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

CÁSSYO JUNIOR REBONATO KIST

SISTEMA WEB PARA LOJAS DE ALUGUEL E VENDA DE ROUPAS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2012

CÁSSYO JUNIOR REBONATO KIST

SISTEMA WEB PARA LOJAS DE ALUGUEL E VENDA DE ROUPAS

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.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2012

ATA Nº: **202**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **CáSSYO JúNIOR REBONATO KIST**.

Às 17:30 hrs do dia 19 de outubro de 2012, Bloco S da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Rúbia E. O. Schultz Ascari (Convidada) e Oriane Gisela Thaler (Convidada), para avaliar o Trabalho de Diplomação do aluno Cássyo Júnior Rebonato Kist, matrícula 1066820, sob o título **Sistema Web para Lojas de Aluguel e Venda de Roupas**; 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 18:30 hrs foi encerrada a sessão.

Beatriz Borsoi

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Rúbia E. O. Schultz Ascari

Profa. Rúbia E. O. Schultz Ascari, M.Sc.
Convidada

Oriane Thaler

Profa. Oriane Gisela Thaler, Grad
Convidada

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

KIST, Cássyo Junior Rebonato. Sistema web para lojas de aluguel e venda de roupas. 2012. 50 f. 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, 2012.

As aplicações *web* facilitam o acesso dos usuários porque as exigências básicas para uso dessas aplicações é um computador com acesso à Internet e com um navegador *web* instalado. Esse tipo de aplicação também é útil para empresas que possuem filiais. Com uso da Internet não há necessidade de uma rede dedicada para conectar os computadores da matriz e das filiais. Considerando esse contexto, verificou-se que o desenvolvimento de um sistema *web* poderia ser relevante para o gerenciamento de uma empresa de aluguel e venda de roupas que possua filiais. A implementação desse sistema também tem o objetivo de apresentar a forma de uso das tecnologias empregadas, destacando-se a linguagem Java para *web*, os *frameworks* Hibernate e PrimeFaces e a biblioteca JQuery.

Palavras-chave: Java para *web*. Aplicações *web*. PrimeFaces. JQuery

ABSTRACT

KIST, Cássyo Junior Rebonato. Web application to manage rental and sale of clothes. 2012. 50 f. Relatório de Estágio Supervisionado - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2012.

The web applications provide easy access for users because the basic requirement to access these applications is a computer with Internet access and a web browser installed. This type of application is also useful for companies that have branches. With use of the Internet there is no need for a dedicated network to connect computers and array of subsidiaries. Considering this context, it was found that the development of a web-based system could be relevant to the management of a company that rents and sells clothes and has offices. The implementation aims to show how to use the technologies employed, especially Java web and the frameworks PrimeFaces and Hibernate, and the JQuery library.

Keywords: JavaEE. Web applications. PrimeFaces. JQuery.

LISTA DE FIGURAS E QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

API	*Application Programming Interface*
CGI	*Common Gateway Interface*
CSS	*Cascading Style Sheet*
DAO	*Data Access Objects*
DDL	*Data Definition Language*
DOM	*Document Object Model*
EIS	*Executive Information System*
EJB	*Enterprise JavaBeans*
EJB	*Enterprise Java Beans*
FTP	*File Transfer Protocol*
HTML	*Hyper Text Markup Language*
IBM	I*nternational Business Machines*
IDE	*Integrated Development Environment*
IIS	*Internet Information Services*
JavaEE	*Java Enterprise Edition*
JDBC	*Java Database Connectivity*
JIT	*Just Int Time*
JNDI	*Java Naming and Directory Interface*
JSF	*Java Server Faces*
JSP	*Java Server Page*
JTA	*Java Transaction API*
JVM	*Java Virtual Machine*
MVC	*Model-View-Controller*
PHP	*PHP Hypertext Preprocessor*
RIA	*Rich Internet Application*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
XHTML	*eXtensible Hyper Text Markup Language*
XML	*eXtensible Markup Language*
XSD	*XML Schema Document*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto do sistema desenvolvido como resultado deste trabalho, os seus objetivos e a sua justificativa. E por fim está a organização dos capítulos que compõem o texto.

1.1 CONSIDERAÇÕES INICIAIS

Uma empresa que fabrica e aluga roupas, comumente denominada de atelier de costura ou de locação, geralmente disponibiliza um grande número de peças para que seus clientes possam escolher. Além de locar peças prontas e de solicitar a confecção de peças que adquirirá, o cliente pode solicitar a confecção de uma peça que alugará. A solicitação de confecção pelo cliente de uma peça que ele alugará é definida como primeira locação.

Para as peças locadas é importante manter o histórico das locações para que o cliente possa ser informado dos usos anteriores da respectiva peça. Para determinadas peças, especialmente confecções de vestuário feminino, é necessário haver um controle para que a locação não se torne repetida em eventos semelhantes ou eventos de público semelhante, por exemplo.

A existência de matriz e filiais nesse tipo de negócio aumenta a vida útil das peças a serem locadas. Isso porque as peças que já foram alugadas em uma das lojas podem ser disponibilizadas para aluguel em outras lojas, proporcionando maior rentabilidade para o empresário.

O gerenciamento das peças locadas e mesmo da confecção, quando a empresa possui matriz e filiais, pode ser facilitado pelo uso de um sistema *web*. Assim, este trabalho se refere ao desenvolvimento de um sistema *web* contará com um servidor (computador na matriz) que disponibilizará acesso aos terminais (cliente *web*) nas filiais. O controle de estoque e o gerenciamento do banco de dados são centralizados, mas o acesso e a manutenção do sistema são distribuídos. A manutenção, nesse caso, se refere às operações realizadas no banco de dados, como os cadastros e o controle das locações.

O sistema desenvolvido é composto basicamente de cadastros, controle de aluguel e de distribuição das peças entre a matriz e as filiais da empresa. A implementação do sistema está

baseada no padrão de projetos MVC (*Model-View-Controller*). Esse padrão será usado para facilitar a implementação e mesmo a manutenção, em termos de código, do sistema desenvolvido. Os dados para a modelagem do sistema são provenientes de uma empresa de aluguel e confecção de roupas que possui matriz e filiais.

O desenvolvimento desse sistema é uma oportunidade de aprendizado pelas tecnologias utilizadas na implementação, destacando-se a linguagem Java, os *frameworks* Hibernate e PrimeFaces, a biblioteca JQuery e o padrão de projetos MVC. Além de fornecer uma forma de controle e gerenciamento para empresas de aluguel e venda de roupas sob medida.

1.2 OBJETIVOS

O objetivo geral se refere ao resultado principal a ser obtido com a realização deste trabalho. Os objetivos específicos complementam o objetivo geral.

1.2.1 Objetivo Geral

Implementar um sistema *web* para gerenciar aluguel e venda de roupas de uma empresa com matriz e filiais utilizando a linguagem Java.

1.2.2 Objetivos Específicos

Dentre os objetivos específicos, destacam-se:

- Apresentar a forma de uso do padrão de projetos MVC no desenvolvimento de uma aplicação Java para *web*.
- Apresentar a forma de uso das tecnologias de suporte ao desenvolvimento Java para *web*, incluindo o *framework* PrimeFaces e a biblioteca JQuery.

1.3 JUSTIFICATIVA

O desenvolvimento de um sistema *web* para uma empresa de aluguel e de venda de roupas se justifica fortemente quando a empresa é composta por matriz e filiais. Com um sistema *web*, a comunicação entre elas fica mais eficaz. Além da comunicação, o controle das operações realizadas e a visão geral do negócio se tornam mais fáceis e efetivas. É também uma possibilidade de aumentar a vida útil das peças locadas porque elas podem ser redistribuídas entre a matriz e as filiais.

Para o desenvolvimento de aplicações para *web,* diversas tecnologias podem ser utilizadas incluindo dos CGIs (*Common Gateway Interface*) com HTML (*HyperText Markup Language*) às aplicações Internet ricas. As linguagens de programação também são distintas, como, por exemplo, .Net, Java e PHP (*PHP Hypertext Preprocessor*).

Para a implementação do sistema resultado deste trabalho, optou-se pela linguagem Java pelas funcionalidades que a mesma oferece no desenvolvimento de sistemas para *web*. Essas funcionalidades estão, também, relacionadas às tecnologias agregadas ao uso da linguagem. Dentre essas tecnologias estão os *frameworks* e os padrões de projeto.

Os padrões de projeto, embora sejam conceituais, definem a forma de empregar tecnologias na resolução de determinados problemas. Esses padrões não estão relacionados à linguagem, mas fornecem uma maneira de resolver problemas recorrentes em programação, otimizando o uso dos recursos que as linguagens e as tecnologias possuem.

Para o desenvolvimento deste trabalho serão utilizados os *frameworks* PrimeFaces e Hibernate, a biblioteca JQuery para interface e o padrão de projetos MVC. Eles serão utilizados visando agilizar a implementação do sistema e prover facilidade de interatividade do usuário.

Este trabalho está centrado na implementação do sistema, uma primeira versão da modelagem foi desenvolvida como trabalho de estágio. No estágio também foram implementados alguns cadastros, mas houve mudanças nas tecnologias e toda a codificação foi refeita.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos, dos quais este é o primeiro. O Capítulo 2 apresenta o referencial teórico que fundamenta as tecnologias e os conceitos utilizados no

desenvolvimento do trabalho. O Capítulo 3 apresenta os materiais e o método empregado na modelagem e na implementação do sistema. No Capítulo 4 está a modelagem e a implementação obtidas como resultado da realização deste trabalho. No Capítulo 5 está a conclusão e por fim estão as referências utilizadas.

2 JAVA PARA WEB

Este capítulo apresenta os conceitos relacionados à plataforma JavaEE (*Java Enterprise Edition*) no desenvolvimento de aplicações *web* e ao padrão de projetos MVC, que são utilizados na implementação do sistema resultado deste trabalho.

2.1 A PLATAFORMA JAVAEE

A plataforma JavaEE, também conhecida como J2EE ou JEE, está voltada para o desenvolvimento de aplicações multicamadas, baseadas em componentes que são executados em um servidor de aplicações. A plataforma JavaEE é considerada um padrão de desenvolvimento porque é necessário seguir determinadas regras para que o produto de software gerado seja compatível com JavaEE. O desenvolvimento de software para esse ambiente é realizado por meio da linguagem de programação Java e de um conjunto de ferramentas de desenvolvimento e de tecnologias associadas.

A plataforma JavaEE é multicamadas, cujas camadas podem estar distribuídas em várias máquinas. Geralmente conhecida como três camadas, mas não necessariamente somente três. Por exemplo: camada cliente que executa na máquina do cliente; camada *web* que executa no servidor JavaEE; camada de negócio que executa no servidor JavaEE; e camada EIS (*Executive Information System*) que executa no servidor EIS. A Figura 1 apresenta uma visão geral das camadas típicas desse padrão.

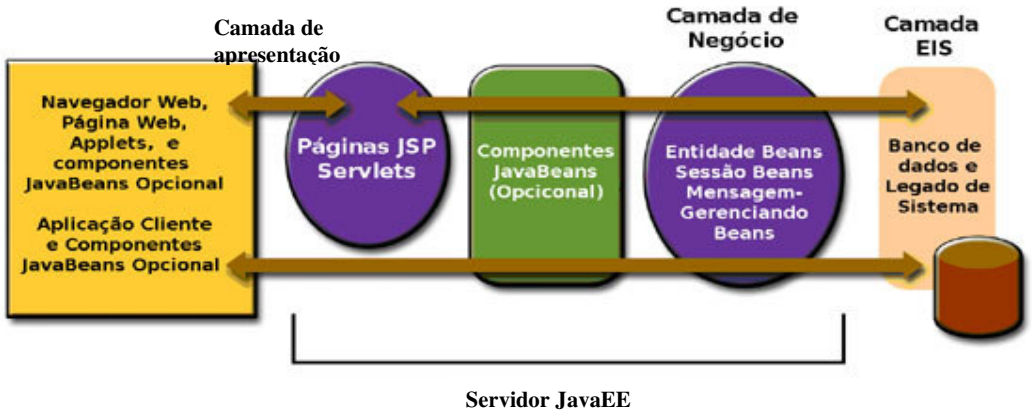

Figura 1 - Apresentação das camadas e dos contêineres
Fonte: Adaptado de Miranda e Silva (2006).

A plataforma JavaEE trabalha com o conceito de contêiner de componentes. Um contêiner fornece serviços aos componentes, tais como, segurança. Dentre os exemplos de contêiner estão o servidor JavaEE; contêiner EJB (*Enterprise JavaBeans*); contêiner *web*; contêiner aplicação cliente; e contêiner *applet*.

