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

TIAGO DE MELLO

SISTEMA PARA REPRESENTAÇÕES COMERCIAIS USANDO HIBERNATE E VRAPTOR

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2013

TIAGO DE MELLO

SISTEMA PARA REPRESENTAÇÕES COMERCIAIS USANDO HIBERNATE E VRAPTOR

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

Orientador:Prof. Luis Carlos Ferreira Bueno

PATO BRANCO
2013

ATA Nº: **221**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **TIAGO DE MELLO**.

Às 17:30 hrs do dia 24 de setembro de 2013, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Luís Carlos Ferreira Bueno (Orientador), Vinicius Pegorini (Convidado) e Wendel Goes Pedrozo (Convidado), para avaliar o Trabalho de Diplomação do aluno Tiago de Mello, matrícula 1172360, sob o título **Sistema WEB para Gerenciamento de Representantes Comerciais**; 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 18:35 hrs foi encerrada a sessão.

Prof. Luís Carlos Ferreira Bueno, M.Sc.
Orientador

Prof. Vinicius Pegorini, Esp.
Convidado

Prof. Wendel Goes Pedrozo, M.Sc.
Convidado

Prof. Eliane Maria De Bortoli Fávero, M.Sc.
Coordenadora do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

A melhor maneira de prever o futuro é inventá-lo.

Alan Kay

RESUMO

MELLO, Tiago de.Sistema Web para representações comerciais usando Hibernate e VRaptor. 2013. 68 f. Monografia (Trabalho de Conclusão de Curso) -Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas,Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2013.

A necessidade das empresas por aplicações que possam ser acessadas em vários lugares e por todos dentro da organização tem aumentando cada vez mais, e sendo assim a substituição de aplicações *desktop* por aplicações Web também. Com base nesse contexto, verificou-se a possibilidade de desenvolver uma aplicação Web para empresas de representação comercial que já possuem um sistema convencional, pensando na necessidade dessas empresas em ter váriospedidos de vendafeitosem várias cidades e o sistemaestar em um único lugar. O trabalho teve como base um sistema desenvolvido em Visual Basic .NET usado por duas empresas de Pato Branco, que foi refeito usando Java Enterprise Edition, VRaptor3, Hibernate e Kendo UI. Esse trabalho também tem como objetivo implantar a aplicação resultante em umambiente *cloud* para ser acessado pelas duas empresas.

Palavras-chave: Internet, Java, representações comerciais, *cloud*.

ABSTRACT

MELLO, Tiago de. Sistema Web para representações comerciais usando Hibernate e VRaptor. 2013. 68 f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2013.

The need of companies for applications that can be used in many places and by all the employees of the organization has increasing, and therefore the replacement of desktop applications for Web applications too. Based on this context, verified the possibility of developing a Web application for commercial representation companies that already have a conventional system, thinking about the necessity of these companies which has multiple sales orders made in various places and the system is in a single place. The work was based on a system developed in Visual Basic .NET used by two companies of PatoBranco, it was remade using Java Enterprise Edition, VRaptor3, Hibernate and Kendo UI. This work also aims to deploy the application resulting in a cloud environment to be accessed by both companies.

Keywords: Internet, Java,commercial representation, *cloud*.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

CRUD	*Create*, *Read*, *Update* e *Delete*
CSS	*CascadingStyleSheets*
DER	Diagrama de Entidade e Relacionamentos
EJB	*Enterprise JavaBeans*
GB	*Gigabyte*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
J2EE	*Java 2PlataformEntrepriseEdition*
JEE	*Java EntrepriseEdition*
JDBC	*Java Database Connectivity*
JSP	*Javaserver Pages*
JST	*Java System Toolkit*
LDAP	*Lightweight Directory Access Protocol*
MVC	*Model-View-Controller*
NIST	National Institute of Standards and Technology
REST	Estado Representacional de Transferência
SGBD	Sistema de Gerência de Banco de Dados
SIG	Sistema de Informação Gerencial
SOA	*Service OrientedArchitecture*
SQL	*Structured Query Language*
SSL	*Secure Socket Layer*
SWT	*Standard Widget Toolkit*
TB	*Terabyte*
UI	*User Interface*
URI	*Uniform Resource Identifier*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o que motivou o início do desenvolvimento do projeto, assim como o contexto em que foi desenvolvido. Ao longo do capítulo estão os objetivos e a justificativa desse trabalho, e por fim está a organização dos capítulos que compõe o texto.

1.1 CONSIDERAÇÕES INICIAIS

Este desenvolvimento teve como base um sistema previamente desenvolvido para duas empresas de representação comercial de Pato Branco, que atuam em segmentos distintos, o que exige configurações personalizáveis pelo usuário. O sistema legado encontra-se desenvolvido em Visual Basic .NET com banco de dados no formato Access, e deverá ser implementado com novas tecnologias para rodar em ambiente *web* neste trabalho de TCC.

Segundo CANDELORO (2009), um dos principais dilemas das empresas que utilizam serviços de representantes comercias é como controlar melhor as vendas feitas pelos seusvendedores. A fim de solucionar esse problema, um grande avanço seria a utilização de um sistema para auxílio no gerenciamento. Entretanto muitas dessas empresas não utilizam nenhum sistema ou então possuem um que não atende bem suas necessidades da empresa.

Conforme PRAHALAD e KRISHNAN (2008), devemos considerar os custos ocultos de sistemas arcaicos e inflexíveis hoje existentes na maioria das empresas e considerar o enorme potencial de adotar um sistema mais moderno. Sendo assim, o novo modelo do sistema terá que trazer melhorias e a possibilidade de rodar em ambiente *web,* oferecendo assim maior mobilidade aos representantes e empresas.

Atualmente grande parte dos sistemas para representantes comerciais rodam em ambiente *desktop*, e as opções para *web* já existem, mas são incompletas. Em geral, os sistemas já existentes disponibilizam apenas cadastros e relatórios. O objetivo do sistema a ser desenvolvido é juntar mais funcionalidades, como gerar e exibir gráficos para facilitar a análise dos dados da empresa e também

uma atenção ao processo interno da empresa com status de inativos das entidades, status de andamento das vendas e status do pagamento das comissões.

Tendo como base um sistema já existente e as deficiências dos atuais, vislumbrou-se apossibilidade de desenvolver um sistema de informação que permita o gerenciamento de representantes comerciais e automação de vendas para autônomos e/ou empresas de pequeno e médio porte.

Para que o sistema rode em ambiente Web,foiutilizado a linguagem Java EE(*Enterprise Editon*) com banco PostgreSQL. Para o desenvolvimento do sistema foram utilizados conceitos da linguagem de modelagem UML(*UnifiedModelingLanguage*). A mesma foi usada para especificar, construir e visualizar as classes do sistema. Também aplicados os conceitos da estrutura MVC(*ModelViewControl*) através do frameworkVRaptor. Com base nesses conceitos foi implementados os princípios de abstração, herança e encapsulamento, além de agilizar a codificação, garantir a boa estruturação do código e facilitar a manutenção.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um sistema computacional utilizando o ambiente de desenvolvimento Java EE e banco de dados PostgreSQL, para o gerenciamento de representantes comerciais e automação de vendas para autônomos e/ou empresas de pequeno e médio porte.

1.2.2 Objetivos Específicos

Para atender o objetivo geral deste trabalho, pretende-se:

- Estudar o programa já existente, verificar as mudanças necessárias para rodar em ambiente *web* e melhorias para satisfazer as necessidades das empresas;
- Criar Diagrama de Entidade e Relacionamento para modelar os dados que serão manipulados pelo *software*;
- Criar Diagrama de Classes com base na UML para modelar classes e relacionamentos;
- Desenvolver um *software* de gerenciamento de representantes comerciais e automação de vendas, utilizando o ambiente de desenvolvimento Java EE e banco de dados PostgreSQL. Codificação dos cadastros de classes, clientes, comissões, fábricas, produtos, transportadoras, vendedores, despesas, pedidos e representantes, além de relatórios e gráficos para consulta dos dados cadastrados.
- Configurar um ambiente Jelastic e implantar, nesse ambiente, o sistema Web desenvolvido.

1.3 JUSTIFICATIVA

O Presentetrabalho tem como objetivo final a migração de um *software* já estabelecido em ambiente *desktop* para Web, resultando na modernização do sistema legado, proporcionando assim mobilidade paraas empresas que trabalham com representantes comerciais.

O processo de desenvolvimento deste sistema é longo e complexo, e nesse foram aplicados diversos conceitos e conhecimentos adquiridos ao longo do curso de graduação, entre os mais importantes estão: desenvolvimento Web, sistemas distribuídos, criação e gerenciamento de banco de dados, análise de sistemas e conceitos de orientação a objeto.

O sistema legado teve o seu desenvolvimento iniciado em meados de 1993 com a versão 4.0 do Visual Basic, passando ainda pelas versões 5.0 e 6.0 e por fim migrado para oVisual Basic .NET. Mesmo que a estrutura de dados, formato de telas, regras de negócio e leiautes de relatórios foram usados como base no

desenvolvimento, o código-fonte teve que serrefeito, pois a estrutura de código-fonte não foi alterada durante as atualizações de versões e existe uma grande diferença entre os padrões de desenvolvimentos usados no Visual Basic 4.0 para o desenvolvimento de aplicações Web.

Com a pretensão do sistema de rodar em ambiente Web, foram utilizadas durante o seu desenvolvimento tecnologias já consolidadas e que atendem as necessidades da aplicação. A versão *EnterpriseEditor* do Java para a camada de controle, pela robustez, padrões de códigos e desenvolvimento, além de não possuir custos. Também foi escolhido um *framework* de MVC sem custo e brasileiroVRaptor, pela baixa complexidade, agilidade no desenvolvimento e suporte para grandes aplicações. O sistema foi disponibilizado pelo ambiente Jelastic, que é uma plataforma Java na nuvem lançada no começo de 2013 e tem uma série de configurações e servidores instalados no Brasil. O banco de dados utilizado foi o PostgreSQL, que também não possui custos e é um banco robusto, além deser compatível com as outras tecnologias utilizadas, principalmente o Jelastic.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos, dos quais este é o primeiro. O segundo apresenta o referencial teórico que fundamenta os conceitos usados no desenvolvimento do projeto. O capitulo 3 apresenta os matérias e método utilizadona construção da aplicação. No quarto capítulo estão os resultados do trabalho, onde é apresentado o sistema resultante assim como: a sua modelagem, implementação e discussões sobre a implementação. No capítulo 5 está a conclusão e por fim as referências utilizadas.

2SISTEMAS DE INFORMAÇÃO GERENCIAL PARA INTERNET

Este capítulo apresenta os conceitos relacionados a sistemas de informações gerencias de modo de geral e especificamente para internet que é foco do sistema construído. Também mostra alguns conceitos relacionados a sistemas *multi-tenancy*(veja os tópicos 2.3.1 e 2.3.2 para maiores informações sobre esse modelo), conceito importante que foi usado no desenvolvimento da aplicação desenvolvida.

2.1 SISTEMAS DE INFORMAÇÃO GERENCIAL

Segundo Stair (2001), um Sistema de Informação é um conjunto de elementos ou componentes inter-relacionados que recebem dados, os processam, propagamdados e informações, e também fornecem uma forma de *feedback*. O principal uso desses sistemas é para processar uma grande quantidade de dados a fim de se obter informações de forma rápida, para isso três etapas são necessárias: a entrada de dados, o processamento desses e a saída para exibi-los.

Atualmente as empresas vêm investindo cada vez mais em Tecnologia da Informação, que é uma grande aliada delas na organização estratégica, tática e operacional da empresa conforme os estudos de Größleret al. (2006)

Sistemas de Informação podem ser desenvolvidos para diversos fins. Os sistemas que focam no apoio a tomada de decisões por parte da gerencia de uma empresa são denominados Sistemas de Informação Gerencial (SIG). Conforme a definição de Pereira etFonseca (1997, p. 241), SIG “são mecanismos de apoioà gestão, desenvolvidos com base na TI e com suporte da informática para atuarcomo condutores das informações que visam facilitar, agilizar e otimizar o processodecisório nas organizações”.

A função da gerência de uma organização é tomar decisões que levem ao sucesso,para isso quanto maior o número de informações disponíveis, maior a probabilidade de decisões acertadas. A combinação de informações preciosas sobre a empresa com procedimentos analíticos em cima desses, resulta em decisões

muito mais acertadasdo que as tomadas apenas com baseno julgamento de executivos experientes e informados (McGee, 1995).

Em suma, Oliveira (2002, p.54) cita vantagens para uma empresa no uso de um SIG são:

- Melhoria na tomada de decisões;
- Melhoria no acesso a informações;
- Descentralização da tomada de decisões na organização;
- Melhoria no fluxo de informações;
- Melhorianas projeções;

2.2SISTEMAS DE INFORMAÇÃO PARA INTERNET

O mundo atual está organizado em sistemas de redes, sobretudo a Internet, a qual é considerada a principal delas, o seu tamanho e importância aumentam à medida que dispositivos, pessoas e organizações se incorporam a ela. Através dessa rede é possível acessar uma variedade de informações, entretenimento, serviços e também aplicações. Para as organizações a *internet* trás várias vantagens, como: reduzir custos de comunicação, desenvolver produtos com base nela, aumentar a fidelidade, desenvolver novos mercados e gerar novas fontes de receitas. (O'BRIEN, 2006)

Com a evolução da *internet* e do uso de tecnologias pelas empresas, o uso de aplicações que possam ser acessadas por meio da *internet* vem se tornando cada vez mais frequente. Esses sistemas, também denominados de aplicações Web vêm tendo um crescimento grande e cada vez mais sistemas que rodam em ambiente *desktop* vem sendo remodelados para ambiente Web.

Segundo Kappel*et al*(2004): "Uma Aplicação Web é um sistema de software baseadoem tecnologias e padrões do World Wide Web Consortium (W3C) que provê recursosespecíficos de Web, como conteúdo e serviços através de uma interface de usuário, oBrowser Web".

Do ponto de vista empresarial, as aplicações Web possuem um importante papel,por permitir transações de dados em tempo real em diferentes localidades,

vários usuários com diferentes papéis e acessos ao sistema e tendo um único sistema rodando de forma centralizada o que simplifica os processos de manutenção e atualização dentro da empresa.

Uma aplicação Web tem aspectos similaresa um site e a uma aplicação tradicional, mas é importante enfatizar que existem diferençasem relação a uma aplicação convencional e também a um site, tais como: aparência, objetividade, interação com o usuário e forma de acesso. Em poucas palavras uma aplicação Web possui características de uma aplicação convencional e outras de sites, e mesmo que um site e um sistema tenham recursos muito diferentes eles devem funcionar juntos e de forma harmoniosa numa aplicação Web.

Segundo Lowe e Henderson-Sellers (2001), a aparência e objetividade de aplicações Web têm uma importância muito maior do que em aplicações convencionais, sendo que é importante que o usuário sinta-se a vontade pra permanecer na página e consiga acessar todas as funcionalidades sem trabalho ou demora. Se o visual não for autoexplicativo, intuitivo e consistente o usuário tende a deixar o sistema rapidamente ou interromper o uso, independente das funcionalidades e recursos funcionarem corretamente.

Do ponto de vista da engenharia de software as aplicações Web e aplicações convencionais são similares, porémno desenvolvimento de aplicações Web existe a preocupação comoutros aspectos além dos usuais, como o ambiente que será executado, o acesso dos usuários, a segurança dos dados, navegação personalizada entre usuários e também a necessidade de oferecer uma interface mais limpa e objetiva.

A diferença entre uma aplicação Web e um site se dá no fato de que uma aplicação Web enfatiza os mesmo aspectos relacionados à interface e apresentação dos sites, mas também nos aspectos de um sistema de informação como o processamento de dados.

Uma aplicação Web pode ser vista como uma mesclagem de uma aplicação tradicional com um site, reunindo aspectos de ambos. Dessa forma a aplicação Web se torna mais completa e complexa, por reunir em um mesmo projeto as necessidades de processamento de dados de um sistema convencional e os recursos visuais chamativos de um site.

2.3*CLOUD COMPUTING*

O termo computação em nuvem surgiu em 2006 em uma palestra de Eric Schmidt, da Google, sobre como a empresa gerenciava suas centrais de dados. Hoje, computação em nuvem, se apresenta como o cerne de um movimento de profundas transformações do mundo da tecnologia. (TAURION, 2009)

O *NationalInstituteof Standards and Technology*(NIST)define a computação em nuvem como um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis que podem ser adquiridos e liberados de forma ágil, simples e sem interação com o provedor de serviços.

Segundo o NIST, *CloudComputing* é dividido em três modelos de serviço:

- Software como Serviço (*Software as a Service* - SaaS): o recurso oferecido é uma aplicação que roda por meio de um navegador de *internet*;
- Plataforma como Serviço (*Plataform as a Service* - PaaS): o recurso oferecido é uma plataforma para aplicações adquiridas ou criadas que usem linguagens e ferramentas suportados pelo fornecedor do serviço;
- Infraestrutura como Serviço (*Infraestructure as a Service* - IaaS): o recurso oferecidos são recursos de infraestrutura, tais como rede e armazenamento.

2.3.1Software como Serviço

No modelo convencional de aplicações para internet, as companhias utilizam umsistemaem sua própria infraestrutura esão responsáveis por todas as atividades de manutenção, integridade, escalabilidade, disponibilidade, encargos relacionados ao gerenciamento de TI e custo de *hardware* e *software* necessários para manter o sistema funcionando corretamente na empresa. (NITU, 2009)

Partindo de uma linha oposta ao modelo convencional, surgiu o conceito de SaaS (Software como Serviço). Neste modelo, a funcionalidade da aplicação é oferecida através de um modelo de assinatura pela Internet. Sendo assim, o cliente não possui ou administra o *software*, apenas aluga e o acessa remotamente.

