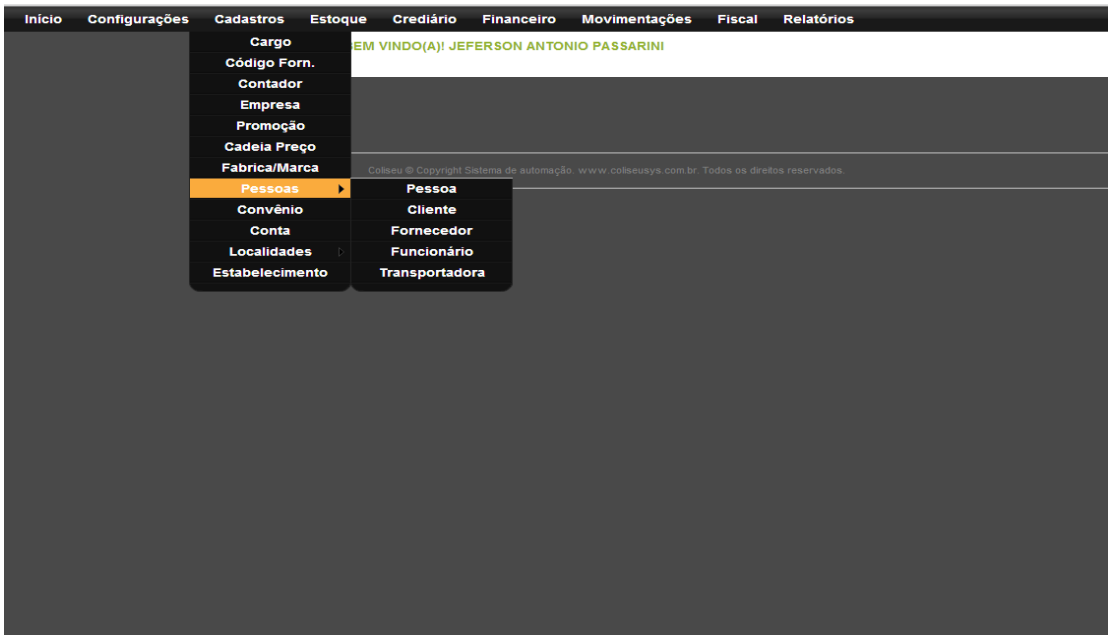

Figura 21 – Navegando no menu

A Figura 22 apresenta a tela de cadastro de produtos. Ela é organizado por grupo, para aproveitar as tributações e códigos fiscais.