A plataforma JavaEE é aberta, baseada em padrões, independente de *hardware* e de sistema operacional que as aplicações desenvolvidas executarão. As aplicações *web* baseadas na plataforma JavaEE usam o padrão de projetos MVC que é composto por três componentes arquiteturais (GAMMA et al., 2000):

a) lógica de apresentação – que define a visão e está relacionada a exibir os dados ou informações da aplicação.

b) lógica de controle - coordena o modelo e a visão e exibe a interface.

c) lógica de negócio – que é representada pelo modelo. O modelo encapsula os dados do aplicativo e as regras de negócio.

Para Prajapati e Dabhi (2009), aplicações *web* desenvolvidas em plataforma JavaEE possuem alta qualidade e atendem mudanças de requisitos de negócio e do cliente. Chung e Lee (2003) exemplificam o desenvolvimento de aplicações *web* com o uso da linguagem Java e de tecnologias relacionadas a XML (*eXtensible Markup Language*). Para a camada de apresentação eles citam o uso de: *eXtensible Hyper Text Markup Language* (XHTML), *Cascading Style Sheet* (CSS), *JavaScript,* XML, *XML Schema Document* (XSD) e *Java Server Page* (JSP). Para a camada de negócios citam o uso de JavaBean. E para a camada de acesso a dados, o MySQL como sistema de gerenciamento de banco de dados e o *driver Database Connectivity*. Como servidor *web* eles incluem o Apache, JSP e Tomcat.

Como a tecnologia Java é orientada a objetos e independente de plataforma, características, como escalabilidade, portabilidade, reusabilidade, segurança, alto desempenho e flexibilidade são inerentes as suas classes e componentes (PRAJAPATI; DABHI, 2009). *Servlet*, componentes *JavaBeans* e EJB (*Enterprise Java Beans*), por exemplo, são classes Java. A linguagem de *script* JSP é usada para criar páginas *web*. Esses elementos são utilizados para implementar o padrão de projetos MVC na plataforma JavaEE.

O Quadro 1 apresenta uma análise de como os atributos de qualidade de componentes são atendidos na plataforma JavaEE, considerando o padrão de projetos MVC.

Atributos	Suporte para atributos de qualidade para componentes MVC		
	Modelo	**Visão**	**Controle**
Desempenho	Compilador JIT (*Just Int Time*) para compilação rápida de *byte code*. Separa *thread* para requisição de usuários. Componentes EJB locais para melhorar desempenho.		
Extensibilidade	Todos os componentes são orientados a objetos.		
Escalabilidade	Componentes EJB são acessíveis por meio da visão/controle	Usa máscara URL (*Uniform Resource Locator)*	Segurança para componentes *Servlet* é realizada por programação
Robustez	Componentes executam sob o controle de JVM (*Java Virtual Machine)*, as classes são carregadas e *byte code* é verificado dinamicamente.		
Flexibilidade	Usa JNDI (*Java Naming and Directory Interface*) baseada em configuração de acesso a banco de dados	Usa páginas principais, inclusão de arquivos/páginas.	Usa métodos de requisição e resposta e cadeias de filtros.
Modularidade	Módulos *jar* EJB	Bibliotecas de *tags*	Filtros.
Reuso	Suporte a objetos EJB remotos	Biblioteca de *tags* customizadas, inclusão de páginas/arquivos e *links*.	Usando métodos de requisição e resposta e cadeias de filtros

Quadro 1 – Suporte a tributos de qualidade em JavaEE utilizando MVC

Fonte: traduzido de Prajapati e Dabhi (2009, p. 1668).

2.2 Padrão de projetos MVC

Uma aplicação JavaEE usa o modelo três camadas: apresentação, negócio e persistência. Na camada de apresentação estão as páginas JSP, a camada de negócio contém a modelagem do domínio do problema e a camada de persistência contém as classes com conhecimento sobre a persistência de objetos no banco de dados.

O *Model-View-Controller* é um padrão de arquitetura que divide a aplicação em controladores que tratam as entradas dos usuários, no modelo que prove as funcionalidades principais e nas visões que apresentam as informações para os usuários (SOUZA, 2008).

Em aplicações complexas, que enviam uma série de dados para o usuário, o desenvolvedor frequentemente necessita separar os dados (*model*) da interface (*view*). Utilizando MVC, alterações feitas na interface não afetarão a manipulação dos dados e estes poderão ser reorganizados sem alterar a interface do usuário. O MVC resolve esse problema por meio da separação das tarefas de acesso aos dados e a lógica do negócio da apresentação e da interação com o usuário. Isso é feito por meio da inserção de um componente entre *model* e *wiew*. Esse componente é o *controller*.

O objetivo básico do padrão de projeto MVC é separar dados ou lógica de negócios (*model*) da interface do usuário (*view*) e do fluxo da aplicação (*controller*). Assim, uma mesma lógica de negócios pode ser acessada e visualizada por interfaces distintas. Na arquitetura MVC, a lógica de negócios (modelo) não sabe quantas nem quais interfaces com o usuário estão exibindo seu estado. A Figura 2 apresenta um esquema de representação da arquitetura MVC utilizando JSP para a camada de interface.

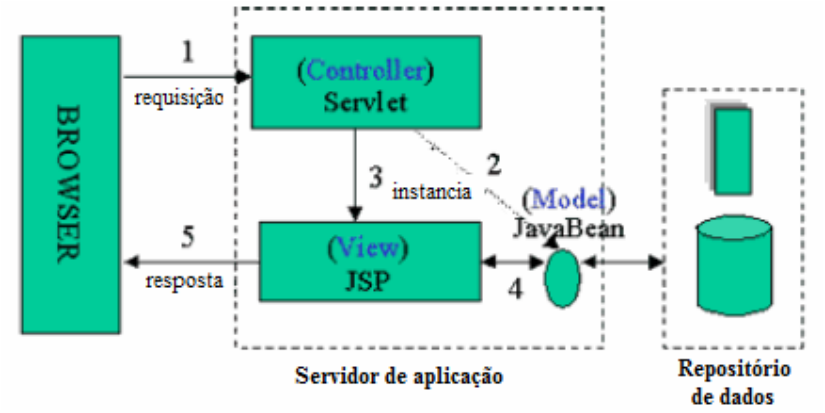

Figura 2 – Arquitetura MVC com JSP
Fonte: traduzido de Khin (2009, p. 280).

Na Figura 2, os números indicam (KHIN, 2009):

1) A requisição é enviada pelo usuário por meio de um navegador *web* (o *browser*), para o controle que tratará essa requisição.
2) O controle (*servlet*) instancia o modelo, realiza as ações necessárias e faz as devidas conexões com o JavaBean para obter as informações solicitadas.
3) O controle envia a representação para a visão por meio das páginas JSP, com as respostas conforme a solicitação do usuário.
4) A página JSP é atualizada ou composta com informações provenientes do modelo.
5) A resposta é enviada ao cliente por meio de uma página JSP, com todos os dados que os métodos produziram ao executar o código do sistema.

De acordo com a representação da Figura 2, o *servlet* age como o controle e está encarregado do processamento das requisições e da criação de quaisquer *beans* ou objetos usados pelo JSP, bem como decidir, dependendo das ações do usuário, quais páginas JSP devem ser encaminhadas para atender a requisição solicitada. Neste modelo não há lógica de processamento na página JSP. Esta é simplesmente responsável por recuperar qualquer objeto ou *bean* que pode ter sido previamente criado pelo *servlet* e extrair o conteúdo dinâmico do *servlet* para inserção em modelos (*templates*) estáticos. Essa abordagem resulta tipicamente

na separação da apresentação do conteúdo, deixando clara a delimitação dos papéis e responsabilidades dos programadores e dos projetistas das páginas (KHIN, 2009).

Na Figura 2, o *browser* representa a visão definida em JSP e fornece a interação do usuário com a visão elaborada em JSP. A visão envia informações para o controlador (*controller*) que altera o modelo. O modelo (*model*) representa a lógica de negócio e envia eventos para a visão (*view*) indicando as mudanças.

As funcionalidades de cada uma das três partes do modelo MVC, de acordo com Bianchini (2008), Prajapati e Dabhi (2009) e Wang (2011) são:

a) Modelo - representa os dados da aplicação e a sua lógica de negócio que determina o acesso aos dados. O modelo mantém o estado persistente dos dados e fornece ao controlador a possibilidade de acesso às funcionalidades da aplicação. Essas funcionalidades são encapsuladas pelo próprio modelo. O modelo notifica a visão que recebe solicitações de alterações e lê o conteúdo do modelo para ser atualizada.

b) Visão - é a camada de interface com o usuário. Nesta camada o usuário vê o estado do modelo e pode manipular a interface para ativar a lógica do negócio. A visão apresenta o conteúdo de um modelo, coleta os dados para o modelo, especifica como esses dados devem ser apresentados e encaminha para o controlador as ações do usuário.

c) Controle - é o componente que define o comportamento da aplicação. Ele recebe e interpreta as ações do usuário e as traduz para ações de atualização do modelo que, por exemplo, ativam processos da lógica de negócio ou alteram o estado do modelo. O controlador se encarrega de selecionar a visualização que será mostrada a partir da interação com o usuário, atualizando a visão apresentada ao usuário. O controlador transforma eventos gerados pela interface em ações de negócio, alterando o modelo.

3 MATERIAIS E MÉTODO

Neste capítulo estão os materiais e o método utilizados no desenvolvimento deste trabalho. Os materiais se referem às tecnologias e às ferramentas utilizadas na modelagem e na implementação do sistema e o método apresenta a sequência das principais atividades realizadas para desenvolvimento do trabalho.

3.1 MATERIAIS

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

a) Astah* Community (ASTAH, 2012) – para a documentação da modelagem que é baseada na UML (*Unified Modeling Language*).

b) Eclipse (ECLIPSE, 2012) – IDE (*Integrated Development Environment*) para desenvolvimento.

c) Linguagem Java (JAVA, 2012) – para a implementação do sistema, complementada por HTML por ser desenvolvimento para *web*.

d) JavaServer Faces 2.0 (JSF, 2012) – para controle entre as classes Java e os JSPs.

e) PrimeFaces (PRIMEFACES, 2012) – para desenvolvimento da interface.

d) Jquery (JQUERY, 2012) – componentes para os *scripts* da interface.

f) MySQL 5 (MYSQL, 2012) – como banco de dados.

g) Hibernate (HIBERNATE, 2012) – *framework* para o mapeamento objeto relacional.

h) Apache Tomcat (TOMCAT, 2012) – como contêiner/servidor *web*.

i) DeZign (DEZIGN, 2012) para modelagem do banco de dados.

O Quadro 2 apresenta as tecnologias utilizadas para modelagem, implementação e execução do sistema. Elas estão distribuídas nas camadas de interface, aplicação (lógica de negócio) e dados.

	Modelagem	Implementação	Execução
Interface (visão)	Não realizada	IDE Eclipse JSF, PrimeFaces, JQuery, (JavaScript, HTML, CSS)	Navegador web com suporte a HTML e JavaScript
Lógica de negócio (controle)	Astah Community	IDE Eclipse Java,	Apache Tomcat
Banco de dados (modelo)	DeZign for Database v6	MySQL Hibernate	MySQL Admin Apache Tomcat

Quadro 2 – Tecnologias utilizadas na definição, implementação e execução do sistema

3.1.1 Astah Community

A ferramenta Astah Commuity é a sucessora da ferramenta Jude Community. É uma ferramenta para modelagem de sistemas com suporte para a UML 2.1 (ASTAH, 2012). Os diagramas podem ser impressos ou exportados como imagem.

3.1.2 IDE Eclipse

A IDE Eclipse (ECLIPSE, 2012) é uma ferramenta de desenvolvimento de código aberto para a construção de programas de computador desenvolvida em Java pela empresa IBM (*International Business Machines)*. Com o uso de *plugins*, a IDE Eclipse pode ser usada não somente para desenvolver em Java, mas também em, por exemplo, C, C++, PHP, ColdFusion, Python e Flex.

3.1.3 Linguagem Java

Java é uma linguagem de programação orientada a objetos, sendo assim, a maior parte dos elementos de um programa Java são objetos (JAVA, 2012). Dentre os elementos que não são objetos estão os tipos básicos, como o *int* e o *float*. O código é organizado em classes, que permite somente relacionamentos de herança simples entre si.

Chamadas a funções de acesso remoto (*sockets*) e os protocolos Internet mais comuns (HTTP, FTP (*File Transfer Protocol*), Telnet, etc.) são suportados em Java, facilitando a implementação de aplicativos baseados na arquitetura cliente/servidor e o desenvolvimento de aplicações *web*. Em aplicações *web*, a camada de interação com o usuário geralmente é desenvolvida em tecnologias agregadas como, por exemplo, JSF (*Java Server Faces*), Flex ou HTML.