Segundo Chong e Carraro (2006),o conceito de SaaS não é recente, mas só ganhou força por volta de 2005 com as melhorias dos recursos para o bom funcionamento dessa tecnologia. Para o desenvolvimento desse modelo, alguns fatores devem ser levados em consideração. As aplicações desse conceito podem ser dividas em quatro níveis de maturidade (apresentados mais abaixo). Na Figura 1 estão exemplificados esses níveis, mostrando três clientes que acessam um mesmo sistema, mas em cada nível o sistema foi implantado de uma forma diferente.

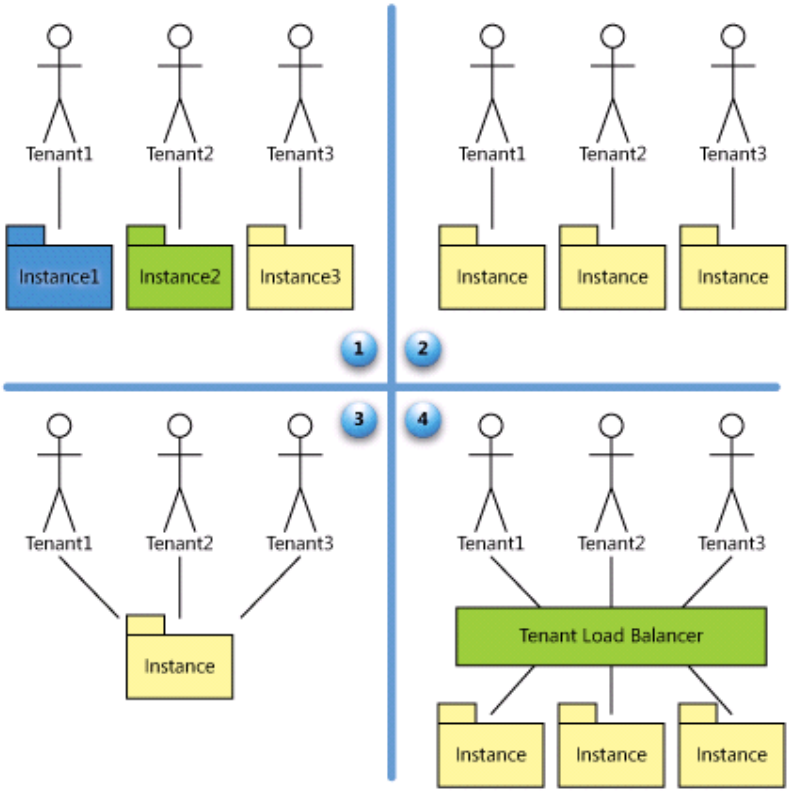

Figura 1 – Níveis SaaS(CHONG e CARRARO, 2006)

No nível 1 (Personalizado), cada cliente tem uma versão única e personalizável do sistema rodando para o uso único dele. (Chong e Carraro, 2006)

No nível 2 (Configurável), existe apenas uma versão do sistema. Para cada cliente existe uma versão separada rodando e o sistema continua sendo apresentado de forma única para cada cliente, no entanto nesse caso é feito através

de configurações e não por versões separadas do sistema como no nível 1. (Chong e Carraro, 2006)

No nível 3 (Configurável com vários *tenants*), o sistema continua sendo configurável, mas nesse caso existe apenas uma versão do sistema rodando e todos os clientes compartilhando dos mesmos recursos de equipamentos. Esse compartilhamento de recursos é transparente para o cliente, no qual o sistema deve identificar pelo cliente conectado quais os dados e configurações que devem ser exibidos. (Chong e Carraro, 2006)

O nível 4 (Escalonável e configurável com vários *tenants*) representa uma evolução do nível anterior. Continua sendo um único sistema rodando para vários clientes, mas nesse caso os recursos são escalonáveis entre os clientes. Nesse modelo os dados dos clientes são mantidos separados e cada cliente tem os seus próprios recursos, visando garantir que o sistema funcione com o mesmo desempenho com um ou com milhares de clientes conectados. (Chong e Carraro, 2006)

2.3.2Sistemas *Multi-tenancy*

*Multi-tenancy*é uma abordagem organizacional para sistemas Saas. O princípio dessa arquitetura é um *tenant* que é um cliente do sistema, onde uma única instância do sistema será disponibilizada para vários clientes (*tenant*). Para um sistema funcionar dessa forma, é necessário o desenvolvimento de uma arquitetura específica para esse fim.(BEZEMER e ZAIDMAN, 2010)

Um *tenant* é um cliente que tem acesso ao sistema, este tem acesso a uma aplicação com as suas configurações e seus dados. Entretanto, a aplicação é única mesmo que para cada cliente fique a impressão de que a aplicação em execução é acessada apenas por ele. É importante diferenciar o conceito de *tenant* do conceito de usuário, um *tenant* faz referência a uma organização que aluga o sistema e um usuário é quem se conecta ao sistema, sendo que todos os usuários de uma organização tem acesso aos dados e configurações do *tenant* da organização.

Sistemas *multi-tenancy*também devem ser diferenciados de sistemas multiusuários, em sistemas multiusuários todos compartilham os mesmo dados entre

si de uma instância do sistema e cada usuário pode ter acesso apenas a algumas partes do sistema. No modelo *multi-tenancy*o usuário tem acesso ao sistema completo, mas apenas dentrodas configurações e dados do *tenant*. Todavia um sistema *multi-tenancy* pode também ser multiusuário, e nesse caso um usuário teria acesso aos dados da sua organização e sobre os dados da organizaçãoteriam as permissões deacesso. (KWOK et al, 2008)

É importante também diferenciar aplicações *multi-tenancy* de aplicações *multi-instance*. Nesse outro modelo o resultado é o mesmo, mas existem várias instancias da mesma aplicação rodando uma para cada cliente e no modelo *multi-tenancy* é uma única instância. (KWOK et al, 2008)

Finalizando aqui este capítulo, que foi o referencial teórico do trabalho usado para o desenvolvimento deste. O próximo capitulo irá apresentar os materiais e o método usado na construção do sistema.

3 MATERIAIS E MÉTODO

Neste capítulo estão detalhados os materiais e o método que foram usados para o desenvolvimento deste trabalho. Os materiais fazem referencia as tecnologias, ferramentas e *frameworks* usados no processo de desenvolvimento do sistema. E o método apresenta as principais atividades desenvolvidas nas etapas de construção do sistema.

3.1 MATERIAIS

Os seguintes materiais foram utilizados ao longo da construção do sistema:

a) Java Enterprise Edition (JEE, 2013) – linguagem de programação usada na codificação do sistema, na versão 7.

b) Vraptor(VRAPTOR, 2013) – *framework* MVC (ModelViewController) usado na codificação do sistema, na versão 3.

c) Hibernate (HIBERNATE, 2013) – *framework* de persistência usada na codificação do sistema, na versão4.0.1.

d) Kendo UI (KENDO, 2013) – *framework javascript* usada na codificação do sistema, na versão*Community*3.1114.

e) Eclipse (ECLIPSE, 2013) –IDE(*IntegratedDevelopmentEnvironment*) usada na codificação do sistema, na versão Juno (4.2).

f) PostgreSQL (POSTGRESQL, 2013) – banco de dados utilizado, na versão 9.4.

g) Apache Tomcat 7.0 (TOMCAT, 2013) – servidor web utilizado para rodar a aplicação, na versão 7.0.42.

h) Jelastic (JELASTIC, 2013) – servidor *cloud* usado para rodar o sistema em ambiente real.

i) Astah (ASTAH, 2013) – programa usado para a modelagem do sistema, na versão*Professional* 6.7.0.

j) Power Architect (ARCHITECT, 2013) – programa usado para a modelagem do banco de dados, na versão *Community*.

k) MVC (*Model*, *View*, *Controller*) – conceito usado no desenvolvimento.

3.1.1 Java Enterprise Edition

Conforme Sampaio (2011), o Java Enterprise Edition é uma das especificações do Java e sendo essa especifica para atender as necessidades dos desenvolvedores deaplicações distribuídas cooperativas.

Para Sampaio (2011), o modelo de programação Java Enterprise Editioné baseado em *containers*, que oferece toda uma infraestrutura para o desenvolvimento de aplicações distribuídas cooperativas. Desta forma, o desenvolvedor já inicia o desenvolvimento com vários recursos prontos,e assim reduz o tempo de desenvolvimento, bem como os riscos do projeto e problemas de manutenção.

Para Sampaio (2011) as aplicações corporativas do modelo Java EE podem ser vistas como um modelo de *tiers* (ou camadas com separação de conceitos)

- Na camada cliente tem o *Web browser* ou *applets*, podendo ter também aplicações Java rodando dentro do Java EE *client container*;
- Na camada de apresentação (ou *Web Tier*) temos os componentes *Web* da arquitetura Java EE, como *JavaserverPages*(JSP), *Servlets*e *JavaServer Faces*;
- Na camada de negócios são rodados os componentes remotos que implementam regras de negócio ou representam entidades de dados, os *Enterprise JavaBeans*e *Web Services*;
- Na camada EIS (*EnterpriseInformation Systems*) tem os servidores de recursos corporativos, como Servidores de Banco de Dados ou *Mainframes*. Normalmente, são consumidos utilizando as *interfacesJava DataBaseConnectie Java ConnectorArchiteture*.

Segundo Ambler e Jewell (2002), o Java Enterprise Editiontem como missão fornecer uma plataforma independente, portável, multiusuário, segura e padronizada de classe corporativa para instalações do lado do servidor escrita na linguagem Java.

Alguns motivos que fazem os desenvolvedores optarem pela linguagem Java:

- Por ela ser orientada a objetos e ser uma linguagem madura, que fará com que as alterações no sistema sejam extremamente fáceis e rápidas, desde que se sigam as boas práticas e recomendações sobre *design* orientado a objetos;
- A enorme quantidade de bibliotecas gratuitas para realizar os mais diversos trabalhos é um ponto fortíssimo para a adoção do Java. Desta forma não é necessário comprar um componente específico, que costuma ser caro;
- O uso do Java é interessante em aplicações que virão a crescer, para a qual a legibilidade do código é importante, onde temos muita conectividade e existem muitas plataformas heterogêneas.

3.1.2 Vraptor3

O *VRaptor*é um *framework* MVC (*Model*, *View*, *Controller*) para *Web*, ou seja, uma biblioteca que fornece um conjunto de classes para auxiliar o desenvolvimento de aplicações com base na arquitetura MVC. (FREIRE*atal*, 2013)

Segundo Cavalcanti (2009), o VRaptor utiliza o padrão de arquitetura MVC em Java focado no desenvolvimento rápido, simples e na fácil manutenção do código. Para maior produtividade, algumas boas práticas são aplicadas, como convenção sobre configuração, injeção de dependências e um modelo Estado Representacional de Transferência (REST).

O VRaptoré uma iniciativa brasileira, de código-aberto, sem custos e possui ampla documentação em português. Para iniciar o desenvolvimento, basta baixar o projeto em branco (https://code.google.com/p/vraptor3/downloads/list) que já possui todas as configurações necessárias. No site do VRaptor são disponibilizados 22 guias (http://vraptor.caelum.com.br/pt/docs/guia-de-um-minuto/) que explicam e exemplificam variados recursos do *framework*, os guias começam mostrando como baixar e rodar o projeto em branco e vão gradualmente explorando recursos mais avançados e complexos.

Segundo Antonio e Ferro (2009):

> "O *frameworkVRaptor* é um controlador MVC desenvolvido por alunos da Universidade de São Paulo (USP) para *web* focado no desenvolvimento ágil. Através da inversão de controle e injeção de dependências, ele elimina o tempo de trabalho que seria perdido com o código repetitivo: validações, conversões, direcionamentos e *lookups*.Ele se adapta perfeitamente para trabalhar com o JSP na camada de apresentação e com *Hibernate* na persistência. Totalmente baseado em anotações do Java 5.0. As anotações garantem uma maneira simples de trabalhar com programação para a *Web*". (ANTONIO; FERRO, 2009, p. 8).

Atualmente o VRaptor é mantido pela Caelum (CAELUM, 2013), mas como se trata de um projeto de código aberto também conta com o apoio de uma extensa comunidade que auxilia no seu desenvolvimento. A Caelum é uma instituição de ensino focada na tecnologia Java que oferece diversos cursos focados em áreas relacionadas a desenvolvimento de sistemas. No site da instituição são disponibilizadas apostilas abertas para todo o público, e uma delas é focada no desenvolvimento usando VRaptor e Hibernate (http://www.caelum.com.br/apostila-vraptor-hibernate/).

O VRaptor encontra-se hoje na versão 3, e mantém uma organizada estrutura de suporte, treinamentos e documentação. Seus benefícios são listados, pela Caelum (2013), como:

- Alta Produtividade: por ser simples e intuitivo;
- Curva de Aprendizado: por ser rápido para aprender sobre o *framework* e começar o desenvolvimento;
- Testabilidade: por ser possível escrever um código modularizado e desacoplado do VRaptor;
- Economia: por focar em alta produtividade e assim economizando horas de trabalho;
- Flexibilidade: possibilidade de utilizar outros *frameworks* em conjunto;
- Soa e Rest – Ready: por facilitar o desenvolvimento para aplicações *REST*[1] *(RepresentationalStateTransfer)*ou orientadas a serviço;
- Melhores práticas de desenvolvimento: por utilizar conceitos que organizam e deixam o código mais limpo.
- Documentação em Português: por possuir uma ampla documentação, assim como fóruns e listas de discussão.

3.1.3 Hibernate

Segundo Antonio e Ferro (2009):

> "*Hibernate* é um *framework* que tem alto desempenho para persistência de objetos usando o modelo relacional e também serviços de consulta (*query*). O modelo de desenvolvimento do *Hibernate* deixa que o programador desenvolva a camada de persistência utilizando o paradigma orientado a objetos, incluindo associação, herança, polimorfismo, composição e *collections*. *Hibernate* proporciona ao desenvolvedor a capacidade de expressar queries em sua própria extensão de SQL (HQL), da mesma maneira que comandos em SQL, ou com um critério de orientação a objetos." (ANTONIO; FERRO, 2009, p. 7).

Diferente de outros *frameworks*, a proposta do *Hibernate* não é esconder o poder do SQL e sim fornecer uma maneira fácil de integrar suas aplicações orientadas a objetos em Java e .Net.

Esta ferramenta provê altodesempenho, escalabilidade, produtividade e flexibilidadepara o desenvolvedor. O *Hibernate* pode ser usado com a grande maioria dos bancos de dados atuais e com os principais servidores de aplicação. E possui diversos recursos, como:

- Opções de fácil uso para formular *queries* dinâmicas e sofisticadas;
- Controle e gerenciamento externo das transações;
- Um sistema avançado de cache que melhora odesempenho do sistema, guardando alguns dados na memória do servidor.
- Acessa qualquer banco de dados através driver *Java DatabaseConnectivity (*JDBC) e também permite acesso a bancos de dados não relacionais;

O uso do *Hibernate* no projeto se deu devido a algumas vantagens da sua utilização:

- Reduzir o tempo de desenvolvimento e o custo direto;
- Reaproveitamento de código;
- Plataforma de código aberto e de uso gratuito;

---

[1]Abreviação de "REpresentationalStateTransfer" é um conjunto de princípios que definem como Web Standards,

Para o desenvolvimento da camada de persistência utilizando *Hibernate*, foram criadas classes Java de acordo com as tabelas existentes no banco de dados. Estas classes posteriormente serão utilizadas para as ações de inclusão, alteração e listagem de dados direto da base. E todas as ações posteriormente realizadas serão responsabilidade do Hibernate.

3.1.4 Kendo UI

Na parte do desenvolvimento de *interface* foi utilizado o *frameworkJavascript*, chamado *KendoUI* (*User Interface*).

KendoUI é um pacote de bibliotecas prontas disponibilizadas para os desenvolvedores de páginas *Web* dinâmicas, utilizando o *Javascript*, *jQuery* e CSS (*CascadingStyleSheets*). Apesar de usar *jQuery* como base no desenvolvimento, ela pode ser denominada um *frameworkJavascript*, sendo que oferece funcionalidades básicas, como arrastar e soltar ou um sistema de *layoutsJavascript*. Possuindo também temas visualmente agradáveis. (ALVAREZ, 2013)

Esse *framework* possui alguns temas já existentes que podem ser livremente usados e alterados, como também disponibiliza uma ferramenta para que possam ser criados novos temas que atenda os padrões de cores do site a ser desenvolvido.

Essa biblioteca foi escolhida pela facilidade no desenvolvimento, pois já conta com vários componentes criados, além de eventos específicos para esses componentes. Ela também oferece estilos prontos, validações na parte do cliente pelo próprio *framework*. Utiliza-se dos padrões de desenvolvimento mais modernos, com base na nova tecnologia HTML5, oferece suporte a todos os navegadores atuais e também aos navegadores para *smartphones*.

A biblioteca é disponibilizada de duas formas distintas, uma comercial e paga, e uma gratuita e de código aberto. A versão comercial além de componentes para *Web* também oferece pacotes para o desenvolvimento de aplicações para

---

HTTP (*Hypertext Transfer Protocol*) e URIs (*UniformResourceIdentifier*) devem ser usados.

desktop e *smartphones*. E a versão gratuita conta apenas com os pacotes *Web*, que foi a utilizada no desenvolvimento na versão 3.1114.

Um ponto crucial na escolha de um *frameworkWeb* é a compatibilidade com os principais navegadores. Durante o desenvolvimento de um CRUD (*Create*, *Read*, *Update* e *Delete*) básico foram testadas as principais funcionalidades do *framework*, assim como os diferentes componentes, e todos rodaram perfeitamente no *InternetExplorer* (versão 7 e superiores), *Safari*, *Chrome*, *Firefox* e *Opera*, e nos navegadores de dispositivos móveis: *Dolphin*, *OperaMini* e *Firefox*.

