MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

DESENVOLVIMENTO DE *WEBSITES* RESPONSIVOS PARA REDE DE HÓTEIS

Bruno Eustáquio Enes Cardoso

Orientador: Prof. Dr. Bruno Ferreira

FORMIGA- MG
2018

BRUNO EUSTÁQUIO ENES CARDOSO

DESENVOLVIMENTO DE *WEBSITES* RESPONSIVOS PARA REDE DE HÓTEIS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - *Campus* Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Bruno Ferreira

FORMIGA- MG

2018

Cardoso, Bruno Eustáquio Enes.
004 Desenvolvimento de Websites Responsivos para Rede de Hoteis / Bruno Eustáquio Enes Cardoso . -- Formiga : IFMG, 2018.
115p. : il.

Orientador: Prof. Dr. Bruno Ferreira
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Responsivo. 2.Web. 3. Java. 4. JSF. 5. PrimeFaces.
I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Me..Naliana Dias Leandro CRB6-1347

BRUNO EUSTÁQUIO ENES CARDOSO

DESENVOLVIMENTO DE *WEBSITES* RESPONSIVOS PARA REDE DE HOTÉIS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais-Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 05 de junho de 2018.

BANCA EXAMINADORA

Prof.° Bruno Ferreira

Prof.° Fernando Paim Lima

Prof.° Manoel Pereira Junior

Agradecimentos

Agradeço primeiramente a Deus, por sempre estar ao meu lado. Senhor, fostes Tu que me ensinastes que nada é impossível, que perante qualquer dificuldade quem acredita no teu amor encontrará o caminho da superação!

Agradeço aos meus familiares, que fizeram de tudo para que eu alcançasse os meus sonhos. Aos meus pais, Messias e Aparecida, irmãos, Natália e Thúlio, e claro, meus sobrinhos, Bernardo e Eduardo, o meu muito obrigado.

Agradeço principalmente a minha querida mãe, pois foi ela que me ensinou a ser o que sou hoje, com seu amor, carisma, carinho e compreensão me criou e me educou, com muitas lutas, derrotas e vitórias, se dedicou para que eu continuasse correndo atrás dos meus sonhos e, mesmo depois de anos se passando, nunca deixou de estar ao meu lado, e sei que nunca deixará.

Agradeço ao meu tio Robson, por ter me ajudado nessa longa caminhada. Sempre esteve ao meu lado, me mostrando não só o melhor caminho a seguir, mas o que me fazia sentir melhor, sem ele, talvez não seria possível realizar este sonho.

Agradeço a minha namorada Laís, por todos esses anos junto comigo, obrigado pela força que me dá para seguir em frente e superar os desafios.

Agradeço a todos os professores e a instituição, pelo aprendizado que levarei e tentarei repassar aos outros, em especial, o professor e amigo Bruno Ferreira, que esteve comigo neste projeto e em tantos outros ao longo da graduação, sempre me ajudando em minhas dificuldades.

É hora de olhar para trás, sentir um pouco de orgulho por todo caminho percorrido, e de agradecer a Deus por ter estado ao meu lado em cada instante desse percurso. Minha gratidão será infinita para com Ele e para com todas as pessoas que sempre me deram forças.

É hora de olhar para frente, ir atrás de novos sonhos, novas metas, novas realizações, novos aprendizados, o qual nunca termina, pois, o conhecimento é a única coisa que levaremos sempre conosco.

E por fim, não gostaria de apenas dizer um obrigado, é muito simples, a gratidão que sinto não poderia ser traduzida em apenas uma palavra. Sou grato a todos aqueles que passaram por minha vida. Familiares, professores, colegas, sem vocês nada disto seria possível. Mas, como faltam palavras para expressar este sentimento, encerro com um obrigado, ou não, melhor, Muito Obrigado, por tudo!

“ Feliz aquele que transfere o que sabe e aprende o que ensina. ”

Cora Corina.

Resumo

Este trabalho apresenta o desenvolvimento de um sistema web para auxiliar o setor hoteleiro, ele propõe funcionalidades tanto para o cliente quanto para a administração do estabelecimento. O cliente acessará os recursos através de um site responsivo, ou seja, estará utilizando as tecnologias mais atuais para que o acesso ao mesmo seja feito de quaisquer dispositivos com tamanhos diferentes de telas. Para o administrativo, o diferencial é o gerenciamento de fluxo de caixa e principalmente o controle de reservas, tudo isso de qualquer lugar através de um dispositivo com acesso à internet. O sistema foi desenvolvido utilizando a linguagem Java e seus módulos para a Web utilizando ainda componentes propícios para tal finalidade como, JSF, *Primefaces* entre outros. O sistema atual já está preparado também para disponibilizar dados para aplicativos moveis através de serviços da Web (*Web Services*), sendo facilmente ampliado para atender novas funcionalidades e dispositivos.

Palavras-chave: Responsivo. sistema. web. internet. JSF. *PrimeFaces*. Java.

Abstract

This work presents the development of a web system to assist the hotel sector, it proposes functionalities for both the client and the administration of the establishment. The client will access the resources through a responsive site, that is, will be using the most current technologies so that access to the same is made of any devices with different screen sizes. For the administrative, the differential is the management of cash flow and especially the control of reservations, all this from anywhere through a device with access to the internet. The system was developed using the Java language and its modules for the Web using also components suitable for such purpose as, JSF, Primefaces among others. The current system is already prepared to make data available for mobile applications through Web services, and is easily extended to meet new features and devices.

Keywords: Responsive. system. web. internet. JSF. PrimeFaces. Java.

Lista de ilustrações

Figura 1 – Arquitetura Lógica Java EE. 26
Figura 2 – Mapeamento das três partes de uma aplicação para o MVC. 28
Figura 3 – Objetos utilizados no MVC e suas interações. 28
Figura 4 – Arquitetura JSF. 30
Figura 5 – Ciclo JSF. 31
Figura 6 – Arquitetura $Tomcat$. 33
Figura 7 – Arquitetura $Hibernate$. 34
Figura 8 – Módulos do $SpringFramework$. 38
Figura 9 – Tamanhos possíveis de telas. 42
Figura 10 – Como devem ser sites responsivos. 43
Figura 11 – Estrutura $Bootstrap$. 44
Figura 12 – Sistema de $grids$ responsivas ($smarttv, desktop, tablet, smartphone$). . . 45
Figura 13 – $MicrosoftVisio$. 47
Figura 14 – $IntelliJ$ IDE. 48
Figura 15 – $MySQLWorkbench$. 49
Figura 16 – Produtos de Trabalho do Processo Unificado em Cada Fase. 50
Figura 17 – Desenvolvimento iterativo e incremental. 51
Figura 18 – Diagrama de Caso de Uso para Cliente. 54
Figura 19 – Diagrama de Caso de Uso para usuário - parte 1. 56
Figura 20 – Diagrama de Caso de Uso para usuário - parte 2. 57
Figura 21 – Lançar reserva. 58
Figura 22 – Dados da empresa. 59
Figura 23 – Cadastro de usuários. 59
Figura 24 – Cadastrar aposento. 60
Figura 25 – Cadastro de pacote. 61
Figura 26 – Cadastro de categoria. 61
Figura 27 – Fluxo de caixa. 62
Figura 28 – Cadastro de nova reserva, dados 1. 63
Figura 29 – Cadastro de nova reserva, dados 3. 63
Figura 30 – Cadastro de nova reserva, dados 3. 63
Figura 31 – Mapa de reservas. 64
Figura 32 – Notificações. 65
Figura 33 – Notificações. 65
Figura 34 – Estrutura das classes Java. 67
Figura 35 – Estrutura das páginas $xhtml$. 68

Figura 36 – Diagrama da tabela categoria. 71
Figura 37 – Novo lançamento. 80
Figura 38 – Pesquisa por data. 84
Figura 39 – Erro com quantidade de diárias. 84
Figura 40 – Erro de datas entre pacotes. 85
Figura 41 – Erro de datas entre pacotes. 86
Figura 42 – Aposentos livres. 89
Figura 43 – Dados do cliente. 92
Figura 44 – Resumo reserva. 92
Figura 45 – Informações. 92
Figura 46 – Informações. 93
Figura 47 – $Poupup$ Reserva. 96
Figura 48 – Adicionando diárias. 98
Figura 49 – Adicionando consumo. 101
Figura 50 – Resumo dos valores. 102
Figura 51 – Confirmação de reserva finalizada. 105
Figura 52 – $Grids$ do $Bootstrap$. 107
Figura 53 – Acesso de um computador. 107
Figura 54 – Acesso de um celular, $design$ responsivo. 108

Lista de quadros

Quadro 1 – Conexão com o banco de dados. 69
Quadro 2 – Classe Java para conexão com o banco de dados. 69
Quadro 3 – Exemplos anotações $Hibernate$ 70
Quadro 4 – Modelo categoria. 72
Quadro 5 – Salvar categoria. 73
Quadro 6 – Método para replicar fator. 74
Quadro 7 – Declarações $CategoriaController$. 74
Quadro 8 – Iniciando o $controller$. 75
Quadro 9 – Montando a hierarquia $TreeNode$. 75
Quadro 10 – Categorias Selecionadas. 76
Quadro 11 – Categorias para caixa de seleção. 76
Quadro 12 – Categorias para caixa de seleção. 76
Quadro 13 – Componente $tree$. 77
Quadro 14 – Modelo lançamento. 78
Quadro 15 – $LancamentoController$. 78
Quadro 16 – Carrega lista de lançamentos. 79
Quadro 17 – Salvando um lançamento. 80
Quadro 18 – Modelo reserva - parte 1. 82
Quadro 19 – Modelo reserva - parte 1. 83
Quadro 20 – Verificação de datas. 85
Quadro 21 – Verificação de aposentos livres. 86
Quadro 22 – Percorredo Aposentos. 87
Quadro 23 – Calculando diárias com pacote. 88
Quadro 24 – Calculando diárias comum. 88
Quadro 25 – Verificando aposentos selecionados. 89
Quadro 26 – Verificando limite de hóspedes. 90
Quadro 27 – Recalculando valores da hospedagem. 90
Quadro 28 – Recalculando valores da hospedagem. 91
Quadro 29 – Salvando a reserva. 93
Quadro 30 – Carregando reservas para o calendário. 94
Quadro 31 – Carregando reservas para o calendário. 95
Quadro 32 – Cancelar Reserva. 97
Quadro 33 – Confirma reserva parte 1. 97
Quadro 34 – Confirma reserva parte 2. 98
Quadro 35 – Recalculando valor da reserva. 99

Quadro 36 – Calculando reservas por dia. 100
Quadro 37 – Calculando o consumo. 101
Quadro 38 – Finalizando a reserva - parte 1. 102
Quadro 39 – Finalizando a reserva - parte 2. 103
Quadro 40 – Finalizando a reserva - parte 3. 104

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CRUD	*Create, Read, Update and Delete*
CSS	*Cascading Style Sheets*
EJB	*Enterprise Java Beans*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HQL	*Hibernate Query Language*
IDE	Ambiente de Desenvolvimento Integrado
Java EE	Java *Enterprise Edition*
JDK	*Java Development Kit*
JPA	*Java Persistence API*
JS	*JavaScript.*
JSF	*JavaServer Faces*

JSP	*Java Server Pages*
MVC	*Model-view-controller*
ORM	*Object-relational Mapping*
PU	Processo Unificado
RAD	*Rapid Application Development*
SQL	*Structured Query Language*
UI	*User Inteface*
UML	*Unified Modeling Language*
XHTML	*eXtensible Hypertext Markup Language*
XML	*Extensible Markup Language*

Sumário

1 INTRODUÇÃO 21
1.1 Justificativa 22
1.2 Objetivos 22
1.2.1 Objetivo Geral 22
1.2.2 Objetivos Específicos 23
1.3 Estrutura do Trabalho 23

2 FUNDAMENTAÇÃO TEÓRICA 25
2.1 Java EE 25
2.2 Padrão MVC 27
2.3 JSF 29
2.4 *Apache Tomcat* 32
2.5 *Hibernate* 33
2.5.1 Uma pra um 35
2.5.2 Muitos pra um 35
2.5.3 Uma pra muitos 36
2.5.4 Muitos pra muitos 36
2.6 *Criteria* 36
2.7 *Spring Security* 37
2.8 *PrimeFaces* 39
2.9 *Websites* Responsivos 40
2.10 *Bootstrap* 43

3 MATERIAIS E MÉTODOS 47
3.1 *Microsoft Visio – UML* 47
3.2 IDE *IntelliJ* 48
3.3 Banco de Dados *MySQL* 49
3.4 Metodologia 49

4 MODELAGEM DO SISTEMA 53
4.1 Apresentação do Sistema 53
4.2 Modelagem 53
4.2.1 Diagrama de Caso de Uso 53
4.2.1.1 Cliente 54
4.2.1.2 Usuário 55
4.2.2 Caso de Uso 58

4.2.2.1 Lançar Reserva (Cliente) . . . 58
4.2.2.2 Cadastrar Empresa . . . 58
4.2.2.3 *CRUD* Usuários . . . 59
4.2.2.4 *CRUD* Aposentos . . . 60
4.2.2.5 *CRUD* Pacotes . . . 60
4.2.2.6 *CRUD* Categorias . . . 60
4.2.2.7 *CRUD* Lançamentos . . . 61
4.2.2.8 *CRUD* Reservas . . . 62
4.2.2.9 Verificar Reservas . . . 64
4.2.2.10 Verificar Notificações . . . 65
4.2.2.11 Gráficos . . . 65

5 DESENVOLVIMENTO . . . 67
5.1 Estrutura das Classes . . . 67
5.2 Criação do Banco de Dados . . . 68
5.3 Fluxo de Caixa . . . 70
5.3.1 Categorias . . . 70
5.3.1.1 Modelo . . . 71
5.3.1.2 Regras de Negócios . . . 72
5.3.1.3 Visão . . . 73
5.3.2 Lançamentos . . . 77
5.3.2.1 Modelo . . . 77
5.3.2.2 Regras de Negócios e Visão . . . 77
5.4 Reservas . . . 81
5.4.1 Modelo . . . 82
5.4.2 Realizando Reservas . . . 83
5.4.3 Mapa Reservas . . . 94
5.4.3.1 Editar Reserva . . . 98
5.4.3.2 Finalizar Reserva . . . 100
5.5 Envio de E-mails . . . 106
5.6 *Design* Responsivo . . . 106

6 TRABALHOS FUTUROS . . . 109

7 CONSIDERAÇÕES FINAIS . . . 111

REFERÊNCIAS . . . 113

1 INTRODUÇÃO

Atualmente, o mercado tem se tornado cada vez mais competitivo e com o surgimento de tecnologias cada vez mais acessíveis, faz-se necessário que as empresas busquem por inovações para alavancar uma maior visibilidade do seu produto e alavamquem os números de vendas e seus lucros. Neste cenário empresas de pequeno e médio porte devem tomar a consciência de que fazem parte de um mercado competitivo. Por exigência do mercado interno e externo as empresas buscam ampliar seus mercados, e uma das alternativas tem sido optar por investimentos em tecnologias da informação como sistema para a Web.

A Web não é somente um meio de comunicação, mas também uma ferramenta de vendas e informações com acessibilidade vinte e quatro horas por dia, seja através de computadores ou celulares, criando uma alta interatividade com os clientes em potenciais, além de aumentar a visibilidade da marca ou produto, podendo atingir um número significativo de pessoas através de sites e redes sociais com baixo custo, sendo assim, um excelente canal para divulgação de produtos, serviços e outras informações.

Também se observa um crescimento considerável no número de empresas presentes na internet ou que, ao menos, já vive a realidade virtual. Segundo o órgão regulamentador da internet brasileira, são mais de 107.000 domínios comercias registrados até a data presente (REGISTRO.BR, 2018). Seja através de sites institucionais ou pela troca de e-mails com fornecedores e clientes, as empresas estão se popularizando pelo meio digital.

Segundo (BITZ, 2018), o mercado hoteleiro é um dos setores que mais sofreram influência nos avanços tecnológicos, destacando *softwares* de gestão para hospedagem. Contudo, vislumbrando esse potencial de mercado e fazendo uso das afirmações de (PERES, 2018), o qual diz que ainda é necessário profissionalizar e automatizar a gestão de setores como o de hotelaria, esse projeto propõe uma aplicação web para suprir tal lacuna de mercado a um preço acessível a empresas de pequeno e médio porte.

A sede da empresa escolhida para a implantação do *software* se encontra em Lavras Novas, um pequeno distrito do município de Ouro Preto – MG que possui aproximadamente 1.500 habitantes de maioria negra, rica em história, folclores e lendas de domínio popular que está localizada a 117 km de Belo Horizonte e 17 km de Ouro Preto.