3.1.4 JavaServer Faces

JavaServer Faces (JSF) é um *framework* padrão orientado a componentes de interface com o usuário para a plataforma JavaEE e é comumente referenciando como um *framework web* baseado em Java (JSF, 2012).

A tecnologia JavaServer Faces simplifica a construção de interfaces com o usuário para aplicações JavaServer. Isso porque os desenvolvedores podem construir aplicações *web* por meio da composição de componentes de interface com o usuário reusáveis, conectar esses componentes com um banco de dados e vincular os eventos gerados no cliente com os manipuladores de eventos que estão no servidor (GLASSFISH, 2012).

3.1.5 PrimeFaces

PrimeFaces é uma suíte de componentes de código fonte aberto para Java Server Faces 2.0 com um conjunto de mais de cem componentes JSF para o desenvolvimento de interfaces ricas (PRIMEFACES, 2012). Dentre as características de PrimeFaces, destacam-se (PRIMEFACES, 2012):

a) Possui um conjunto de componentes de interface gráfica para implementação de *Rich Internet Application* (RIA), como, por exemplo, *DataTable*, *AutoComplete*, *HtmlEditor* e *Charts* (gráficos).

b) Não há necessidade de configuração XML extra e não há dependências requeridas de outras tecnologias.

c) A construção de Ajax é baseada no padrão JSF 2.0 Ajax APIs (*Application Programming Interface*).

d) Possui um s*kinning Framework* com mais de 25 temas.

3.1.6 jQuery

jQuery (JQUERY, 2012) é uma biblioteca JavaScript que simplifica a identificação de *tags* em documentos HTML, a manipulação de eventos, o desenvolvimento de animação e de interações Ajax, facilitando o desenvolvimento *web*. jQuery possui as seguintes características (SILVA, 2010):

a) Uso de seletores CSS para localizar elementos da estrutura de marcação HTML da página;

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

c) Indiferença às inconsistências de renderização entre navegadores;

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

e) Admite programação encadeada, ou seja, cada método retorna um objeto;

f) É extensível, permite a criação e a inserção de novas funcionalidades em bibliotecas existentes.

jQuery se destina a adicionar interatividade e dinamismo às páginas *web* e proporciona facilidades como: adicionar efeitos visuais e animações; acessar e manipular o DOM (*Document Object Model*); buscar informações no servidor sem necessidade de recarregar a página; prover interatividade; alterar conteúdos; modificar apresentação e estilo; e simplificar tarefas específicas de JavaScript.

jQuery foi criada com a preocupação de ser uma biblioteca em conformidade com os padrões *web*, ou seja, compatível com qualquer sistema operacional, navegador e com suporte total para a biblioteca CSS3. Essa biblioteca foi criada e está em acordo com as diretrizes do W3C (*World Wide Web Consortium*), mas cabe ao desenvolvedor escrever os *scripts* de maneira a atender essa conformidade (SILVA, 2010).

3.1.7 MySQL

O MySQL é um servidor e gerenciador de banco de dados relacional, que utiliza a linguagem SQL (MYSQL, 2012). O MySQL possui licença como *software* livre e licença paga. Atualmente esse servidor e gerenciador de banco de dados atende também aplicações de grande porte, embora tenha sido inicialmente projetado e desenvolvido para aplicações de pequeno porte.

3.1.8 Hibernate

O Hibernate (HIBERNATE, 2012) é um *framework* para o mapeamento objeto-relacional escrito na linguagem Java e é também disponibilizado em .Net com o nome NHibernate. Esse *framework* facilita o mapeamento dos atributos entre uma base de dados relacional e o modelo de objetos de uma aplicação por meio do uso de arquivos XML.

O Hibernate visa minimizar a complexidade dos programas Java baseados no modelo orientado a objeto que trabalham com banco de dados no modelo relacional. O Hibernate

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

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

3.1.9 Apache Tomcat

O Apache Tomcat (TOMCAT, 2012) é um contêiner *servlet,* ou seja, um servidor de aplicações utilizado para interpretar aplicações escritas em Java para *web*. Ele não implementa um contêiner EJB, mas abrange parte da especificação JEE com tecnologias como *servlet* e JSP e tecnologias de apoio relacionadas, como segurança, JNDI *Resources* (API para acesso a diretórios) e JDBC *DataSources*.

O Tomcat pode atuar também como servidor *web*, ou pode funcionar integrado a um servidor *web* dedicado como o Apache ou o IIS (*Internet Information Services*). Como servidor *web*, ele provê um servidor HTTP puramente em Java. O servidor inclui ferramentas para configuração e gerenciamento, o que também pode ser feito editando-se manualmente arquivos de configuração formatados em XML.

3.1.10 DeZign For Database v6

DeZign for Databases (DEZIGN, 2012) é uma ferramenta para projeto de banco de dados que auxilia a criar e a manter bancos de dados. Essa ferramenta utiliza diagramas de entidade e relacionamentos para projetar e gerar bases de dados graficamente e sentenças SQLs.

Os principais recursos da ferramenta DeZign for Databases são (DEZIGN, 2012):

a) Modelar visualmente bancos de dados – a modelagem é feita graficamente por meio de diagramas de entidades e relacionamentos. O projeto pode ser apresentado em níveis de detalhes distintos.

b) Gerar bancos de dados – gerar *scripts* DDL (*Data Definition Language*) completos para criar o banco de dados ou gerá-lo diretamente.

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

d) Sincronizar modelos ou bancos de dados – as funcionalidades de sincronização e comparação são possíveis para todos os casos de uso: do modelo para a base de dados, do modelo para *script*, da base de dados para o modelo, do *script* para o modelo e do modelo para o modelo.

3.2 MÉTODO

No método utilizado para o desenvolvimento do sistema são consideradas as três fases análise, projeto e implementação propostas por Rumbaugh, et al. (1997) e as fases de análise, projeto, codificação e testes do modelo sequencial linear de Pressman (2005). Contudo, são acrescentadas outras denominações e mesmo fases visando explicitar as atividades realizadas. As fases definidas para o desenvolvimento do projeto foram:

a) Revisão da modelagem

A modelagem do sistema foi realizada como trabalho de estágio. Para o trabalho de conclusão de curso essa modelagem foi revisada e ajustes foram realizados. A revisão teve como base os processos de negócio de uma empresa de confecção e locação de roupas.

A definição dos requisitos foi baseada nas necessidades e interesse de uma empresa (loja) que fabrica e aluga roupas. Não foram realizadas entrevistas formais ou mesmo questionários porque o autor desse trabalho conhece os procedimentos dessa empresa. E conversas informais foram realizadas com o proprietário e com outros funcionários.

b) Implementação

Para o trabalho de estágio, além da modelagem, também foram implementados alguns cadastros visando estudar as tecnologias que eram, basicamente, Java com VRaptor. No trabalho de estágio foi utilizado o VRaptor para o controle entre as classes Java e os JSPs. Contudo, no prosseguimento da implementação, como trabalho de conclusão de curso, muitas dificuldades foram encontradas com o uso do VRaptor. Assim, fez-se um levantamento de outros *frameworks* que poderiam ser utilizados. As pesquisas indicaram que o uso de JSF com PrimeFaces, IceFaces ou RichFaces facilitaria a implementação.

Optou-se pelo uso de PrimeFaces juntamente com JSF pelos vários comentários favoráveis postados em *blogs* técnicos, ou seja, de programadores. Com o desenvolvimento do trabalho percebeu-se que, efetivamente, o uso do PrimeFaces é relativamente mais fácil, se comparado com o VRaptor. Assim, os cadastros implementados como trabalho de estágio foram completamente refeitos.

c) Testes

Em termos de testes eles estiveram centrados em verificar o código e foram realizados pelo próprio autor deste trabalho. Os testes de requisitos e mesmo de usuário serão realizados posteriormente. Isso porque há interesse em implantar esse sistema em uma empresa de confecção e aluguel de roupas. Essa empresa serviu de base para a definição dos requisitos.

4 RESULTADO

Este capítulo apresenta o sistema obtido como resultado da realização deste trabalho.

4.1 APRESENTAÇÃO DO SISTEMA

O sistema gerenciará os aluguéis de roupas da empresa. As roupas de aluguéis podem ser fabricadas sob medida, mas, neste caso, o valor do primeiro aluguel da roupa é mais elevado. As roupas para locação também podem ser ajustadas às medidas do cliente.

Depois de realizada uma locação, o cliente tem três dias para cancelar o aluguel. Se a locação for cancelada após esse período será cobrado 50% do valor da locação. As roupas disponibilizadas para locação poderão ser vendidas, caso o cliente deseje comprá-las. No caso de venda, há cobrança de juros se houver atraso no pagamento.

Ao ser efetuado um aluguel é definido um valor de entrada. Esse valor é destinado para compra de materiais, quando é o primeiro aluguel ou simplesmente como garantia. O restante do pagamento é realizado na entrega da peça ao cliente. A data de devolução é informada pelo cliente. Se essa data ultrapassar um final de semana, o cliente deverá pagar mais um aluguel da roupa, podendo ser 30% a menos, caso a roupa não tenha sido reservada para locação para outro cliente. Se na devolução a roupa retornar com danos, como rasgos e manchas impossíveis de serem retiradas, será cobrado o valor da roupa do cliente, após ser feita uma avaliação dos danos.

O sistema contará com um controle de estoque das roupas já confeccionadas, as roupas terão referência própria e dados adicionais como cor e tamanho (um vestido curto ou longo, por exemplo).

O controle financeiro da empresa será tratado pelo sistema como entradas e saídas. Contando com um cadastro de contas, boletos, tarifas e despesas em geral. O sistema emitirá um alerta (por *email*) para o cliente das contas a serem pagas com até uma semana de antecedência e o informará também sobre contas vencidas.

O sistema gerenciará também a devolução de roupas danificadas que não tenham conserto. Haverá uma opção para dar baixa na roupa e tirá-la do cadastro de aluguéis. Porém

ficará gravado no sistema qual foi o cliente responsável pelo dano, a data e o valor da roupa, além da identificação da roupa e do valor pago pelo cliente.

O acesso ao sistema será feito por meio de *login*. O cadastro dos usuários é realizado pelo administrador do sistema.

4.2 MODELAGEM DO SISTEMA

A Figura 3 contém o diagrama de casos de uso definido para o sistema. Esse diagrama também mostra que todas as operações serão feitas após o usuário estar logado no sistema. Por meio desse diagrama é possível, também, verificar que o gerente da empresa herda todas as funcionalidades da classe funcionário e assim tem acesso a todas as funcionalidades do sistema.

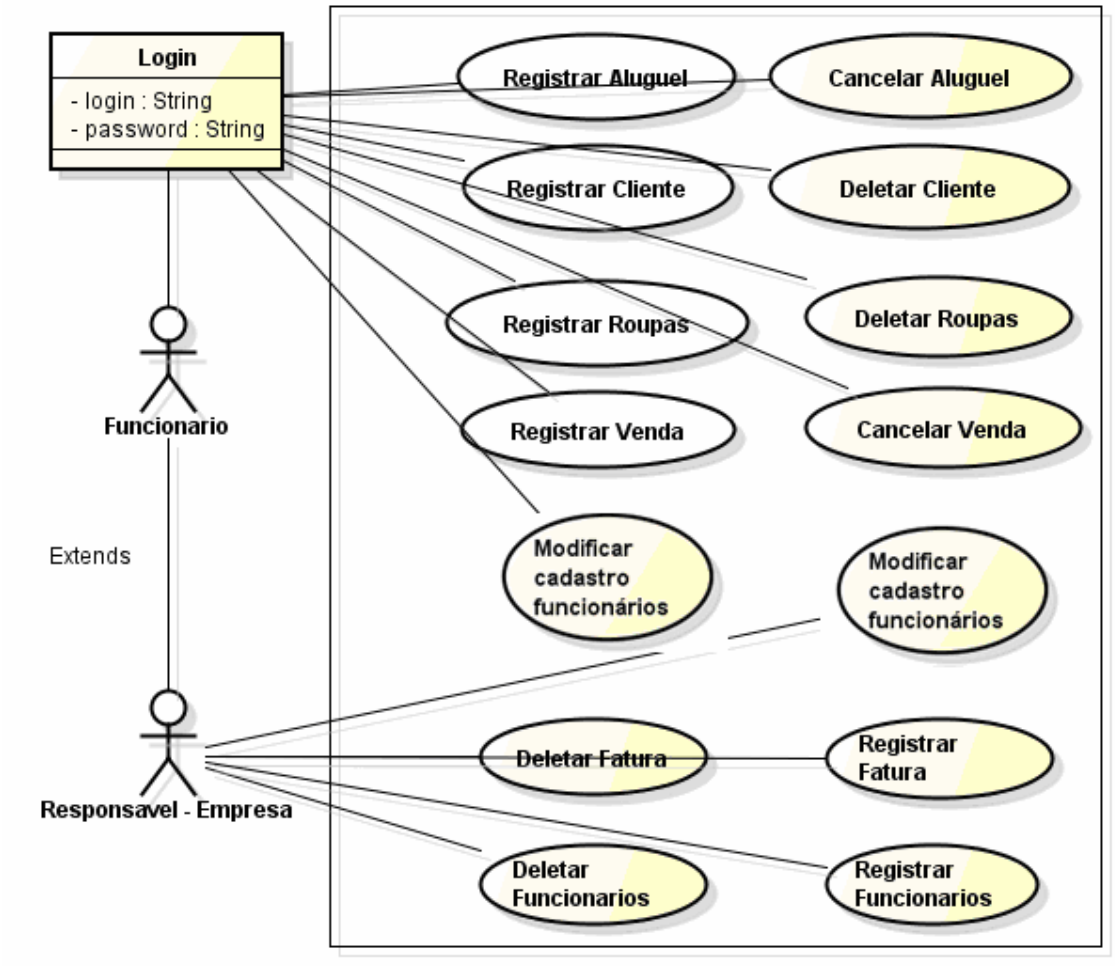

Figura 3 – Diagrama de casos de uso

A Figura 4 contém o diagrama de classes do sistema. Por meio desse diagrama é possível verificar que a única maneira de acesso ao sistema é por *login*. O acesso é realizado por meio da classe *login*. Por questão de segurança somente poderão fazer *login* os usuários que forem cadastrados pelo gerente da empresa usuária. O gerente tem permissões de administrador do sistema.