No desenvolvimento de *interface* foi utilizado o HTML5 (*HyperTextMarkupLanguage*) com os conceitos e padrões mais modernos no desenvolvimento. E como se fez necessário o uso de alguns componentes como *grids* e menus, esses componentes usados foram os do *framework javascriptKendoUI*. Esse *framework* oferece uma variedade de componentes para interfaces de usuário incluindo menus dinâmicos, gráficos, painéis, grades de dados, árvores, janelas, sistemas de *upload* de arquivos, e outros. São 13 tipos diferentes de componentes, com funcionalidades avançadas, documentação completa e exemplos para desenvolvimento.

3.1.5 Eclipse

O Eclipse é uma IDE (*IntegratedDevelopmentEnvironment*)para desenvolvimento em Java. Uma ferramenta completa e robusta, além de ser sem custos e de código aberto o que lhe torna amplamente utilizada para o desenvolvimento de aplicações para *internet*.IDE é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de *software*.

Serson (2007) afirma que, *Eclipse* é uma IDE de código aberto para a construção de programas de computador criado pela IBM. E hoje o *Eclipse* é a IDE mais utilizada no mundo.

O *Eclipse* possui como características marcantes (Serson, 2007):

- O uso de SWT (*Standard Widget Toolkit*) e não do *Swing* como biblioteca gráfica;
- Forte orientação ao desenvolvimento baseado em *plug-ins*;

- Amplo suporte ao desenvolvedor com centenas de *plug-ins* que procuram atender às diferentes necessidades de diferentes programadores.

ConformeBurnette (2006), o *Eclipse* é controlado por uma organização sem fins lucrativos independente, chamada *Eclipse Foundation*. E o autor também alega que o *Eclipse* funciona nos sistemas operacionais atuais mais populares, incluindo Windows, Linux e Mac OS.

3.1.6 PostgreSQL

O PostgreSQL é um SGDB (Sistema Gerenciados de Banco de Dados) livre e sem custos que é amplamente utilizado em conjunto com aplicações Java por ser simples, multiplataforma e sem custos. (DEVMEDIA, 2013)

Segundo Lobo (2008), os SGBD's foram criados para melhorar o gerenciamento dos dados, oferecendo melhorias significativas em relação aos arquivos de armazenamento. Os SGBD's são verdadeiros sistemas que têm a função de armazenar os dados e, também, oferecer mecanismos para o gerenciamento das informações.

De acordo com Leite (2007):

> "Desenvolvido na Universidade de Berkeley - Califórnia (EUA) - e iniciado em 1986, o projeto *Postgre* deu origem ao banco de dados *PostgreSQL* em 1994 com a incorporação da linguagem SQL ao produto inicial[2]. Entre os atuais servidores de banco corporativos do tipo *open source* (de livre distribuição) e de código aberto, o *PostgreSQL* vem se destacando rapidamente. É um servidor que gerencia bancos com uma capacidade de armazenamento quase ilimitada; vai depender do espaço em disco disponível. E, apensar de seu caráter de *software* livre (o que muitas vezes carrega uma certa dose de preconceito) o *PostgreSQL* vem sendo uma séria alternativa a servidores de banco de proprietários. Sendo um banco de distribuição livre, é desenvolvido por uma comunidade de profissionais da área de banco de dados". (LEITE, 2007).

As principais características do *PostgreSQL* são as seguintes (Leite, 2007):

- Banco de dados do tipo objeto-relacional;
- É de livre distribuição e com código aberto;

---

[2] O ancestral do *PostgreSQL* foi o *Ingress* (1977-1985), também desenvolvido na Universidade de Berkeley. Por isto, muitos dizem que *PostgreSQL* significa "*após o Ingress, acrescido de instruções SQL*".

- Capacidade ilimitada para armazenamento de dados;
- Cada tabela pode armazenar até 16TB[3] (cada campo até 1GB[4]);
- Pode ser usado em várias plataformas (Linux, Unix, Windows);
- Conta com conexões SSL (*Secure Socket Layer*);
- Contempla*triggers*, herança, sequências, *storedprocudures*etc.;
- Compatível com linguagens *opensource*, além de *drivers* para outras.

O PostgreSQL possui o conceito de *schemas*que são espaços lógicos que podem se comportar de forma semelhante a bases de dados distintas por poderem ter suas próprias tabelas, índices e afins. Mas, um conjunto de *schemas* faz parte de uma mesma base de dados. O Conceito de *schemas* foi bastante útil para a aplicação desenvolvida porutilizar o conceito de *multi-tenancy*, sendo que a aplicação foi desenvolvida para vários clientes e cada um terá um *schema*distinto com os seus dados.

3.1.7 Apache Tomcat 7.0

O Tomcat é um servidor de aplicações Web que foi usado para rodar a aplicação. Watson (2008) afirma que com o aparecimento da *Web* como plataforma padrão para fornecer aplicações para usuários finais, surge a necessidade de servidores de aplicações. O autor define um servidor de aplicações como um substituto do *software* cliente-servidor tradicionalmente instalado nos computadores dos usuários finais. Sendo assim ele executa as aplicações de forma centralizada, apresentando-as para os usuários em janelas exibidas localmente nos navegadores *Web*. E as aplicações utilizam os dados armazenados em um ou mais servidores de banco de dados.

Um servidor *Web* tem quatro funções básicas (STOUT, 1997):

- Servir páginas *Web*;
- Rodar programas de interconexão (*gateway*) e retornar seus resultados;

---

[3] TB - *Terabyte é* a quantidade da capacidade de armazenamento.
[4] GB - *Gigabyte*, assim como o *Terabyte*, porém com capacidade inferior.

- Controlar o acesso ao servidor;
- Monitorar e registrar estatísticas de acesso ao servidor.

Segundo Watson (2008):

> "As aplicações *Web* podem ser desenvolvidas com várias tecnologias, hoje predominando a Java. As aplicações escritas em Java devem ser compatíveis com a padrão J2EE (*Java 2 Enterprise Edition*), que define como elas devem se empacotadas e disponibilizadas. O J2EE e padrões relacionados são controlados pela *Sun Mycrosystems*e aceitos por praticamente todos os desenvolvedores de software." (WATSON, 2008).

Visto a citação apresentada acima, o servidor de aplicações selecionado para a arquitetura deste trabalho foi o *Tomcat*, pois ele foi totalmente desenvolvido em Java, logo, ele possui todas as características inerentes a um programa Java, pode ser executado em qualquer sistema operacional, como Linux, Windows, Mac OS Solaris, suportam aplicações que usam *Vraptor*, é um *software* livre e isento de custos.

De acordo com 4Linux (2013), o *Tomcat* é um servidor de aplicações capaz de rodar aplicações baseadas em *Servlets*, JSP (*Java Server Pages*), JSTL (*Java Standard Tag Library*), JSF (*Java Server Faces*) e outras tecnologias para aplicações *Web* previstas pelo Java EE.

O *Tomcat* traz recursos topo de linha como (4Linux, 2013):

- *Clustering* ativo-ativo;
- Suporte a *Webservices*;
- *Pools* de conexão a bancos de dados e integração a serviços de diretório LDAP (*LightweightDirectory Access Protocol*), de modo que hoje não existem razões técnicas para se optar por uma solução proprietária.

3.1.8 Jelastic

Jelastic é um servidor nas nuvens para Java que foi usado para implantar o sistema. Segundo JELASTIC (2013), Jelastic é uma das mais novas plataformas para Java nas nuvens. Essa plataforma é simples de usar e possui recursos escaláveis.

O Jelastic é uma plataforma para provedores de hospedagem. Existem diversos provedores que oferecem os serviços dessa plataforma, no Brasil a empresa provedora do serviço é aWebSolute (WEBSOLUTE, 2013). Para usar o serviço é necessário criar uma conta sem custos num dos provedores, que fornecem o acesso por um período de testes de 30 dias e após esse período é necessário pagamento para continuar o uso deste.

A forma de pagamento é diferente de outros serviços similares, sem mensalidade fixas e com base nas configurações escolhidas e trafego de dados. É oferecido um painel de controle por onde é configurado o ambiente, adicionado aplicações e também realizado os pagamentos. Nesse painel podem ser configuradosum ou vários ambientes com um valor mínimo e máximo de memória, qual o servidor de aplicações, qual o banco de dados e qual a URL do ambiente. Com base nas configurações escolhidas é definido um custo mínimo e máximo mensal para o ambiente criado, que pode ser alterado a qualquer momento. O pagamento é feito por um controle pré-pago de créditos, onde são adicionados créditos à conta e esses são descontados diariamente. O ambiente tem um custo mínimo e é esse o custo quando a aplicação não for acessada em nenhum momento, e um custo máximo se a aplicação for acessada com grande frequência 24 horas por dia. Se a aplicação estiver parada, então não existe custo algum.

O custo mais acessível e não fixo, assim como a compatibilidade com os outros materiais usados no desenvolvimento foram os principais fatores para a escolha deste servidor.

3.1.9Astah

Astah foi usada para a criação do diagrama de classe. Através dela é possível criar modelagens com base na UML (*UnifiedModelingLanguage*). Essa ferramenta possui recursos de geração de código-fonte em Java com base nos diagramas criados, assim como criação dos diagramas com base no código-fonte em Java. Os diagramas criados podem ser exportados em formato imagem.

O Astah é a ferramenta sucessora de uma famosa ferramenta chamada Jude. Possui várias versões para distribuição, a vesão*Community* é sem custos e

possui recursos limitados. A versão *Professional* é completa e possui um período *trial* de 20 dias. Existem também versões para Windows, Linux e Mac.

3.1.10 Power Architect

É uma ferramenta para banco de dados, que permite trabalhar de forma gráfica com um modelo de dados. Possui recursos como: geração de código SQL com base em Diagramas de Entidade-Relacionamento, criação de Diagramas de Entidade-Relacionamento com base em banco já existentes, compatibilidade com um grande número de SGBDs (Sistema de Gerenciamento de Banco de Dados), arrastar e soltar, conexão com várias bases de dados simultâneas, salvar como projeto ou em formato XML.

Essa ferramenta foi usada para a geração do banco de dados, e também do Diagrama de Entidade-Relacionamento. É livre e possui versão sem custos ou com recursos mais avançados. Também versões para *desktop* e servidores, assim como Windows, Mac e Linux.

3.1.11 MVC

Segundo Jobstraibizer (2009), o conceito MVC, que traduzido significa Modelo, Visão e Controle, separa em três camadas uma aplicação.

MVC é um conceito de engenharia de software para desenvolvimento e design, que propõe a separação de uma aplicação em três partes: modelo, visão e controle. O modelo é usado para definir e gerenciar o domínio da informação, a visão é responsável por exibir os dados ou informações da aplicação e o controle controla as duas camadas anteriores, exibindo a interface correta ou executando algum trabalho complementar para a aplicação. (Gonçalves, 2007)

Na camada *Model*, temos os modelos da aplicação, no qual serão definidas suas propriedades e atributos. Também é o local onde deve ser incluída a camada

de persistência (DAO - *Data Access Object*). Basicamente essa camada trata as definições e acessos relacionados a banco de dados.

A camada *View*representa a interface com o usuário, ela é a responsável pela apresentação do *Model*. Uma boa prática é de que nessa camada exista essencialmente para definir o leiaute da aplicação e que não possua lógicascomplexas, a fim de facilitar manutenções, e assim a *interface* poderá ser alterada por qualquer *web designer* sem influenciar no restante da aplicação.

A camada *Controller* processa e responde a eventos, que podem ser ações do usuário ou do sistema, fazendo interações com o *Model*. Toda a lógica da aplicação é feita nessa camada, por exemplo, validações e atribuições. Ou seja, temos aqui toda a regra de negócio da aplicação.

Para exemplificar, segundo Sampaio (2007):

> "Imagine um aplicativo que acrescenta nome e *email* das pessoas a um *grid* (uma tabela com múltiplas colunas). Vamos ver o papel dos três elementos:
> - *View*: Apresenta um *grid* contendo as colunas "nome" e "*e-mail*", além de um formulário com campos para que o usuário acrescente novas pessoas ao *grid*. Haverá um botão que, ao ser acionado, solicitará ao *Controlle*r a adição da pessoa ao *grid*.
> - *Model*: Armazena a lista de pessoas e possui métodos para adicionar, remover ou simplesmente obter a lista de pessoas.
> - *Controller:* Recebe a ação do usuário na*View,* atualiza o*Model*e manda a *View*atualizar sua exibição, de modo que a pessoa apareça no *grid".* (SAMPAIO, 2007, p. 80).

Esse conceito visa dividir o desenvolvimento de sistemas em camadas separadas que trabalham em conjunto formando um projeto completo. O uso desse conceito auxilia o processo de desenvolvimento, por propor uma melhor organização de código-fonte. Segundo Singh*atal* (2002), o MVC é o padrão de arquitetura de software mais recomendado para aplicações interativas.

3.2 MÉTODO

Os procedimentos para definir o sistema estão baseados nas fases propostas por Pressman (2002) para o modelo sequencial linear que são análise, projeto, codificação e testes. A seguir detalhamos as fases definidas para este trabalho:

a) Levantamento de requisitos –Para o levantamento de requisitos foi analisado o sistema na sua versão para *desktop* e também realizado entrevistas com as duas empresas de representação comercial de Pato Branco que utilizam o sistema. Com base no estudo do sistema e nas entrevistas, foi possível levantar e documentar os requisitos para o início da construção do projeto.

b) Análise – Durante a análise foram verificadas as mudanças necessárias nas telas, regras e comportamento do sistema legado para o funcionamento em ambiente Web, assim como algumas mudanças solicitadas pelo cliente. Essas mudanças foram descritas para posteriormente serem utilizadas na codificação.

c) Projeto –Foram criados dois diagramas para a documentação e visualização das entidades do sistema: diagrama de classes e Diagrama de Entidade-Relacionamento.

d) Implementação (codificação) do sistema –Para a codificação do sistema foi seguido modelo da arquitetura MVC (*ModelViewControl*). Utilizado o *framework*VRaptor para a camada de controle, Hibernate para a camada de modelagem e Kendo UI para a camada visual. Foram criadas classes genéricas e abstratas para facilitar a codificação da parte de controle.

A documentação da biblioteca Kendo UI foi muito útil em vários aspectos para auxiliar na documentação e poder utilizar vários recursos e componentes a fim de deixar o sistema visualmente mais agradável, assim como mais dinâmico ao usuário. Como o sistema será usado por duas empresas e estará rodando em um único local, é necessário que o sistema acesse duas bases de dados distintas com base na empresa do usuário que realizou o acesso, para isso foi necessário usar recursos dos*frameworks*Hibernate e VRaptor, como guardar componentes na sessão do usuário, pegar a URL digitada pelo usuário, criar conexões com o banco de dados em tempo de execução.

e) Realização dos testes –Os testes realizados foram através da revisão de código-fontee testes unitários nas telas do sistema após a fase de codificação ter sido concluída.

f) Implantação do sistema –O sistema foi implantado em ambiente *cloud* e disponibilizado para as duas empresas que irão utilizá-lo. O sistema será único e rodará em um servidor *cloud*, assim disponível para as duas empresas e todos os seus vendedores. O banco de dados também está no mesmo ambiente, sendo que cada empresa tem a sua própria base com seusdados.

4RESULTADOS

Este capítulo apresenta o sistema obtido como resultado do trabalho.

4.1 DESCRIÇÃO DO SISTEMA

O sistema é uma nova versão para ambiente Web de um sistema *desktop*. O sistema legado foi desenvolvido em Visual Basic.NET e já utilizado por duas empresas de representação comercial de Pato Branco. O foco é auxiliar a representação comercial a ter um controle maior sobre os seus pedidos, assim como quem comprou, quem vendeu e comissões geradasde forma remota e acessível de qualquer lugar.

Os cadastros contidos são de clientes, vendedores, transportadoras, produtos, classes de produtos, usuários, fábricas, comissões, pedidos e despesas. Todos esses cadastros possuem uma tela de listagem com opções de ordenação e filtro, assim como a geração de relatório nos formatos pdf ou xls. Todos os cadastros são necessários para o controle dos pedidos de venda do sistema, assim como as comissões para cada pedido.

Os usuários são de dois tipos, o vendedor e o administrador. Os vendedores são os responsáveis por ter o contato direto com o cliente e assim gerar pedidos de venda e os administradores são os responsáveis por toda a representação. Dessa forma, os vendedores têm acesso apenas aos dados dos seus clientes e dos seus pedidos. E os administradores têm acesso a todas as telas do sistema. Esses usuários são cadastrados pelo sistema e possuem um *login* e senha de acesso.

O pedido é o cadastro principal do sistema, ele representa um pedido de venda realizado por um cliente da representação. Esse pedido pode ser cadastrado no sistema por um vendedor ou por um usuário administrador da representação. O vendedor só pode ver os seus pedidos, cadastrar novos pedidos e passar o status para “Em Revisão”. Já os administradores podem ver todos os pedidos e podem passar o pedido para qualquer status, respeitando as regras de cada etapa. Um pedido é cadastrado com o status “Em edição” e passa por uma série de status ate

ser fechado, mas apenas pode ter seus dados alterados quando “Em edição”, “Em Revisão”, “Reprovado” ou “Reprovado pela fábrica”.

Com as edições feitas o status é passado para “Em Revisão” onde os administradores fazem as revisões e mudampara “Reprovado” ou “Revisado”, se for reprovado poderá ser alterado pelo vendedor novamente e depois passado para “Em Revisão”, e se for revisado indica que o pedido está correto.Depois de revisado o pedido pode ser passado para “Enviado para fábrica”, quando o usuário do sistema tem a opção de enviar o pedido de venda para fábrica por e-mail. Após ser enviado, o pedido ainda pode passar para “Reprovado pela fábrica” ou “Aprovado pela fábrica”, quando aprovado o pedido está concluído e quando reprovado esse pode ser alterado e enviado novamente. Depois que um pedido é aprovado pela fábrica, o sistema se encarrega de calcular os valores das comissões do vendedor e da representação conforme as configurações de comissões do sistema. Ainda existe o status “Cancelado” que pode ser atribuído a um pedido a qualquer momento, mas depois de atribuído este não poderá mais ser alterado.