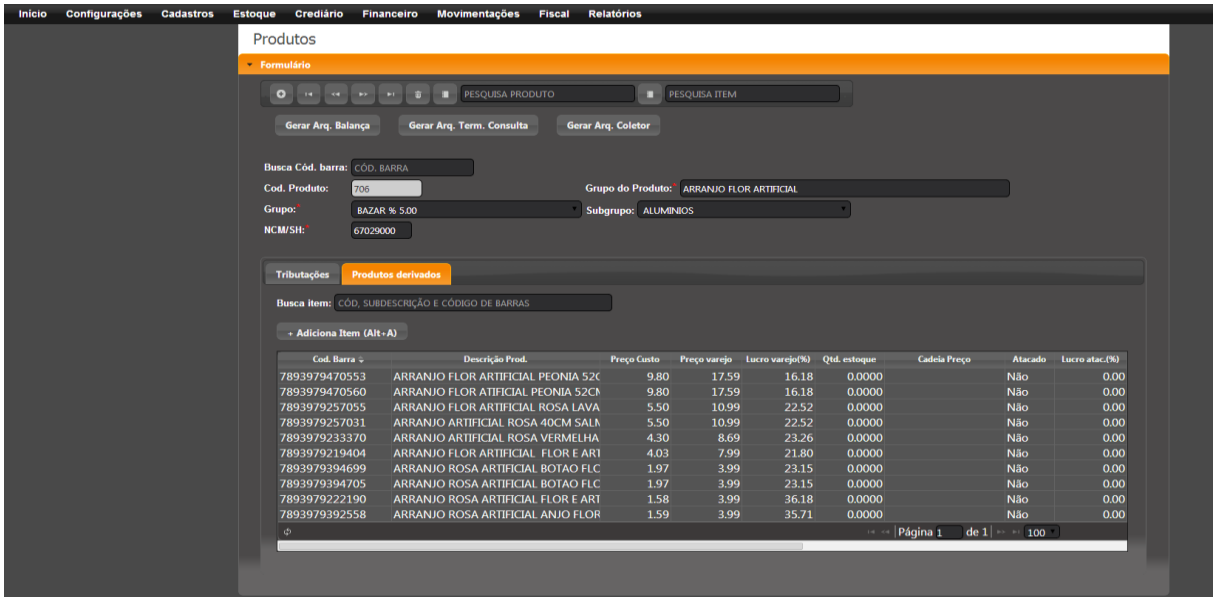

Figura 22 – Formulário de cadastro de produtos

O formulário de cadastro de produtos (Figura 22) pode ser usado para inserir novos produtos, consultar produtos, mudar tributações e vincular cadeia de preços. Na parte superior da tela de cadastro, encontra-se os botões de navegação, excluir, ir para lista, adicionar e um campo para pesquisa. Um pouco mais abaixo, botões para gerar arquivos de balança, terminal de consulta e coletor de dados.

Os campos que estão com um asterisco em vermelho no *label*, são campos obrigatórios. Caso o usuário tente salvar sem preencher esses campos, o sistema informará que eles são obrigatórios e devem ser preenchidos. Nessa tela há duas abas, a de dados tributários e produtos derivados.

Ainda na parte administrativa está a parte de lançamento de notas de saída e de entrada. A Figura 23 mostra a tela de nota de saída.

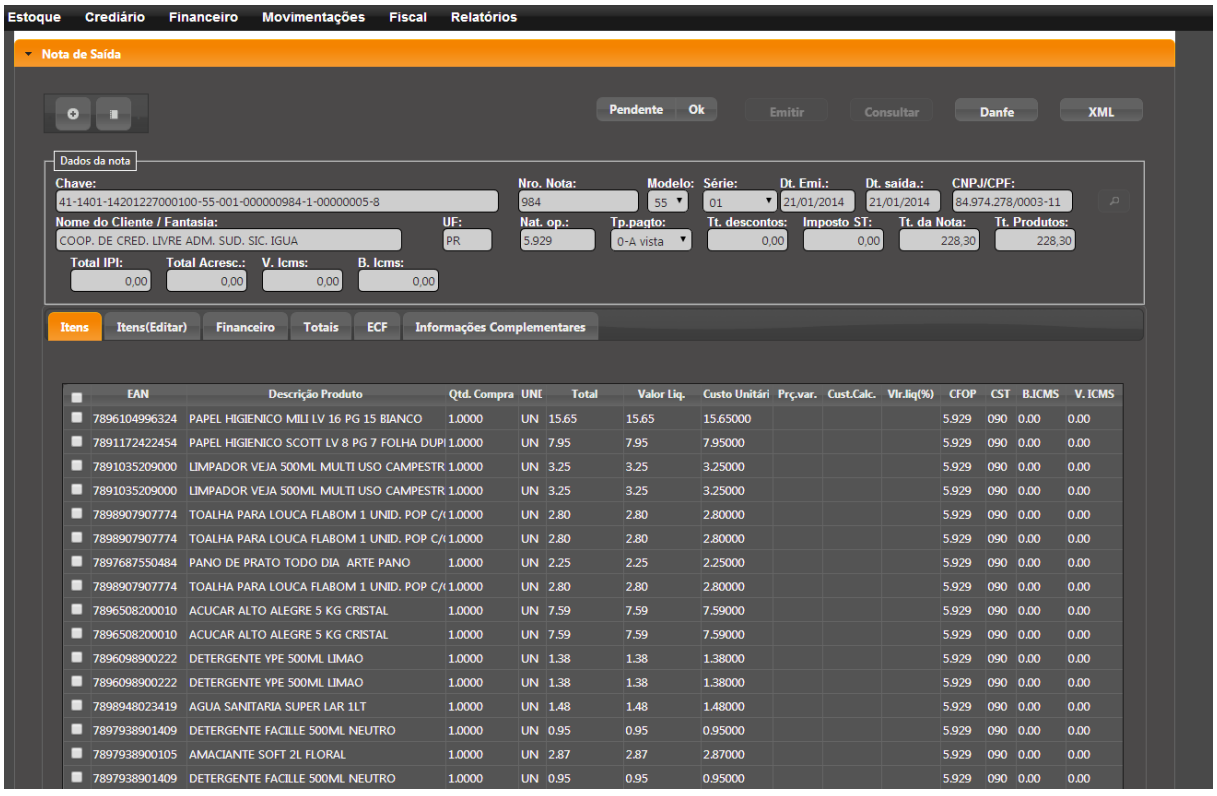

Figura 23 – Formulário de nota de saída

Na Figura 23, na parte superior, além dos botões de adicionar e listar, estão botões de emissão, consulta de nota, através deles é realizado uma requisição nos *webservices* da receita estadual e federal, também é possível imprimir o danfe (documento auxiliar de nota fiscal eletrônica) e realizar o *download* do XML da nota. Abaixo estão os dados do cabeçalho da nota e as abas para listar os itens, edita-los, financeiro para cadastrar as faturas de pagamento, totais da nota, ECF onde é