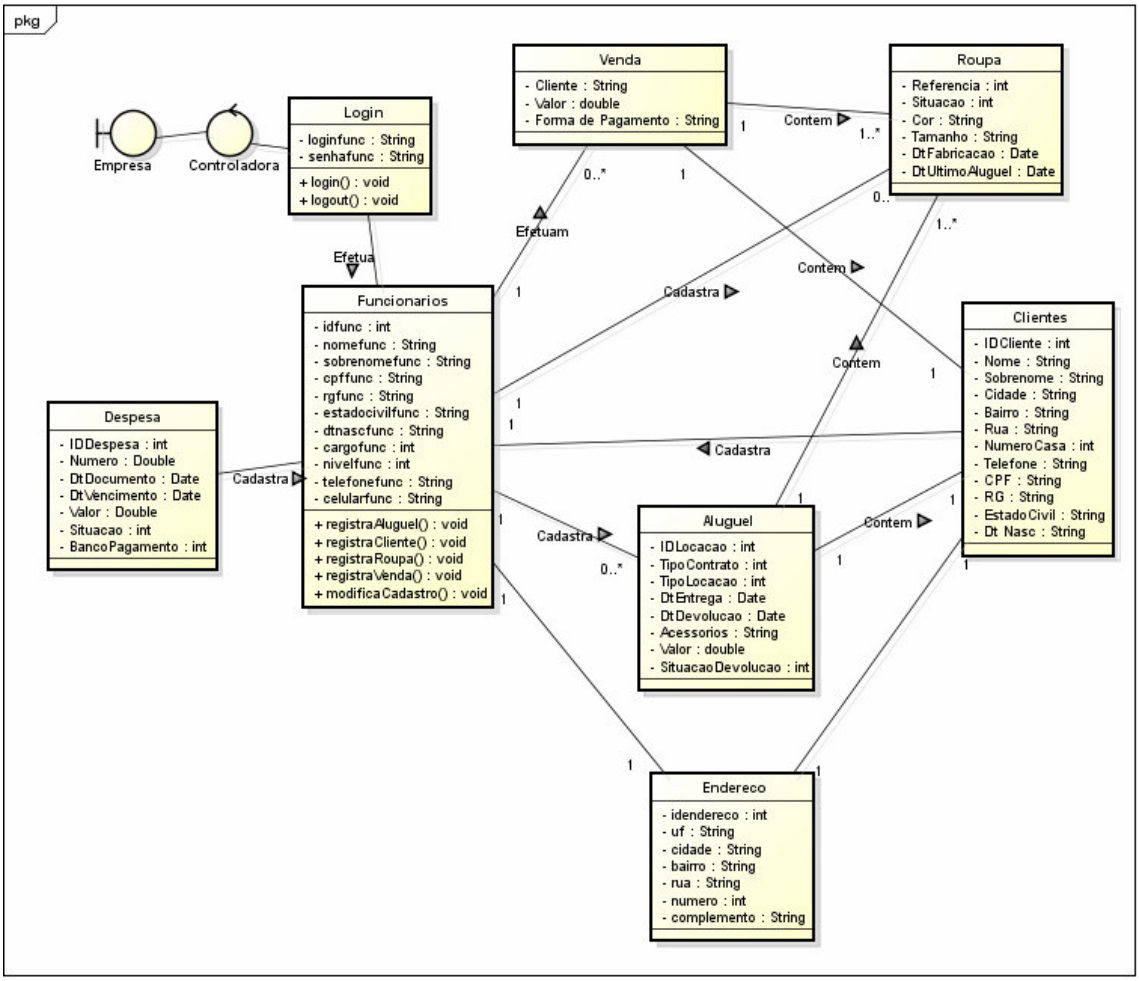

Figura 4 – Diagrama de classes do sistema

O sistema implementado realiza processos para chegar a resultados. Para exemplificar as funções representadas por esses processos foram utilizados diagramas de sequência. A Figura 5 contém o diagrama de sequência do processo de aluguel de roupas.

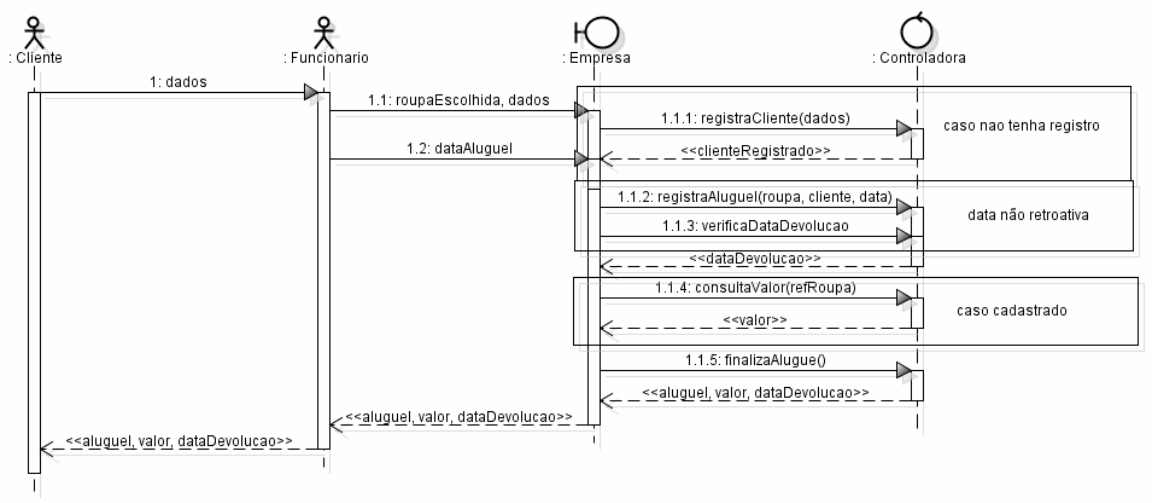

Figura 5 – Diagrama de sequência para o processo de aluguel de roupas

A Figura 6 ilustra o processo para efetuar *login* no sistema. Se o *login* está correto, o sistema realiza o *login* e informa que o mesmo foi efetuado.

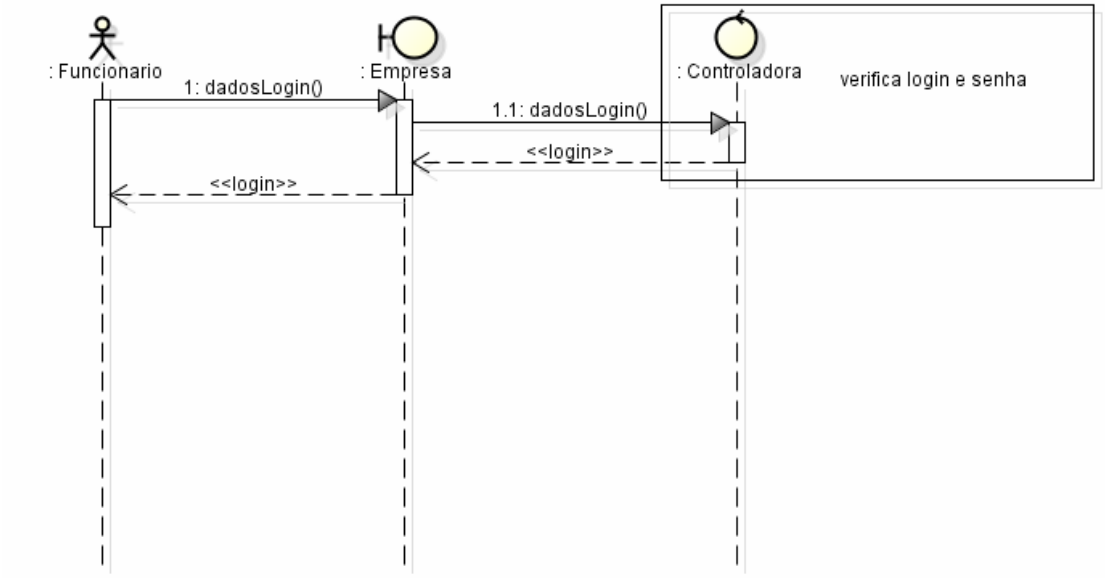

Figura 6 – Diagrama de sequência para login no sistema

Na Figura 7 está o processo de registro do cliente. Nesse processo, o cliente informa os seus dados ao funcionário que o cadastra no sistema. O sistema faz a verificação se o cliente já está cadastrado, caso ele não esteja é permitido o cadastro do mesmo.

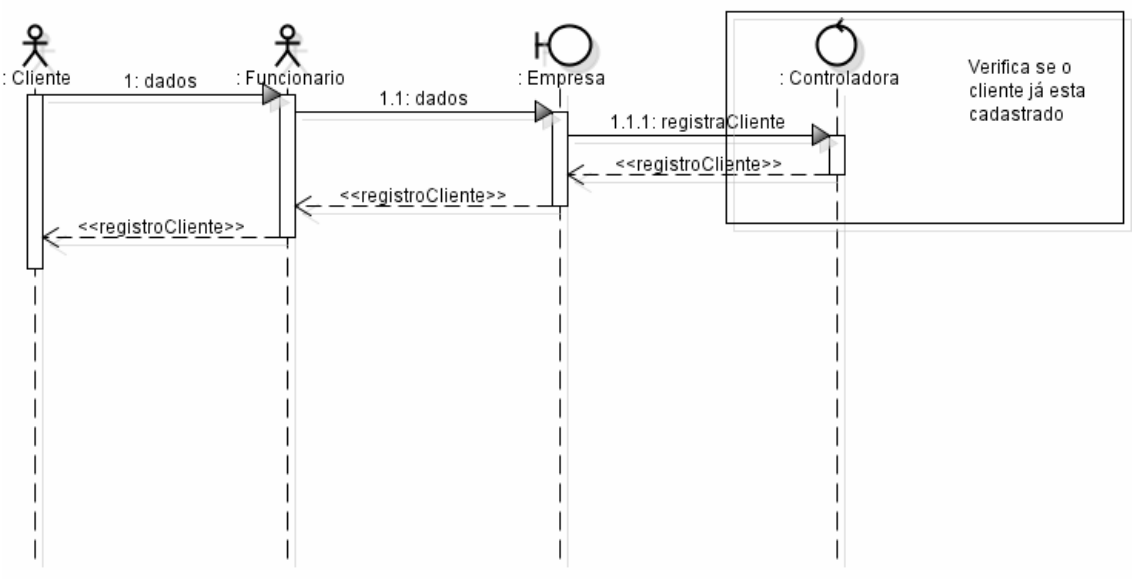

Figura 7 – Diagrama de sequência para cadastro de clientes

O diagrama da Figura 8 mostra o processo para cadastro de uma despesa. O gerente da empresa foi tratado como um funcionário neste diagrama, porém somente o gerente poderá cadastrar estas despesas. O sistema recebe os dados da despesa, verifica se ela já existe, caso não exista permite o cadastro da mesma.