Desde 1990, turismo é a principal atividade econômica do distrito, onde se é dado ênfase ao ecoturismo, gastronomia e atividades culturais (SILVA, 2013). Assim, é notável o investimento nas áreas de hospedagem, bares e restaurantes, e por estar situada em pleno parque Nacional do Itacolomi com suas belezas naturais atraiu um público variado sendo os praticantes de esportes radicais e também os que buscam a calma e a simplicidade do interior.

O *software* desenvolvido irá atender tanto clientes quanto funcionários de empresas do ramo. Para ajudar nos testes e validar o software, uma empresa real fez parte do seu desenvolvimento.

1.1 Justificativa

Com o crescimento da economia de cidades como Lavras Novas a partir do turismo, existem vários hotéis e pousadas instaladas na cidade sem ferramentas de gerenciamento adequadas, uma dessas é a Pousada do Rei Arthur. Ela está situada no final da rua principal da cidade, a pousada é a única temática da cidade e região, em estilo de "Torres de Castelo", lembrando o período medieval e com decoração também ambientada por este período.

Uma das principais justificativas para a realização deste projeto é otimizar as funcionalidades oferecidas pela empresa, tanto para clientes, quanto para os funcionários. A empresa em questão não contém nenhuma tecnologia diferenciada para lhe auxiliar com reservas, fluxo de caixas, entre outras finalidades. Tudo é feito com a ajuda do planilhas desenvolvidas no *software* Excel e papéis pelos funcionários, essa situação pode gerar problemas como no fluxo de caixa, erros nas reservas, entre outros.

Os clientes de hotéis, pousadas e outros segmentos, atualmente procuram facilidade e agilidade em qualquer processo que tenham que realizar, como evitar filas em *check-in*, solicitação de serviço via *mobile* entre outras funcionalidades. Empresas que não acompanhar esse desenvolvimento tecnológico para oferecer serviços diferenciados aos hóspedes, estará em rota de colisão com a modernidade e deixará de ser competitivo no mercado atual.

O cenário globalizado exige uma precisão maior em quase todas as áreas, empresas buscam *softwares* personalizados que possam atender às suas necessidades de forma mais objetiva. Existem vários sistemas de hotelaria no mercado, porém apresentam algumas funcionalidades que não se encaixam nos negócios da empresa além de demandarem alto investimento financeiro. Assim, o impacto de um sistema de gerenciamento irá proporcionar para a empresa uma redução considerável de tempo, através de operações simplificadas, melhorando sua gerência e atendendo as necessidades específicas da organização sem grande investimento.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo o desenvolvimento de um sistema web que auxilie e proporcione um melhor gerenciamento das funcionalidades de empresas do ramo hoteleiro.

1.2.2 Objetivos Específicos

- Realizar análise de negócio do cliente;
- Realizar a modelagem do sistema;
- Pesquisar ferramentas e tecnologias para atender o projeto;
- Aprender ferramentas para o desenvolvimento de websites responsivos;
- Implementar fluxo de caixa;
- Implementar controle de reservas;
- Implementar interfaces que permitam aos usuários uma simplicidade nas decisões;
- Implementar regras de acesso ao website dos funcionários;
- Criar interfaces responsivas para as funcionalidades dos clientes;
- Permitir que clientes reservem suas acomodações diretamente de forma online;
- Criar uma página Home para apresentar a empresa.

1.3 Estrutura do Trabalho

Este documento está organizado em capítulos, sendo que este é o primeiro, onde são apresentadas as ideias iniciais do projeto, com os objetivos e a justificativa.

No capítulo 2 será apresentado a fundamentação teórica com base nas tecnologias usadas para o desenvolvimento do sistema web. O capítulo 3 contém os matérias e método utilizado para a realização do trabalho. O capítulo 4 contém a modelagem do sistema. O capítulo 5 apresenta os métodos e as principais funcionalidades do sistema. O capítulo 6 apresenta os trabalhos futuros, por fim, o capítulo 7 contém as considerações finais. Após são apresentadas as referências bibliográficas utilizadas como base para os textos.

2 FUNDAMENTAÇÃO TEÓRICA

As aplicações Web de hoje em dia possuem regras de negócio bastante complicadas. Codificar essas muitas regras já representam um grande trabalho. Além dessas regras, conhecidas como requisitos funcionais de uma aplicação, existem outros requisitos que precisam ser atingidos através da nossa infraestrutura: persistência em banco de dados, transação, acesso remoto, *web services*, gerenciamento de conexões HTTP, cache de objetos, gerenciamento da sessão web, balanceamento de carga, segurança, entre outros. São chamados de requisitos não-funcionais (CAELUM, 2015).

Se o desenvolvedor também for o responsável por escrever código que trate todos os requisitos mencionados acima, o custo de implementação seria muito mais alto e menos seguro. Tendo isso em vista, a empresa *Sun Microsystems* [1] criou uma série de especificações que, quando implementadas, podem ser usadas por desenvolvedores para tirar proveito e reutilizar toda essa infraestrutura já pronta, chamada Java EE.

O Java EE (*Java Enterprise Edition*) consiste de uma série de especificações bem detalhadas, dando uma receita de como deve ser implementado um software que faz cada um desses serviços de infraestrutura.

A ideia é que o desenvolvedor possa criar uma aplicação que utilize esses serviços. Como esses serviços são bem complicados, a equipe não perderá tempo implementando essa parte do sistema. Existem implementações tanto open source quanto pagas, ambas de boa qualidade.

Abaixo serão listadas as principais funcionalidades do Java EE que foram utilizadas no sistema desenvolvido com outras tecnologias que são compatíveis com o Java EE focado, claro, em desenvolvimento Web.

2.1 Java EE

Como já dissemos, o Java EE é uma plataforma ou ambiente para desenvolvimento de aplicações de grande porte e aplicações web que possui bibliotecas e funcionalidades que implementam *softwares* baseados na linguagem Java (DEVMEDIA, 2014a).

Ela oferece ao cliente e ao desenvolvedor recursos para as criações de sistema com segurança, escalabilidade, integridade, confiabilidade entre outros. Essa plataforma tem uma série de tecnologias que têm objetivos distintos, sendo que as mais conhecidas são:

[1] <https://www.oracle.com/br/sun/index.html>

- **Servlets:** São componentes Java executados no servidor que tem o objetivo de gerar conteúdo (HTML e XML) dinâmico para web;

- **Java Server Pages (JSP):** Especialização de *servlets* que permite aplicações em Java serem mais robustas e tenham facilidades no desenvolvimento;

- **Java Server Faces (JSF):** É um *framework* web com o padrão MVC (*Model, View* e *Controller*) baseado em Java que ajuda a simplificar o desenvolvimento de interfaces (telas do sistema) através de um modelo de UI;

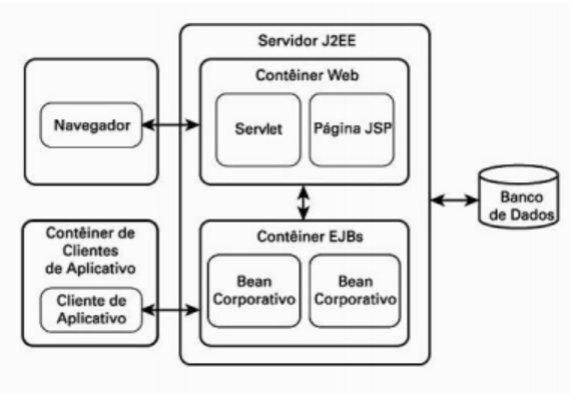
- **Java Persistence API (JPA):** É uma API padrão do Java que usa o conceito de mapeamento objeto relacional, sendo utilizada para a persistência dos dados. Essa ferramenta traz muita produtividade, pois consegue desenvolver aplicações que trabalham com banco de dados sem escrever nenhuma linha SQL;

- **Enterprise Java Beans (EJB):** São componentes que executam em um container de aplicação e que oferecem a facilidade e produtividade no desenvolvimento de componentes distribuídos, transacionados, seguros e portáteis.

Para fornecer a comunicação entre componentes, APIs, persistência, serviços entre outros, o Java EE oferece os containeres para abrigá-los. Os containeres são implementados pelos fornecedores de servidor de aplicativos Java EE, que disponibilizam um container para cada tipo de componente como: *Applet*, cliente de aplicativo, web e EJBs.

Os componentes Web e EJB são distribuídos, gerenciados e executados em um servidor Java EE. Veja como funciona na Figura 1.

Figura 1 – Arquitetura Lógica Java EE.



Fonte: (DEVMEDIA, 2014a).

Os contêineres armazenam, oferecem serviços (gerenciamento ciclo de vida, segurança, implementação, e comunicação com outros componentes) e recursos como um ambiente em tempo de execução compatível para os componentes Java EE conseguirem trabalhar.

2.2 Padrão MVC

Na fase de Projeto começamos a nos preocupar com a arquitetura da aplicação. Nesta fase damos realmente valor à tecnologia, diferente da fase de análise onde ainda estamos esboçando o problema a ser resolvido. Definimos a plataforma e como os componentes do sistema irão se organizar. Evidentemente que os requisitos ainda são importantes, pois, por exemplo, um sistema Web ou então uma aplicação de tempo real deverá influenciar na arquitetura,

O Engenheiro Civil Christopher Alexander (ALEXANDER, 1997) descreveu um padrão como sendo um problema que se repete inúmeras vezes em um mesmo contexto e que contenha uma solução para resolver tal problema de modo que esta solução possa ser utilizada em diversas situações. O termo padrões de projeto ou *Design Patterns*, descreve soluções para problemas recorrentes no desenvolvimento de sistemas de software orientados a objetos.

O uso de um padrão de projeto tem como intuito seguir um modelo pré-determinado de desenvolvimento que busca a melhor adaptação às necessidades dos desenvolvedores.

Desenvolver uma aplicação utilizando algum padrão de projeto pode trazer alguns dos seguintes benefícios:

- Aumento de produtividade;
- Uniformidade na estrutura do *software*;
- Redução de complexidade no código;
- As aplicações ficam mais fácies de manter;
- Facilita a documentação;
- Estabelece um vocabulário comum de projeto entre desenvolvedores.

O padrão MVC é utilizado em muitos projetos devido à arquitetura que possui, o que possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o *Model*, o *Controller* e a *View*, executa o que lhe é definido e nada mais do que isso (DEVMEDIA, 2013b).

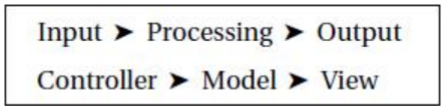
A utilização do padrão MVC traz como benefício isolar as regras de negócios da lógica de apresentação, a interface com o usuário. Isto possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem que haja a necessidade da alteração das regras de negócios, proporcionando assim muito mais flexibilidade e oportunidades de reuso das classes.

Uma das características de um padrão de projeto é poder aplicá-lo em sistemas distintos. O padrão MVC pode ser utilizado em vários tipos de projetos como, por exemplo, *desktop*, *web* e *mobile*.

O padrão arquitetural *Model-View-Controller* (MVC) é uma forma de quebrar uma aplicação, ou até mesmo um pedaço da interface de uma aplicação, em três partes: o modelo, a visão e o controlador.

O MVC inicialmente foi desenvolvido no intuito de mapear o método tradicional de entrada, processamento, e saída que os diversos programas baseados em GUI utilizavam. No padrão MVC, teríamos então o mapeamento de cada uma dessas três partes para o padrão MVC conforme ilustra a Figura 2:

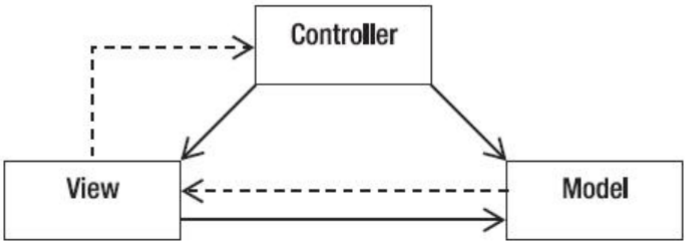
Figura 2 – Mapeamento das três partes de uma aplicação para o MVC.



Fonte: (DEVMEDIA, 2013b).

A Figura 3 demonstra que a entrada do usuário, a modelagem do mundo externo e o *feedback* visual para o usuário são separados e gerenciados pelos objetos Modelo (*Model*), Visão (*View*) e Controlador (*Controller*).

Figura 3 – Objetos utilizados no MVC e suas interações.



Fonte: (DEVMEDIA, 2013b).

O controlador (*Controller*) que interpreta as entradas do mouse ou do teclado enviado pelo usuário e mapeia essas ações do usuário em comandos que são enviados para o modelo (*Model*) e/ou para a janela de visualização (*View*) para efetuar a alteração apropriada. Por sua vez o modelo (*Model*) gerencia um ou mais elementos de dados, responde a perguntas sobre o seu estado e responde a instruções para mudar de estado.

O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema que está se tentando resolver. Por fim, a visão (*View*) gerencia a área retangular do *display* e é responsável por apresentar as informações para

o usuário através de uma combinação de gráficos e textos. A visão não sabe nada sobre o que a aplicação está atualmente fazendo, tudo que ela realmente faz é receber instruções do controle e informações do modelo e então exibir elas. A visão também se comunica de volta com o modelo e com o controlador para reportar o seu estado.

Tão importante quanto explicar cada um dos objetos do padrão arquitetural MVC é explicar como é o seu fluxo tipicamente. Primeiramente o usuário interage com a interface (por exemplo, pressionando um botão) e o controlador gerenciar esse evento de entrada da interface do usuário. A interface do usuário é exibida pela visão (*view*), mas controlada pelo controlador. O controlador não tem nenhum conhecimento direto da *View*, ele apenas envia mensagens quando ela precisa de algo na tela atualizado. O controlador acessa o modelo, possivelmente atualizando ela de forma apropriada para as ações do usuário (por exemplo, o controlador solicita ao modelo que o carrinho de compras seja atualizado pelo modelo, pois o usuário incluiu um novo item). Isto normalmente causa uma alteração no estado do modelo tanto quanto nas informações.

Por fim, a visão usa o modelo para gerar uma interface com o usuário apropriada. A visão recebe as informações do modelo. O modelo não tem conhecimento direto da visão. Ele apenas responde a requisições por informações de quem quer que seja e requisita por transformações nas informações feitas pelo controlador. Após isso, o controlador, como um gerenciador da interface do usuário, aguarda por mais interações do usuário, onde inicia novamente todo o ciclo.

Portanto, a principal ideia do padrão arquitetural MVC é a separação dos conceitos - e do código. MVC é como a clássica programação orientada a objetos, ou seja, criar objetos que escondem as suas informações e como elas são manipuladas e então apresentar apenas uma simples interface para o mundo.

Existem diversos *frameworks* para Java que implementam o padrão MVC e são muito utilizados em diversos projetos, entre eles temos o JSF, que será explicado no próximo tópico.

2.3 JSF

Segundo (BUENO, 2015), o JSF nada mais é que um *framework* onde é efetuada a elaboração de interfaces de usuários para sistema web, colocando componentes em um formulário e ligando os a objetos Java, sendo assim ele faz a separação entre a lógica e regras de negócio a navegação e conexões com serviços externos seguindo o modelo MVC. Tem como ponto forte a possibilidade de um grande número de componentes e um design bastante flexível por isso esse *framework* vem se acomodando nas novas tecnologias.

> O JSF (Java Server Faces) é a tecnologia padrão do j2EE 1.4 (ou superior) para criar aplicações web. Ele herda das tecnologias JSP e Servlets e extende

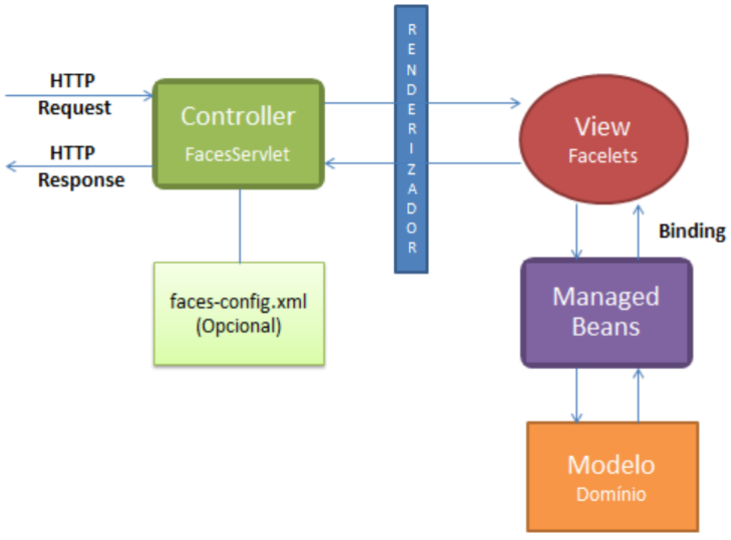
> seus conceitos com um ciclo de vida e um conjunto de componentes e recursos sofisticados e focados no desenvolvimento RAD para Web (GOMES, 2008).