vinculado cupons fiscais e informações complementares.

A figura 24 pode-se visualizar a tela de nota de entrada, que é similar à de nota de saída, sem a aba de ECF e com a inclusão de botões para a geração de etiquetas dos produtos, na aba de Itens(Editar) é lançado os itens, o sistema controla o lançamento de acordo com a tributação e o fornecedor de cada produto, mostrado o valor de imposto e sugerindo um preço de venda, conforme a margem de lucro cadastrada para o produto, descontando as tributações e custos operacionais.

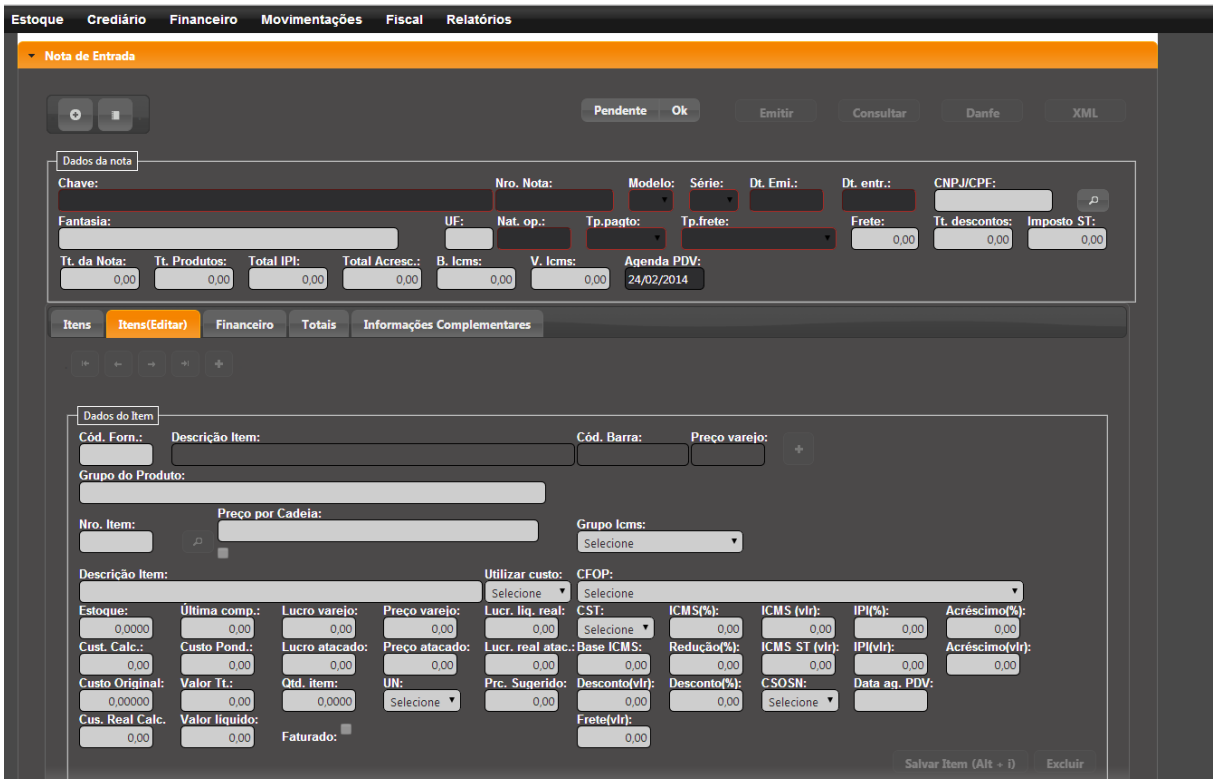

Figura 24 – Formulário de nota de entrada

Na Figura 25 apresenta a tela principal do frente de caixa, através dessa tela são realizadas as vendas, soma dos itens, visualiza-se informações do produto, comunicação com o servidor e geração dos arquivos do PAF.