O sistema conta com algumas configurações para a geração de comissões que são definidas pelos administradores conforme as necessidades de cada empresa. No cadastro de fábricas existe o valor da comissão que a própria fábrica paga em cima dos pedidos de venda, e em cada item de um fábrica também tem um valor de comissão. No qual algumas fábricas tem uma comissão para todos os itens e outras para cada item em específico. Assim, o usuário pode deixar o valor 0 (zero)para o valor de porcentagem da fábrica ou do item, e ao realizar o cálculo das comissões os valores iguais a 0 (zero) serão ignorados. Além das comissões da fábrica, cada vendedor também tem um valor específico para as comissões de uma fábrica ou de um item de uma fábrica.

Para o pagamento de comissões existe uma configuração no sistema, que permiteduas formas de pagamentos: por nota ou por pedido. Quando um pedido é gerado, o cliente pode pagar em uma única vez ou em várias. Quando for comissões por nota, a comissão será paga aos vendedores apenas quando o cliente realizar um pagamento, uma vez que o valorda comissão será calculado com base no valor da parcela paga pelo cliente. E no caso de comissões por pedido, o vendedor tem o valor total da comissão repassado pela representação sempre que o pedido for fechado (mesmo que ainda não tenha sido pago).

Para o controle e facilidade da representação existem relatórios e consultas específicas para os administradores do sistema. Na consulta de cliente, o usuário pode verificar todas as compras de um cliente, quanto tempo está sem comprar, valor total comprado e valor total comprado num período. Na consulta de pedidos, os pedidos num período, valor dos pedidos, valor das comissões, comissões da fábrica, comissões de um vendedor e também gerar um recibo para o acerto mensal da representação com os vendedores.

A aplicação foi desenvolvida para rodar em ambiente *cloud*, onde várias empresas podem se conectar a mesma aplicação rodando no mesmo lugar.Para acessar a aplicação a empresa possui um endereço único, onde a última parte do endereço é o nome da empresa cliente do sistema. Por exemplo, se o nome da empresa forsuper-pedidos, o endereço de acesso basicamente assim: www.sistema-unico-rodando.com.br/super-pedidos, onde o que identifica a empresa é o que vem depois do "/" (barra) e o endereço pode ser qualquer um que seja configurado no ambiente da aplicação. Para acessar o sistema o cliente precisa saber o endereço e possuir um usuário e senha, todos fornecidos por um administrador daaplicação. Esse endereço pode ser um domínio que o cliente possua e que deseje utilizar ou um genérico para qualquer empresa. O administrador irá criar uma nova base com o nome do cliente e será o mesmo nome que irá ser usado na última parte do endereço, sem a necessidade de parar a aplicação. Para o primeiro acesso é fornecido um usuário com uma senha simples, e cabe ao cliente alterar a senha desse usuário pelo sistema.

4.2 MODELAGEM DO SISTEMA

O diagrama de classes é a documentação necessária para a criação de todas as entidades do sistema, nesse diagrama estão os nomes das classes e todos os atributos e métodos das classes. Durante a codificação as classes geradas devem seguir o que está modelado nesse diagrama.

A figura 2 contém o diagrama de classes do sistema. Esse é o diagrama simples, apenas com os nomes e os relacionamentos das entidades do sistema.

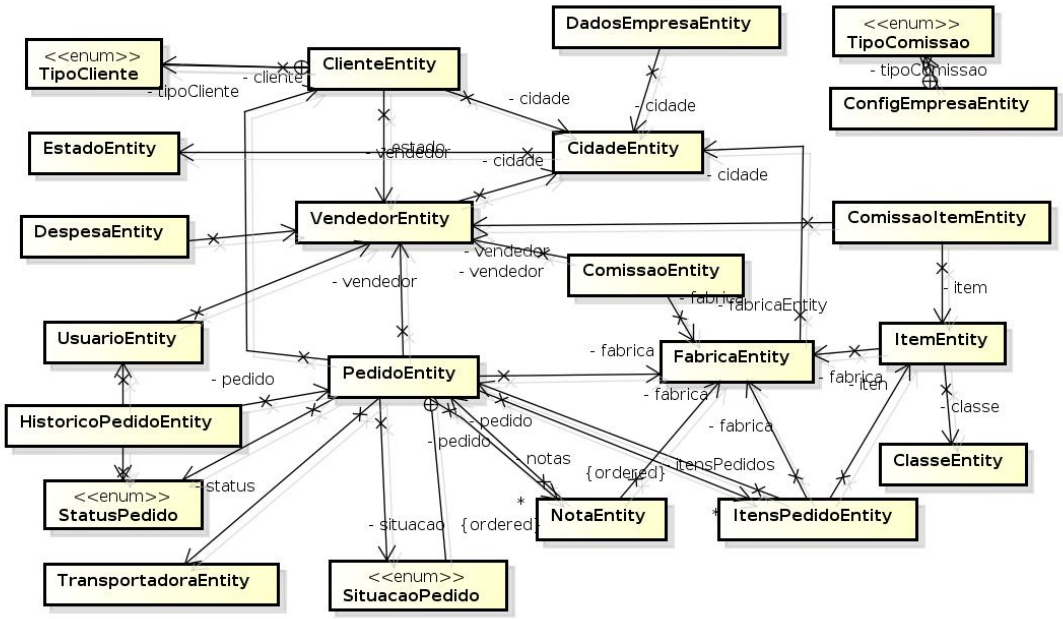

Figura 2 - Diagrama de classes simples

A figura 3 contém o diagrama de classes do projeto. Esse é o diagrama completo, mostrando todas as propriedades e função das entidades do sistema e também o relacionamento entre essas.

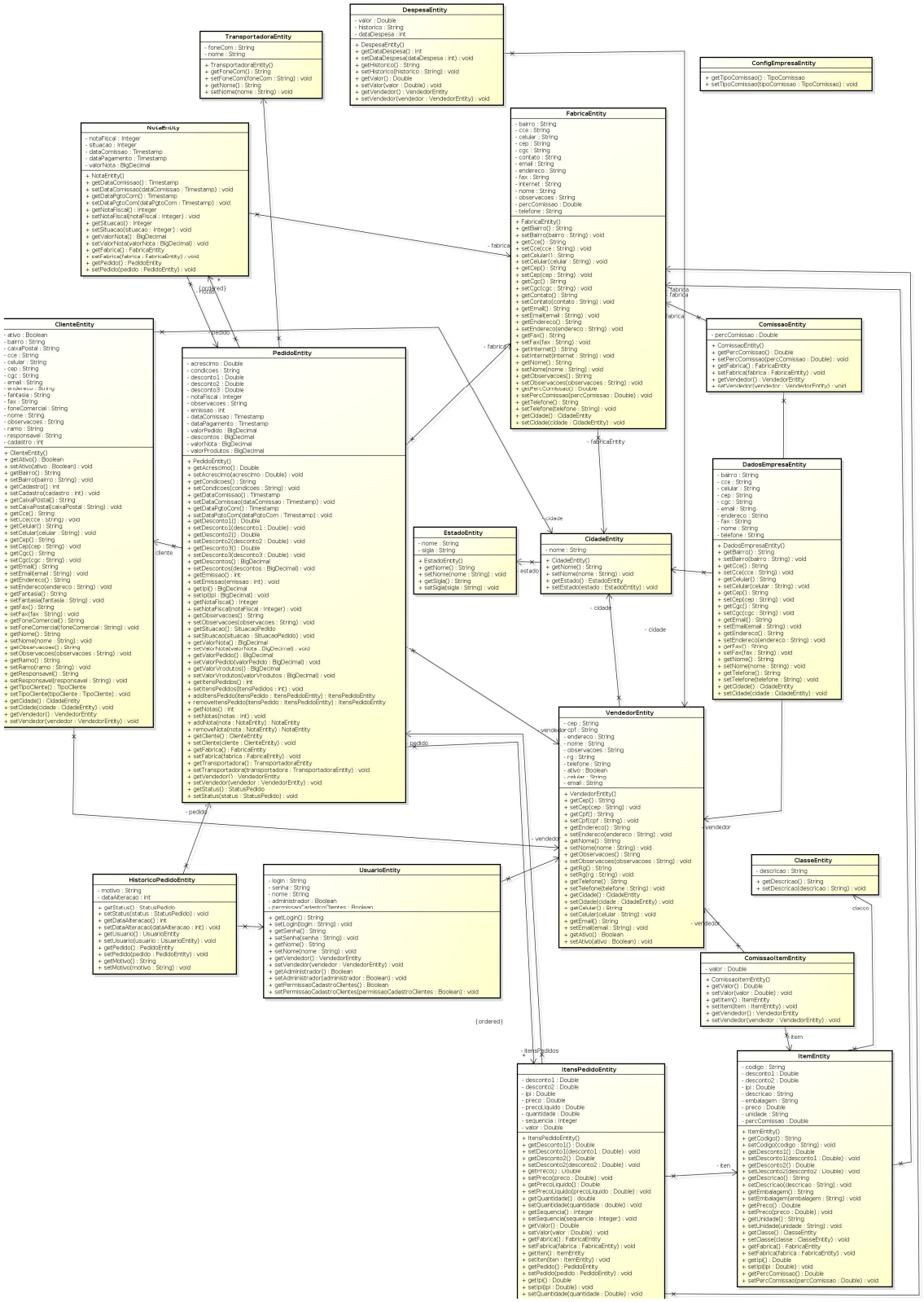

Figura 3 - Diagrama de classes completo

O DER (Diagrama de Entidade e Relacionamento) modela as tabelas de um banco de dados, definindo campos, tipo de dados de cada campo, chaves primárias,

e relacionamentos entre as tabelas determinando as chaves secundárias ou estrangeiras. Na Figura 4 observam-se todas as tabelas que armazenam os dados do sistema e seus respectivos campos e interligações.

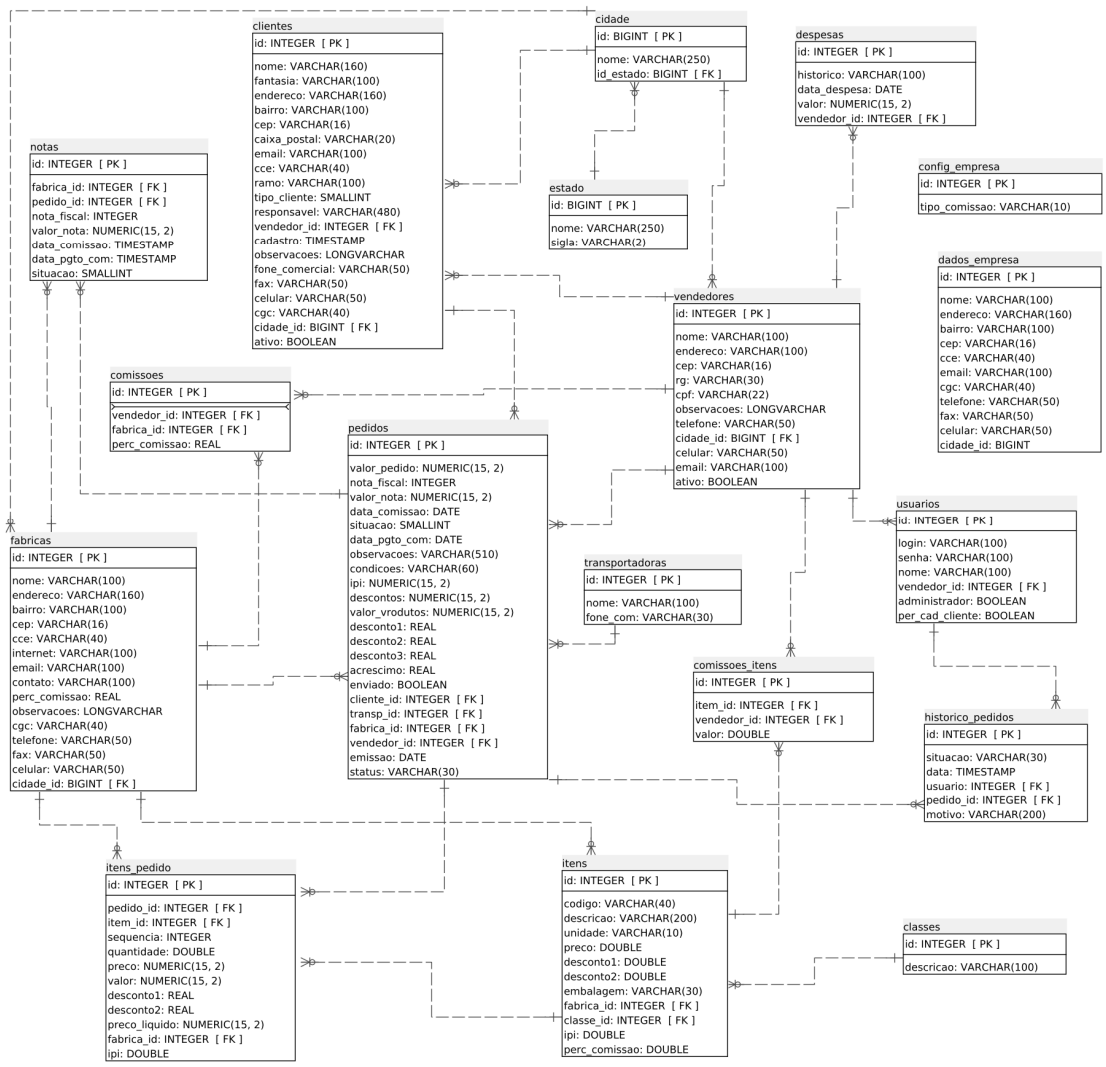

Figura 4 - Diagrama de Entidade e Relacionamento

4.3 APRESENTAÇÃO DO SISTEMA

Após efetuar o *login* o usuário é direcionado para a página inicial do sistema (Figura 5). No sistema legado (Figura 6) os cadastros eram acessados via menu superior ou botões de atalho, já na nova versão são abertas em abas e acessadas

pelo menu localizado na lateral esquerda da janela. Ainda na nova versão existe o nome do usuário na parte superior esquerda, o botão de sair na parte superior direita e no topo o nome da empresa de representação.

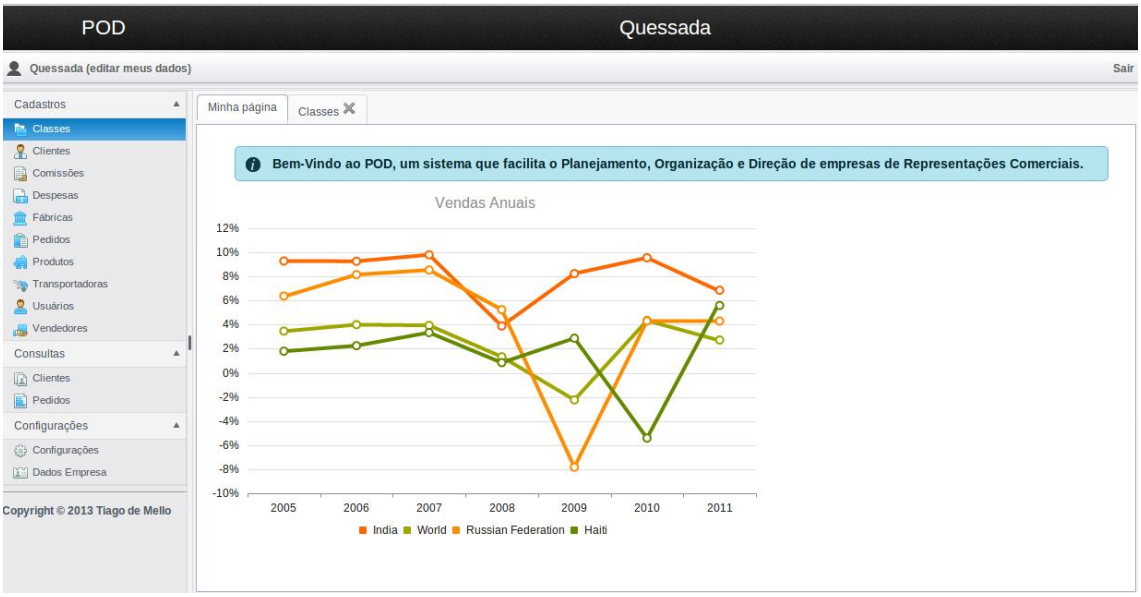

Figura 5 - Tela inicial nova

Figura 6 - Tela inicial antiga

Todas as telas do sistema são abertas em abas, na maioria dos casos com uma listagem dos dados referente ao cadastro ou consulta. Tanto as abas quanto os *grids* de listagem usados são da biblioteca Kendo UI, no caso das abas foram

criados métodos para abrir e a possibilidade de fechar uma aba com um botão (funcionalidade que não faz parte da biblioteca e foi codificada). E nos *grids*foramconfiguradas as opções de filtragem e ordenação de todas as colunas, além de adicionado a opções para exportar os dados em formatoPDF ou XLS. No sistema legado as listagens não tinham opções de filtro ou ordenação. Também foi configurada a paginação nas listagens para que o tempo da consulta fosse reduzido, buscando apenas os registros na página. Na Figura 7 o exemplo da listagem da tela de cadastro de pedidos e na Figura 8 a mesma tela no sistema legado.