Segundo (ALMEIDA, 2012), o JSF é baseado no padrão de projeto MVC (*ModelView-Controller*), o que torna o desenvolvimento de sistemas menos complicado, uma de suas melhores vantagens é a clara separação entre a visualização e regras de negócio (modelo).

O JSF utiliza a arquitetura MVC, porém com uma organização um pouco diferente. Ele trabalha com a classe *javax.faces.webapp.FacesServlet*, que exerce o papel de *Controller* da arquitetura MVC, controla as requisições, roteando o tráfego e administrando o ciclo de vida dos *beans* e componentes de interface do usuário (UI) (DEVMEDIA, 2014b).

Assim, como podemos visualizar na Figura 4, a *FacesServlet* recebe a requisição originada no navegador e chama a *View*, representada por nossas páginas XHTML. Conforme nossa *View* é processada, esta efetua a chamada do que for necessário aos *Managed Beans*, através de *bindings* entre componentes e *beans*. Os *Managed Beans* se encarregam de acessar as classes do Modelo, que são o domínio da aplicação.

Figura 4 – Arquitetura JSF.

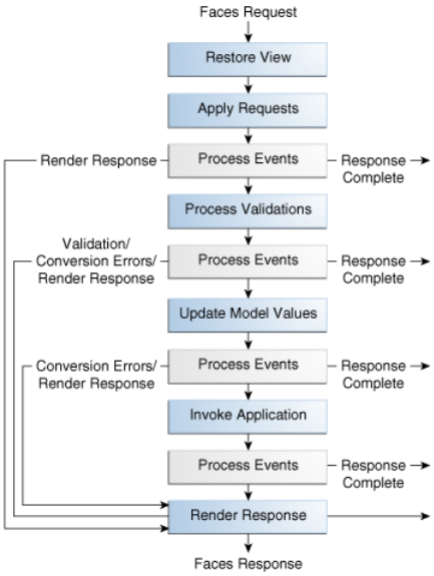


Fonte: (DEVMEDIA, 2014b).

Para (SOUZA, 2014), as seguintes requisições são manipuladas no ciclo de vida: Iniciais e postagens, quando o usuário faz a solicitação de uma requisição inicial de uma página, ocorre então respectivamente ocorre a solicitação da mesma pela primeira vez. Na postagem ocorre o processamento de um formulário que contém uma página previamente carregada em um *browser*.

Com a requisição inicial sendo manipulada pelo ciclo, ocorre a restauração da "*view*" e reformula as fases de resposta. Quando ocorre essa ação as fases "*Restore View*" e "*Render Response*" são executadas, isso porque não há um usuário ou ação para ocorrer processamento, em muitas das vezes um clique em um *link* gera a primeira de uma página JSF. A Figura 5 Ilustra todo o ciclo de vida do JSF.

Figura 5 – Ciclo JSF.



Fonte: (SOUZA, 2014).

> A fase do $RestoreView$ inicia quando uma requisição em uma página JSF é gerada por um click sobre um link ou botão, durante esse processo a implementação constrói um $view$ da página em manipulação. Para processar uma simples requisição é permitido resgatar todas as informações necessárias (SOUZA, 2014).

Segundo (SOUZA, 2014), sendo uma requisição inicial na página o JSF criará uma página vazia, nisto o ciclo passa para a fase em que ocorrerá a renderização da resposta, mas isso enquanto a "*view*" vazia está sendo populada por componentes referenciado por "*tags*".

A fase de *Render Response* é chamada se ocorrer qualquer problema, e a página é re-exibida com uma mensagem de erro sendo apresentada ao usuário. Se não ocorrer nenhum problema, a fase de "*Update Model Values*" é chamada.

O conceito de (SOUZA, 2014), informa que o JSF possui várias características que oferecem ganhos em desenvolvimento web:

- O Desenvolvedor tem permissão de criar UIs por meio de conjunto de componentes UIs predefinidos;

- Tem um conjunto de *tags* JSP para fazer o acesso aos componentes;

- Reutiliza componentes da própria página;

- Os eventos são associados ao lado cliente juntamente com os manipuladores de eventos do lado do servidor;

- Na construção de aplicações web são fornecidas separações de funções;

- Em alguns de seus componentes é utilizado *Ajax* para melhorar o desempenho e a eficácia em alguns processos.

2.4 *Apache Tomcat*

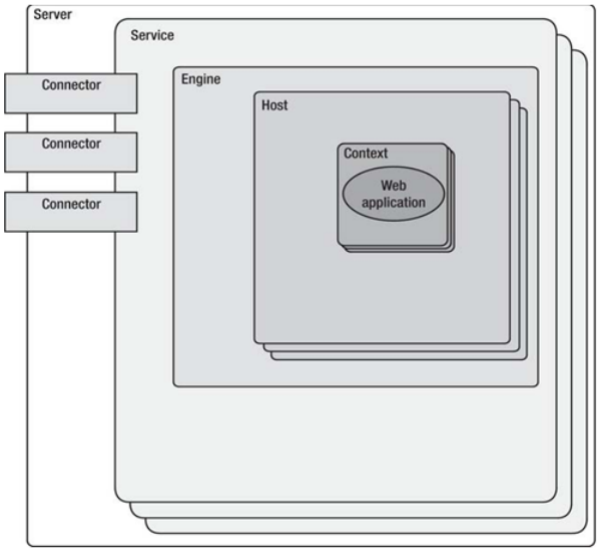
O servidor *Apache Tomcat* é um container Web de código fonte aberto baseado em Java que foi criado para executar aplicações Web que utilizam tecnologias *Servlets* e JSPs.

O servidor *Tomcat* foi criado inicialmente como um subprojeto da *Apache-Jakarta*, no entanto, devido a sua alta popularidade, acabou sendo designado para um projeto separado da *Apache*, sendo assim mantido por um grupo de voluntários da comunidade de código aberto do Java.*Apache Tomcat* é um servidor bastante estável com todas as características que um container comercial de aplicações web possui (DEVMEDIA, 2013a).

Uma instancia do *Tomcat* é o componente de mais alto nível na hierarquia do container *Tomcat*. Apenas uma única instância do *Tomcat* pode existir em uma única JVM. Esta abordagem torna todas as outras aplicações Java, rodando numa mesma máquina física num servidor *Tomcat*, seguras caso o *Tomcat* ou a JVM trave. Podemos ter múltiplas instâncias numa mesma máquina física, porém como processos Java separados rodando em portas separadas.

A instância do *Tomcat* consiste de um grupo de aplicações de container, no qual tem-se uma hierarquia muito bem definida. O componente principal desta hierarquia é o *Catalina Servlet Engine*. O *Catalina* trata-se da implementação do *Container Servlet* conforme especificado no *Java Servlet* API. *Tomcat* 7 nada mais é do que a implementação do *Servlet* API 3.0, a última especificação da *SUN*. A Figura 6 mostra o relacionamento dos principais componentes da arquitetura do *Tomcat*.

Figura 6 – Arquitetura $Tomcat$.



Fonte: (DEVMEDIA, 2013a).

2.5 *Hibernate*

A maioria dos sistemas desenvolvidos em uma empresa precisa de dados persistentes, portanto persistência é um conceito fundamental no desenvolvimento de aplicações. Se um sistema de informação não preservasse os dados quando ele fosse encerrado, o sistema não seria prático e usual.

Quando falamos de persistência de dados com Java, normalmente falamos do uso de sistemas gerenciadores de banco de dados relacionais e SQL, porém existem diversas outras alternativas para persistir dados.

Mapeamento objeto relacional (*object-relational mapping*, ORM, O/RM ou O/R *mapping*) é uma técnica de programação para conversão de dados entre banco de dados relacionais e linguagens de programação orientada a objetos.

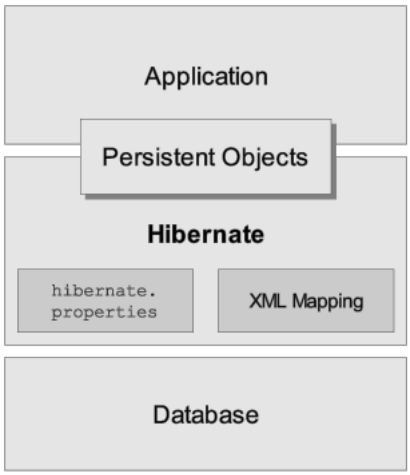
Em banco de dados, entidades são representadas por tabelas, que possuem colunas que armazenam propriedades de diversos tipos. Uma tabela pode se associar com outras e criar relacionamentos diversos. Em uma linguagem orientada a objetos, como Java, entidades são classes, e objetos dessas classes representam elementos que existem no mundo real, uma tecnologia para isso seria o *Hibernate*.

Segundo (SOUZA, 2014), o *Hibernate* é um *framework* de persistência com uma utilização muito grande no mundo Java, onde permite o mapeamento de classes em Java para tabelas de um banco de dados.

O *Hibernate* não tem apenas a função de mapeamento de objeto relacional, possui também um mecanismo poderoso de consulta de dados, podendo ter uma redução de tempo importante no desenvolvimento da aplicação. O *Hibernate* tem uma arquitetura simples formada por conjunto de interfaces. A Figura 7 nos mostra a descrição da arquitetura de alto nível do *Hibernate*.

Hibernate é uma camada que reside na JVM e garante o mapeamento dos objetos JAVA ocultos na JVM sob modelo relacional ou modelo de dados. Hibernate também garante a transferência das classes Java nas entidades de dados e, portanto, dados dos objetos nas entidades e tabelas (KIOSKEA, 2013).

Figura 7 – Arquitetura $Hibernate$.



Fonte: (SAUVé, 2000).

Hibernate implementa a especificação EJB 3.0/JPA, que permite que o mapeamento objeto/relacional passe a ser descrito na própria classe Java através do mecanismo de *annotations*, nativo da linguagem Java a partir da versão 5.0 do JDK. Assim como descrito na especificação EJB, as entidades são classes Java comuns (*POJOs*) e o conceito de entidade do EJB 3 é o mesmo utilizado pelas entidades persistentes do Hibernate (DEVMEDIA, 2009).

Annotations podem ser divididas em duas categorias:

a) **Anotações de mapeamento** lógico – permitem descrever o modelo de objetos, a associação entre classes, etc.;

b) **Anotações de mapeamento** físico – descreve o esquema físico, tabelas, colunas, índices, etc.

Toda classe Java que será persistida em um banco de dados através do *Hibernate* é considerada uma entidade e é declarada utilizando a anotação @*Entity* (acima da declaração da classe).

A anotação @*Id* permite que o usuário defina qual propriedade é a chave primária da sua entidade. A propriedade pode ser preenchida pela própria aplicação ou ser gerada pelo Hibernate (recomendado). É possível definir a estratégia para geração graças à anotação @GeneratedValue. Quatro tipos são possíveis:

1) **TABLE** – utiliza uma tabela com a informação dos últimos IDs para geração dos próximos;
2) **IDENTITY** – utilizado quando você possui bancos que não suportam sequence, mas suportam colunas identidade (por exemplo, SQL Server);
3) **SEQUENCE** – utilizando uma sequence para gerar a chave primária;
4) **AUTO** – Utiliza uma das estratégias acima de acordo com o banco de dados.

Vários tipos de associações, comuns em banco de dados, são representados pelas anotações do *Hibernate*. Veremos abaixo as principais associações:

2.5.1 Uma pra um

É possível associar entidades através de um relacionamento um-para-um usando a anotação @*OneToOne*. Existem três casos para utilização deste tipo de relacionamento: ou as entidades associadas compartilham os mesmos valores da chave primária; ou uma chave estrangeira é armazenada por uma das entidades (note que a coluna de chave estrangeira no banco de dados deve ter uma restrição de valor único para suportar a multiplicidade um-para-um); ou uma tabela associativa é usada para armazenar o *link* entre as duas entidades (uma restrição de valor único deve ser definida em cada FK para garantir a multiplicidade um-para-um).

2.5.2 Muitos pra um

As associações muitos-para-um são declaradas em nível de propriedades com a anotação @*ManyToOne*.

Podemos usar a anotação @*JoinColumn* para dar nome a tabela criada, essa anotação é opcional. Caso não seja preenchida, o Hibernate cria o nome através da concatenação do nome do relacionamento no lado da tabela muitos, do sublinhado (_) e do nome da coluna representante da chave primária no lado da tabela um.

2.5.3 Uma pra muitos

Associações um-para-muitos são declaradas em nível de propriedades com a anotação *@OneToMany*. Como esse relacionamento é bidirecional, na classe que representa o lado "muitos", utilizamos a anotação *@OneToMany(mappedBy=...)*.

2.5.4 Muitos pra muitos

Uma associação muitos-para-muitos é definida logicamente usando a anotação *@ManyToMany*. Você também pode descrever uma tabela associativa e a condição de ligação usando a anotação *@JoinTable*. Se a associação é bidirecional, um lado poderá atualizar as informações do relacionamento na tabela associativa enquanto o outro lado será ignorado em tal atualização.

De acordo com (ESJUG, 2014) implementar um mapeamento objeto relacional pode ser considerada uma tarefa complexa, porém esta tecnologia possui algumas vantagens:

- Produtividade - com a eliminação dos códigos SQL no código fonte, as classes passam a ser mais simples e com isso o sistema é desenvolvido em menor tempo;
- Manutenibilidade – por reduzir o número de linhas do código fonte do sistema, menor será o trabalho de manutenção do sistema;
- Desempenho – o tempo economizado no desenvolvimento, pode ser dedicado a programar otimizações do sistema;
- Independência de Fornecedor – por mais que um banco de dados utilize a mesma linguagem SQL, alguns comandos, tipos de dados, podem ser diferentes de um banco para outro.

2.6 *Criteria*

Quando utilizamos o *Hibernate* para realizar consultas no banco de dados, temos a oportunidade de trabalhar com SQL, com HQL e também com *Criteria Query* API. A API (*Application Programming Interface*) *Criteria* do *Hibernate* fornece uma maneira elegante de construção de query dinâmica para consultas no banco de dados. A API é uma alternativa as consultas HQL (*Hibernate Query Language*) e também ao SQL tradicional (BALLEM, 2017).

Usando SQL como consulta, trabalhamos diretamente com tabelas e colunas que devem ser descritas na consulta através de strings. O HQL modificou a forma como as consultas são realizadas quando ao invés de trabalharmos com tabelas e colunas, usamos classes e atributos das classes (variáveis), construindo assim o relacionamento orientado a objeto entre banco de dados e aplicação. Essa técnica possibilita a consulta através de objetos, porém continuamos ainda escrevendo a consulta em linhas strings e não em código Java.

Já a API *Criteria* une as vantagens do HQL com a praticidade de escrever consultas com código Java através da interface *org.hibernate*. *Criteria*, abolindo assim o uso de longas strings. A interface *Criteria* define os métodos necessários para trabalhar com *Criteria* e por ser uma consulta de forma programática, erros no código podem ser sinalizados ainda em tempo de compilação, coisa que com HQL ou SQL não é possível observar.

2.7 *Spring Security*

No mundo de TI, segurança é uma palavra que tem que ser levada em conta com um nível alto de relevância. Isso significa tantas coisas diferentes em tantos diferentes contextos, que por fim, deve ser levado sobre uma proteção de recursos sensíveis e valiosos contra o uso malicioso (SCARIONI, 2013).

A ferramenta *Spring Security*, possibilita que a autenticação fique forte e altamente personalizável e ainda estabelece uma estrutura de controle de acesso que é o padrão de *facto* para a proteção de aplicativos baseados no *framework Spring* (CHHATPAR, 2006).

Com o aumento da produção de *software* os problemas são frequentes, a *framework Spring* possibilita solucionar inúmeros problemas comuns presentes no desenvolvimento J2EE e Java JEE. O objetivo dessa ferramenta é lembrar de fazer a coisa certa na programação orientada a objetos: projetar aplicações usando interfaces (CHHATPAR, 2006).