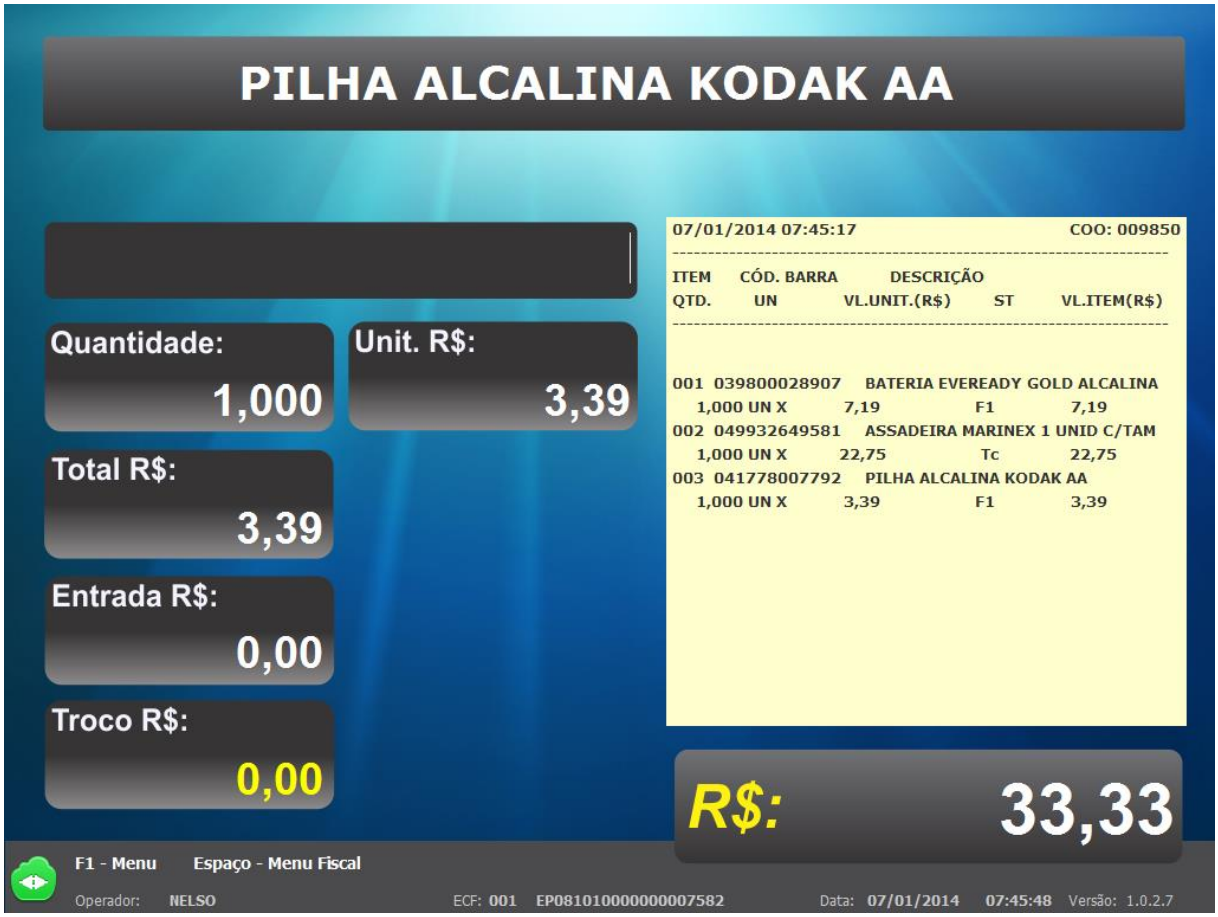

Figura 25 – Tela principal do frente de caixa

4.4 Exemplo de implementação do sistema

Nesta seção está exemplificada a implementação do sistema. O objetivo é apresentar a forma de uso das tecnologias.

Dada a amplitude do sistema, foi optado por apresentar apenas algumas partes do código, como por exemplo: faturamento de itens da nota, comunicação do frente de caixa com servidor, persistência de dados e controle de conexões da aplicação com o banco de dados.

Na Listagem 1 está o código de grande importância para a parte de faturamento da nota fiscal.