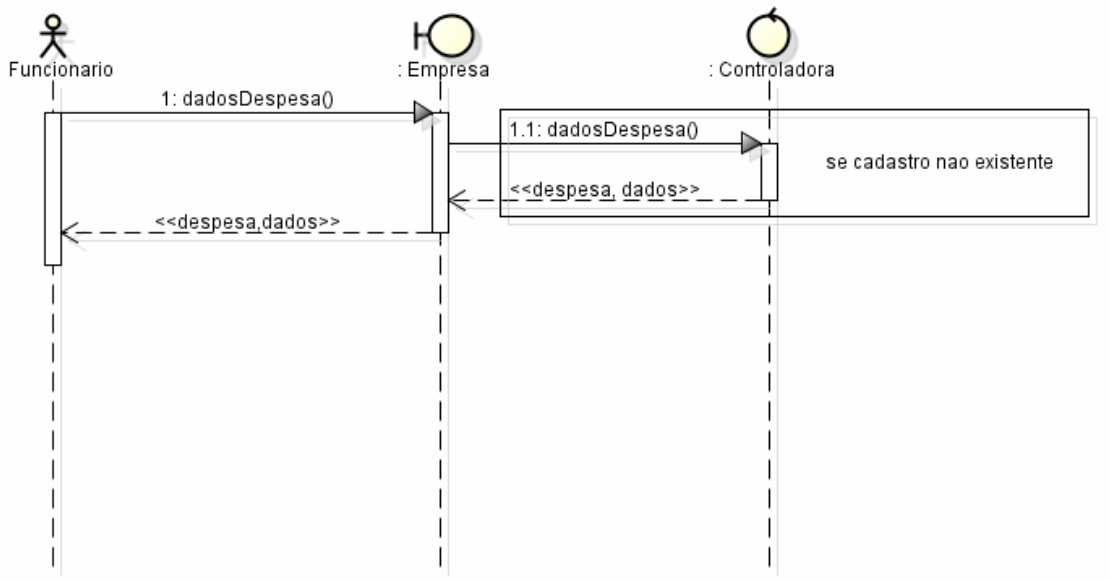

Figura 8 – Diagrama de sequência para cadastro de uma despesa

O diagrama da Figura 9 representa a venda de uma roupa. Nessa venda o cliente informa a roupa e o cliente. Se o cliente não estiver cadastrado, o sistema solicita seu registro e após isso é solicitada a data da compra da roupa para registrar a venda no sistema.

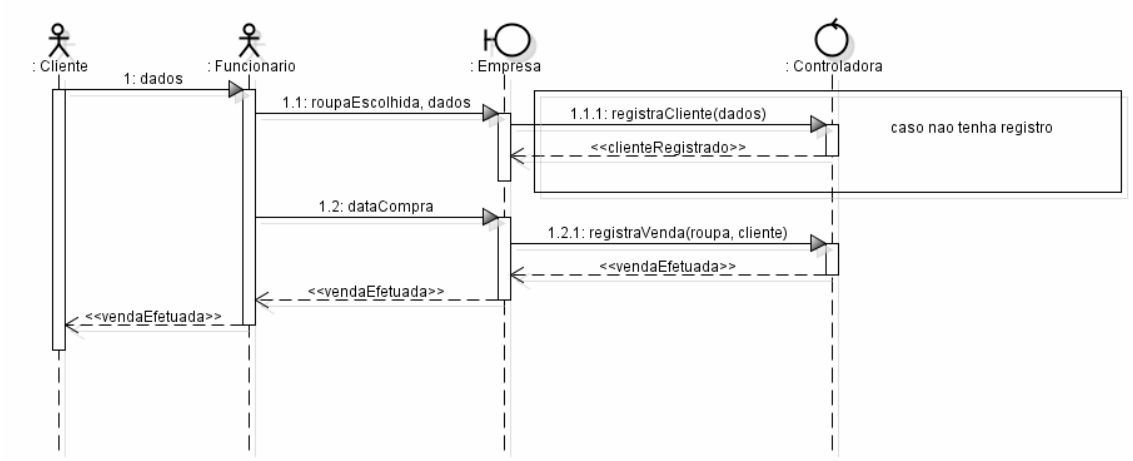

Figura 9 – Diagrama de sequência para a venda de roupas

Na Figura 10 está o diagrama que representa a sequência de ações para o registro de uma roupa. Para cadastrar uma roupa, o funcionário informa os dados e o sistema verificará se a mesma já está cadastrada. Caso não esteja, permitirá fazer o cadastro da mesma, informando se o cadastro foi efetuado com sucesso.

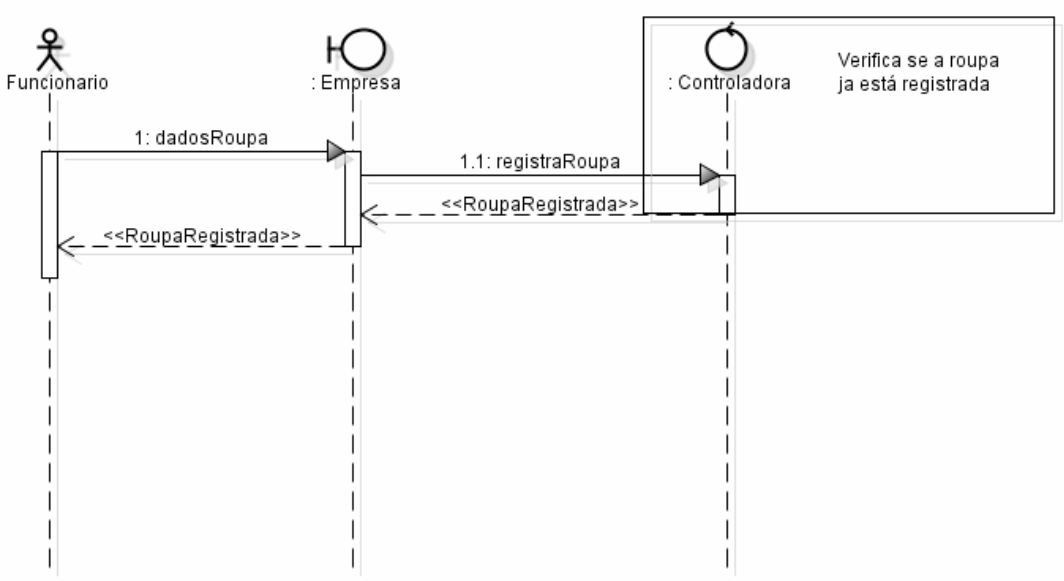

Figura 10 – Diagrama de sequência para cadastrar uma roupa no sistema

4.3 DESCRIÇÃO DO SISTEMA

Ao acessar o sistema é apresentada uma tela de *login*. Nessa tela, o usuário informa os dados para acesso que são o identificador do usuário (*login*) e a respectiva senha. A Figura 11 apresenta a tela de *login*.

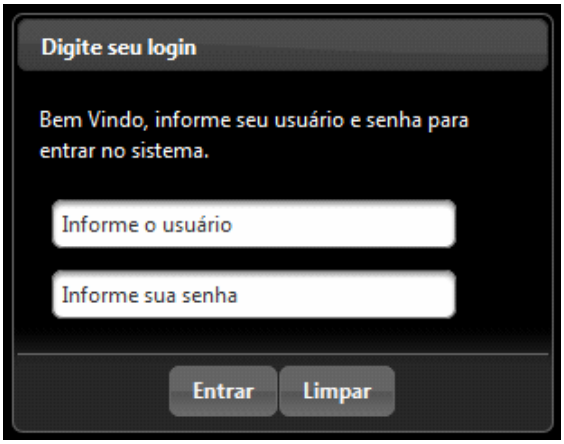

Figura 11 – Tela de login no sistema

Caso os dados informados para *login* estejam corretos, o sistema redirecionará o usuário para a tela inicial do sistema (Figura 12). Nessa tela há um menu de opções que apresentam as funcionalidades disponíveis do sistema.

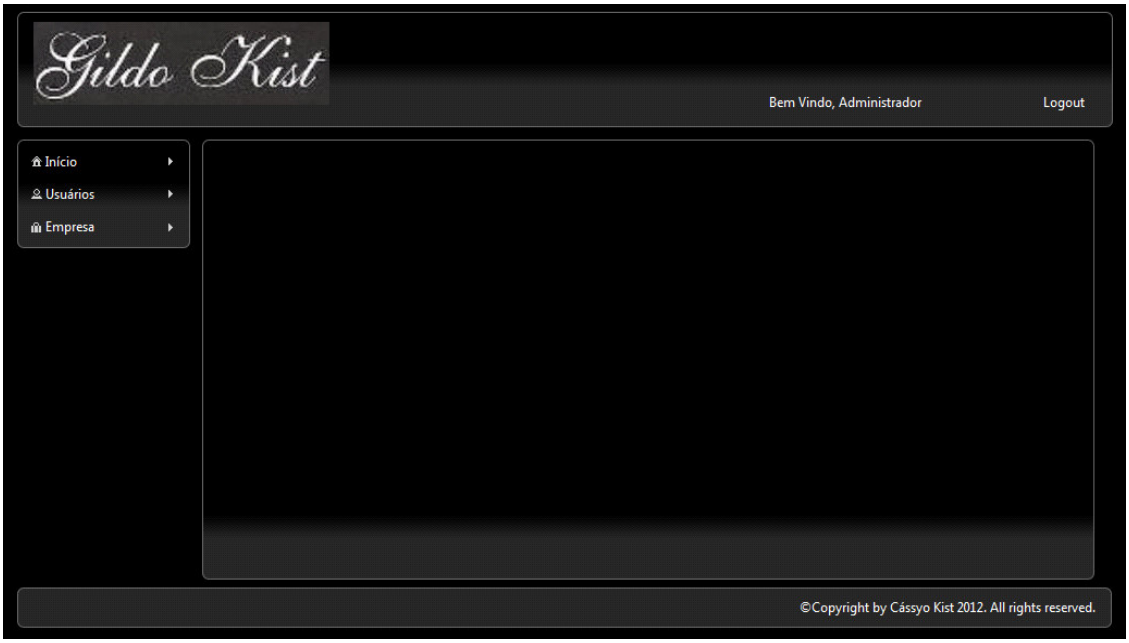

Figura 12 – Tela inicial do sistema

A Figura 13 apresenta a forma de cadastro de usuário. Para isso basta estar logado com um usuário administrador (Admin) e selecionar a opção “Usuários” - “Cadastrar” que o sistema mostrará a tela de cadastro de usuário.

Figura 13 – Cadastro de usuário

Nesta tela após serem informados os dados do usuário e clicar em “Salvar”, o sistema efetuará o processo de inclusão do usuário no banco de dados. Caso este processo tenha sido efetuado sem nenhum problema, o sistema redirecionará o conteúdo da página via Ajax para a listagem de usuários cadastrados e será emitida uma mensagem de que o usuário foi cadastrado com sucesso, como mostra a Figura 14.

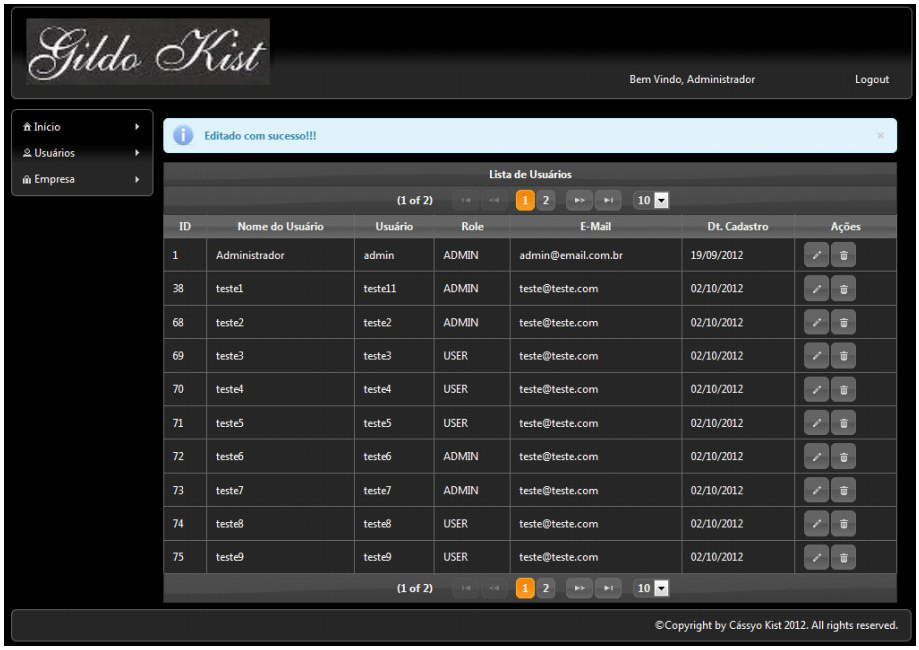

Figura 14 – Tela de cadastro de usuários

A Figura 15 apresenta a forma de cadastro de aluguel. Para isso basta estar logado com um usuário administrador (Admin) ou um usuário (User).