Atualmente mantido pela empresa Interface21, o *Spring* é um *framework* que traz diversos benefícios as aplicações, aumentando a produtividade no desenvolvimento de aplicações além de promover um grande aumento de performance em tempo de *runtime* e facilitar o trabalho com testes unitários (SCHITINI, 2011).

O *Spring Framework* é composto por recursos organizados em cerca de 20 módulos, tais módulos podem ser implementados separadamente ou em conjunto com outros, isto permite ao *Spring* ser aplicado nos mais variados tipos de aplicações, sendo estas de qualquer porte (CALçADO, 2008), (JOHSON, 2011).

O núcleo da Estrutura *Spring* é baseado no princípio da Inversão de Controle (IoC). Aplicações que seguem este princípio usam configurações que descrevem as dependências entre seus componentes. Depois, fica a cargo da estrutura IoC satisfazer as dependências configuradas. A "inversão" significa que a aplicação não controla sua estrutura; é responsabilidade da estrutura IoC fazê-lo.

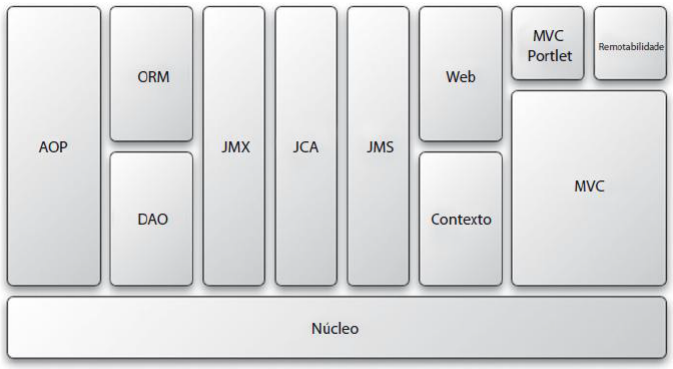
Segundo (VUKOTIC, 2009):

> O JSF (A implementação da DI (Injeção de dependência) põe o foco o pouco acoplamento: os componentes de sua aplicação devem presumir o mínimo possível dos outros componentes. A maneira mais fácil de se obter o baixo acoplamento no Java é codificar para interfaces.

O *Framework Spring* é composto de vários módulos bem definidos conforme mostra a Figura 8. Segundo (WALLS, 2008) "Ao serem tomados como um todo, estes módulos fornecem tudo o que é necessário para desenvolver aplicativos para soluções corporativas." Mas não é necessário ter o *Framework Spring* como base para desenvolver aplicações.

Os módulos do *Spring* são livres podendo ser escolhidos o que melhor se adaptarem ao aplicativo em desenvolvimento. De acordo com WALLS, 2008 O *Spring* oferece pontos de integração com outros *frameworks* e bibliotecas, não tendo que desenvolver por si próprio." Os módulos do *Spring* são demonstrados na Figura 8.

Figura 8 – Módulos do $SpringFramework$.



Fonte: (WALLS, 2008).

Dentro da *Spring* existem vários projetos que auxiliam no desenvolvimento. O *Spring Security* é um dos projetos mais maduros e amplamente utilizado. Fundado pela *SpringSource* em 2003 ele é usado para garantir inúmeros ambientes exigentes, como: agências governamentais, aplicações militares e bancos centrais. Para que o usuário possa usá-lo com confiança em seus projetos a *Apache* 2.0 libera a licença (ROBERT, 2012).

Spring Security tem o foco em tornar a parte de autenticação e autorização uma coisa simples de fazer. Ele tem uma variedade muito grande de opções e ainda é bastante extensível.

Com algumas poucas configurações já podemos ter uma autenticação via banco de dados, LDAP ou mesmo por memória. Sem falar nas várias integrações que ele já suporta e na possibilidade de criar as suas próprias.

Quanto a autorização, ele é bem flexível também. Através das permissões que atribuímos aos usuários autenticados, podemos proteger as requisições web (como as telas do nosso sistema, por exemplo), a simples invocação de um método e até a instância de um objeto.

2.8 *PrimeFaces*

Atualmente, existem diversas organizações que trabalham na criação de componentes personalizados, como exemplo, podemos citar a *Oracle* (*ADF Faces Rich Client*), *IceSoft* (*IceFaces*), *Red Hat* (*RichFaces*), *Prime Technology* (*PrimeFaces*) e etc.

As bibliotecas de componentes terceiras incluem muitos componentes interessantes, como tabelas de dados avançadas, menus suspensos, botões, barras de progressão, diálogos, componentes para captura de datas e cores, etc.

> A interface de um sistema, seja gráfica ou não, pode ser entendida como a estrutura que transmite ao usuário a ideia real do que é possível realizar com aquele determinado aplicativo computacional. Uma forma de implementar uma interface gráfica para web que possam ser denominadas como interface rica é por meio do uso do $framework PrimeFaces$. Esse $framework$ é de código fonte aberto, utilizado para auxiliar no desenvolvimento de páginas web que utilizem a tecnologia JSF ($JavaServerFaces$) (DEVMEDIA, 2012).

O *Primefaces* é um *framework* para interface gráfica, composto por mais de 100 componentes para o JSF, ou seja, é uma biblioteca de componentes para JSF, com suporte nativo a *Ajax* e HTML 5. É de fácil utilização, possui boa documentação e tem ricos exemplos de códigos disponibilizados na web para auxílio na utilização dos mesmos.

PrimeFaces é uma biblioteca leve, todas as decisões tomadas são baseadas em manter as aplicações tão leves quanto possível. Normalmente a adição de uma solução de terceiros poderia trazer uma sobrecarga, mas, no entanto, este não é o caso com *PrimeFaces*. É apenas um único *.jar* com nenhuma dependência e nada para configurar(TECNOLOGY, 2014).

Segundo (VENTURINI, 2011), o *PrimeFaces* oferece um conjunto de componentes com versões estáveis e de código aberto para o JSF 2.0 e permite que sejam inseridos em seu conjunto, outros componentes através de especificações em JSF. Ele está organizado em três módulos:

a) *User Inteface (UI) Components* – compreende os componentes que contém as funcionalidades encapsuladas de *AJAX*, *Javascript* e gráficos animados;

b) *Optimus* – módulo que oferece soluções para facilitar desenvolvimento com JSF. Também contém componentes de extensões de segurança;

c) *FacesTrace* – módulo encarregado das funções relacionadas ao desempenho das aplicações baseadas em JSF.

Segundo (FEITOSA, 2010), "*PrimeFaces* é uma biblioteca enxuta e com um único arquivo, não sendo necessário nenhuma configuração e não tem nenhuma dependência.". O *framework* permite muitas possibilidades de criação de *layout* para aplicações web e temas gráficos

que podem ser alterados facilmente, evitando a necessidade de utilizar componentes baseados em outras tecnologias.

Segundo (PRIMEFACES, 2018), o *framework* é considerado melhor do que muitos outros, tendo em vista que:

a) simplicidade e desempenho: *PrimeFaces* é leve, todas as decisões tomadas são baseadas em manter o *PrimeFaces* tão leves quanto possível;

b) facilidade de uso: os componentes do *PrimeFaces* são desenvolvidos com um princípio de design que afirma que "Uma boa UI deve ocultar a complexidade, mas manter a flexibilidade";

c) possui o *Mobile UI kit* para criar aplicações web móveis para dispositivos portáteis baseados em navegadores *webkit*, por exemplo, *IPhone*, *Palm*, *Android* entre outros;

d) possui um rico conjunto de componentes de interface: *DataTable, AutoComplete, HTMLEditor*, gráficos e etc.;

e) não é necessária nenhuma configuração extra de XML e não há dependências;

f) os componentes são construídos com *AJAX* no padrão JSF 2.0;

g) possui mais de 30 temas de *templates*;

h) possui boa documentação com exemplos práticos.

2.9 *Websites* Responsivos

Ao construir um site, o *designer* deve refletir sobre aspectos que vão muito além da escolha de cores, disposição dos ícones, preferências de seu cliente, entre outros. Ele deve conhecer previamente para quem vai desenvolver (empresa, tipo de mercado), entender um pouco sobre o público-alvo, ou seja, saber qual a idade média deste público, qual o segmento de mercado a qual está servindo, se será um site somente para homens ou mulheres, ou se irá atender à uma região específica (SILVA, 2014a).

Para isto, o designer necessita analisar tendências, explorar outros sites que atendam ao mesmo público, mesmo que por exemplo, seja um site para comunicação de governos com a população, ou um site de *e-commerce*, ele precisa entender o foco de cada um deles.

Faz-se necessário ainda, que o *designer* seja capaz de observar detalhadamente sites pré-existentes, imagens, entre outros para então selecionar estilos e formatos mais adequados e que melhor se encaixem ao público destinado, e dependendo de todos estes fatores e o que foi resumido e esboçado para este site a ser desenvolvido, o *designer* poderá buscar ferramentas que adaptem-se e tragam maior facilidade ao processo de construção do site.

Para contemplar todos estes fatores e ferramentas que um designer deve conhecer, seria necessário falar de *Design*, *Web Design*, *Design* Responsivo, *Design* Estratégico, *Marketing*, Semiótica, Engenharia Semiótica, IHC, e tantas outras teorias, algumas mais antigas que outras, mas que estão em constante desenvolvimento e evolução e que podem auxiliar e aumentar o conhecimento dos *designers*.

Para entender melhor, *design* Responsivo é um princípio de desenvolvimento para Web cujo objetivo é adaptar o *layout* das páginas a qualquer dispositivo, tela e resolução, com objetivo de garantir a boa experiência do usuário, possibilitando navegação e leitura confortáveis sem comprometer o conteúdo. (SILVA, 2014b) explica o conceito de *Design* Responsivo:

> Antes de qualquer coisa, é necessário que fique muito claro que $design$ responsivo não diz respeito simplesmente e somente à adaptação do $layout$ ao tamanho da tela.
>
> Vai muito além disso, pois o conceito de design responsivo na sua forma ampla deve ser entendido como $design$ capaz de responder às características do dispositivo ao qual é servido. Responder, neste contexto, tem sentido de movimentar-se expandindo e contraindo.
>
> Em outras palavras, o $design$ responsivo ou $layout$ responsivo expande e contrai com a finalidade de se acomodar de maneira usável e acessível à área onde é visitado ou, mais genericamente, ao contexto onde é renderizado, seja um $smartphone$, um $tablet$, um leitor de tela, um mecanismo de busca etc. (SILVA, 2014b).

O conceito foi atualizado e fundamentado em 2010 no artigo "*Responsive Web Design*" escrito por (MARCOTTE, 2010) no blog "*A List Apart*". O autor propõe que, em vez de desenvolver um *design* para cada dispositivo, deveria ser projetado um único código que adaptasse o *layout* para as diferentes telas, por meio de tecnologias padronizadas (*HyperText Markup Language [HTML] e Cascading Style Sheets [CSS]*). Antes do *Design* Responsivo, era comum a criação de uma ou mais versões *mobile* do mesmo site, o que dificultava a manutenção do conteúdo.

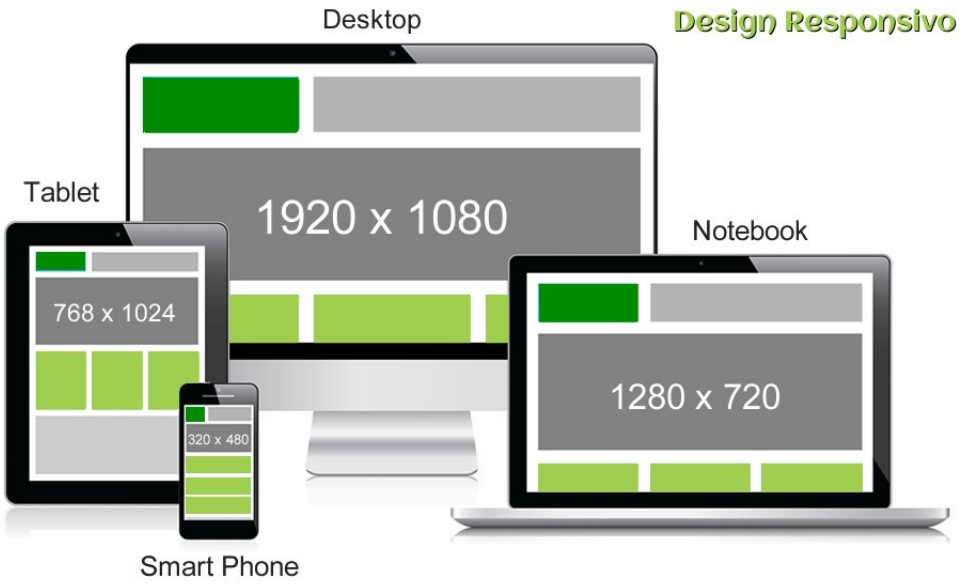
Uma explanação sobre a evolução dos *layouts* de web é pertinente para entender o assunto a ser tratado. (SILVA, 2014b):

> Em 1992, no NCSA Centro Nacional de Aplicações para Supercomputadores da Universidade de Illinois , foi desenvolvido o primeiro navegador gráfico para a web, que durante quatro anos foi responsável pela popularização da internet. Foi com o navegador denominado Mosaic que tudo começou.

Ao longo destes anos, outras tecnologias como o HTML1, 2, 3, 4 e o 5, HTML (*Hypertext Markup Language*) é uma linguagem utilizada para estruturação e apresentação de conteúdo de uma tela, entre outras evoluíram juntamente com as novas possibilidades e novas formas de se utilizar a Internet. Uma das últimas evoluções, foi o CSS - *Cascading Style Sheets* - que possibilitou a separação da marcação de conteúdos de sua apresentação. Simplificando a explanação, neste caso, HTML é a ferramenta para estruturar o site e o CSS para apresentar o conteúdo.

O *design* responsivo é também um termo bastante novo e tem o propósito de servir layouts para *desktop* e dispositivos móveis, considerando que foi Ethan Marcotte que falou sobre este tema pela primeira vez em 2010. Em seu livro, (SILVA, 2014b) citando Ethan, explanou que por causa do crescimento de usuários que utilizam dispositivos móveis, algumas empresas estavam prejudicando-se por ainda possuírem um único site a ser acessado por diferentes dispositivos. Com o objetivo de simplificar o entendimento do *design* responsivo, buscou-se exemplos dos diferentes tamanhos possíveis a que um site deve se ajustar e respectivamente, como deve ser um site responsivo.

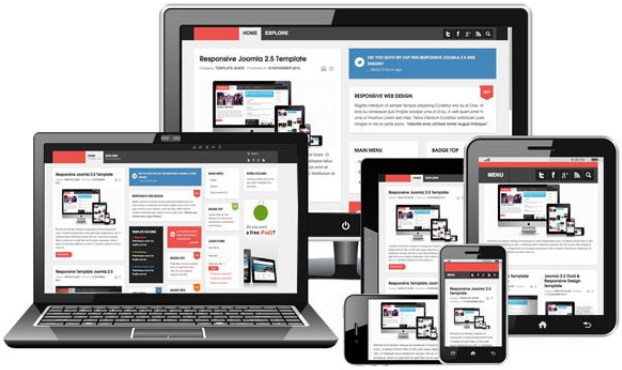
Figura 9 – Tamanhos possíveis de telas.



Fonte: Disponível em: < https://wsidm.com.br/blog/como-construir-um-website-responsivo-estrategia-seo/> acesso em 14/04/2018.

Na Figura 9 estão representados apenas alguns tamanhos possíveis de telas, pois existe hoje no mercado, *Androids* com uma infinidade de tamanhos diferentes. Para adaptar-se a esta infinidade de tamanhos, com a utilização do *design* responsivo, é possível obter um site que, mesmo mudando o tamanho e tipo de dispositivo, pode ser visto de maneira que o usuário não necessite ficar utilizando o *zoom* para poder enxergar seu conteúdo.

Figura 10 – Como devem ser sites responsivos.



Fonte: Disponível em: < http://www.divulgacaodesites.com.br/desenvolvimento-de-sites-responsivos.html > acesso em 14/04/2018.

Na Figura 10 é possível perceber um site responsivo em diferentes dispositivos exemplificados.

2.10 *Bootstrap*

Em 2011, o *Bootstrap* foi criado como uma solução interna para resolver inconsistências de desenvolvimento dentro da equipe de engenharia do *Twitter*. Basicamente, não havia uma definição de estrutura de código na forma escolhida pelos engenheiros do *Twitter* para desenvolver a plataforma.

O desenvolvimento e a engenharia da web são um trabalho especializado muitos dizem que é uma arte, e cada engenheiro tem a sua própria maneira de codificar. Isso funciona em alguns casos, mas quando há vários engenheiros trabalhando no mesmo projeto com abordagens de codificação ligeiramente diferentes, as inconsistências são inevitáveis. Com o tempo, as inconsistências de engenharia da web podem se transformar em problemas de codificação sérios que criam incerteza e aumentam o tempo de manutenção.