Na Listagem 2 e 3 visualiza-se parte do código que envia recebe uma lista de objetos do frente de caixa via *rest* e, chama os métodos para persistência no banco de dados, e retorna se foi efetuado com sucesso ou não.

```
@Override
public void faturaItensNota(Nfeide nfeide, String operacaoRealizar) {
    try {
        Set<Nfedetitem> nfedetitems = new HashSet<Nfedetitem>(nfeide.getNfedetitems());
        List<Filialitem> filialitems = new ArrayList<>();
        boolean marcarFaturaItens = false;
        // percorrendo todos os itens da nota
        for (Nfedetitem nfedetitem : nfedetitems) {

            Filialitem filialitem = nfedetitem.getFilialitem();
            String hql;
            Sittributariaentr sittributariaentr = new Sittributariaentr();
            // Verifica se o tipo do cfop alimenta o estoque
                hql = "from Sittributariaentr where cfop.idcodcfop = :cfop ";
                Query query = null;
                query = manager.createQuery(hql);
                query.setMaxResults(1);
                query.setParameter("cfop", nfedetitem.getCfop().getIdcodcfop());
                sittributariaentr = (Sittributariaentr) query.getSingleResult();
            //Verifica se esta faturada
            if(!nfedetitem.isFaturado()){

                // calculando os custos
                filialitem = calculaItemNota(nfedetitem);
                if(nfedetitem.isAlteraporcadeia()){
                    filialitemRepository.alteraporcadeia(filialitem);
                }
                marcarFaturaItens = true;
            }
                if (sittributariaentr.isEstoque()) {

                    if (operacaoRealizar.equals("somar")) {
                        filialitem.setQtdestoque(filialitem.getQtdestoque().add(nfedetitem.getQcom()));

                        Historicoitem historicoitem = new Historicoitem(0,
                                sessionUser.getFilial(), nfeide.getNfeemientr().getFornecedor(), nfeide.getDsaient(),
                                nfedetitem.getVuncom(), nfedetitem.getLucrovarejo(),
                                nfedetitem.getLucroatacado(),
                                nfedetitem.getPrecovarejo(),
                                nfedetitem.getPrecoatacado(),
                                nfedetitem.getLucroatacado(),
                                nfedetitem.getLucrovarejo(), nfedetitem.getQcom(),
                                nfedetitem.getCustoreal(), nfedetitem, nfedetitem
                                        .getFilialitem().getPrecovarejo());

                        manager.persist(historicoitem);
                    } else {
                        filialitem.setQtdestoque(filialitem.getQtdestoque().subtract(nfedetitem.getQcom()));
                    }
                }
                filialitems.add(filialitem);
            }
            if(marcarFaturaItens){
                marcaFaturadoItensNota(nfeide.getIdnfeide());
            }
            if (!filialitems.isEmpty()) {
                filialitemRepository.saveList(filialitems);
            }
        } catch (PersistenceException e) {
            IndexController.logger.warn(MessageException.persistenceException+ e.getMessage()+ "), <b>Causa: " +
                    "Metodo lancamentoNota class NfeideBusiness </b> "+ e.getCause());
        }
```

Listagem 1 – Parte do código que realiza faturamento da nota fiscal

O método de faturamento da nota (Listagem 1) recebe um objeto da nota chamado “nfeide” e uma *String* denominada “operacaoRealizar”, através disso é possível extrair os itens da nota do objeto “nfeide”, verificando a situação tributária de cada item e faturando, caso ainda não tenho sido. Com a *String* “operacaoRealizar” verifica-se se deve somar ou subtrair do estoque, finalizando isso é persistido no banco de dados.

```
@Override
public void setEcfvendacabecalho() {
    try {
        for (Versaopdvs versaopdvsL : versaoPdvsLocalList) {
            if (versaopdvsL.getTabelasatualiarena().getNometb().equals("ecfvendacabecalho")) {
                for (Versaopdvs versaopdvsS : versaoPdvsServerList) {
                    if (versaopdvsS.getTabelasatualiarena().getNometb().equals("ecfvendacabecalho")) {
                        if (versaopdvsL.getVersao() > versaopdvsS.getVersao()) {
                            Long totalReg = repositoryEcfvendacabecalho.getRecordEcfvendacabecalho(versaopdvsS.getDttimealter());
                            if (totalReg > 0) {
                                List<Ecfvendacabecalho> ecfvendacabecalho = null;
                                String resultado = "";
                                int j = 0;
                                for (int i = 0; i <= totalReg; i += 25) {
                                    Pageable topTen = new PageRequest(j, 25);
                                    j++;
                                    ecfvendacabecalho = repositoryEcfvendacabecalho.getListEcfvendacabecalho(versaopdvsS.getDttimealter(), topTen);

                                    Response responseEcfCab = restfulie.at("http://" + Session.getInstance().getEcfmovimento().getPdvs().getIpservidor()
                                            + ":8080/CP/ecfvendacabecalho/setEcfvendacabecalho")
                                            .accept("application/xml").as("application/xml").post(ecfvendacabecalho);

                                    resultado = responseEcfCab.getResource().toString();
                                }
                                if (resultado.equals("Sucesso")) {
                                    ultimoEcfVendaCab = ecfvendacabecalho.get(ecfvendacabecalho.size() - 1).getDttimealter();
                                    restfulie.at("http://" + Session.getInstance().getEcfmovimento().getPdvs().getIpservidor()
                                            + ":8080/CP/pdvs/atualizaVersoesServerPdv")
                                            .accept("application/xml").as("application/xml").post(versaopdvsL);
                                    falhaComun = false;
                                } else {
                                    falhaComun = true;
                                    TrataException.trataAlertaLog(resultado);
                                }
                            }
                        }
                    }
                }
            }
        }
    } catch (Exception e) {
        falhaComun = true;
        TrataException.trataErroLog(e.getMessage());
    }
}
```