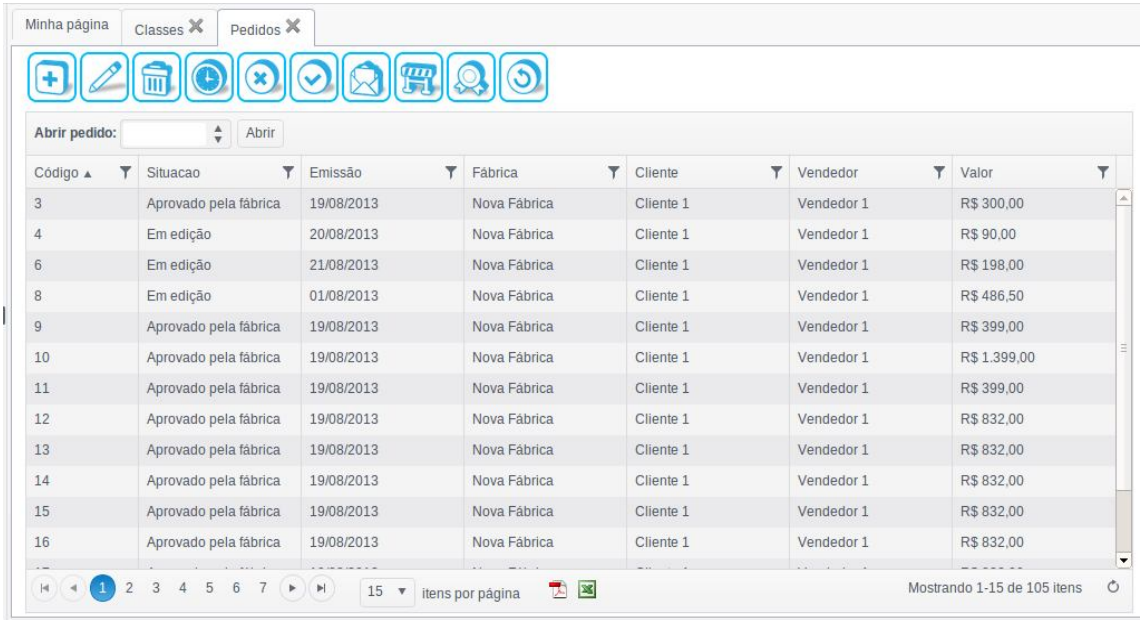

Figura 7 - Listagem de Pedidos

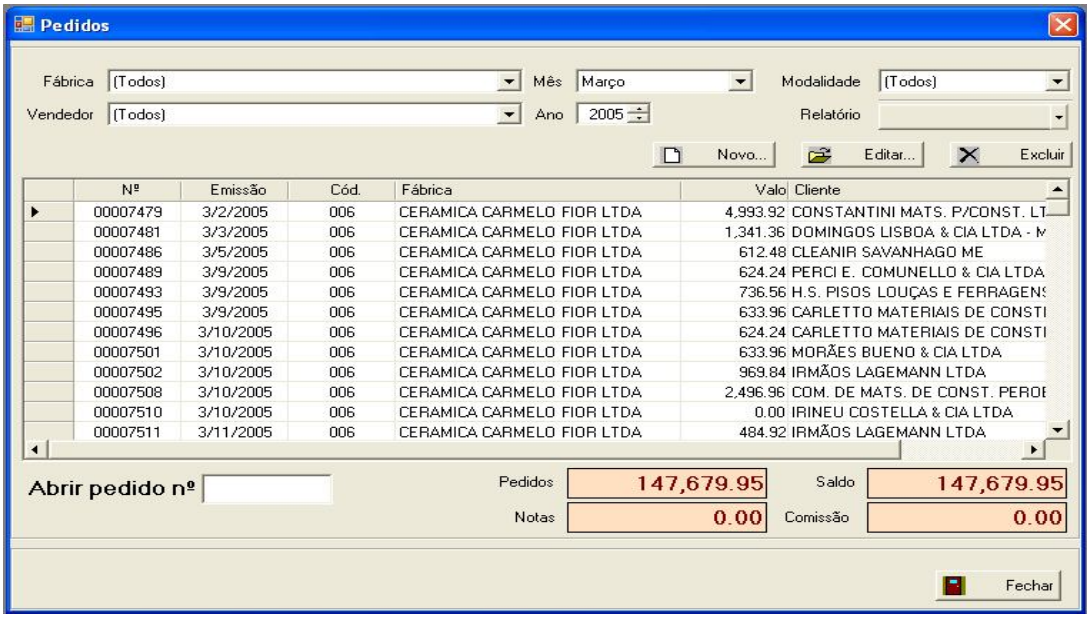

Figura 8 - Listagem de Pedidos

As telas de cadastros são abertas em janelas sobre a aba do *grid*, da mesma forma que no sistema *desktop*. Em alguns casos os cadastros foram divididos em abas dentro da janela para aproveitar melhor o espaço na tela, como no cadastro de pedidos (Figuras 9, 10 e 11).

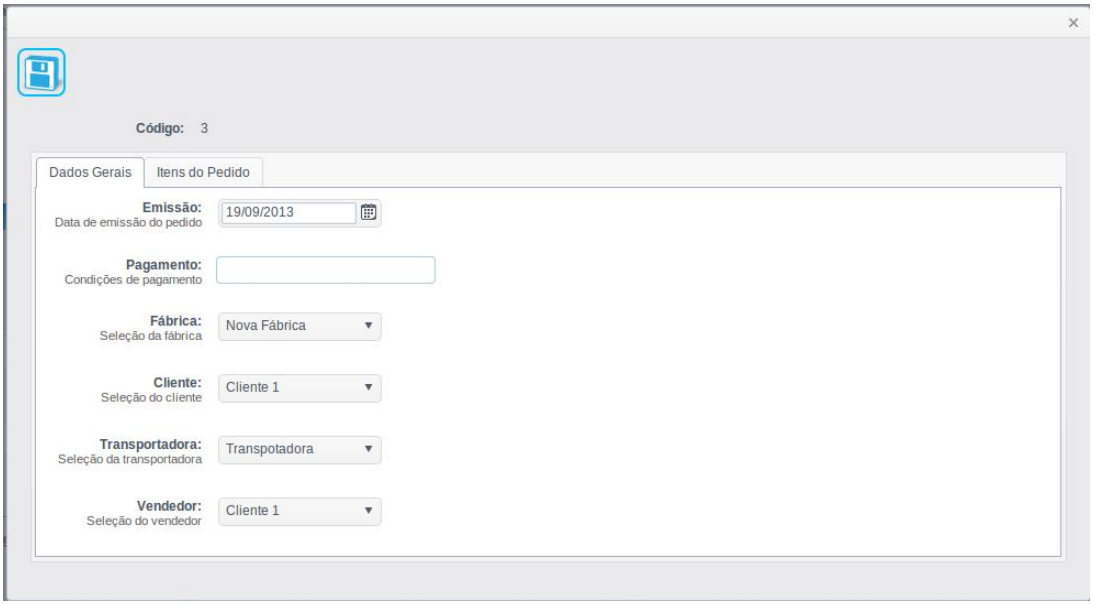

Figura 9 - Cadastro pedidos 1

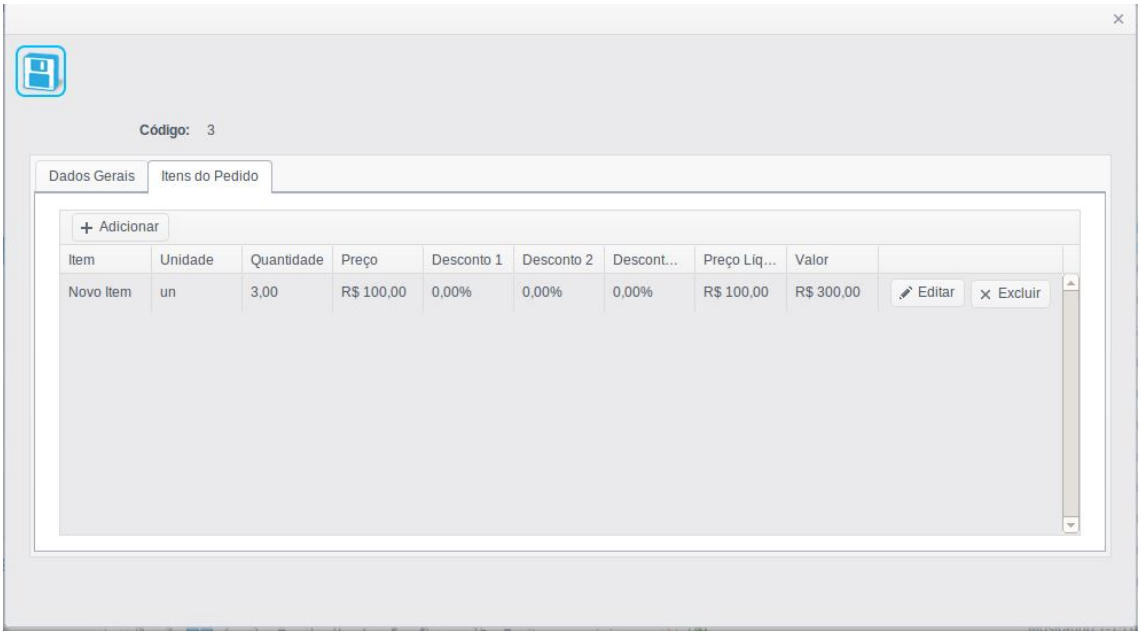

Figura 10 - Cadastro pedidos 2

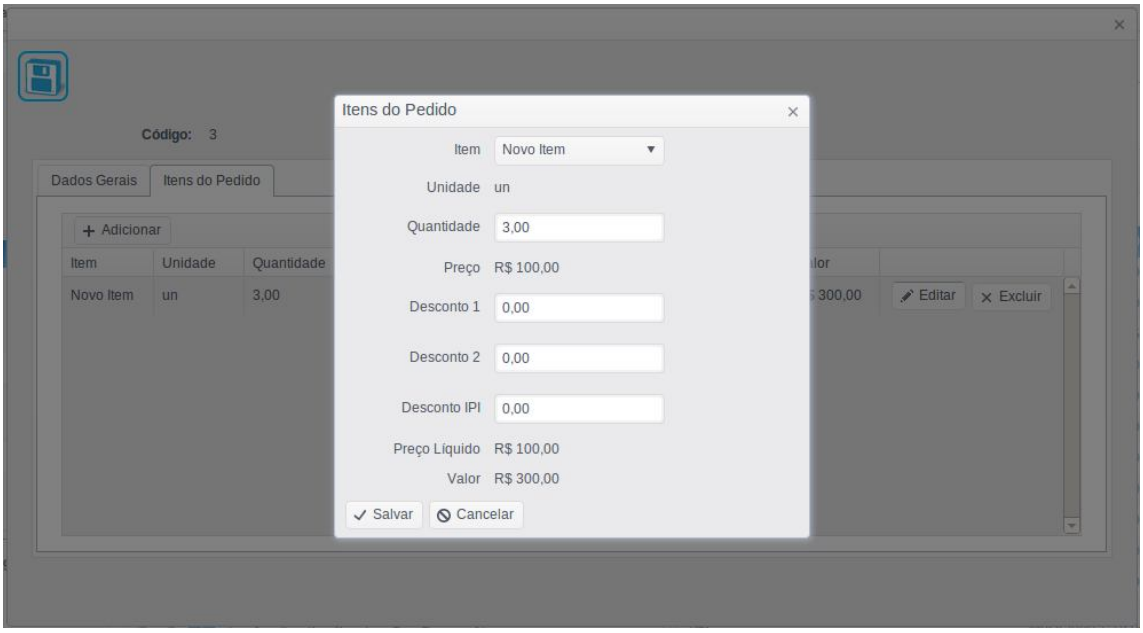

Figura 11 - Cadastro de pedidos 3 (adicionar item)

Para comparação, na Figura 12 está a tela de cadastro de pedidos da versão antiga do sistema.

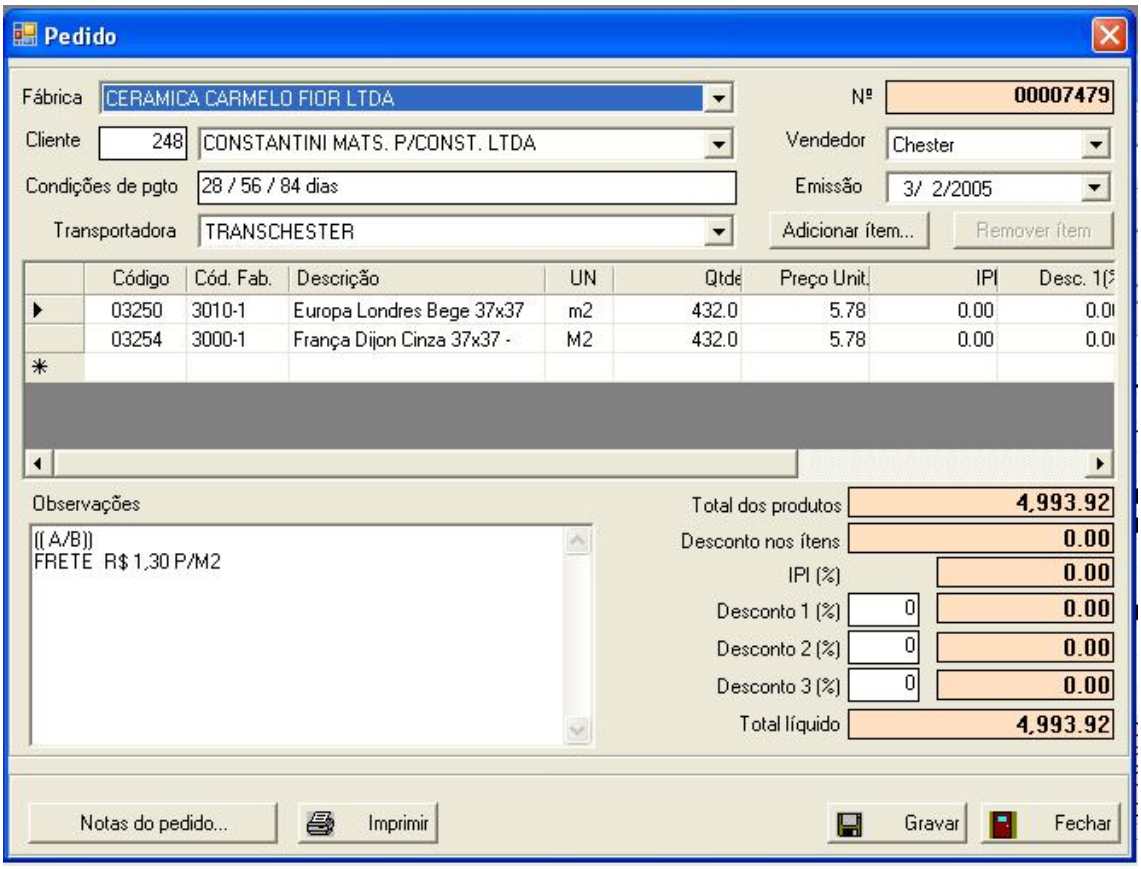

Figura 12 - Cadastro de pedidos

Toda vez que uma ação é executada corretamente uma mensagem é apresentada ao usuário para notificar que ocorreu tudo certo. E quando acontece algum erro, tanto por falha do sistema ou por validação lógica, também é exibida uma mensagem ao usuário. A mensagem de sucesso (Figura 13) é uma caixa na cor verde, de avisos na cor amarela e de erros na cor vermelha. E quando houver problemas na validação de campos obrigatórios a mensagem (Figura 14) é exibida ao lado do campo.

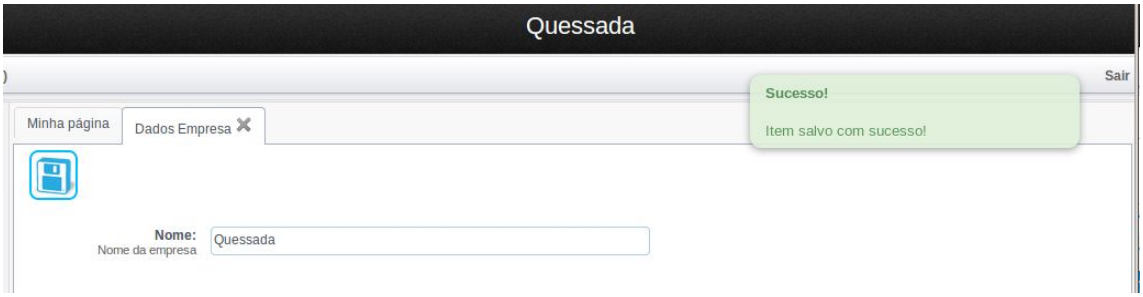

Figura 13 - Mensagem de sucesso

Figura 14 - Mensagem de campo obrigatório

No sistema alguns relatórios levam o nome da empresa e outros dados, como cidade e estado. O nome localizado no topo também faz referencia a empresa do usuário. Portanto, existe uma tela específica para o cadastro dos dados da empresa (Figuras 15 e 16). No sistema legado a tela (Figura 17) já existia, a principal diferença foi o recurso do Google Maps que foi adicionada nessa tela, assim como em todas as telas onde exista um endereço para ser preenchido.