O *Bootstrap* foi uma ferramenta desenvolvida originalmente por Mark Otto e Jacob Thorton, então engenheiros do *Twitter*. A iniciativa do *Bootstrap* certamente foi bem-sucedida no *Twitter*, permitindo que toda a equipe trabalhasse com maior rapidez e eficiência e menos inconsistências.

Embora inicialmente uma solução interna no *Twitter*, Mark e Jacob rapidamente perceberam que tinham descoberto algo muito maior. Em agosto de 2011, a estrutura *Bootstrap* foi lançada como um projeto de *software* livre no *Github*. Em alguns meses, milhares de desenvolvedores de todo o mundo contribuíram com o código e o *Bootstrap* se tornou o projeto de

desenvolvimento de *software* livre mais ativo do mundo. Desde então, tornou-se cada vez mais conhecido e se transformou na "estrutura *front-end* mais popular para o desenvolvimento inicial de projetos móveis com capacidade de resposta na web".

Podemos definir o *bootstrap* como: É uma coleção de vários elementos e funções personalizáveis para projetos da web de código aberto (*opensource*), empacotados previamente em uma única ferramenta. Em palavras simples, é um conjunto de ferramentas para facilitar o desenvolvimento de sites e sistemas web. Ao projetar um site com o *Bootstrap*, os desenvolvedores podem escolher quais elementos querem usar. E, o mais importante, podem ter a certeza de que os elementos escolhidos não conflitarão entre si e serão totalmente responsivos funcionando tanto para *desktop* como quaisquer outros tamanhos de tela. É como um quebra-cabeças, exceto que cada peça se encaixa perfeitamente com as outras, quaisquer que sejam as peças escolhidas.

A estrutura do *boostrap* é simples e seu pacote contém três tipos diferentes de arquivos (CSS, *JavaScript* e *Fonts*), que vêm devidamente organizados em suas pastas com mostra a Figura 11.

Figura 11 – Estrutura $Bootstrap$.

Fonte: Disponível em: < https://getbootstrap.com/ > acesso em 14/04/2018.

Montar um *layout* é simples e rápido utilizando sua documentação. Como toda a estrutura do CSS já vem definida, basta procurar o componente necessário e adicionar seu código. Em poucos minutos seu *layout* toma forma e está pronto para uso. E o mesmo acontece com o *JavaScript*.

O *bootstrap* funciona com um sistema de *grids* que possibilita sua divisão em até doze colunas de mesma largura. Essas grids são nativamente responsivas, você pode definir *grids* diferentes para cada tipo de resolução.

Algumas vantagens de usar o *framework Bootstrap* (ZEMEL, 2013):

Figura 12 – Sistema de $grids$ responsivas ($smarttv, desktop, tablet, smartphone$).

Fonte: Disponível em: < https://getbootstrap.com/ > acesso em 14/04/2018.

- É otimizado para o desenvolvimento de layouts responsivos;
- Possui uma interface super amigável e moderna;
- Padronização de códigos;
- Aumento da produtividade;
- Atualmente possui uma grande diversidade de temas;
- Grande quantidade de *plugins* adaptados ou desenvolvidos para o *framework*;
- Integração com qualquer linguagem de programação;
- Sistema responsivo;
- Um dos *frameworks* mais utilizados no desenvolvimento de portais e sistemas do mundo;
- Possui documentação detalhada e de fácil entendimento;
- *Download* facilitado e totalmente grátis.

Desvantagens:

- Seu código terá de seguir os "padrões de desenvolvimento *Bootstrap*";
- Tema padrão e comum do *Bootstrap* (caso não faça ajustes visuais, ou troque o tema seu projeto se parecerá com outros que também utilizam o *Bootstrap*).

3 MATERIAIS E MÉTODOS

A seguir serão descritos as ferramentas e o método que foram utilizados para a modelagem e desenvolvimento do sistema

3.1 *Microsoft Visio – UML*

O *Microsoft Visio* é um programa da *Microsoft* que permite compor diversos tipos de gráficos de organização diferentes, como fluxogramas, organogramas e diagramas em geral. Esta é uma versão de teste do programa e requer um cadastro para receber a chave para teste que pode ser efetuado utilizando sua conta da *Microsoft*.

Ele disponibiliza diversas ferramentas para auxiliar no gerenciamento de projetos e milhares de figuras, formas e desenhos diferentes, permitindo que você possa desenhar desde o planejamento de uma estrutura de rede para aplicação em uma empresa até um organograma empresarial, além de WBS, fluxograma, diagramas de modelagem para desenvolvimento, diagrama de caso de uso entre outros (TECHSOUP, 2017).

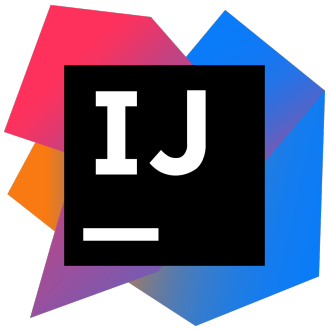
O *Microsoft Visio* é uma excelente ferramenta para desenvolver projetos e, como a maioria dos programas da *Microsoft*, conta com uma ampla gama de opções. A montagem dos diagramas é inteiramente feita por “clicar e arrastar” imagens e mesmo inserções de textos ou ligações entre figuras são extremamente simples por contarem com botões para a função.

A interface do programa é muito “visual”, o que permite que o uso de suas ferramentas se torne intuitivo, mesmo para usuários iniciantes. Não traz maiores dificuldades em seu uso, pois é uma ferramenta cuja organização de menus e conteúdos traz familiaridade, especialmente para usuários do pacote *Office da Microsoft*. Da mesma forma, as funcionalidades são intuitivas e mesmo que em um primeiro momento você não entenda perfeitamente como aplicá-las, a utilização de certa forma demonstra o que é preciso para completar o uso da função.

Figura 13 – $Microsoft Visio$.

Fonte: Disponível em: < http://logos.wikia.com/wiki/MicrosoftVisio > acesso em 20/04/2018.

Figura 14 – $IntelliJ$ IDE.



Fonte: Disponível em: < https://houndsoft.blogspot.com.br/2017/05/intellij-idea-ultimate-v201712-final.html > acesso em 20/04/2018.

3.2 IDE *IntelliJ*

IntelliJ IDEA é um Java IDE por *JetBrains*, para computadores que oferece suporte para todos os desenvolvedores que querem trabalhar com *Frameworks*, serviços corporativos e dispositivos móveis (TARGETWARE, 2017).

Para programação Web o *IntelliJ* IDEA possui várias ferramentas que irão ajudar, usando ferramentas como *Spring* MVC, *Webflow*, Reproduzir, *Grails*, *Web Services* , JSF , *Struts* e *Flex*, além de incluir assistência a códigos de programação HTML5, CSS3, SASS, LESS, *JavaScript*, *CoffeeScript*, *Node.js* e *ActionScript*.

IntelliJ IDEA permite escrita de código sem complicações. Ele pratica uma abordagem não-intrusiva intuitiva para ajudá-lo a escrever, depurar, refatorar, testar e aprender o seu código. Graças à sua profunda compreensão das linguagens e tecnologias, *IntelliJ* IDEA oferece um segundo par de mãos quando você precisar deles.

A IDEA cria um ambiente conveniente, onde todos os membros da equipe podem trabalhar juntos de forma eficiente. Integração transparente com uma grande variedade de sistemas de controle de versão permite que os membros da equipe permaneçam em sincronia com as mudanças dos outros, garantindo que todas as contribuições serão produtivas. *IntelliJ* IDEA pode coexistir com outras IDEs populares, como o *Eclipse* e ferramentas de gerenciamento de projetos como o *Maven*, assim sua equipe pode usar cada ferramenta, onde é melhor aplicável.

Para melhorar ainda mais a produtividade da equipe global, *IntelliJ* IDEA coopera com *JetBrains TeamCity*, uma integração contínua e um poderoso servidor de compilação.

IntelliJ IDEA permite à sua empresa fornecer produtos de *softwares* complexos dentro do cumprimento das normas de qualidade e de cronogramas apertados. O IDE constantemente valida a qualidade do código e oferece correções imediatas para problemas encontrados em to-

dos os níveis - a partir de instruções individuais para a arquitetura global, usando as inspeções de código avançada e análise da matriz de dependência. Seja um problema padrão de codificação, problemas potenciais de desempenho, ou violação do contrato, *IntelliJ* IDEA exibe um aviso e corrige o problema, ajudando a produzir de forma limpa, código de primeira linha em menos tempo do que nunca (TARGETWARE, 2017).

3.3 Banco de Dados *MySQL*

O *MySQL* é um sistema gerenciador de banco de dados relacional de código aberto usado na maioria das aplicações gratuitas para gerir suas bases de dados. O serviço utiliza a linguagem SQL (*Structure Query Language* – Linguagem de Consulta Estruturada), que é a linguagem mais popular para inserir, acessar e gerenciar o conteúdo armazenado num banco de dados (PISA, 2012).

O *MySQL Workbench* é uma ferramenta visual para design, desenvolvimento e administração de base de dados *MySQL*. Essa ferramenta é originária do *DBDesigner* (VESPA, 2010).

A ferramenta é muito boa e completa. Possui as opções de abrir conexão, editar dados, editar scripts SQL, gerenciar conexões, novo modelo de dados, modelo de dados a partir da base (engenharia reversa), modelo de dados de um *script* SQL, criação de instâncias de servidor, importação/exportação de base, gerenciamento de segurança e gerenciamento de instâncias.

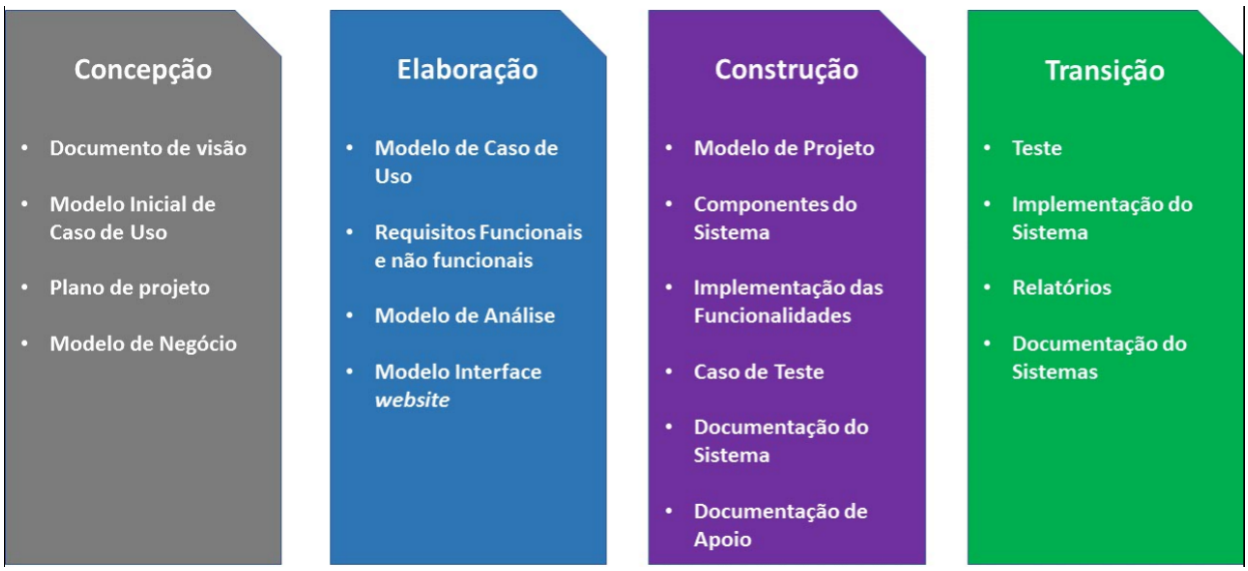
Figura 15 – $MySQLWorkbench.$

Fonte: Disponível em: < http://grandeportal.blogspot.com.br/2014/06/instalando-mysql-e-workbench-no-ubuntu.html > acesso em 20/04/2018.

3.4 Metodologia

A metodologia de trabalho será baseada no Processo Unificado (PU), o qual é um processo de desenvolvimento de *software* compostos por quatro etapas básicas. Cada etapa dita um conjunto de atividades necessárias para transformar requisitos do usuário em um sistema de *software*. O PU de desenvolvimento de sistemas combina ciclos iterativo e incremental para a construção de software. A Figura 16 mostra as atividades básicas em cada etapa.

Figura 16 – Produtos de Trabalho do Processo Unificado em Cada Fase.



Fonte: Elaborada pelo autor.

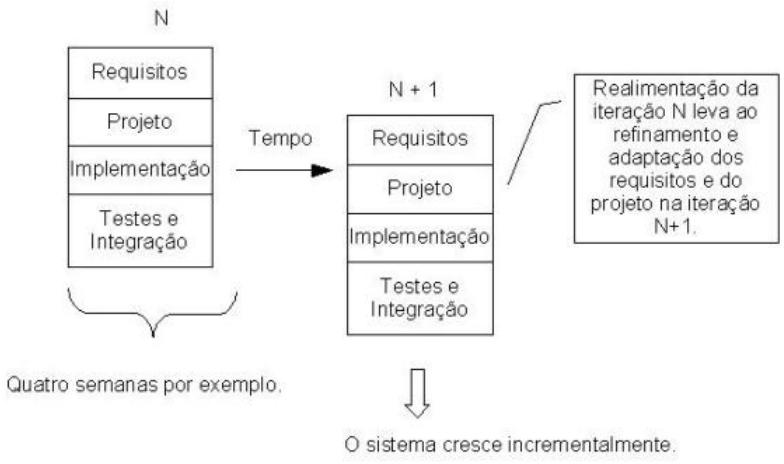
O Processo Unificado (PU) surgiu como um processo popular para o desenvolvimento de *software* visando à construção de sistemas orientados a objetos (o RUP – *Rational Unified Process* é um refinamento do PU). É um processo iterativo e adaptativo de desenvolvimento e vem ganhando cada vez mais adeptos devido a maneira organizada e consistente que permite conduzir um projeto.

O PU utiliza um paradigma evolucionário para o desenvolvimento de softwares. O ciclo de vida iterativo é baseado em refinamentos e incrementos sucessivos a fim de convergir para um sistema adequado. Em cada iteração incrementa-se um pouco mais o produto, baseando-se na experiência obtida nas iterações anteriores e no *feedback* do usuário, Figura 17. Cada iteração pode ser considerada um miniprojeto de duração fixa, sendo que cada um destes inclui suas próprias atividades de análise de requisitos, projeto, implementação e testes (DEVMEDIA, 2016).

O resultado de cada iteração é um sistema executável, porém incompleto. Ele não está pronto para ser colocado em produção e pode continuar nesta situação ainda por muitas iterações. Vale ressaltar também que cada iteração produz um sistema com qualidade de produto final, e não um protótipo.

Ao invés de combater a inevitável mudança que ocorre no desenvolvimento de *software* (principalmente nas fases iniciais), o PU prega uma atitude de aceitar a mudança e a adaptação como fatores inevitáveis e, de fato, essenciais. Não se deve tentar especificar completa e corretamente o sistema em uma tacada só, com a ideia de criar um conjunto congelado de requisitos.

Figura 17 – Desenvolvimento iterativo e incremental.



Fonte: (DEVMEDIA, 2016)

Em cada iteração é escolhido um pequeno subconjunto de requisitos, os quais são rapidamente projetados, implementados e testados pelos usuários. Isso leva a uma realimentação rápida - baseada em dados concretos de usuários, desenvolvedores e testes – que possibilita modificar ou adaptar a compreensão dos requisitos do projeto. Os usuários finais podem ver um sistema parcial, utilizá-lo e assim terão mais subsídios para criticar ou aprovar. Esse ciclo de avaliações e detecção de falhas não traduz um erro, mas sim, representam um modo hábil de progredir e descobrir o que é de real valor para os interessados no projeto.

Além de melhorar a compreensão dos requisitos, a implementação precoce possibilita detectar se o projeto está no caminho certo ou, por exemplo, se alguma mudança na arquitetura central deve ser feita.

Consequentemente o trabalho progride por meio de uma série de ciclos estruturados em construção-realimentação- adaptação. É melhor resolver e pôr à prova as decisões arriscadas e fundamentais de projeto logo no início e o desenvolvimento iterativo fornece o mecanismo para isso. Processo Unificado foi criado para ser um processo ágil de desenvolvimento e prega uma abordagem realística para a condução de um projeto. Ao contrário do modelo em cascata, onde cada etapa do ciclo de vida é realizada integralmente e de uma só vez (e que é mais apropriado para a construção de prédios do que para *softwares*), no PU as atividades são repetidas quantas vezes forem precisos, em ciclos organizados.