Listagem 2 – Parte do código que envia os dados para o servidor

Na frente de caixa o método que realiza o envio do cupom fiscal (Listagem 2), depois de finalizado, verifica a versão do objeto que está no servidor e compara com a versão do frente de caixa, se a versão do caixa for maior que a versão do servidor, ele envia os dados a partir da data e horário da última comunicação, isso feito com uma *Thread* para interromper a continuação da venda.

```
@Post
@Consumes("application/xml")
@Path("setEcfvendacabecalho")
public void setEcfmovimento(List<Ecfvendacabecalho> ecfvendacabecalhos) {
    try {
        repositoryEcfvendacabecalho.saveList(ecfvendacabecalhos);

        result.use(representation()).from("Sucesso").serialize();
    } catch (PersistenceException ex) {
        result.use(representation()).from(ex.getMessage()).serialize();

    }

}

@Post
@Consumes("application/xml")
@Path("setEcfvendadetalhe")
public void setEcfvendadetalhe(List<Ecfvendadetalhe> ecfvendadetalhes) {
    try {
        repositoryEcfvendadetalhe.saveList(ecfvendadetalhes);

        result.use(representation()).from("Sucesso").serialize();
    } catch (PersistenceException ex) {
        result.use(representation()).from(ex.getMessage()).serialize();

    }

}

@Post
@Consumes("application/xml")
@Path("setEcftotaltipopagamento")
public void setEcftotaltipopagamento(
        List<Ecftotaltipopagamento> ecftotaltipopagamentos) {
    try {
        repositoryEcftotaltipopagamento.saveList(ecftotaltipopagamentos);

        result.use(representation()).from("Sucesso").serialize();
    } catch (PersistenceException ex) {
        result.use(representation()).from(ex.getMessage()).serialize();

    }

}
```

Listagem 3 – Parte que o recebe os dados do frente de caixa

Do lado o servidor ele recebe os dados enviado do frente de caixa (Listagem 3) e chama o método de persistência no banco de dados, se for realizado retorna uma *String* com a mensagem “Sucesso” e com isso é atualizado a versão do servidor.

Na Listagem 4 está o método genérico que realiza a persistência no banco de dados uma lista de objetos, o JPA junto com o Hibernate se responsabiliza de fazer a inserção dos dados no banco e, o VRaptor abre e fecha as conexões com o banco de dados, como apresentado na Figura 30.

```
@Override
public List<T> saveList(List<T> entity) {
    List<T> retorno = new ArrayList<>();
    try {
        manager.getTransaction().begin();

        if (entity != null) {
            Iterator<T> i = entity.iterator();
            while (i.hasNext()) {
                T savedEntity = i.next();
                savedEntity = manager.merge(savedEntity);
                retorno.add(savedEntity);
            }
        }
        manager.getTransaction().commit();
    } catch (PersistenceException e) {
        if (manager.getTransaction().isActive()) {
            manager.getTransaction().rollback();
        }
        IndexController.logger.warn(MessageException.persistenceException
                + e.getMessage() + "), <b>Causa: </b> " + e.getCause());
        throw new PersistenceException(
                MessageException.persistenceException + e.getMessage()
                        + "), <b>Causa: </b> " + e.getCause());
    }
    return retorno;
}
```