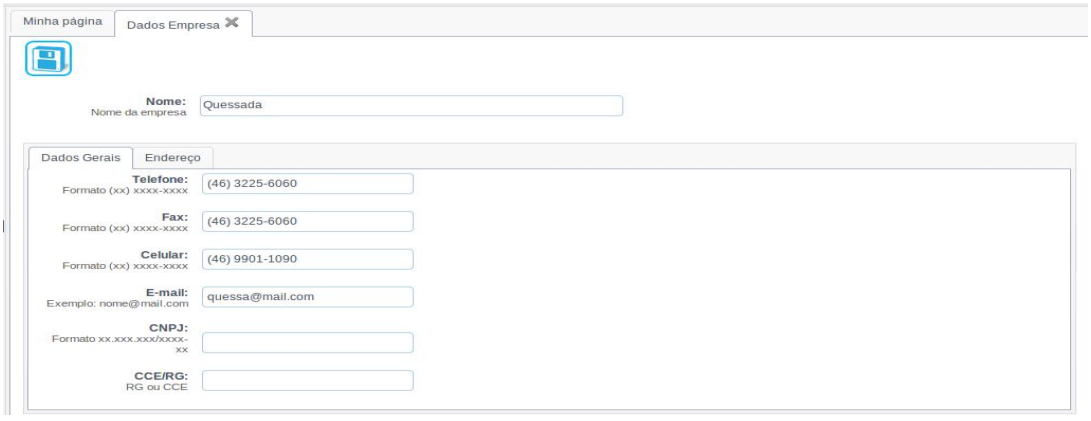

Figura 15 - Dados da empresa 1

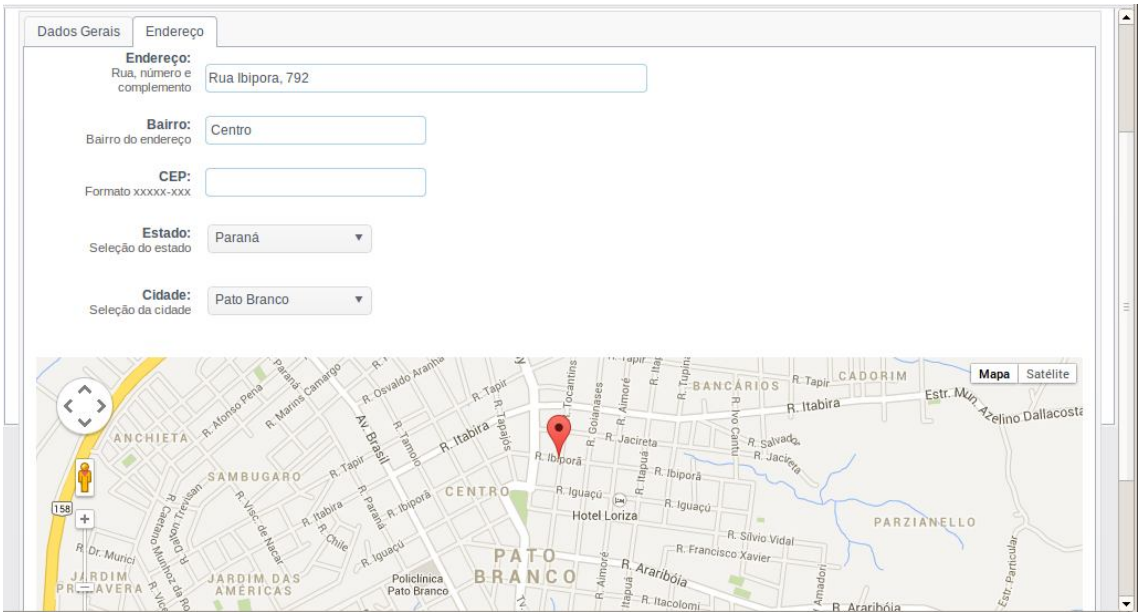

Figura 16 - Dados da empresa 2 (mapa)

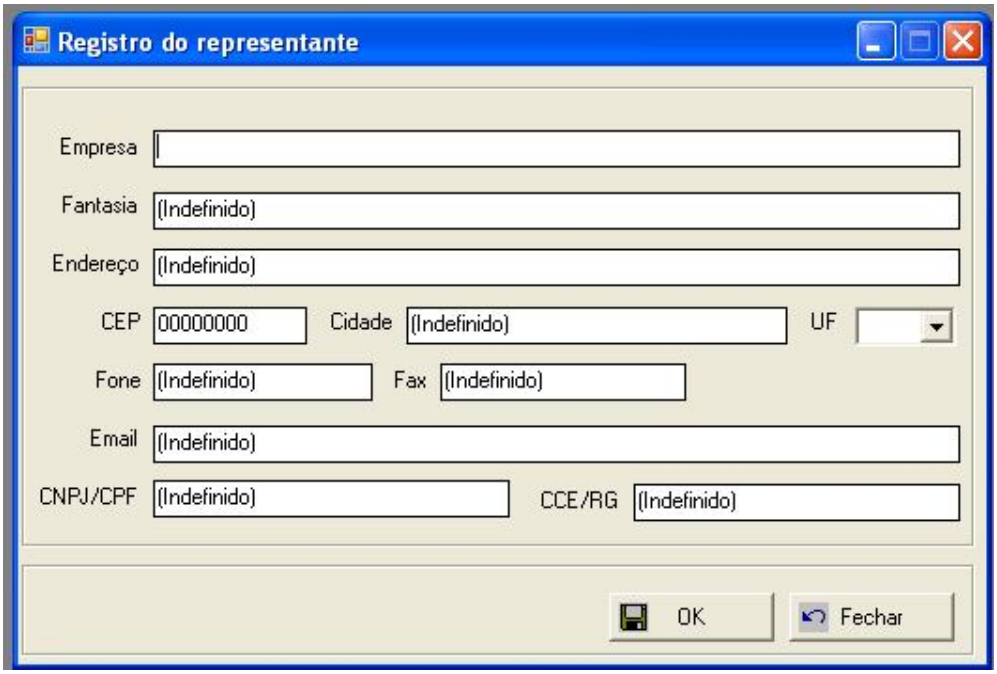

Figura 17 - Registro do representante

De acordo com a Figura X e Figura Y é possível observar que a tela para configuração dos dados da empresa não segue o mesmo padrão das outras telas do sistema com listagem e inclusão, nesta não existe listagem ejá é aberta com os campos do cadastro. Na tabela no banco de dados sempre existirá apenas um registro com os dados da empresa de representação.

4.4IMPLEMENTAÇÃO DO SISTEMA

Para o desenvolvimento da aplicação foi utilizado recursos da arquitetura MVC, na figura abaixo é possível observar o fluxo dos processos realizados pelo sistema.

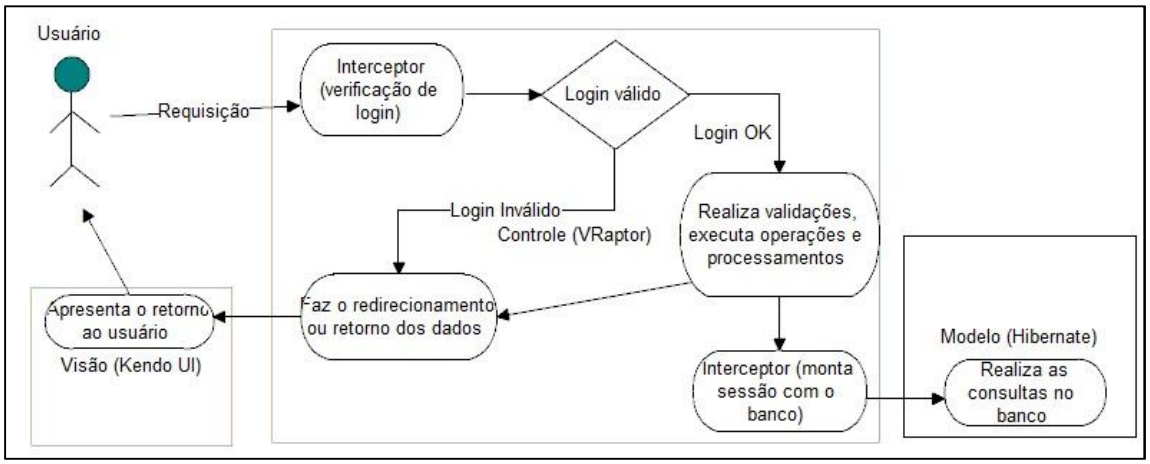

Figura 18–Quadro com as camadas do sistema

A Figura 18 mostra as camadas do sistema.Na camada de Visão foi usado o Kendo UI e é onde se define como as informaçõesserão apresentadas ao usuário. Na camada de Modelo foi usado o Hibernate e é onde são definidas as estruturas de dados do sistema e também onde são realizadas ações na base de dados. E na camada de controle foi usado o VRaptor e é onde feito todo o processo lógico do sistema, validações, controle de acesso, criação de sessões, redirecionamento de telas e execução de operações.

As listagens de códigos a seguir visam exemplificar os usos das tecnologias usadas durante o desenvolvimento. Com o intuito de demonstrar a forma que foi a realizada a codificação dessas no projeto.

Na tela inicial do sistema (Quadro1) foi desenvolvida uma JSP utilizando os recursos do HTML5 e CSS3. Todas as páginas serão abertas em cima da página inicial e, portanto as referências para indicar os usos de outros estilos ou *scripts* já estão mapeadas na tela inicial e não serão referenciados novamente, assim como não haverá a necessidade de replicar o *layout* do sistema em todas as telas.

```
{...}
```

```
        <script type="text/javascript">
        function init() {
                kendo.culture("pt-BR");

                createSplitters();
                createTabs();
                createPopup();

                $("#panelbar").kendoPanelBar({
expandMode: "multiple"
         });
        }

        $(document).ready(init);
        </script>
</head>

<body>
        <div id="vertical" style="height: 100%; width: 100%;">
                <div id="top-pane">
                        <div id="header" class="pane-content">
                                <hgroup>
                                        <h1 class="site_title">POD</h1>
                                        <h2 class="section_title">${dadosEmpresa.nome}</h2>
                                </hgroup>
                        </div>
                </div>
                <div id="second-top-pane">
                        <div id="secondary_bar" class="pane-content">
                                <divclass="user">
                                        <p>${user.nome} (<a href="#">editar meus dados</a>)</p>
                                        <a class="logout_user"
href="<c:urlvalue='/logout'/>">Sair</a>
                                </div>
                        </div>
                </div>
                <div id="middle-pane">
                        <div id="horizontal" style="height: 100%; width: 100%;">
                                <div>
                                        <!—Espaço para os menus de cadastros -->
                                </div>
                                <div id="center-pane">
                                        <div id="tabstrip"></div>
                                </div>
                        </div>
                </div>
        </div>
        <div id="popUp"></div>
</body>
</html>
```

Quadro 1 - Página inicial do sistema

Ainda no código acima é possível notar o uso da biblioteca Kendo UI para a criação do *layout* da tela inicial do sistema, sendo que cada área do sistema é representada por um elemento *div* que tem um *id* configurado, esse *id* é usado para mapear a posição onde o componente estará disposto na tela. A criação desses componentes é feita ao inicializar a tela, sendo que para isso foi adicionado um *listener* para esse evento que irá chamar o método init. Dentro do método init é realizado a chamada aos métodos responsáveis por criar os componentes da biblioteca Kendo, sendo que esses métodos estão codificados no arquivo default.js que está separado do código da interface visual.

Além dos componentes da biblioteca Kendo também foram usadas folhas de estilo em cascata para tornar o visual do sistema mais agradável.Esses estilos estão

no arquivo layout.css separado do código-fonte da interface. Além deste existe outro arquivo com estilos do sistema, que é o default.css que possui estilos usados em várias telas do sistema. Esses estilos vão definir as fontes do sistema, estilos dos *links*, efeitos de sombra, as imagens e textos da parte superior do sistema e outras configurações que sejam relacionadas às características visuais do sistema.

Abaixo estão listados os métodos usados na criação de componentes da biblioteca Kendo UI que estão no arquivo default.js.

No Quadro 2 é possível observar o método para a criação dos *splitters* que servem para separar em painéis a tela, sendo que esses painéis podem ter seu tamanho alterado com o arrastar do mouse. Para configurar o componente é necessário passar o nome do elemento *div* no qual o componente será localizado e também podem ser definidas algumas características, como tamanho, se será na horizontal ou vertical e outras que podem ser encontradas na documentação da biblioteca.

```
// *** SPLITTERS ***
function createSplitters() {
        $("#vertical").kendoSplitter({
         orientation: "vertical",
         panes: [
             { collapsible: false, resizable: false, size: "48px"},
             { collapsible: false, resizable: false, size: "38px"},
             { collapsible: false, resizable: false, size: "100%"}
         ]
    });
    $("#horizontal").kendoSplitter({
         panes: [
             { collapsible: false, resizable: true, size: "200px"},
             { collapsible: false, resizable: true, size: "100%"}
]
});
}
```

Quadro2 - Criação de painéis

No Quadro 3 estão os métodos para a criação das abas do sistema. As abas são os elementos onde serão abertas as telas do sistema e já foram criados os métodos para criar a primeira aba que é a página inicial do sistema e fixa. Também foram criados métodos para adicionar uma nova, excluir uma existente e para adequar o conteúdo ao tamanho da tela do usuário (expandContentDivs). O componente usado para as abas não possui um botão para fechar, portanto esse foi adicionado manualmente, e quando clicado sobre a imagem adicionada no título da aba, é chamado um método para fechar a aba. Ao adicionar uma nova aba, é verificado se ela já foi aberta e caso sim apenas será exibida a aba já aberta, não permitindo assim abrir a mesma tela em duas abas separadas.

```
//*** TABS ***
vartabStrip;
vartabStripElement;
```

```
varremoveImg      =      "      <imgsrc='./public/default/images/close.gif'      alt='close'
onclick='removeTab($(this).closest(\"li\"))'/>";

function createTabs() {
       tabStripElement = $("#tabstrip").kendoTabStrip();
tabStrip = tabStripElement.data("kendoTabStrip");

    // adicionar a primeira aba
tabStrip.append({
              text: "Minha página",
contentUrl: "/pod/tabindex"
    });

tabStrip.select(tabStrip.tabGroup.children("li")[0]);
       expandContentDivs(tabStripElement.children(".k-content").last());
}

function expandContentDivs(divs) {
divs.height(tabStripElement.innerHeight()       -       tabStripElement.children(".k-tabstrip-
items").outerHeight() - 16);
}

function addNewTab(title, url) {
       if (! containsTab(title)) {
              tabStrip.append({
                     text: title + removeImg,
                 encoded: false,
contentUrl: url
        });
       }

       selectTab(title);
       expandContentDivs(tabStripElement.children(".k-content").last());
}

function removeTab(tab) {
varotherTab = tab.next();
otherTab = otherTab.length ? otherTab : tab.prev();

tabStrip.remove(tab);
tabStrip.select(otherTab);
}

function containsTab(title) {
       title += " ";
       var tabs = tabStrip.tabGroup.children("li");
       for (vari=0; i<tabs.length; i++) {
              if (tabs[i].textContent == title ) {
                     return true;
              }
       }

       return false;
}

function selectTab(title) {
       title += " ";
       var tabs = tabStrip.tabGroup.children("li");
       for (vari=0; i<tabs.length; i++) {
              if (tabs[i].textContent == title) {
                     tabStrip.select(tabs[i]);
              }
       }
}
```

Quadro3 - Criação das abas

Como grande parte das telas do sistema terá um *grid* para listagem dos dados, foi criado um único método (Quadro4) para a criação desses grids. Com base nos parâmetros passados, os grids são criados. Sempre existirá um *id* que é usado para se obter os itens selecionados ou abrir uma tela de alteração. No método de criação do *grid* já está adicionada as configurações para paginação, ordenação e

filtragem no servidor (que foi necessária para reduzir o tempo das consultas na base de dados, trazendo apenas os dados da página atual). Também já são adicionados os botões de exportação e evento para abrir a tela de edição com um clique duplo sobre uma linha.

```
//*** GRID ***
varexportPDF = "<imgsrc='./public/default/images/pdf.png' alt='Exportarem PDF'
title='Exportarem PDF' class='export-button'onclick='removeTab($(this).closest(\"li\"))'/>";
varexportXLS = "<imgsrc='./public/default/images/xls.png' alt='Exportarem XLS'
title='Exportarem XLS' class='export-button'onclick='removeTab($(this).closest(\"li\"))'/>";

function createGrid(id, url, fields, columns, toolbar, dlbclick, detailTemplate, detailInit,
selectable) {
	varselectableTipe = selectable != null ? selectable : "multiple";
	$(id).kendoGrid({
		sortable: true,
		resizable: true,
		filterable : true,
		selectable: selectableTipe,
		columns : columns,
		toolbar : toolbar != null ? toolbar : "",
		detailTemplate: detailTemplate,
		detailInit: detailInit,
		dataBound: dataBound,
		editable: "inline",
		dataSource : {
			serverPaging: true,
			serverSorting: true,
			serverFiltering: true,
			page: 1,
	pageSize: 15,
	 batch: true,
			transport: {
				contentType: "application/json",
				read: url,
				update: url.replace("data","update"),
				parameterMap: function(data, operation) {
					if (operation == "update") {
						return {entity: kendo.stringify(data.models[0])};
					}else {
						var param = "";
						param += "page=" + data.page;
						param += "&pageSize=" + data.pageSize;
						param+= "&filter=" +
kendo.stringify(data.filter);
						param += columnsToJson("sort", data.sort);
						return param;
					}
			}
		 },
		schema: {
				data: "items",
				total: "totalCount",
		    model: { id: 'id', fields: fields }
		 },
		 sort: { field: "id", dir: "asc" }
		},
		pageable: {
			pageSize: 15,
			pageSizes: [15, 30, 50, 100, 500, 1000],
			refresh: true,
			input: false
		}
	});

	$(id + " .k-pager-refresh").before("<a href='#' onclick='exportGridPDF(\"" + url +
"\", \"" + id + "\")' class='export-button'>" + exportPDF + "</a>");
	$(id + " .k-pager-refresh").before("<a href='#' onclick='exportGridXLS(\"" + url +
"\", \"" + id + "\")' class='export-button'>" + exportXLS + "</a>");

	resizeGrid(id, 60);

	if (dlbclick != false) {
```

```
            $(id).delegate("tbody>tr", "dblclick", function(){openEditWindow(id,
url.replace("data","edit"));});
        }
}
```