Não há um plano detalhado para todo um projeto. Há um plano de alto nível (chamado Plano de Fases) que estima a data de término do projeto e outros marcos de referência principais, mas ele não detalha os passos de granularidade fina para se atingir tais marcos.

Um plano detalhado (chamado Plano de Iterações) somente planeja a iteração a ser feita em seguida. O planejamento detalhado é feito de forma adaptativa, de iteração para iteração.

4 MODELAGEM DO SISTEMA

Este capítulo apresenta a modelagem realizada para o desenvolvimento do sistema e *website* proposto para rede hoteleira.

4.1 Apresentação do Sistema

O controle de reserva e de fluxo de caixa são as principais funcionalidades do sistema proposto, mas é importante lembrar que existem outras funcionalidades importantes para que essas duas principais funcionem corretamente.

Outro ponto que vale ressaltar é que os funcionários da empresa não serão os únicos beneficiados, os clientes contaram com um novo site que segue as mais novas tendências de *marketing* do mercado digital e principalmente a vantagem de ser um site responsivo, ou seja, que se adapte a qualquer tela.

4.2 Modelagem

Para que seja possível o desenvolvimento de um sistema que atenda as reais necessidades do cliente, é preciso realizar uma modelagem correta das informações que serão controladas pelo sistema, demonstrando estes dados de forma a facilitar todo o processo de implementação, o que pode ser por meio de técnicas de modelagem de *software*, na utilização em metodologias de desenvolvimento ou em ferramentas de apoio.

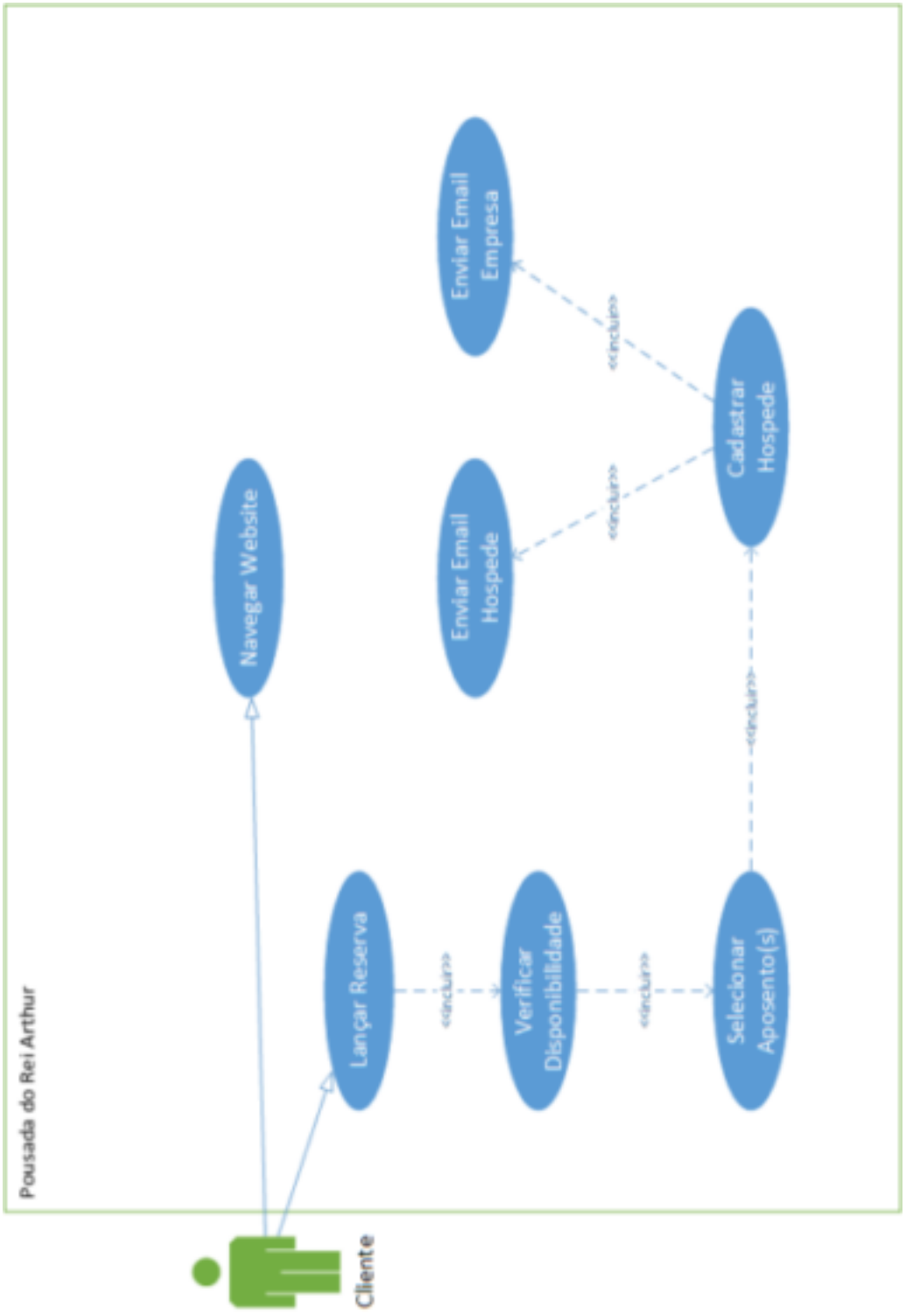
4.2.1 Diagrama de Caso de Uso

Um diagrama de caso de uso exibe um conjunto de casos de uso e atores (um tipo especial de classe) e seus relacionamentos. Eles são utilizados para que se possa identificar como cada um dos elementos do sistema irá se comportar, deixando o sistema de uma forma mais fácil de compreender, pois apresentam uma visão externa de como esses elementos podem ser utilizados no contexto (BOOCH JAMES RUMBAUGH, 2005).

A seguir são mostrados os casos de uso do *website* e do sistema em questão com seus respectivos atores.

4.2.1.1 Cliente

Figura 18 – Diagrama de Caso de Uso para Cliente.



Fonte: Elaborada pelo autor.

Conforme apresentado na Figura 18, existe somente um ator no qual seria um possível hóspede, ele poderá navegar pelo *Website* desenvolvido neste trabalho, conhecendo um pouco mais sobre a empresa, observando suas descrições, aposentos, entre outros pontos.

Outra funcionalidade possível para este ator é o ato de realizar a sua reserva, como descrito no diagrama acima, para que a reserva seja concluída, é preciso passar pelos seguintes etapas: verificar disponibilidade (com a data selecionada), selecionar aposentos (livres), cadastrar hóspede e o envio de e-mail de confirmação.

4.2.1.2 Usuário

A Figura 19 e Figura 20 nos mostram as funcionalidades que o usuário do sistema é capaz de realizar, depois de fazer o *login*, existem várias opções em sua maioria, *CRUD's* de suma importância para o funcionamento da reserva e os principais, verificação de reservas (mapa elaborado) e o fluxo de caixa (lançamentos).

Figura 19 – Diagrama de Caso de Uso para usuário - parte 1.

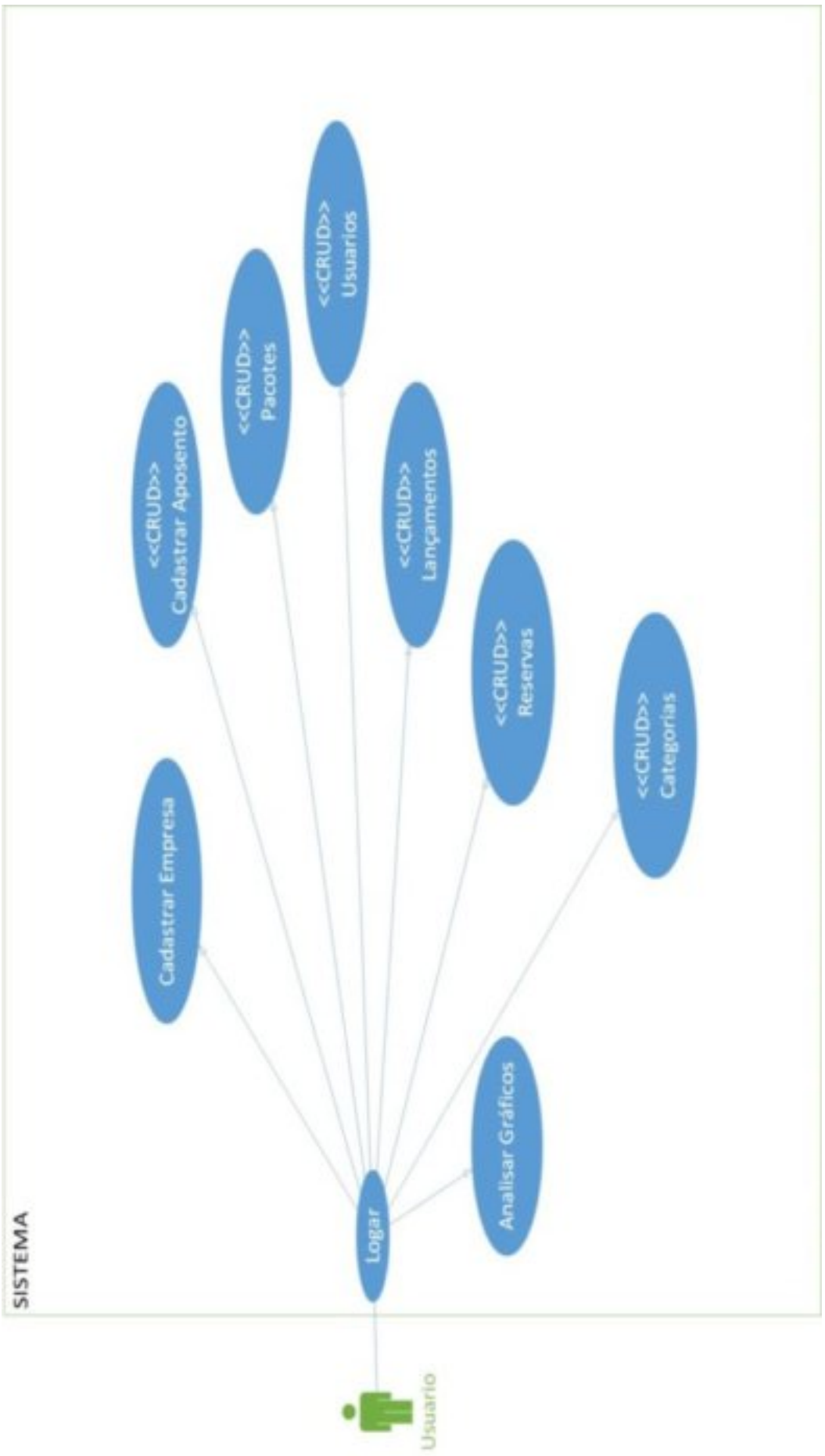
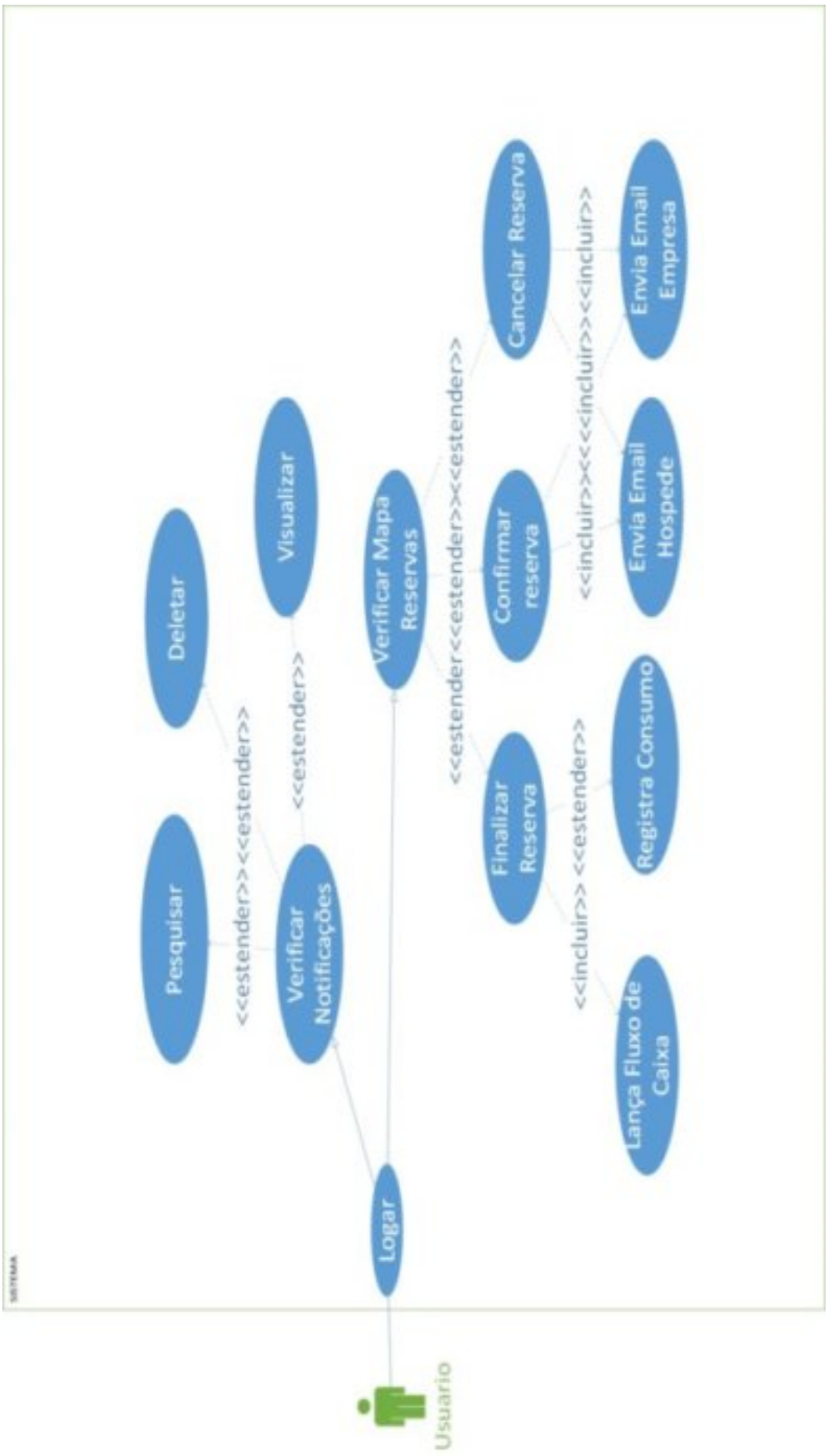


Figura 20 – Diagrama de Caso de Uso para usuário - parte 2.



4.2.2 Caso de Uso

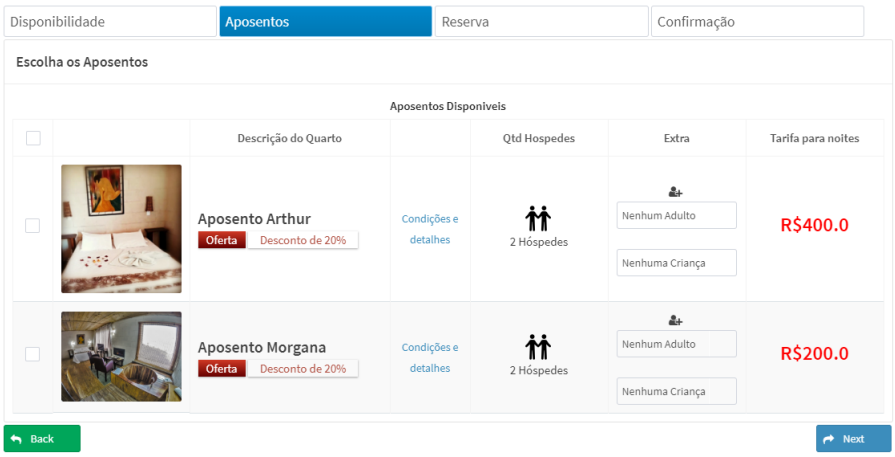
A seguir serão apresentados os principais casos de uso do sistema, com os seus respectivos fluxos principais, os quais serão utilizados para o desenvolvimento.