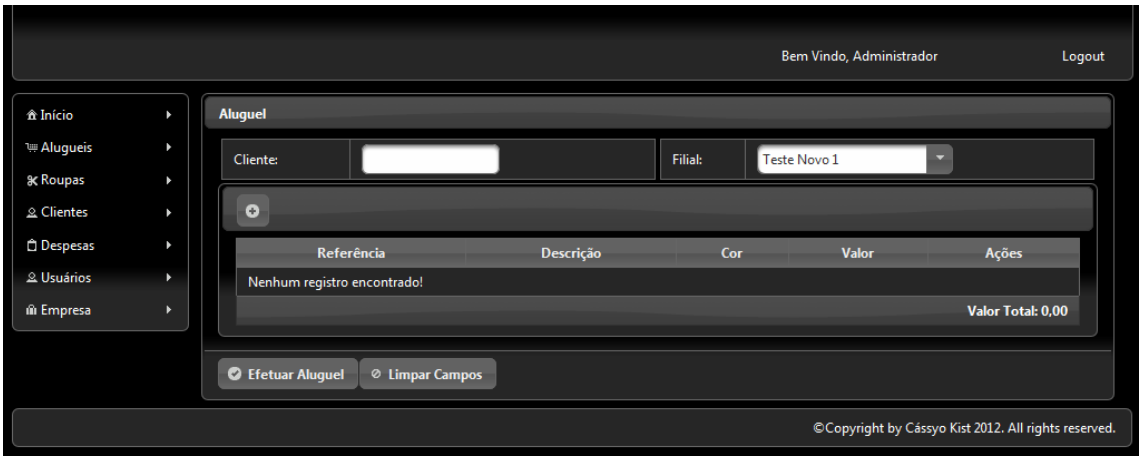

Figura 15 – Cadastro de Aluguel

Para efetuar um aluguel (Figura 16) basta selecionar o cliente, digitando as iniciais do nome. Se o cliente estiver cadastrado aparecerá no "*auto-complete*" do campo as opções para selecionar. Em seguida é necessário escolher a filial que o usuário deseja efetuar o aluguel. Selecionados cliente e filial, clicando no ícone com o símbolo "+" acima da tabela podem ser adicionadas as peças para aluguel. Depois de adicionadas todas as peças desejadas, basta clicar em "Efetuar Aluguel" que o sistema salvará o registro do aluguel.

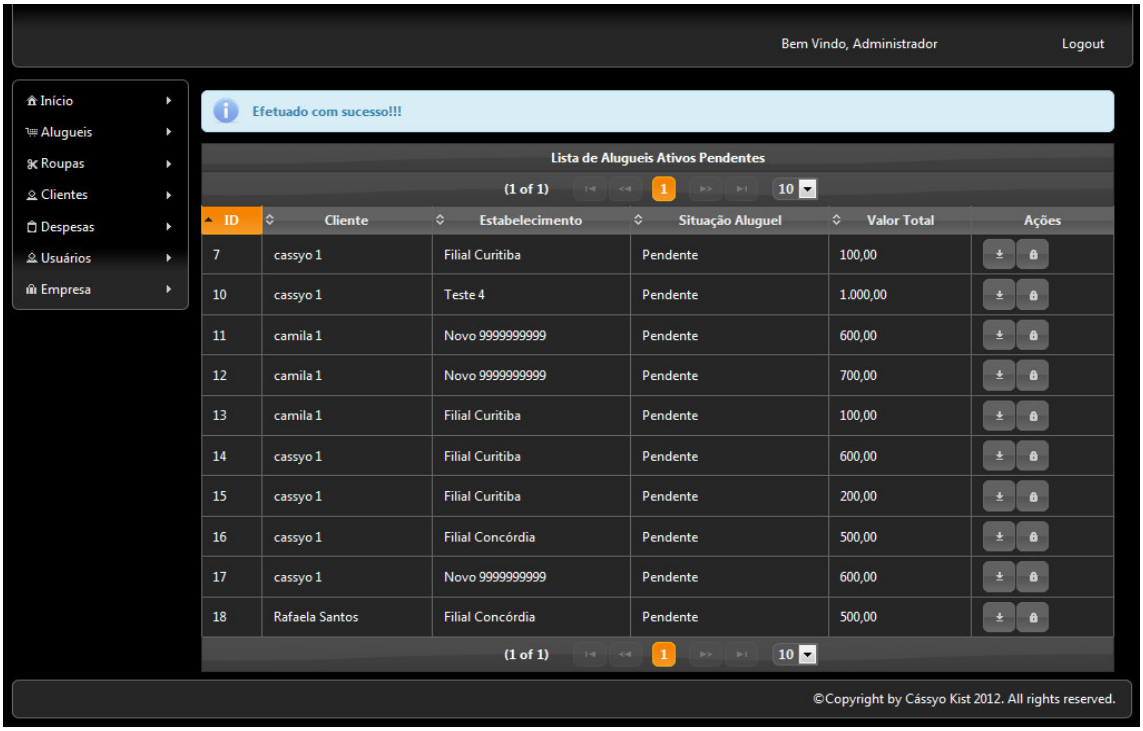

Figura 16 – Tela para efetuar aluguéis

4.4 IMPLEMENTAÇÃO DO SISTEMA

As listagens de código a seguir visam exemplificar a forma de implementação do sistema. O objetivo é mostrar o uso das tecnologias empregadas na implementação do sistema. A Listagem 1 apresenta o código para *login* ao sistema. Na tela de *login* são informados dados para acesso ao sistema.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<h:head>
      <title><ui:insert name="title">Login</ui:insert></title>
      <link type="text/css" rel="stylesheet"
            href="#{request.contextPath}/css/default.css" />
      <link type="text/css" rel="stylesheet"
            href="#{request.contextPath}/css/style.css" />
</h:head>
<h:body>
<h:form prependId="false">
<div id="loginPainel">
<p:panel header="Digite seu login" id="boxDlg">
<p>Bem Vindo, informe seu usuário e senha para entrar no sistema.</p>
<div><h:panelGrid columns="2" cellpadding="5">
<p:inputText id="username" value="#{loginMB.lf.usuario}" required="true"
size="40" requiredMessage="Usuário incorreto!" />
<p:watermark for="username" value="Informe o usuário" />
<p:password id="password" value="#{loginMB.lf.senha}" required="true"
size="40" requiredMessage="Senha incorreta!" />
<p:watermark for="password" value="Informe sua senha" />
</h:panelGrid>
<f:facet name="footer">
<p:commandButton action="#{loginMB.login}" value="Entrar"
onclick="PrimeFaces.cleanWatermarks();"
oncomplete="PrimeFaces.showWatermarks();"
update="boxDlg, messages" />
<p:commandButton type="reset" value="Limpar" />
</f:facet>
</div>
<p:panel>
<p:messages id="messages" display="icon" />
</div>
</h:form>
</h:body>
</html>
```

Listagem 1 – Código para login ao sistema

O código da Listagem 1 está em HTML, com sua estrutura básica *head* e *body*. Todos os códigos relacionados ao sistema são colocados na *tag body*. O formulário que está denominado com a *tag form* necessita de um *action* e um *method*, que por padrão é o *POST*, para funcionar. Os campos "Login" e "Senha" usarão uma caixa de texto com a estrutura HTML *<p:inputText>,* para que o usuário informe os respectivos dados. No *action* é informado o caminho que o sistema acessará a página. O *method* especifica a forma que os dados serão transferidos para o servidor, podendo ser *post* ou *get*. Neste projeto foi optado pelo *post* porque os dados não são mostrados na URL. Desta forma, eles não ficam visíveis para o usuário e é mantido o conceito de *URL clean*, que se baseia em manter a URL o mais legível possível.

Na *tag* *<p:messages>* há uma variável chamada "messages". Essa variável é interpretada pelo PrimeFaces, que carrega uma mensagem de retorno que é incluída na variável *FacesContext* do JSF na classe *Managed Bean*. Na tag *<p:inputText>* do *login* e da *senha* é informado um *value.* Esse *value* é informado para que o JSF vincule os dados com um *form* (Listagem 2), que nada mais é que uma classe com os *gets* e *sets* dos atributos que a página "conversa" com o *Managed Bean*, o real motivo do *form* existir e manter os dados encapsulados no *Managed Bean* que contem os métodos das conexões com o banco de dados para capturar os dados da tabela User.

```
package br.com.usandoprimefaces.form.user;
import java.io.Serializable;
import br.com.usandoprimefaces.bean.user.User;
public class UserForm extends User implements Serializable{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public UserForm() {
		super();
	}

}
```

Listagem 2 – Classe Form

A Listagem 3 apresenta o código responsável por apanhar os dados informados na tela do *login* e verificar no banco de dados se esses dados estão corretos, para assim ser possível o acesso ao sistema.

```
public String login() {

		User usuarioLogado = null;
```

```
	try {
	LoginDAO dao = new LoginDAOImpl();
	usuarioLogado = dao.efetuaLogin(lf);
	if (usuarioLogado != null) {
	HttpSession session = (HttpSession) FacesContext
	.getCurrentInstance().getExternalContext()
	.getSession(false);
	session.setAttribute("usuarioLogado", usuarioLogado);
	return "/protected/index.xhtml";
	} else
	FacesContext.getCurrentInstance().addMessage(null,
	new FacesMessage("Usuário não autenticado!"));
	}
	} catch (Exception e) {
		e.getMessage();
	}
	return null;
}
```

Listagem 3 – Login ao sistema

No código apresentado na Listagem 3, as informações de *login* são enviadas para o método que é carregado da classe *DAOImpl*. Esse método espera como parâmetro um objeto do tipo *user* e verifica se ele está correto. Caso o método carregado retorne *null*, significa que o *login* ou a senha informados são inválidos. E então o método redireciona o usuário para a página do *login* do sistema com uma mensagem de erro. Se as informações de *login* estão corretas, o usuário é adicionado em uma *session* com o nome de "usuarioLogado" e redirecionado para a página *index*, que é a tela inicial do sistema. Nessa listagem de código também são efetuados os tratamentos de erros que podem ser gerados. Caso um erro ocorra uma mensagem é adicionada na *FacesContext* do JSF e o usuário é redirecionado para a tela de *login* do sistema.

Na Listagem 4 está o código que cria a *query* de consulta para verificação dos dados e retorna o usuário caso encontrado, ou uma mensagem de erro.

```
package br.com.usandoprimefaces.daoimpl.login;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.criterion.Restrictions;

import br.com.usandoprimefaces.bean.user.User;
import br.com.usandoprimefaces.dao.login.LoginDAO;
import br.com.usandoprimefaces.form.login.LoginForm;
import br.com.usandoprimefaces.util.HibernateUtils;

public class LoginDAOImpl implements LoginDAO {

/* Método que verifica o usuario no banco */
	@Override
public User efetuaLogin(LoginForm form) {

	User usuario = null;
```

```
        SessionFactory session = HibernateUtils.getSessionFactory();
        Session sess = session.openSession();
        try {
              sess.beginTransaction();
              usuario = (User) sess.createCriteria(User.class)
        .add(Restrictions.eq("usuario", form.getUsuario()))
        .add(Restrictions.eq("senha", form.getSenha()))
        .uniqueResult();
        sess.getTransaction().commit();
        } catch (Exception ex) {
              System.out.println("Erro ao conectar " + ex.getMessage());
              sess.close();
        } finally {
              if (sess != null) {
                    sess.close();
              }
        }
              return usuario;
        }
}
```

Listagem 4 – Código para consulta de usuário ao banco de dados

No código apresentado na Listagem 4 está o tratamento do método que carrega a classe *DAOImpl*. Nesse método é verificado se o usuário informado está correto. Primeiro, utiliza-se uma variável *usuario* do tipo *User* e uma *SessionFactory* do Hibernate para pesquisar no banco de dados. Com a *criteria query* é criado o código SQL de consulta no banco para verificar se os dados do usuário estão corretos. Em seus parâmetros são passados os dados recuperados da página que foram informados pelo usuário e adicionado em forma de *restrictions*, e é esperado um *UniqueResult* que irá retornar um resultado único do banco de dados. Se atender essas condições, as exceções são tratadas e o *usuario* é retornado.

Na Listagem 5 é apresentado o código para capturar os dados informados pelo usuário Administrador, para assim ser efetuado o cadastro do usuário.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- template="template/principal.xhtml" -->
<ui:composition template="../../template/principal.xhtml"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">

<ui:define name="conteudo">
<h:form id="userForm" prependId="false">
<p:messages id="messages" autoUpdate="true" closable="true" />
<p:panel id="painelUser">
<f:facet name="header">Cadastro e Edição de Usuário  </f:facet>
<table width="100%">
<tr valign="top">
<td width="48%"><p:panelGrid styleClass="boxInterno" columns="2">
<h:outputLabel for="usuario" value="Usuario: " />
```

```
<p:inputText id="usuario" value="#{userMB.form.usuario}" required="true" />
<h:outputLabel for="senha" value="Senha: " />
<p:inputText id="senha" value="#{userMB.form.senha}"
required="true" type="password" />
<h:outputText value="Role: " />
<p:selectOneMenu id="role" required="true" value="#{userMB.form.role}">
<f:selectItem itemLabel="Selecione Um" itemValue="" />
<f:selectItem itemLabel="Administrador" itemValue="ADMIN" />
<f:selectItem itemLabel="Usuário" itemValue="USER" />
</p:selectOneMenu>
</p:panelGrid></td>
<td width="48%"><p:panelGrid styleClass="boxInterno" columns="2">
<h:outputLabel for="nome" value="Nome: " />
<p:inputText id="nome" value="#{userMB.form.nomeUser}" required="true" />
<h:outputLabel for="email" value="E-mail: " />
<p:inputText id="email" value="#{userMB.form.email}" required="true" />
<h:outputLabel for="dtCadastro" value="Dt. Cadastro: " />
<p:calendar disabled="true" required="true"
value="#{userMB.form.dtCadastro}" id="dtCadastro">
<f:convertDateTime pattern="dd/MM/yyyy" />
</p:calendar>
</p:panelGrid></td>
</tr>
</table>
<f:facet name="footer">
<p:commandButton action="#{userMB.salvar}" value="Salvar"
icon="ui-icon-circle-check" update="messages, painelUser" />
<p:commandButton value="Limpar Campos" type="reset"
icon="ui-icon-cancel" />
</f:facet>
</p:panel>

</h:form>
</ui:define>
</ui:composition>
```