Listagem 4 – Código que realiza a persistência de uma lista de objetos

O método responsável para efetuar a persistência de dados, em forma de uma lista (Listagem 4), recebe uma lista, inicia uma transação, se a lista não for vazia transformar em um *Interator*, salva no banco de dados e para finalizar confirma a persistência, se ocorrer algum erro volta os dados.

```java
package br.com.coliseu.resources;

import javax.annotation.PostConstruct;

@Component
public class CriadorDeEntityManager implements ComponentFactory<EntityManager> {

    private final EntityManagerFactory factory;
    private EntityManager manager;

    public CriadorDeEntityManager(EntityManagerFactory factory) {
        this.factory = factory;
    }

    @PostConstruct
    public void abre() {
        this.manager = factory.createEntityManager();
    }

    public EntityManager getInstance() {
        return this.manager;
    }

    @PreDestroy
    public void fecha() {
        this.manager.close();
    }

}
```

Listagem 5 – Código que abre e fecha as conexões com o banco de dados

Através de uma injeção de dependência com a anotação na classe “@Component” (Listagem 5) o Vraptor abre a conexão com o banco de dados, e após a confirmação dos dados no banco, ele fecha a conexão.

5 CONCLUSÃO

O objetivo deste trabalho está centrado na análise, projeto e implementação de um sistema para supermercado. Devido à amplitude do sistema, a modelagem e mesmo a implementação foram organizadas em partes. Essas partes e sua organização compõem uma arquitetura, seja de tecnologias utilizadas para implementar e executar o sistema, seja de módulos de código.

A linguagem Java, agregada por *frameworks* e a biblioteca JQuery, é tecnologia escolhida para implementar o sistema. O sistema é composto de uma parte *desktop* e outra *web,* sendo a maior parte em ambiente *web*. O referencial teórico centrou-se em aplicações *web* enfatizando a aplicabilidade da linguagem Java nesse tipo de aplicação.

O desenvolvimento deste trabalho oportunizou um aprendizado considerável, tanto em termos de modelagem como de implementação utilizando a linguagem Java, especialmente com o desenvolvimento para *web*.

A modelagem de um sistema amplo, não necessariamente considerado de grande porte, é uma tarefa de muito aprendizado, especialmente se o programador não possui muita experiência com esse tipo de atividade.

Apesar da incerteza e da insegurança que são gerados quando há um cliente real, o resultado foi muito interessante. Ao mesmo tempo que ter um cliente que efetivamente usará o sistema acarreta mais responsabilidade para que se tenha considerado tudo o que é necessário. Ter alguém que criticará o que está sendo feito com interesse de uso e visão prática faz com que o aprendizado seja maior.

Desenvolver um sistema como este, em que se pode aplicar todos os conceitos referentes ao ciclo de vida do desenvolvimento de um software, é uma oportunidade de colocar teorias efetivamente em prática. E isso foi proporcionado pela realização deste trabalho, tanto em termos da modelagem do sistema como da implementação, que está aproximadamente 70% concluída. Com esse período permitiu pensar em como utilizar as tecnologias de maneira a não ocasionar inconsistências ou erros.

A realização deste trabalho auxiliou a agregar informações sobre análise e desenvolvimento do projeto de sistemas, possibilitando entender: a visão geral de um sistema, os requisitos levantados junto à usuário e clientes, diagramas de caso de uso, de classes e entidades relacionais e implementação de códigos. Além disso foram realizados minuciosos testes manuais, devido à necessidade fiscal que engloba

o sistema e também foram usados dois clientes pilotos, um supermercado de Quedas do Iguaçu (Supermercado Ubialli) e outro de Pato Branco (Supermercado Econômico Norte), com isso foi possível corrigir pequenas falhas, antes dos teste e homologação junto ao órgão de teste do PAF, que foram realizados em agosto de 2013 pela empresa LTS de Chapecó-SC (LTS, 2013). Com aprovação dos teste e homologação do *software,* foi possível a implantação em mais clientes, atualmente o projeto está em mais de 25 clientes no estado do Paraná e Mato Grosso do Sul.

REFERÊNCIAS BIBLIOGRÁFICAS

ASTAH. **Astah community**. Disponível em <http://astah.net/editions/community >. Acesso em 18/02/2014.

BOMFIM JÚNIOR, F. T. **JSP: a tecnologia para java na internet**. São Paulo: Érica, 2002.