4.2.2.1 Lançar Reserva (Cliente)

Os hóspedes acessam a página de reservas online, escolhem as datas que entrada e saída, o sistema verifica se existe algum aposento livre e retorna com os mesmos com um pre-cálculo do valo total.Os hóspedes seleciona o (s) aposento (s) desejado e é redirecionado para uma tela de cadastro de hóspedes, logo após ele adicionar os seus dados, é redirecionado para uma página com algumas informações e pedindo para o hóspede confirmar, como mostra a Figura 21.

Após a confirmação o sistema envia e-mails, para o hóspede com os dados para deposito bancário e o valor que precisa ser depositado ou transferido e a data limite para o deposito, a empresa recebe uma confirmação de pre-reserva.

Figura 21 – Lançar reserva.



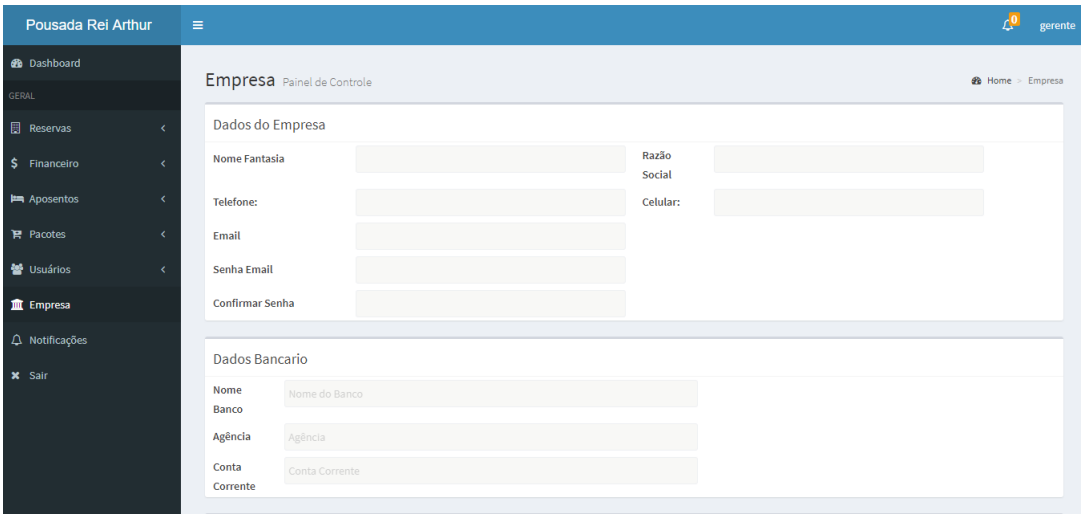
Fonte: Elaborada pelo autor.

Um *link* com a descrição do aposento também é disponibilizado da tela, caso o hóspede clique na imagem do aposento, o mesmo será redirecionada para a página de fotos, para que possa ver melhor os aposentos da empresa.

4.2.2.2 Cadastrar Empresa

O usuário do sistema cadastra os dados da empresa, como razão social, nome fantasia, endereço e também os dados da conta bancária que serão usados no envio de e-mail, e o próprio e-mail e senha para que o sistema consiga enviar os e-mails necessários, como mostra a Figura 22.

Figura 22 – Dados da empresa.

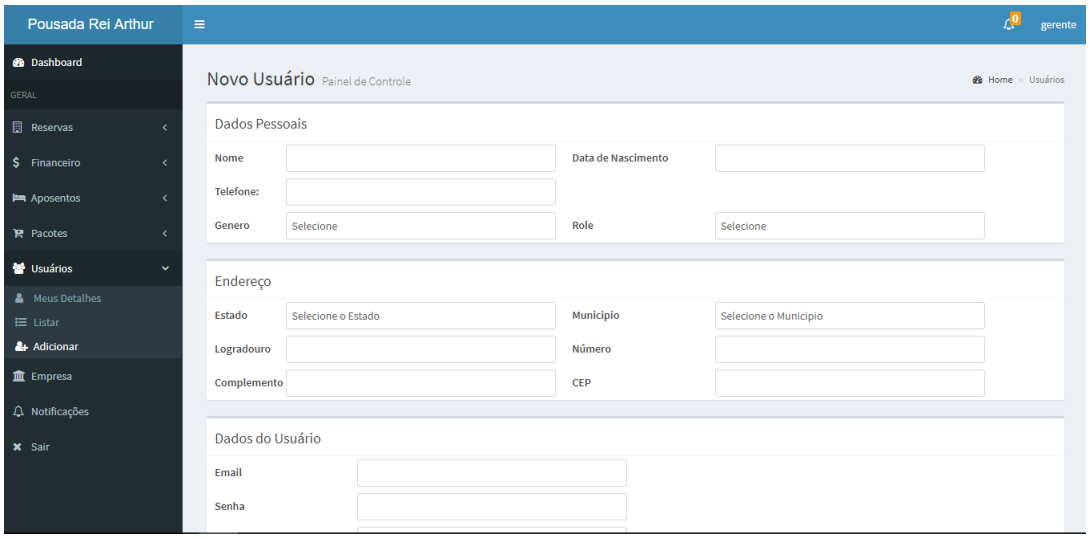


Fonte: Elaborada pelo autor.

4.2.2.3 *CRUD* Usuários

Aqui é possível cadastrar usuários para terem acesso ao sistema através de um e-mail válido e uma senha, os dados pessoais também são necessários. Também é possível excluir, editar e listar os usuários com algum critério de pesquisa ou simplesmente listar todos, como mostra a Figura 23.

Figura 23 – Cadastro de usuários.

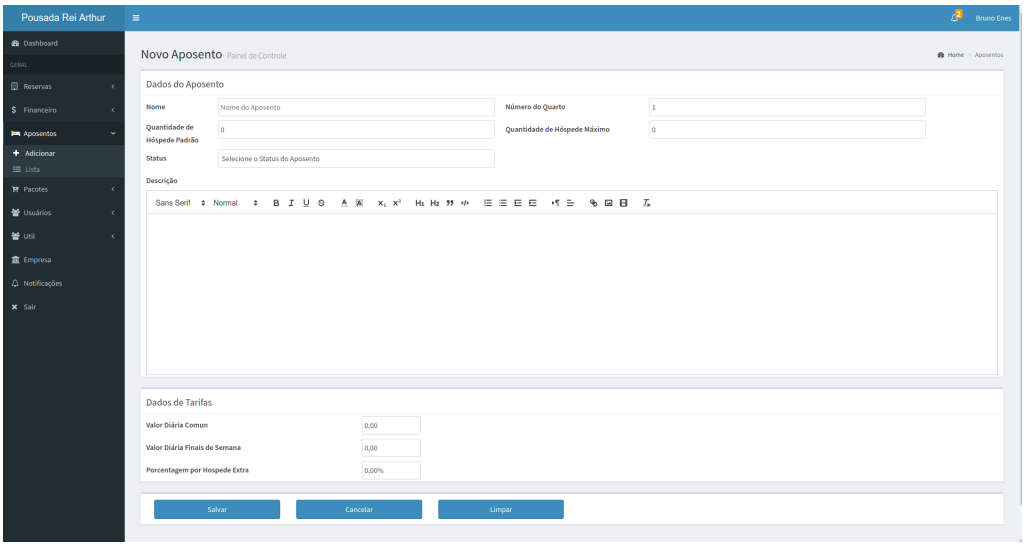


Fonte: Elaborada pelo autor.

4.2.2.4 *CRUD* Aposentos

Nesse *CRUD*, o usuário cadastra os aposentos da empresa com seu nome e número e seu status (operante, em obra, inativo), a quantidade de hóspede padrão, quantidade máxima, sua descrição e seus valores que podem ser o padrão, que valem de domingo a quinta, valores de finais de semana, sexta e sábado, e a porcentagem cobrada por hóspede a mais. É possível deletar, editar e listar os aposentos com critérios de pesquisas ou não, como mostra a Figura 24.

Figura 24 – Cadastrar aposento.



Fonte: Elaborada pelo autor.

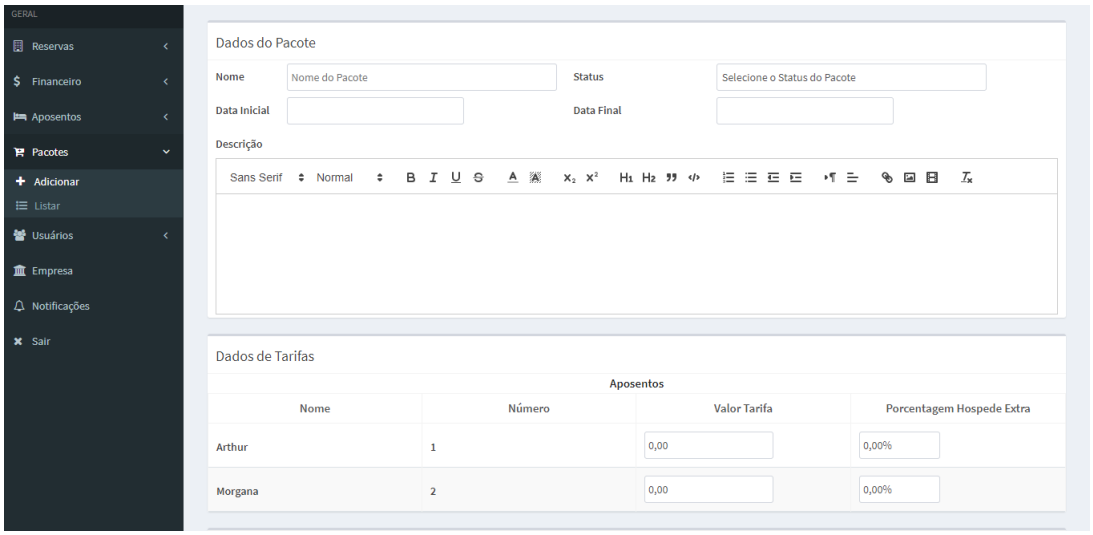
4.2.2.5 *CRUD* Pacotes

Este *CRUD* é para criar um pacote fechado com datas iniciais e finais, é preciso cadastrar um nome, descrição, a data inicial e final, se o pacote está ativo ou não, e um valor, o qual é cadastrado por aposento, é listado os aposentos criados e cada um terá o seu valor especifico. Também é possível deletar, editar e listar os pacotes com critérios de pesquisas ou não, como mostra a Figura 25.

4.2.2.6 *CRUD* Categorias

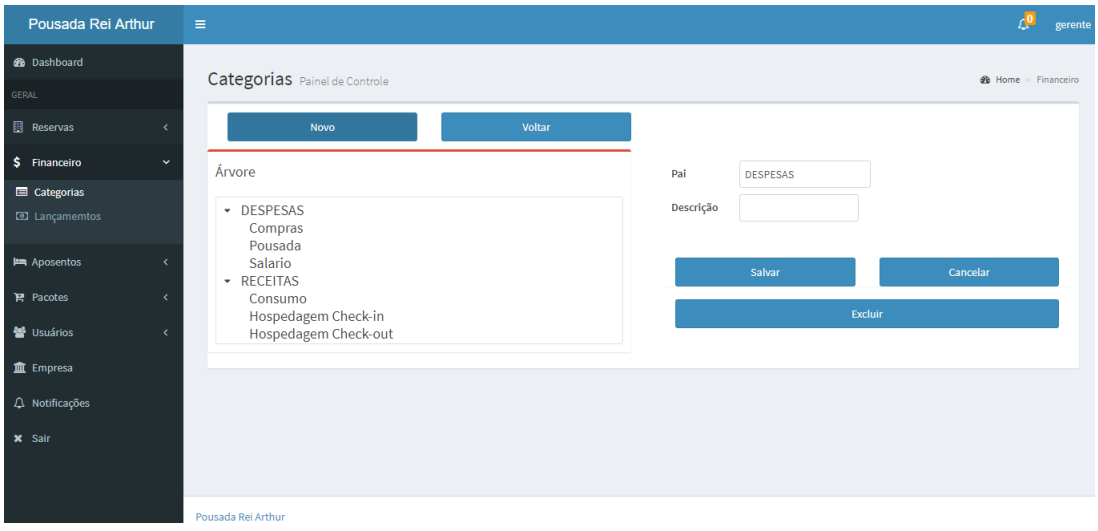
O usuário adiciona categorias para serem usados no fluxo de caixa, que podem ser despesas ou receitas, é necessário ter um nome e um ‘pai’ para essa nova categoria. Quando for lançar uma entrada ou saída no fluxo de caixa, o usuário terá que informar a categoria. Também é possível editar e excluir, como mostra a Figura 26.

Figura 25 – Cadastro de pacote.



Fonte: Elaborada pelo autor.

Figura 26 – Cadastro de categoria.



Fonte: Elaborada pelo autor.

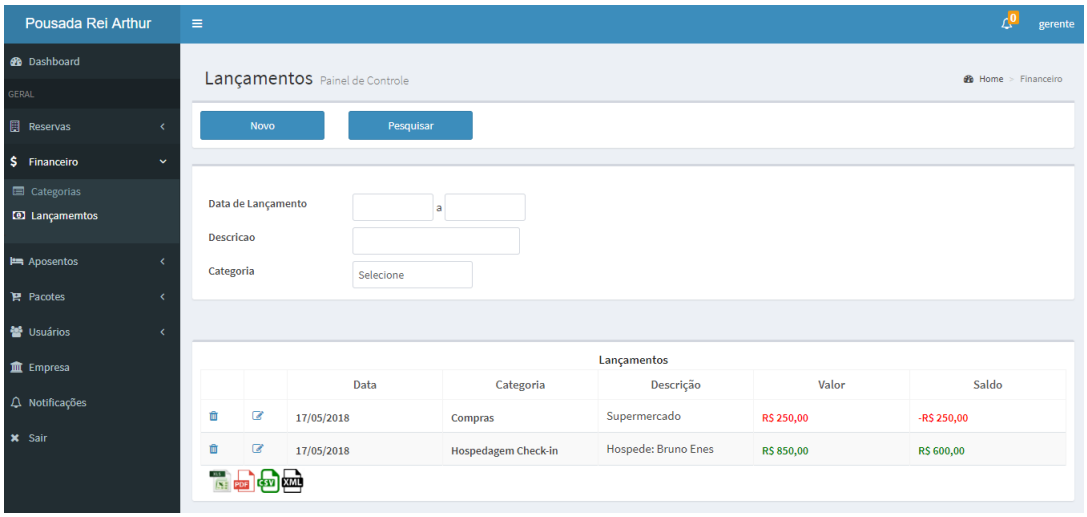
4.2.2.7 *CRUD* Lançamentos

Neste *CRUD* é feito o fluxo de caixa, é listado todas as entradas e saídas realizadas, ordenadas por data de cadastro, a lista contêm ainda a sua categoria, descrição, valor e o saldo total até o memento.

Para adicionar um novo lançamento, é preciso passar a data do evento, categoria, valor e descrição, assim que confirmada, o sistema verifica se é uma despesa ou receita e faz o cálculo no saldo atual da empresa. O usuário também pode editar ou excluir um lançamento, como

mostra a Figura 27.

Figura 27 – Fluxo de caixa.



Fonte: Elaborada pelo autor.

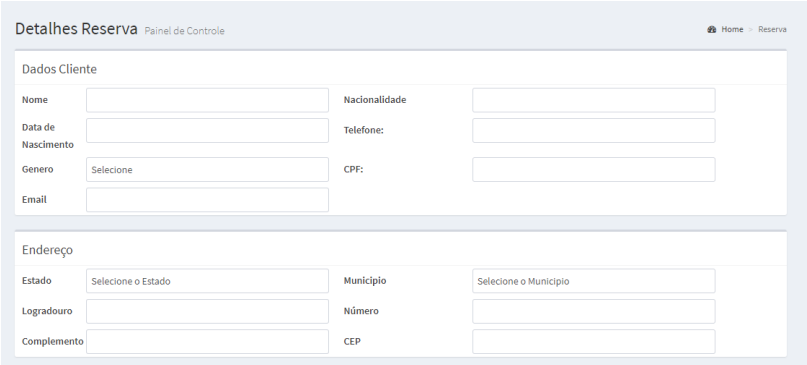
4.2.2.8 *CRUD* Reservas

O sistema de reservas não será realizado somente pelo hóspede, caso seja preciso, o usuário também poderá adicionar uma reserva, principalmente se houver alteração dos valores para uma determinada reserva, cujo os valores sejam diferentes dos que estarão cadastrados no sistema (descontos ou outros imprevistos).

Para que o cadastro de uma nova reserva seja feito, é preciso passar todos dos dados que são pedidos na reserva online, a única diferença é a adição do valor, onde o funcionário que irá calcular, não o sistema, e claro, selecionar o (s) aposento (s) desejado.