Listagem 5 – Código para cadastro de usuário

O código constante na Listagem 5 segue o mesmo padrão da "Listagem 1 – Código para *login* ao sistema". Para cada campo do cadastro de usuários foi utilizado um *label* para informar o campo e um *input* para informar ao usuário e uma caixa de texto para informar os dados. Essa caixa contém um nome e um *value* para ser utilizado pelo *Manage Bean* e para que esses dados possam ser utilizados na página JSF.

Na Listagem 6 é informado o código em que o usuário é "passado" para a classe responsável por chamar os métodos que salvam o usuário no banco de dados.

```
public String salvar() {
      UserDAO dao = new UserDAOImpl();
      User user = new User();
      HttpSession session = (HttpSession) FacesContext.getCurrentInstance()
      .getExternalContext().getSession(false);
      User usuario = (User) session.getAttribute("usuarioLogado");
      try {
// Função usada para passar os dados do Object UserForm para o User
      if (usuario.getRole().name().equalsIgnoreCase(Role.ADMIN.name())) {
```

```
      BeanUtils.copyProperties(user, form);
      dao.save(user);
      FacesContext.getCurrentInstance().addMessage(  null, new
FacesMessage(
      "Salvo com sucesso!!!"));
      return "/protected/user/listaUser.xhtml";
      } else {
      FacesContext.getCurrentInstance().addMessage(  null, new
FacesMessage(
      "Você não tem permissão para executar esta ação!!!"));
      }
      } catch (IllegalAccessException e) {
FacesContext.getCurrentInstance().addMessage(  null,new FacesMessage("Não
foi possível salvar o objeto. " + e.getMessage()));
      } catch (InvocationTargetException e) {
      FacesContext.getCurrentInstance().addMessage(  null,new
FacesMessage("Não foi possível salvar o objeto. " + e.getMessage()));
}
      return null;
      }
```

Listagem 6 – Código para salvar o cadastro de usuário

No código apresentado na Listagem 6, são criadas uma instância de *User* para informar os dados recebidos da página JSF, uma instância do DAO para acessar os métodos existentes e uma HttpSession para obter o usuário logado. No código da Listagem 6 no *action* do botão "Salvar" na página JSF contém o *Managed bean* e o nome do método salvar que existe nele. Logo após o método ser invocado o sistema verifica se o usuário conectado é um administrador (Admin). Caso seja um administrador é executado um *save* que recebe como parâmetro o usuário que foi informado na página. Em seguida é adicionada a mensagem de sucesso no cadastro e então o sistema redireciona o usuário através do "*return*" para a página de listagem de usuários.

O código apresentado na Listagem 7 é responsável por chamar os métodos para salvar, excluir, listar, buscar e editar objetos. No pacote *DAO* (*Data Access Objects*) estão todas as interfaces do sistema. Na interface apresentada na Listagem 6 é invocado o método *save* para salvar o cadastro do usuário.

```
package br.com.usandoprimefaces.dao.user;
import java.util.List;
import br.com.usandoprimefaces.bean.user.User;
public interface UserDAO {
      public List<User> listar();
      public User save(User user);
      public void delete(User user);
      public User getById(int id);
      public void update(User user);
}
```

Listagem 7 – Código para chamar métodos de inclusão, exclusão e edição de objetos

Na Listagem 8 está o código do método para salvar o cadastro invocado.

```
@Override
public User save(User user) {

	SessionFactory session = HibernateUtils.getSessionFactory();
	Session sess = session.openSession();
	try {
		sess.beginTransaction();
		sess.merge(user);
		sess.getTransaction().commit();
		return user;
	} catch (HibernateException ex) {
		System.out.println("Erro ao salvar objeto: " +
ex.getMessage());
		sess.getTransaction().rollback();
		sess.close();
	} finally {
		if (sess != null) {
			sess.close();
		}
	}
	return null;
}
```

Listagem 8 – Código para salvar cadastro de usuário invocado

Na Listagem 8 está o método *save* invocado na Listagem 6. Esse método inicia uma transação com o banco de dados por meio de uma *SessionFactory* do *Hibernate*. Na execução dessa transação é atribuída a entidade *User*, que foi passada por parâmetro na chamada do método. Após isso é utilizado o *merge* para gravar no banco de dados. O *merge* verifica se o registro sendo salvo já está no banco. Se não estiver o mesmo é cadastrado e é realizado um *commit* da transação, retornando o registro que foi cadastrado.

Na Listagem 9 é apresentado o código para capturar os dados informados pelo usuário do sistema para ser efetuado o cadastro do aluguel.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition template="../../template/principal.xhtml"
	xmlns="http://www.w3.org/1999/xhtml"
	xmlns:ui="http://java.sun.com/jsf/facelets"
	xmlns:h="http://java.sun.com/jsf/html"
	xmlns:f="http://java.sun.com/jsf/core"
	xmlns:p="http://primefaces.org/ui">

	<ui:define name="conteudo">
	<h:form id="form">
	<p:messages id="messages" autoUpdate="true" closable="true" />
	<p:panel id="painelAluguel">
	<f:facet name="header">Aluguel</f:facet>
	<table width="100%">
	<tr valign="top">
	<td width="48%"><p:panelGrid styleClass="boxInterno" columns="2">
```

```
<h:outputLabel value="Cliente:" for="cliente" />
<p:autoComplete id="cliente" value="#{aluguelMB.selectedCliente}"
completeMethod="#{aluguelMB.autoCompleteCliente}" var="cliente"
itemLabel="#{cliente.nome}" itemValue="#{cliente}"
converter="clienteConverter" maxResults="10" />
</p:panelGrid></td>
<td width="48%"><p:panelGrid styleClass="boxInterno" columns="2">
<h:outputText value="Filial:" />
<p:selectOneMenu id="estabelecimento" required="true"
value="#{aluguelMB.form.estabelecimento.idestabelecimento}">
<f:selectItems value="#{aluguelMB.estabelecimentos}"
var="estab" itemLabel="#{estab.nomeEstab}"
itemValue="#{estab.idestabelecimento}" />
</p:selectOneMenu>
</p:panelGrid></td>
</tr>
</table>
<p:panel id="painelListaRoupas" widgetVar="painelRoupas"
header="Vestidos selecionados:">
<f:facet name="header">
<p:commandButton id="btnAdicionarRoupa" icon="ui-icon-circle-plus"
onclick="dlgAdicionarRoupa.show()">
<p:tooltip for="btnAdicionarRoupa">Adicionar Roupa</p:tooltip>
</p:commandButton>
</f:facet>

<p:dataTable id="dataTableRoupa" var="roupasDataTable"
value="#{aluguelMB.roupasSelecionadas}"
emptyMessage="Nenhum registro encontrado!">
<p:column>
<f:facet name="header">
<h:outputText value="Referência" />
</f:facet>
<h:outputText value="#{roupasDataTable.referencia}" />
</p:column>
<p:column>
<f:facet name="header">
<h:outputText value="Descrição" />
</f:facet>
<h:outputText value="#{roupasDataTable.descricao}" />
</p:column>
<p:column>
<f:facet name="header">
<h:outputText value="Cor" />
</f:facet>
<h:outputText value="#{roupasDataTable.cor}" />
</p:column>
<p:column>
<f:facet name="header">
<h:outputText value="Valor" />
</f:facet>
<h:outputText value="#{roupasDataTable.valor}">
<f:convertNumber minFractionDigits="2" />
</h:outputText>
</p:column>
<p:column>
<f:facet name="header">
<h:outputText value="Ações"></h:outputText>
</f:facet>
```

```
      <p:commandButton id="btnExcluirRoupaPanel" icon="ui-icon-trash"
      onclick="confirmacaoExcluir.show()">
      <p:tooltip for="btnExcluirRoupaPanel">Excluir roupa do aluguel</p:tooltip>
      <f:setPropertyActionListener value="#{roupasDataTable}"
      target="#{aluguelMB.selectedRoupa}" />
      </p:commandButton>
      </p:column>
      <f:facet name="footer">
      <div align="right">
      <h:outputLabel value="Valor Total:" for="valorTotal" />
      <h:outputLabel id="valorTotal" value="#{aluguelMB.valorTotal}">
      <f:convertNumber minFractionDigits="2" />
      </h:outputLabel>
      </div>
      </f:facet>
      </p:dataTable>
      </p:panel>
      <f:facet name="footer">
      <p:commandButton action="#{aluguelMB.efetuarAluguel}"
      value="Efetuar Aluguel" icon="ui-icon-circle-check"
      update="messages, painelListaRoupas" />
      <p:commandButton value="Limpar Campos" type="reset"
      icon="ui-icon-cancel" />
      </f:facet>
      </p:panel>
      </h:form>

<!-- INICIO DO DIALOG USADO PARA ADICIONAR ROUPAS AO DATATABLE -->
      <h:form id="formModal">
      <p:dialog header="Selecione a roupa que deseja adicionar"
      widgetVar="dlgAdicionarRoupa" resizable="false"
      id="dialogAdicionarRoupa" modal="true" showEffect="fade"
      hideEffect="explode">

      <p:dataTable id="roupas" paginator="true" rows="15"
       paginatorTemplate="{CurrentPageReport}  {FirstPageLink} {PreviousPageLink}
       {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
       rowsPerPageTemplate="5,10,15" emptyMessage="Nenhum registro encontrado!"
       var="roupa" value="#{aluguelMB.roupas}"
       selection="#{aluguelMB.selectedRoupa}"
      sortBy="#{roupa.referencia}" selectionMode="single"
      rowKey="#{roupa.idroupa}" ajax="true">

      <p:column>
      <f:facet name="header">
      <h:outputText value="Referência" />
      </f:facet>
      <h:outputText value="#{roupa.referencia}" />
      </p:column>
      <p:column>
      <f:facet name="header">
      <h:outputText value="Descrição" />
      </f:facet>
      <h:outputText value="#{roupa.descricao}" />
      </p:column>

      <p:column>
      <f:facet name="header">
      <h:outputText value="Cor" />
```

```
        </f:facet>
        <h:outputText value="#{roupa.cor}" />
        </p:column>
        <p:column>
        <f:facet name="header">
        <h:outputText value="Valor" />
        </f:facet>
        <h:outputText value="#{roupa.valor}">
        <f:convertNumber minFractionDigits="2" />
        </h:outputText>
        </p:column>
        </p:dataTable>
        <f:facet name="footer">
        <p:commandButton id="btnDlgConfirmaAdicao" value="Adicionar Roupa"
        icon="ui-icon-circle-check" oncomplete="dlgAdicionarRoupa.hide()"
        actionListener="#{aluguelMB.adicionarRoupaPanel}"
        update=":form:dataTableRoupa" />
        <p:commandButton id="btnDlgCancelaAdicao" icon="ui-icon-cancel"
        onclick="dlgAdicionarRoupa.hide()" type="button" value="Cancelar" />
        </f:facet>
        </p:dialog>
        </h:form>
        <!-- MENSAGEM PARA CONFIRMAÇÃO DE EXCLUSÃO DA ROUPA DA TABLE -->
        <p:confirmDialog id="confirmacaoDialogExcluir"
        message="Tem certeza que deseja EXCLUIR esta roupa do aluguel?"
        header="Confirmação" severity="alert" widgetVar="confirmacaoExcluir">
        <p:commandButton id="btnConfirmExcluir" value="Sim"
        oncomplete="confirmacaoExcluir.hide()" icon="ui-icon-circle-check"
        actionListener="#{aluguelMB.excluirRoupaPanel}"
        update=":form:dataTableRoupa" />
        <p:commandButton id="btnDeclineExcluir" value="Cancelar"
        onclick="confirmacaoExcluir.hide()" type="button"
        icon="ui-icon-cancel" />
        </p:confirmDialog>
        </ui:define>
</ui:composition>
```

Listagem 9 – Código para efetuar aluguel

No código da Listagem 9, o *action* do botão “Efetuar Aluguel” na página JSF contém o *Managed bean* e o nome do método que existe nele e é responsável para efetuar o aluguel. Logo após o método ser invocado, o sistema verifica se o usuário conectado é um administrador (Admin) ou um usuário (User), caso seja é executada uma função da BeanUtils.copyProperties que apanha os dados do *form* e os passa para o objeto aluguel. Em seguida são atribuídos valores aos campos e é chamado o método *efetuarALuguel da classe DAO* que recebe como parâmetro o aluguel que foi criado e populado. Em seguida é adicionada a mensagem de sucesso no cadastro e então o sistema redireciona o usuário através do “*return*” para a página de listagem de aluguéis ativos e pendentes que é o padrão dos novos aluguéis.

Na Listagem 10 é informado o código em que o aluguel é “passado” para a classe responsável por chamar os métodos que efetuam o aluguel no banco de dados.