Quadro4 - Criação do *Grid*

Os estilos são necessários para dar uma boa aparência ao sistema. Para criar a barra superior com os dados do sistema e o usuário foram criados estilos (Quadro5). Foramadicionadas imagens como plano de fundo dos componentes, usadas fontes específicas, efeitos de sombra e também os estilos relacionados ao posicionamento para que os textos e imagens fiquem no seu devido lugar na tela.

```
/* Header */

#top-pane {
        background: blue url(../images/header_bg_old.png) repeat-x;
}

#header {
height: 48px;
width: 100%;
}

#header h1.site_title {
float: left;
margin: 0;
font-size: 22px;
display: block;
width: 220px;
height: 48px;
font-weight: normal;
text-align: center;
line-height: 48px;
color: #fff;
}

#header h2.section_title {
float: center;
margin: 0;
font-size: 22px;
display: block;
height: 48px;
font-weight: normal;
text-align: center;
line-height: 48px;
color: #fff;
}

/* Secondary Header Bar */
#secondary_bar {
height: 38px;
width: 100%;
background: #F1F1F4 url(../images/secondary_bar.png) repeat-x;
}

#secondary_bar .user {
float: left;
width: 100%;
height: 38px;
}

.user p {
margin: 0;
padding: 0;
color: #666666;
font-weight: bold;
display: block;
float: left;
height: 35px;
```

```
line-height: 35px;
text-indent: 25px;
text-shadow: 0 1px 0 #fff;
background: url(../images/icn_user.png) no-repeat center left;
margin-left: 5px;
}

.user a {
text-decoration: none;
color: #666666
}

.user a:hover {
color: #77BACE;
}

.user a.logout_user {
color: #666666;
font-weight: bold;
float: right;
display: block;
height: 35px;
margin-right: 5px;
line-height: 35px;
}

.user a.logout_user:hover {
color: #77BACE;
}
```

Quadro5 - Estilo *layout*

Para os formulários do sistema também foram criados alguns estilos (Quadro6) para que os *labels*fiquem à esquerda e os campos à direita, e também tenha um texto pequeno junto ao *label* com mais informações, além dos estilos para os componentes de entrada de dados, com o intuito de obter uma aparência mais agradável e manter o mesmo padrão visual do resto do sistema.

```
.myform label {
	display: inline-block;
	font-weight: bold;
	text-align: right;
	width: 140px;
	margin-right: 10px;
}

.myform .small {
	color: #666666;
	display: block;
	font-size: 11px;
	font-weight: normal;
	text-align: right;
	width: 140px;
}

.myform input {
	font-size: 12px;
	padding: 4px 2px;
	border: solid 1px #aacfe4;
	width: 200px;
	margin: 2px 0 20px 0px;
}

.myforminput.big, .big {
	font-size: 12px;
	padding: 4px 2px;
	border: solid 1px #aacfe4;
	width: 400px;
	margin: 2px 0 20px 0px;
}

.myformtextarea {
```

```
        font-size: 12px;
        padding: 4px 2px;
        border: solid 1px #aacfe4;
        width: 400px;
        margin: 2px 0 20px 0px;
}

.formField {
        padding: 4px 2px;
        margin: 2px 0 20px 0px;
}

.displayField {
        font-size: 12px;
        padding: 4px 2px;
        vertical-align: super;
}
```

Quadro6 - Estilo formulário

Para a codificação de uma tela de quadro (Quadro7) no sistema a quantidade de linhas de código se torna pequena, sendo que muitos métodos e estilos estão codificados em bibliotecas separadas. No código abaixo é possível notar que foi configurado na tela uma variável chamada module que é usada ao longo do código para a criação dos *ids* dos componentes.Essa técnica é útil ao ser realizada a cópia desse arquivo, pois só será necessário alterar o valor dessa variável e o resto do código já estará pronto. Além do *id* do *grid* e do caminho para a chamada do método de listar do cadastro, também são passadas por parâmetros ao método de criação do *grid*quais as colunas que irão aparecer na listagem, configurando qual é o campo, o seu título e o seu tamanho.

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<c:setvar="module" value="classe" />

<script type="text/javascript">

        var fields = {
        id: { type: 'number' },
descricao: { type: 'string' }
};

        var columns = [{
                field: "id",
                title: "Código",
                width: 100
        }, {
                field: "descricao",
                title: "Descriçao"
        }];

        $(document).ready(function() {
                createGrid("#grid_${module}", "<c:url value='/cadastros/${module}/data'/>",
                            fields, columns);
                resizeGrid("#grid_${module}");
        });

</script>

<div class="botoes">
        <!-- Incluir -->
        <a href="#" onclick="javascript: openNewWindow('<c:url
value='/cadastros/${module}/form'/>');" title="Incluir">
                <imgsrc="${pageContext.servletContext.contextPath}
/public/default/images/icn_insert.png" class="icon" /></a>
```

```
	<!-- Editar -->
	<a href="#" onclick="openEditWindow('#grid_${module}', '<c:url
value='/cadastros/${module}/edit'/>');" title="Editar">
		<imgsrc="${pageContext.servletContext.contextPath}
/public/default/images/edit.png" class="icon" /></a>

	<!-- Excluir -->
	<a href="#" onclick="deleteFromGrid('#grid_${module}', '<c:url
value='/cadastros/${module}/delete'/>');" title="Excluir">
		<imgsrc="${pageContext.servletContext.contextPath}
/public/default/images/remove.png" class="icon" /></a>
</div>
<div id="grid_${module}"></div>
```

Quadro7 - Exemplo listagem

Na codificação do formulário (Quadro8) de cadastro o código já se torna um pouco extenso de acordo com a quantidade de campos existentes. No exemplo abaixo é possível notar que o uso de uma variável com o nome do módulo também foi utilizado nas páginas de cadastro. Ao inicializar a tela é chamado um método *javascript*para a criação da configuração de validação dos campos. Descendo no código foi adicionado o botão de salvar, sendo que este vai chamar um método que irá realizar as validações, submeter o formulário e retornar a mensagem de *feedback* ao usuário.

Por fim, é criado um formulário com os campos do cadastro. Existe umaestrutura de condição para a exibição do código, para que esse campo só seja exibido quando for uma alteração e terá o seu valor no formato*label*. Pode-se notar que os componentes estão todos dentro de um elemento *div* que está com o estilo myform, o qualpossui os estilos padrões para formulários no sistema. Nas telas de cadastros do sistema, cada campo do cadastro tem um *label*com o nome do campo e uma descrição com mais detalhes sobre o campo junto. Essa descrição detalhada está junto ao *label*, com letras menores e é definida dentro de um elemento *span*com a classe CSS *small*. E por último, abaixo do *span*está o componente de entrada de dados e assim completando uma linha do cadastro. Se houver mais campos do cadastro, todos vão seguir a mesma lógica tendo um *label*, um *span* e o componente de entrada de dados.

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:setvar="module" value="classe"/>

<script type="text/javascript">
	function init() {
		$("#form_${module}").validate({
			onsubmit: false,
			rules : {
				"entity.descricao" : {
					required : true
				}
			}
```

```
                });
        }

        $(document).ready(init);
</script>

<div class="botoes">
        <!-- Salvar -->
        <a href="#" onclick="submitForm('#form_${module}', '#grid_${module}', '<c:url
value='/cadastros/${module}/save'/>');"><imgalt="Salvar"
        src="${pageContext.servletContext.contextPath}/public/default/images/icn_save.png"
class="icon"/></a>
</div>

<div>
        <form id="form_${module}">
                <div class="myform">
                        <c:if test="${entity.id != null}">
                                <label>Código:</label>
                                <span class="id">${entity.id}</span>
                                <input type="hidden" name="entity.id" class="k-textbox"
value="${entity.id}"/>
                                <br/>
                                <br/>
                        </c:if>
                        <label for="descricao">Descriçao:
                        <span class="small">Descrição da classe</span></label>
                        <input type="text" maxlength="100" name="entity.descricao" class="k-
textbox big" value="${entity.descricao}"/>
                </div>
        </form>
</div>
```

Quadro8 - Exemplo formulário

Na camada de controle da aplicação o uso do *framework* VRaptor e também do Hibernate agilizou e simplificou bastante o desenvolvimento. Um bom exemplo do uso em conjunto de ambos os *frameworks* foi na codificação da classe HibernateTransactionInterceptor (Quadro 9). Essa classe é responsável por iniciar uma nova transação toda vez que um método é chamado, se o método for executado corretamente então é realizado o *commit*, ou então o rollback.

Para essa funcionalidade foi criada uma classe que implementa a interfaceInterceptor do VRaptor que irá interceptar todas as chamadas à métodos do sistema. E ao executar um método é sempre iniciada uma nova transação no banco de dadosatravés dos recursos do Hibernate, e assim evitando replicar esse tipo de código ao longo do sistema e evitando problemas em métodos que executem várias ações na base de dados.

```
package br.edu.utfpr.infra.hibernate;

import org.hibernate.Session;
import org.hibernate.Transaction;

importbr.com.caelum.vraptor.Intercepts;
importbr.com.caelum.vraptor.Validator;
importbr.com.caelum.vraptor.core.InterceptorStack;
importbr.com.caelum.vraptor.interceptor.Interceptor;
import br.com.caelum.vraptor.resource.ResourceMethod;

@Intercepts
public class HibernateTransactionInterceptor implements Interceptor {

    private final Session session;
```

```
    private final Validator validator;

    public HibernateTransactionInterceptor(Session session, Validator validator) {
this.session = session;
this.validator = validator;
    }

    public void intercept(InterceptorStack stack, ResourceMethod method, Object instance) {
Transaction transaction = null;
        try {
            transaction = session.beginTransaction();
stack.next(method, instance);
            if (!validator.hasErrors()) {
transaction.commit();
            }
        } finally {
            if (transaction != null &&transaction.isActive()) {
transaction.rollback();
            }
        }
    }

    public boolean accepts(ResourceMethod method) {
        return true; // Will intercept all requests
}
}
```

Quadro9 – Exemplo de uso do VRaptor com o Hibernate

Nos códigos de listagem e cadastro foi possível verificar que nos códigos da camada visual que existiam algumas *urls* que eram passadas aos métodos *javascript*. Essas *urls* fazem referencia a métodos da camada de controle, a chamada a esses métodos é feita via *urls*. Essa relação e direcionamento das *urls* aos métodos Java é realizado pelo *framework*VRaptor.

Para que uma classe tenha os seus métodos acessíveis dessa forma é necessário uma anotação *@Resource* que é observada na classe (Quadro 10) de controle do cadastro de classes.

```
@Resource
public class ClasseController extends AbstractController<ClasseEntity> {

        private final static String MODULE = "classe";

        private GeneralDaogeneralDao;

        public ClasseController(Result result, GenericDao<ClasseEntity>dao,
               {...}
        }

        @Path("/cadastros/" + MODULE + "/list")
        public void list() {
                super.list();
        }

        @Path("/cadastros/" + MODULE + "/data")
        public void getData(int page, intpageSize, List<SortDescription> sort,
FilterContainer filter) {
                super.getData(page, pageSize, sort, filter);
        }

        @Path("/cadastros/" + MODULE + "/form")
        public void form() {
                super.form();
        }

        @Path("/cadastros/" + MODULE + "/edit")
        public void edit(Long id) throws Exception {
                super.edit(id);
```

```
	}

	@Path("/cadastros/" + MODULE + "/save")
	public void save(ClasseEntity entity) {
		validate(entity);
		super.save(entity);
	}

	@SuppressWarnings("unchecked")
	private void validate(ClasseEntity entity) {
		 List<Criterion> filters = new ArrayList<Criterion>();
		filters.add(Restrictions.ne("id", entity.getId()));
		filters.add(Restrictions.ilike("descricao", entity.getDescricao()));

		 List<ClasseEntity>lista =
(List<ClasseEntity>) generalDao.find(ClasseEntity.class, filters);
		if (! lista.isEmpty()) {
			throw new ExpectedException(
"Já existe uma classe cadastrada com a mesma descrição!");
		}
	}

	@Path("/cadastros/" + MODULE + "/delete")
	public void delete(List<Long> items) {
		super.delete(items);
	}

	@Path("/cadastros/" + MODULE + "/exportPDF")
	public InputStreamDownloadexportPDF(List<Column> columns, List<SortDescription> sort,
FilterContainer filter) throws IOException {
		return super.exportPDF(columns, sort, filter);
	}

	@Path("/cadastros/" + MODULE + "/exportXLS")
	public InputStreamDownloadexportXLS(List<Column> columns, List<SortDescription> sort,
FilterContainer filter) throws Exception {
		return super.exportXLS(columns, sort, filter);
	}
}
```

Quadro10 - Exemplo de classe de controle

No código acima é possível observar que a classe herda os métodos de uma classe abstrata (AbstractController) que possui a implementação de todos os métodos da classe, exceto o de validação. Todo o método da classe tem uma anotação em cima, é essa anotação que indica qual a *url* que fará referência ao método abaixo. Dessa forma todas as ações relacionadas ao cadastro de classes ficam em uma única classe e da camada visual é possível chamar os métodos diretamente, e assim organizando e simplificando o código-fonte.

Todos os métodos da classe de controle do cadastro de classes estão codificados na classe AbastractController (Quadro 11), essa classe implementa de forma genérica as ações básicas de uma tela de cadastro e assim evita repetições de código, deixa o código mais limpo e organizado e também agiliza a codificação.

```
{...}

@Resource
@SuppressWarnings("serial")
public abstract class AbstractController<T extends AbstractEntity> implements
IAbstractController<T> {

	{...}

	public void getData(int page, intpageSize, List<SortDescription> sort,
FilterContainer filter, final String... args) {
```

```
JSONArray items = JSONArray.fromObject(dao.listAll(persistentClass, page,
pageSize, sort, filter), config);
                JSONObjectjson = new JSONObject();
                json.put("totalCount", dao.getCount(persistentClass, filter));
                json.put("items", items);
                response.setHeader("Content-Type", "application/json");
                result.use(Results.http()).body(json.toString());
        }

        public void edit(Long id) throws Exception {
                T entity = (T) dao.find(persistentClass, id);
                if(entity == null) {
                        throw new ExpectedException(
"Esse item já foi excluído, atualize a sua listagem!");
                }
                result.include("entity", entity);
                result.redirectTo(this.getClass()).form();
        }

        public void save(T entity) {
                if(entity.getId() != null) {
                        dao.update(entity);
                } else {
                        dao.save(entity);
                }
                result.use(json()).from("OK").serialize();
        }

        public void delete(List<Long> items) {
                dao.delete(items, persistentClass);
                result.use(json()).from("OK").serialize();
        }

        public InputStreamDownloadexportXLS(List<Column> columns, List<SortDescription> sort,
FilterContainer filter) throws Exception {
        byte[] arquivo = GridUtils.getExportXls(columns,
dao.listAll(persistentClass, sort, filter));
        ByteArrayInputStreamconteudoStream = new ByteArrayInputStream(arquivo);
return new InputStreamDownload(conteudoStream,
"application/vnd.ms-excel", "Arquivo.xls", true, arquivo.length);
        }

       {...}

}
```

Quadro11 - Classe de controle genérica

No código acima é possível observar que a classe genérica de controle também possui a anotação do VRaptor para indicar que é um recuso, e espera que toda classe que herda dela especifique uma classe de referência. Essa classe que está ao lado do nome da classe abstrata é o que indica qual a entidade que será processada. Como todas as entidades no sistema tem uma classe pai em comum (AbstractEntity), essa classe passada por parâmetro precisa ser filha dessa entidade.

Nos métodos list e form seu corpo está em branco, pois esses métodos apenas servem para redirecionar para uma página. Eles fazem referencia a um arquivo jsp no sistema, o list, por exemplo,que é chamado ao abrir uma nova aba e vai redirecionar para o arquivo que tenha o código de listagem que irá abrir dentro da aba do sistema.

O método getData é o responsável por realizar a busca na base de dados para preencher um *grid* de listagem, no inicio do método existe uma configuração usada em conjunto com JSON para excluir algumas propriedades desnecessárias das listagens. Esse método vai chamar o método de listagem que irá pegar os dados da página atual, e depois o método para verificar o total de itens na base que é exibido no *grid*. Esses dados são usados para formar um JSON com as propriedades totalCount e items, essas duas propriedades estão configuradas também no método de criação do *grid*na camada visual. Após montar esse objeto, é configurado o tipo de retorno de dados e enviado para a tela.

Nós métodos *save*, *edit* e *delete*estão codificados os métodos para salvar, buscar um item para alteração e excluir um item da base de dados. Dentro dos métodos são usados os métodos do *framework* Hibernate para realizar as ações na base de dados. Todos os métodos tem o retorno de sucesso para indicar que tudo o ocorreu bem. E no caso do edit, é verificado se o item que será alterado ainda existe na base de dados e se existir é adicionado o próprio item no retorno para preencher os dados do formulário nas telas de cadastro.

E por fim, tem os métodos para exportação dos itens do *grid*. Esses métodos montam os arquivos e retornam o arquivo gerado para a tela, e é iniciado o download do arquivo no computador do usuário que fez a requisição.

Para o sistema poder rodar em ambiente *cloud* se fez necessário desenvolver uma lógica de acesso, a aplicação roda em um único lugar e vários clientes se conectam a ela simultaneamente e em um número variável. Para esse funcionamento cada cliente do sistema possui um*schema* dentro da mesma base de dados, *squema* é um conceito do PostgreSQL onde cada *squema* possui suas própria tabelas. No caso do sistema existe um *squema* para cada cliente e todos os*squemas* tem a mesma estrutura de tabelas, mas com os dados referentes a um cliente especifico.

Quando um cliente se conecta ao sistema, ele tem um endereço próprio que contém o nome do *squema*, e quando esse endereço é acessado,é criada na sessão do usuário uma conexão com o banco de dados para o*squema* do endereço usado para acesso. Com essa implementação, para fornecer acesso a um novo cliente, só é necessário criar um novo *squema* e o fornecer o nome deste ao novo cliente.Sendo assim, o endereço da aplicação é sempre o mesmo e no fim contém o nome do *squema*, por exemplo: www.aplicacao.com.br/{nome-do-squema}.