BEVILAQUA, A. LACERDA, R. C. LIMA, S. A. **Vraptor: o framework de aprendizado java *web***, 2010, p. 1-5.

CAVALCANTI, l. **VRaptor 3**. Disponível em <http://www.infoq.com/br/articles/VRaptor3> Acesso em 22/09/2010.

CAELUM. **VRaptor 3.** Disponível em <http://vraptor.caelum.com.br/pt/>. Acesso em 15/01/2014.

CHO, E. S.; KIM, S. D.; RHEW, S. Y.; LEE, S. D.; KIM, C. G. **Object-oriented web application architectures and development strategies**. Asia Pacific Software Engineering Conference and and International Computer Science Conference 1997 (APSEC '97/ICSC '97), 1997, p. 322 – 331.

DEITEL, H.M.; DEITEL, P.J. **Java: como programar**. Porto Alegre: Bookman, 2001.

DEZIGN. **DeZign for databases**, version 6.0. Disponível em <http://www.datanamic.com/dezign/index.html>. Acesso em 13/02/2014.

DUHL J., **Rich internet applications**, IDC white papers. Disponível em <http://www.idc.com>, 2003.

ECLIPSE. **Eclipse IDE Juno**. Disponível em: < http://www.eclipse.org/juno/>. Acesso em: 12/12/2013.

FIELDS, D. K.; KOLB, M. A. **Desenvolvendo na web com java server pages**. Rio de Janeiro: Ciência Moderna, 2000.

FRATERNALLI, P., PAOLINI, P., **A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications**, In: Extending Database Tecnology (EDBT 98), p. 421-435, 1998.

FUKUDA, H., YAMAMOTO, Y. **A system for supporting development of large scaled rich internet applications**. In: 23rd IEEE/ACM International Conference on Automated Software *Engine*ering (ASE 2008), 2008, p. 459-462.

HARTKE NETO, R. **Curso de JSP: versão 1.0.** Universidade Federal de Santa Catarina. Programa Especial de Treinamento: Florianópolis, 2002

HIBERNATE, **Hibernate - JBoss Community**, disponível em <http://www.hibernate.org>, acesso em 20/09/2010.

INFOBLOG. **Hibernat**e. Disponível em <http://infoblogs.com.br/view.action?contentId=_18077&Hibernate.html>. Acesso em 10/02/2014.

JBOSS 7. **Apliction server**. Disponível em <http://www.jboss.org/jbossas>. Acesso em 09/02/2014.

JQUERY, **Jquery**. Disponível em <http://jquery.com>. Acesso em 19/02/2014.

LTS. **Laboratório de testes de software**. Disponível em < http://bell.unochapeco.edu.br/lts/?p=1676/>. Acesso em 20/02/2014.

MARQUES, A. C.; CAETANO, J. S. **Novas tecnologias na educação: reflexão sobre a prática**. Maceió: EDUFAI, 2002.

MAVEN. **Apache maven**. Disponível em <http://maven.apache.org/>. Acesso em 19/02/2014.

OLIVEIRA, E., PEREIRA, M. J. V., HENRIQUES, P. R. **Compreensão de aplicações web: o processo e as ferramentas**. 6a. Conferência da Associação Portuguesa de Sistemas de Informação. Bragança, 2005, p.1-14.

PGADMIN. **pgAdmin**. Disponível em <http://www.pgadmin.org>. Acesso em 28/12/2013.

POSTGRESQL. **Postgresql**. Disponível em <http://www.postgresql.org/about/>. Acesso em 10/02/2014.

RECEITA. **Federal**. Disponível em < http://www.receita.fazenda.gov.br/Legislacao/Decretos/Ant2001/Ant1999/Decreto70235/default.htm>. Acesso em 10/02/2014.

RICCA, F. **Analysis, testing and re-structuring of web applications**, In: International Conference on Software Maintenance (ICSM'2004), p. 474-478, 2004.

RUMBAUGH, J. et al. **Modelagem e projeto baseado em objeto**. Rio de Janeiro: Campus, 1997.

SILVA, M. S. JQuery. **A biblioteca do programador javascript**. São Paulo: Novatec 2010.

SHARMA, V.; SHARMA, R. **Desenvolvendo sites de e-commerce**. São Paulo: Makron Books, 2001.

SQL MAGAZINE. **SQL developer**. Disponível em <http://sqldeveloper.solyp.com>. Acesso em 12/12/2013.