```
public String efetuarAluguel() {

AluguelDAO dao = new AluguelDAOImpl();
EstabelecimentoDAO daoEstab = new EstabelecimentoDAOImpl();

Aluguel aluguel = new Aluguel();

HttpSession session = (HttpSession) FacesContext.getCurrentInstance()
.getExternalContext().getSession(false);
User usuario = (User) session.getAttribute("usuarioLogado");

try {

/*
* Verifica se o usuário e ADMIN ou USER antes de continuar o método
*/
if (usuario.getRole().name().equalsIgnoreCase(Role.ADMIN.name())
|| usuario.getRole().name()
.equalsIgnoreCase(Role.USER.name())) {
// Função usada para passar os dados do Object AluguelForm para
// o
// Aluguel
BeanUtils.copyProperties(aluguel, form);

// Setando os valores para o aluguel ser efetuado
aluguel.setValorTotal(getValorTotal());
aluguel.getRoupas().addAll(getRoupasSelecionadas());
aluguel.setCliente(selectedCliente);
aluguel.setEstabelecimento(daoEstab.getById(form
.getEstabelecimento().getIdestabelecimento()));
aluguel.setPendente(true);
aluguel.setDesativado(false);

// Efetuando o aluguel por padrão sera setado como pendente
dao.efetuarAluguel(aluguel);

FacesContext.getCurrentInstance().addMessage(null,
new FacesMessage("Efetuado com sucesso!!!"));
return "/protected/aluguel/listaAluguelAtivoPendente.xhtml";
} else {
FacesContext
.getCurrentInstance()
.addMessage(
null,
new FacesMessage(
"Você não tem permissão para executar esta ação!!!"));
}

} catch (IllegalAccessException e) {
FacesContext.getCurrentInstance().addMessage(
null,
new FacesMessage("Não foi possível efetuar o aluguel: "
+ e.getMessage()));
} catch (InvocationTargetException e) {
FacesContext.getCurrentInstance().addMessage(
null,
```

```
new FacesMessage("Não foi possível efetuar o aluguel: "
+ e.getMessage()));
}

return null;
}
```

Listagem 10 – Código do Managed Bean para efetuar o aluguel e outras funcionalidades

No código apresentado na Listagem 10 é criada uma instância de Aluguel para armazenar os dados recebidos da página JSF, uma instância do DAO para acessar os métodos existentes e uma HttpSession para obter o usuário logado.

O código apresentado na Listagem 11 é responsável por chamar os métodos para as listagens, "*auto-complete*" do cliente, efetuação de aluguéis, desativação de aluguéis e ativação de aluguéis. No pacote *DAO* estão todas as interfaces do sistema. Na interface apresentada na Listagem 11 é invocado o método *efetuarALuguel* para salvar o aluguel no banco de dados.

```
public br.com.usandoprimefaces.dao.aluguel;

import java.util.List;

import br.com.usandoprimefaces.bean.aluguel.Aluguel;
import br.com.usandoprimefaces.bean.cliente.Cliente;

public interface AluguelDAO {
        public List<Aluguel> listarAtivoPendente();
        public List<Aluguel> listarAtivoBaixado();
        public List<Aluguel> listarDesativados();
        public List<Cliente> autoCompleteCliente(String strPesquisa);
        public Aluguel efetuarAluguel(Aluguel aluguel);
        public void desativarAluguel(Aluguel aluguel);
        public void ativarAluguel(Aluguel aluguel);
        public void baixarAluguel(Aluguel aluguel);
}
```

Listagem 11 – Código para chamar métodos de listagem, desativação, ativação e efetuação do aluguel e autocomplete do cliente

Na Listagem 12 é representado o método *efetuarALuguel* invocado na Listagem 11. Esse método inicia uma transação com o banco de dados por meio de uma *SessionFactory* do *Hibernate*. Na execução dessa transação é atribuída a entidade *Aluguel*, que foi passada por parâmetro na chamada do método. Após isso é utilizado o *merge* para gravar no banco de dados. O *merge* verifica se no local em que será salvo o aluguel no banco não há registro. Se não houver cadastra e faz um *commit* da transação, retornando o aluguel que foi cadastrado.

```
@Override
public Aluguel efetuarAluguel(Aluguel aluguel) {
	SessionFactory session = HibernateUtils.getSessionFactory();
	Session sess = session.openSession();
	try {
	sess.beginTransaction();
	sess.merge(aluguel);
	sess.getTransaction().commit();
	return aluguel;
	} catch (HibernateException ex) {
	System.out.println("Erro ao efetuar aluguel: " + ex.getMessage());
	sess.getTransaction().rollback();
	sess.close();
	} finally {
	if (sess != null) {
	sess.close();
	}
	return null;
	}
```

Listagem 12 – Código para efetuar o aluguel no banco de dados

5 CONCLUSÃO

O objetivo deste trabalho de conclusão de curso foi a implementação de um sistema para aluguel e venda de roupa. A aplicação desenvolvida é voltada para *web*, utilizando recursos que caracterizam uma aplicação Internet como rica, as denominadas RIAs. Dessa forma, o referencial teórico do trabalho esteve centrado em conceitos relacionados a esse tipo de aplicação e tecnologias para o seu desenvolvimento.

Como estágio, o autor deste trabalho fez uma primeira versão da modelagem e implementou alguns cadastros. Com o decorrer do desenvolvimento, especialmente para funcionalidades mais complexas, ou seja, não cadastros simples, muitas dificuldades foram encontradas na implementação. Assim, decidiu-se buscar alternativas tecnológicas para a implementação do sistema. Optou-se pelo uso de JSF com PrimeFaces pelas muitas referências favoráveis ao uso dessas tecnologias encontradas na Internet e pelas opiniões de programadores conhecidos. Desta forma, a codificação do sistema teve que ser reiniciada.

Com as mudanças realizadas em termos de tecnologias, para o desenvolvimento do sistema foram utilizadas as tecnologias Java para *web*, com *frameworks* que facilitam o desenvolvimento como o Hibernate para persistência de dados no banco, Java Server Faces 2.0 que procura encapsular toda a API e PrimeFaces que disponibiliza uma grande facilidade para implementação da interface utilizando Java Script, Ajax e Jquery, assim como suas validações.

Optou-se pelo desenvolvimento para *web* pelo fato da facilidade de interligar as filiais e a matriz e por ter um único banco de dados para toda a rede de lojas. Isso porque a empresa que utilizará esse sistema é composta por matriz e filiais. Assim, os gerentes podem fazer consultas e análises da sua loja sem precisar entrar em contato direto com outras pessoas, agilizando o processo.

Com o decorrer do desenvolvimento deste projeto foi adquirido muito conhecimento em Java para *web* e nas outras tecnologias utilizadas. A necessidade de pesquisar sobre as tecnologias que foram utilizadas permitiu analisar os códigos elaborados de outra maneira. Verificou-se nessa busca pelo aprendizado que uma boa maneira de melhorar a elaboração do próprio código é utilizar APIs, sem esquecer que é importante o conhecimento dos conceitos fundamentais. Dentre esses conceitos destacam-se os relacionados à orientação a objetos e das próprias linguagens, incluindo o Jquery e JavaScript.

Com o desenvolvimento deste trabalho percebeu percebeu-se que muitas vezes é necessário projetar o desenvolvimento do restante do sistema para que as interferências entre as funcionalidades sejam visualizadas e analisadas.

Como trabalhos futuros, destacam-se especialmente a implementação de todas as funcionalidades definidas para o sistema, inclusive as que surgirão com a realização dos testes. Os testes de usuários serão realizados pela empresa que utilizará o sistema. Os testes para verificar consistência e validações do sistema serão realizados pelo próprio autor deste trabalho.

REFERÊNCIAS

ASTAH. **Astah community**. Disponível em: <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em: 08 ago. 2012.

BIANCHINI, Sandro. **Avaliação de métodos de desenvolvimento de aplicações web**. Dissertação ICMC/USP, 2008.

CHUNG, Sam; LEE, Yun-Sik. **Modeling web applications using Java and XML related technologies.** Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03), 2003, p. 322-331.

DEZIGN. **DeZign for databases**. Disponível em: <http://www.datanamic.com/dezign/index.html>. Acesso em: 25 ago. 2012.

ECLIPSE. **Eclipse IDE**. Disponível em: <http://www.eclipse.org>. Acesso em: 05 set. 2012.

GAMMA, Erich; JOHNSON, Ralph; HELM, Richard; VLISSIDES, John. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GLASSFISH. **JavaServer Faces specification**. Disponível em: <http://javaserverfaces-spec-public.java.net/>. Acesso em: 10 ago. 2012.

HIBERNATE, **Hibernate - JBoss community**. Disponível em: <http://www.hibernate.org>. Acesso em: 02 ago. 2012.

JAVA. **Java**. Disponível em: http://www.oracle.com/us/technologies/java/index.html. Acesso em: 12 ago. 2012.

JQUERY, **Jquery**. Disponível em: <http://jquery.com>. Acesso em: 12 ago. 2012.

JSF. **JavaServer Faces**. Disponível em: <http://www.javaserverfaces.org/>. Acesso em: 20 set. 2012.

KHIN, Su Su. **Business processes solution with apache struts framework.** 2009 International Conference on Computer Engineering and Technology (ICCET). IEEE Computer Society, 2009, p. 280-282.

MIRANDA, Divino Gomes; SILVA, Marcos Alberto Lopes da. **Novos recursos para desenvolvimento na plataforma JavaEE 5 com a especificação EJB 3.0**, 2006. Disponível em: http://www.linhadecodigo.com.br/Artigo.aspx?id=1085. Acesso em: 09 jul. 2012.

MYSQL. **MySQL**. Disponível em: <http://www.mysql.com>. Acesso em: 10 de set. 2012.

PRAJAPATI, Harshad B.; DABHI, Vipul K. **High quality web-application development on JavaEE platform**. IEEE International Advance Computing Conference (IACC 2009), 2009, p. 1664-1669.

PRESSMAN, Roger. **Engenharia de software**. Rio de Janeiro: MacGraw-Hill, 2005.

PRIMEFACES. **PrimeFaces quickstart tutorial-part1**. Disponível em: http://java.dzone.com/articles/primefaces-quickstart-tutorial. Acesso em: 18 ago. 2012.

RUMBAUGH, James; BLAHA, Michael; PREMERLANI, William; EDDY, Frederick; LORENSEN, William. **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Campus, 1997.

SILVA, Maurício Samy. JQuery. **A biblioteca do programador JavaScript**. Novatec, 2010.

SOUZA, Vítor. **Introdução ao desenvolvimento web em Java**, 2008. Disponível em: http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt. Acesso em: 7 set. 2011.

TOMCAT. A**pache Tomcat**. Disponível em: <http://tomcat.apache.org/>. Acesso em: 12 set. 2011.

WANG, Guanhua. **Application of lightweight MVC-like structure in PHP.** 2011 International Conference on Business Management and Electronic Information (BMEI 2011), IEEE, 2011, p. 74-78.