O sistema desenvolvido já foi implantado utilizando um ambiente Jelastic, na Figura 19 é possível ver as configurações usadas para a criação desse ambiente. Na configuração é definido o servidor de aplicações, o banco de dados, o escalonamento vertical e horizontal, se será usado o recurso de alta disponibilidade (melhora o escalonamento horizontal) e se vai ser configurado um endereço público (diferente do disponibilizado por padrão).

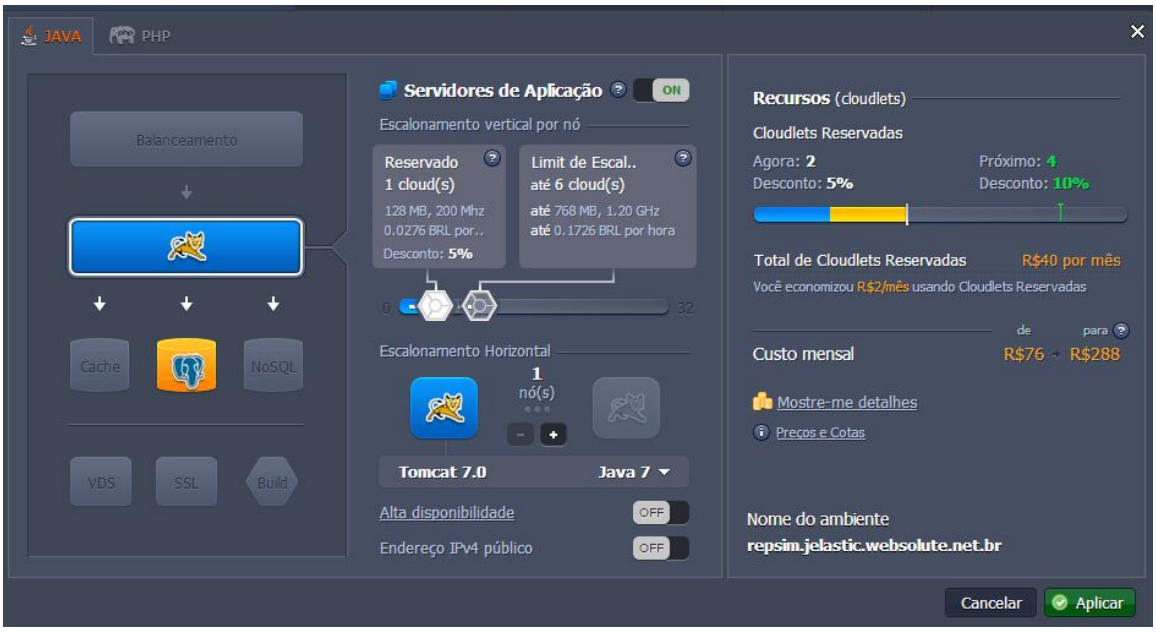

Figura 19 - Topologia ambiente Jelastic

4.5 DISCUSSÕES

O sistema resultante teve como base de desenvolvimento um já existente, e com a nova versão algumas modificações podem ser notadas. O simples fato da mudança de plataforma, de *desktop* para Web já resultou em grandes mudanças.

Quando é feito uma aplicação para rodar em ambiente *desktop*, é necessário saber para qual sistema operacional a aplicação será focada e se limitando apenas a um público, ou então tendo a necessidade de mais de uma versão do sistema. Já no caso de uma aplicação Web, essa pode ser usada em qualquer computador ou dispositivo móvel, desde que possua um navegador Web. Essa diferença, por outro lado, trás um custo ao desenvolvimento, uma vez que cada navegador trata as páginas de uma forma distinta. Como normalmente o objetivo de aplicações Web é

proporcionar mobilidade, é interessante funcionar corretamente nos mais variados navegadores disponíveis. Isso acarreta um custo de teste e desenvolvimento alto para garantir que o sistema rode em vários navegadores.

Já olhando no ponto de vista do usuário, uma grande mudança é vista no fato de que a aplicação pode ser acessada por qualquer dispositivo com acesso a Internet, sem a necessidade de possuir a aplicação instalada nesse. Outra vantagem é na questão da atualização de versões que pode ser feita automática ou de uma única vez, sem a necessidade de instalar uma nova versão ou rodar um instalador de atualização. Também existe a questão de um computador falhar e os dados nele serem perdidos, outro ponto é que toda vez que o computador for trocado ou atualizado os dados precisam ser salvos e o sistema instalado novamente.

Entretanto essa mobilidade tem um custo, primeiramente financeiro, pois é necessário que essa aplicação esteja rodando em um servidor disponível para todos os usuários e esse servidor terá um custo. Outro ponto que pode ser problemático é o fato de que o sistema só poderá ser acessado se o dispositivo tiver acesso à Internet, se por algum motivo uma empresa ficar sem conexão o sistema também estará inacessível.

E por fim, existe o custo de tempo pelo trafego, os dados serão enviados ao usuário final via rede e o tempo para esse envio depende da conexão do usuário. Comparando as duas versões do sistema, é possível notar que a primeira possui telas de listagens que apresentam todos os registros em uma única página e com extrema rapidez, já no caso da versão mais atual foi feito a separação por páginas para reduzir o tempo de listagem e mesmo assim o tempo para carregar uma página com 15 registros é maior do que para carregar 1000 registros na versão *desktop*.

Outras mudanças também podem ser notadas entre as duas versões. Na versão atual o separador decimal para número é ‘,’ e não mais ‘.’. Os relatórios gerados pelos sistemas são arquivos em formato PDF, já na versão antiga existe um visualizador de relatórios. Sendo que os relatórios na versão atual têm um tempo de geração maior, pelo mesmo motivo da diferença nas consultas.

Em termos gerais o sistema trouxe mudanças visuais que o tornaram visualmente mais agradável, além disto, trouxe a mudança de plataforma para rodar em ambiente Web, novas opções de consultas, recurso do Google Maps, nova forma de comissão por itens e fluxo de status do pedido. Mas, a versão para *desktop* ainda possui algumas vantagens como, formas de navegações específicas, o tempo das

consultas e a estabilidade por estar em funcionamento há bem mais tempo e já ter sido atualizada algumas vezes.

5 CONCLUSÃO

O objetivo deste trabalho de conclusão de curso foi a implementação e implantação de um sistema Web para empresas de representação comercial. O desenvolvimento teve como base um sistema pronto e já utilizado por duas empresas do ramo de Pato Branco, e o sistema resultante foi implantado em um ambiente *cloud* para ser acessado por ambas. Sendo assim, o referencial teórico do trabalho foi focado em conceitos relacionados a aplicações gerenciais para internet e ambiente *cloud*.

O contato direto com o cliente, a realização de uma análise e modelagem do sistema foi uma experiência novae produtiva, que por meio dessa foram adquiridos novos conhecimentos, além da possibilidade de aplicar os conceitos absorvidos ao longo do curso.

Os *frameworks*VRaptor3, Hibernate e Kendo eram desconhecidos. O uso desses no trabalho trouxe novos conhecimentos do ponto de vista de desenvolvimento junto com conceitos de Orientação a Objetos, MCV e Java, que foram de grande importância no projeto e tiveram horas de estudo dedicadas as suas metodologias e boas práticas.

O sistema foi desenvolvido focado em rodar em ambiente *cloud*, para isso foram estudados os conceitos de aplicações Web e aplicação nas nuvens, além de formas de desenvolvimento desse tipo de sistema.

O uso dos *frameworks* escolhidos foi bastante útilno desenvolvimento do projeto, o*framework*VRaptor é simples, ágil e auxilia bastante no desenvolvimento, entretanto no decorrer do projeto viu-sea necessidade desobrescrever algumas classes e interfaces do framework para o funcionamento da aplicações para ambiente nas nuvens e em conjunto com a biblioteca Kendo UI. O uso do Hibernate foi muito acertado e facilitou consideravelmente o desenvolvimento, além de ser essencial para o funcionamento da aplicação no ambiente*cloud*. Quanto ao Kendo UI, observou-se queé uma ferramenta agradável visualmente e com excelente documentação, porém possui apenas 13 componentes que é um número pequeno em relação aos concorrentes.

O projeto desenvolvido conseguiu manter a maioria das telas e ações do sistema legado da mesma forma, e em casos específicos,remodelagens e alterações

na forma de interação com o usuário foram necessárias, como alguns cadastros que foram divididos em abas e relatórios que tiveram a forma de acesso alterada. A interface do sistema foi refeita e se tornou mais agradável, e também trouxe novos recursos, como filtros e ordenação nas listagens,o que por outro lado ocasionounum custo maior no tempo na listagem dos dados.

Para as empresas irá trazer novas funcionalidades como uma nova forma de comissão (por item), consultas de itens vendidos e controle de status nos pedidos que foram verificados como sendo necessários para os clientes. Outra necessidade solucionada com a nova implementação, foi a possibilidade dos vendedores usarem os sistema e eles mesmos cadastrarem os pedidos de venda, uma vez que no sistema legado apenas os administradores tinham acesso ao sistema e os todos os pedidos eram cadastrados por eles. Na nova versão também foram criadas novas consultas, como a de clientes, que foram desenvolvidas para facilitar e agilizar as análises de dados que eram feitas manualmente com base nos relatórios impressos, tais como, verificar quais os clientes que estão sem realizar pedidos nos últimos seis meses.

Contudo o sistema também possui suas limitações.Se a empresa estiver sem acesso a Internet não poderá continuar a usar o sistema e irá ficar parada durante esse tempo. O sistema legado está rodando há mais de 10 anos e é totalmente confiável, já a versão atual mesmo com os testes realizados está suscetível a possíveis falhas. Comparado com o sistema atual, a nova versão é mais lenta no acesso aos dados por conta do acesso local ser muito mais rápido do que um acesso remoto.O sistema roda sobre um servidor *cloud*, que tem um custo, portanto as empresas terão um custo mensal para manter o sistema rodando.

O projeto está desenvolvido com todos os cadastros do sistema legado, implantado no ambiente *cloud* e pronto para uso. Entretanto, o desenvolvimento do projeto ainda terá continuidade, existem outros recursos que foram pensados e acabaram não sendo codificados, como por exemplo: a alteração dos dados pessoais do vendedor, possibilidade de solicitar um novo usuário, consulta de melhores vendedores e atalhos rápidos para os relatórios mais usados. Também existe a possibilidade de ser feita uma versão *mobile* que trabalhe em conjunto com a versão Web, e também automatizar o processo de cadastrar uma nova base para um cliente e fornecer acesso a esse.

REFERÊNCIAS

4LINUX. **Curso Tomcat: servidores de aplicações Java EE com Tomcat**. Disponível em: <http://www.4linux.com.br/cursos/curso-tomcat.html>. Acesso em: 1ago. 2013.

ALVAREZ, M. A. **Introdução a Kendo Ui**.Disponível em: <http://www.criarweb.com/artigos/introducao-kendo-ui.html>. Acesso em: 17 ago. 2013.

ANTONIO, Erik Aceiro e FERRO, Milene. **Análise Comparativa Entre os Principais Frameworks de Desenvolvimento JAVA**. Disponível em: <http://wright.ava.ufsc.br/~alice/conahpa/anais/2009/cd_conahpa2009/papers/final139.pdf>. Acesso em: 30ago. 2013.

APACHE. **Apache Tomcat**. Disponível em: <http://tomcat.apache.org/index.html>. Acessado em: 20 mar. 2013.

ARCHITECT. **Power Architect**. Disponível em: <http://www.sqlpower.ca/page/architect_download_os>. Acesso em: 31 ago. 2013.

ASTAH. **Astah**. Disponível em:<http://astah.net/download>.Acessoem: 31 ago. 2013.

BEZEMER, C.; ZAIDMAN, A. **Multi-tenant SaaS applications: maintenance dream or nightmare?**. Nova York: ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), 2010.

BURNETTE, Ed. **Eclipse IDE: guia de bolso**. Porto Alegre: Bookman, 2006.

CAELUM. **CAELUM**. Disponível em: <http://www.caelum.com.br/>.Acesso em: 28 set. 2013.

CANDELORO, Raúl. **A Hora da Virada: Como Assumir o Controle de Sua Empresa e Fazê-La Voltar a Dar Lucro em Tempos Difíceis.** São Paulo: Elsevier, 2009.

CAVALCANTI, Lucas. **Vraptor3**. Disponível em: <http://www.infoq.com/br/articles/VRaptor3>. Acesso em: 02 ago. de 2013.

CAELUM. **Apresentação técnica**. Disponível em: <http://vraptor.caelum.com.br/pt/>. Acessoem: 02 ago. de 2013.

CHONG ,Frederick; CARRARO**,**Gianpaolo.**Architecture Strategies for Catching the Long Tail**.Disponível em: <http://msdn.microsoft.com/en-us/library/aa479069.aspx>. Acesso em: 05 set. 2013.

DEVMEDIA, E. **Conheça o Apache Tomcat**. Disponível em: <http://www.devmedia.com.br/conheca-o-apache-tomcat/4546> Acesso em: 27 ago. 2013.

ECLIPSE. **Eclipse Juno**. Disponível em:<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junosr2>. Acesso em: 31 ago. 2013.

FREIRE, A., SILVEIRA. P. **VRaptor - SimpleAndQuick*Web* Framework**. Disponível em: <https://wiki.softwarelivre.org/pub/WSL/WSL2004/anais-2004.pdf#page=39>. Acesso em: 23 ago. 2013.

GONCALVES, Edson. **Desenvolvendo aplicações web com JSP, Servlets, Java Server Faces, Hibernate, EJB3 Persistence e Ajax**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2007.

GRÖßLER, A.; GRÜBNER, A.; MILLING, P. M. **Organisational adaptationprocesses to external complexity.International Journal of Operations &Production Management**.Vol. 26, n. 3, p. 254-281, 2006.Disponívelem:<http://arxiv.org/ftp/arxiv/papers/1110/1110.4299.pdf>. Acesso em: 05 set. 2013.

HIBERNATE. **Hibernate 4.0**. Disponível em:<http://www.hibernate.org/downloads>.Acesso em: 31 ago. 2013.

JEE. **Java Enterprise Edition**. Disponível em:<http://www.oracle.com/technetwork/java/javaee/downloads/index.html>. Acesso em: 31 ago. 2013.

JELASTIC. **Jelastic**.Disponível em:<http://jelastic.com/pt/>. Acesso em: 31 ago. 2013.

JOBSTRAIBIZER, Flávia. **Guia Profissional PHP**. São Paulo: Digerati Books, 2009.

KAPPEL, G.;MICHLMAYR, E.;PRÖLL,B.;REICH, S.;RETSCHITZEGGER, W. **Web Engineering - Old wine in new bottles?**. Munique: ICWE 2004.

KENDO. **Kendo UI**. Disponível em:<http://www.kendoui.com/download.aspx>. Acessoem: 31 ago. 2013.

KWOK, T.; NGUYEN, T.; LAM, L**.A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application**.International Conference on Services Computing. Washington: IEEE Computer Society, 2008. p. 179-186.

LOWE, David B.; HENDERSON-SELLERS, Brian.**Characteristics of Web Development Processes**.Disponível em: <http://services.eng.uts.edu.au/~dbl/archive/2001-low01c.pdf>. Acesso em: 05set. 2013.

MCGEE, J.; PRUSAK, L.**Gerenciamento Estratégico da Informação**.Rio de Janeiro: Editora Campos, 1995.

NIST.**Definition of Cloud Computing v15**.Disponível em: < http://csrc.nist.gov/groups/SNS/cloud-computing/>. Acessadoem 05 set. 2013.

NITU.**Configurability in SaaS (software as a service) applications**.Pune, India: Proceedings of the 2nd annual India Software Engineering Conference (ISEC), 2009.

OLIVEIRA, Djalma de Pinho Rebouças de. **Sistemas de informação gerenciais: estratégias, táticas, operacionais**. 8. ed., São Paulo: Atlas,1992.

PEREIRA, Maria José Lara de Bretãs; FONSECA, João Gabriel Marques. **Faces da Decisão: as mudanças de paradigmas e o poder da decisão**. São Paulo: Makron Books, 1997.

POSTGRESQL. **PostgreSQL**. Disponível em:<http://www.postgresql.org/download/>.Acesso em: 31 ago. 2013.

PRAHALAD, C. K.; KRISHNAN, M. S. **Nova Era Da Inovaçao.** São Paulo: Elsevier, 2008.

PRESSMAN, R. S. **Engenharia de software**. 5 ed. São Paulo: McGraw Hill, 2002.

SAMPAIO, Cleuton. **Guia do Java: Enterprise Edition5: desenvolvendo aplicações corporativas**. Rio de Janeiro: Brasport, 2007.

SAMPAIO, C. **Java Enterprise Edition6: desenvolvendo aplicações corportivas**.Rio de Janeiro: Brasport, 2011.

SINGH, Inderjeet; STEARNS, Beth; JOHNSON, Mark.**Designing Enterprise Applications with the J2EE Platform. SecondEdition**. 2nd ed. New Jersey, USA: Addison-Wesley, 2002.

STAIR, R. M. **Princípios de Sistemas de Informação: Uma abordagem gerencial**. 8 ed. Rio de Janeiro: LTC, 2001.

STOUT, R. **Dominando a World Wide*Web***. São Paulo: Makron Books, 1997.

TAURION, C. **CloudComputing: Computação em Nuvem: Transformando o mundo da tecnologia da informação**. Rio de Janeiro: Brasport, 2009.

TOMCAT. **Tomcat 7.0**. Disponível em:<http://tomcat.apache.org/download-70.cgi>. Acesso em: 31 ago. 2013.

VRAPTOR. **Vraptor3**. Disponível em:<https://code.google.com/p/vraptor3/downloads/list>. Acesso em: 31 ago. 2013.

WEBSOLUTE. **WEBSOLUTE.** Disponível em:<http://www.websolute.com.br/>. Acesso em: 29 set. 2013.