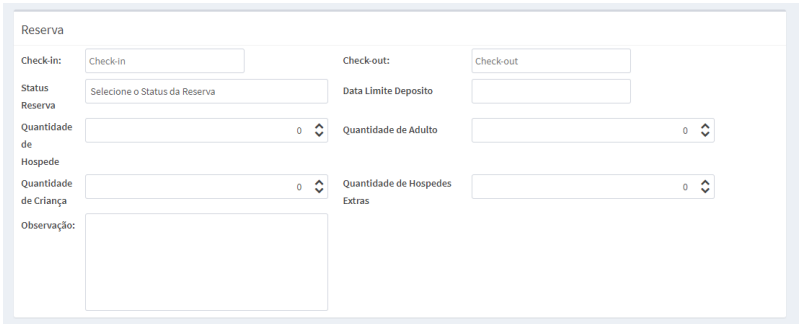
O usuário também poderá excluir e listar uma determinada reserva, de acordo com algum critério ou não, como mostram a Figura 28, Figura 29 e Figura 30.

Figura 28 – Cadastro de nova reserva, dados 1.



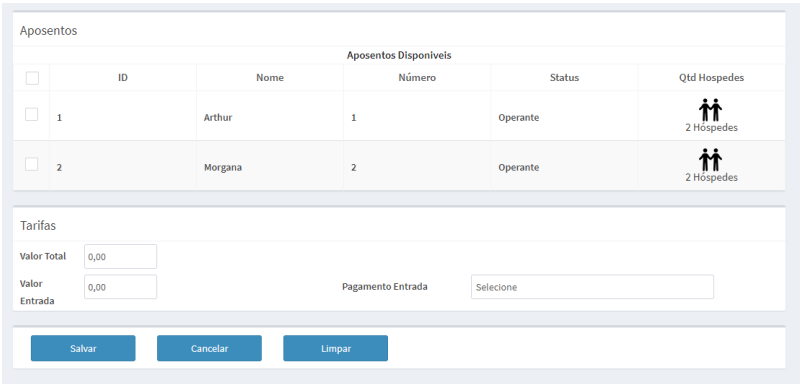
Fonte: Elaborada pelo autor.

Figura 29 – Cadastro de nova reserva, dados 3.



Fonte: Elaborada pelo autor.

Figura 30 – Cadastro de nova reserva, dados 3.



Fonte: Elaborada pelo autor.

4.2.2.9 Verificar Reservas

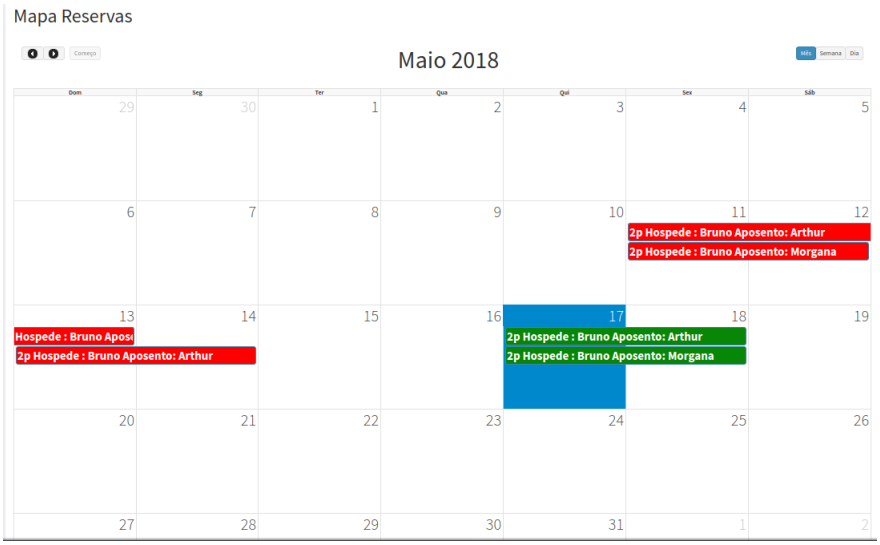
A verificação de reserva é uma das funcionalidades mais importantes desse sistema, é possível ver o mapa de reservas de acordo com algum critério desejado (finalizadas, confirmadas, canceladas ou aguardando pagamento), nessa etapa também que o usuário poderá confirmar, cancelar, editar ou finalizar uma reserva.

Para confirmar uma reserva, o usuário terá que receber o comprovante de deposito ou verificar se existe a transferência para a conta bancária da empresa, casos algum desses itens esteja ok, o usuário acesso ao sistema, seleciona a reserva desejada e faz a confirmação de pagamento, assim que confirmar, o sistema já faz o lançamento no fluxo de caixa como entrada. Logo após a confirmação, o sistema envia um e-mail para o hóspede e para a empresa confirmando a reserva.

No cancelamento, o fluxo de caixa não é alterado, por decisão da empresa, pois na hora do cancelamento pode ser que o dinheiro não seja devolvido no momento exato, assim o fluxo de caixa terá que ser alterado manualmente.

A edição também é fácil de realizar, podendo aumentar os dias que o hóspede deseja ficar, assim o sistema irá calcular o novo valor da reserva em questão. Quando uma reserva for finalizada, é possível adicionar o valor do consumo que o hóspede poderá ter tido, depois de adicionado este valor, o usuário irá selecionar o meio de pagamento do mesmo, caso seja em dinheiro, o sistema irá lançar no fluxo de caixa como entrada colocado na descrição do lançamento o nome do hóspede e qual reserva ele pertence, Figura 31.

Figura 31 – Mapa de reservas.

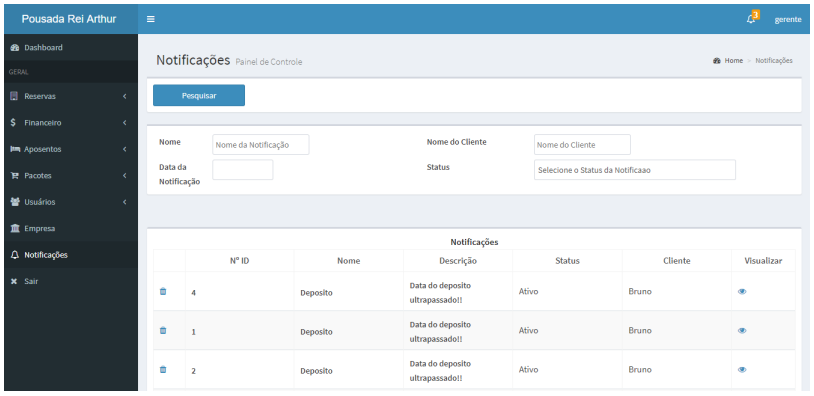


Fonte: Elaborada pelo autor.

4.2.2.10 Verificar Notificações

O usuário verifica as notificações que o sistema cria, essas notificações são criadas quando o sistema verifica que uma determinada reserva não está confirmada e a data limite de deposito já foi ultrapassado. O usuário poderá marcar a notificação como visualizada, excluí-la e pesquisar por algum critério disponível, Figura 33.

Figura 32 – Notificações.

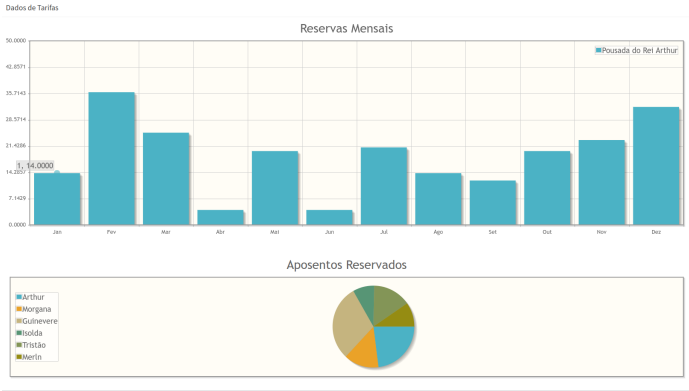


Fonte: Elaborada pelo autor.

4.2.2.11 Gráficos

O usuário poderá verificas alguns gráficos gerados pelo sistema, no atual sistema, os gráficos de quantidade de reservais mensais nos doze meses do ano poderá ser analisada e também qual o aposento está sendo mais alocado atá a data presente da consulta, Figura 33. Para trabalhos futuros, mais gráficos poderão ser criados de acordo com a demanda da empresa.

Figura 33 – Notificações.



Fonte: Elaborada pelo autor.

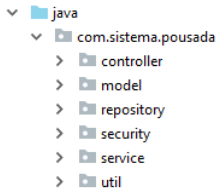
5 DESENVOLVIMENTO

Neste capitulo serão apresentados como foi realizado o desenvolvimento do sistema, a organização do código, a criação do banco de dados com o relacionamento do JPA entre as classes Java, as regras de negócio das principais funcionalidades do sistema e como elas foram implementadas.

5.1 Estrutura das Classes

Como foi utilizado o conceito de orientação a objetos, cada objeto identificado na modelagem do sistema foi implementado como uma classe no projeto, as quais foram organizadas em pacotes, cada uma segundo as suas funções. Ainda seguindo o modelo MVC como explicado anteriormente, temos as separações entre o modelo, a visão e o controle.

Figura 34 – Estrutura das classes Java.



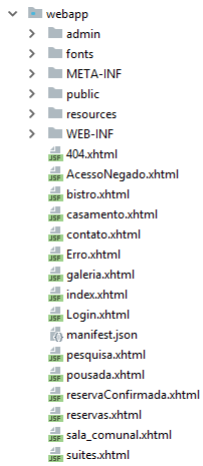
Fonte: Elaborada pelo autor.

Como demonstrado na Figura 34, as classes Java foram divididas em vários pacotes, o *controller*, como o próprio nome diz, é onde encontra os controladores das páginas, qualquer ação do usuário será tratada neste pacote. O pacote *model* é onde se encontra as classes que são 'convertidas' em tabela do banco de dados através do *Hibernate*.

Repository temos o acesso ao banco de dados, a classe desse pacote tem a função de acessar o banco de dados fazendo as consultas necessárias usado *Hibernate* ou *Criteria*. O pacote *security* é realizado todos funções de segurança, tanto para o *login* do usuário do sistema, quanto para a separação de que páginas são seguras ou são acessíveis para todos na Web.

No pacote *service*, são implementadas as classes que intermediam os *controllers* e os *repositories*, nelas são feitas as validações necessárias para salvar, deletar, entre outras funções. O pacote útil contém classes que com funções genéricas, ou seja, funções que podem ser usadas por qualquer *controller* ou *service*, como por exemplo um validador de e-mail, CPF, entre outras.

Figura 35 – Estrutura das páginas $xhtml$.



Fonte: Elaborada pelo autor.

A Figura 35 nos mostra a estrutura das páginas Web, vale ressaltar nesta imagem a pasta admin, nelas se encontram todas a páginas que são usadas no sistema, com os *CRUD's* citados anteriormente. As outras páginas mostradas são as que ficam de acesso livre na Web.

Observa-se que as páginas estão no formato *.xhtml*, isso acontece pelo fato de que ao desenvolvemos aplicações *serve-side* precisamos de uma tecnologia de visualização que seja processada pelo servidor e renderizada em um formato que nosso navegador entende. No JSF 2.0, usamos XHTML como sendo esta tecnologia. Pois ele é um formato que estende o HTML, adicionando regras do XML.

5.2 Criação do Banco de Dados

Como explicado anteriormente, a criação do banco de dados é feita pelo próprio sistema utilizando JPA, no Quadro 1 é mostrado como é feita a conexão com o banco.

Este arquivo de configuração é chamado de *persistence.xml*, é nele que ocorre a conexão com o banco de dados, na primeira linha é atribuído um nome para ele, ReiArthur, isto serve para identificar na classe Java de conexão.

As primeiras propriedades são sobre qual banco de dados está sendo utilizado, como pode observar, está sendo utilizado *MySQL*, então passamos o *schema*, usuário, senha e qual o driver de conexão. Abaixo está configurado algumas propriedades do *Hiberante*, como *debug*, entre outras.

Outra propriedade importante é a do c3p0, que é um *pool* de conexão, ele trabalha controlando o acesso múltiplo ao banco de dados.

Quadro 1 – Conexão com o banco de dados.

```
<persistence-unit name="ReiArthur">
   <!-- provedor/implementacao do JPA -->
   <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

   <properties>
      <!-- dados da conexao -->
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost/reiarthur" />
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password" value="root" />
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"
/>

      <!-- propriedades do hibernate -->
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect" />
      <property name="hibernate.connection.provider_class"
            value="org.hibernate.connection.C3P0ConnectionProvider" />

      <!-- controla multiplos acesso -->
      <property name="hibernate.c3p0.max_size" value="20" />
      <property name="hibernate.c3p0.min_size" value="5" />
      <property name="hibernate.c3p0.acquire_increment" value="1" />
      <property name="hibernate.c3p0.idle_test_period" value="300" />
      <property name="hibernate.c3p0.max_statements" value="50" />
      <property name="hibernate.c3p0.timeout" value="300" />
   </properties>
</persistence-unit>
```

Fonte: Elaboração do autor.

Para que o JPA leia o nosso *persistence.xml*, usamos a classe *EntityManagerFactory* como mostra o Quadro 2.

Quadro 2 – Classe Java para conexão com o banco de dados.

```
@ApplicationScoped
public class EntityManagerProducer {

        private EntityManagerFactory factory;

        private static EntityManagerProducer instance;

        public EntityManagerProducer() {
                factory = Persistence.createEntityManagerFactory("ReiArthur");
        }

        @Produces @RequestScoped
        public EntityManager createEntityManager() {
                return factory.createEntityManager();
        }

        public void closeEntityManager(@Disposes EntityManager manager) {
                manager.close();
        }

        public EntityManager getEntityManager(){
           return factory.createEntityManager();
    }

    /** Verifica se já existe uma instancia do objeto */
    public static synchronized EntityManagerProducer getInstance(){
            if (instance == null)
               instance = new EntityManagerProducer();


return instance;
    }
}
```

Fonte: Elaborada pelo autor.

É criado um *EntityManagerFactory* com o nome de *factory* através do nome do arquivo *persistence.xml*, assim já está feita a conexão. Para a criação de uma tabela, foi usado as anotações do *Hibernate*, o Quadro 3 mostra um exemplo de classe com as anotações necessárias.

Quadro 3 – Exemplos anotações $Hibernate$

```
@Entity
@Table
public class Banco  implements Serializable {

   @Id
   @GeneratedValue
   private Long id;

   @NotNull
   @Column(nullable = false, length = 200)
   private String nome;

   @NotNull
   @Column(nullable = false, length = 30)
   private String contaCorrente;

   @NotNull
   @Column(nullable = false, length = 30)
   private String agencia;

   //getters e setters omitidos
```

Fonte: Elaboração do autor.

As anotações *@Entity* e *@Table* do *Hiberante* irá mostrar ao JPA que aquela classe ira tornar uma tabela no banco de dados, cada atributo da classe vira um atributo da tabela.

A anotação *@Id* é uma obrigatoriedade do *Hibernate*, logo após temos a anotação *@GeneratedValue*, ela é utilizada para informar que a geração do valor do identificador único da entidade será gerenciada pelo provedor de persistência.

Vários outros atributos foram utilizados no desenvolvimento do sistema, como mostrado no tópico sobre o *Hibernate*, o mesmo foi um *framework* de sua importância para este projeto, ajudando principalmente na agilidade de codificação.

5.3 Fluxo de Caixa

5.3.1 Categorias

Em um aplicativo financeiro, as categorias servem para classificar os lançamentos financeiros que forem realizados. Podemos entender como lançamento financeiro qualquer registro de despesas ou receita que se deseje registra o sistema.

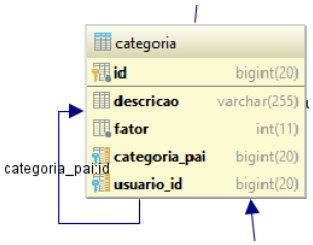
No sistema desenvolvido, as categorias foram organizadas em uma árvore, expressando subdivisões das categorias. As principais categorias são DESPESAS e RECEITAS.

Este cadastro de categorias é um dos mais complexos do sistema, pois, além de trabalhar com entidades autor referenciadas no *Hibernate*, usa também o componente *Tree* do *Primefaces*.

5.3.1.1 Modelo

Um dos pontos que torna o cadastro de categorias muito interessante é a camada de acesso a dados. Foi construído uma entidade autor referenciada, ou seja, uma tabela com um relacionamento consigo mesmo. Isso é necessário para construir a estrutura de árvore para as categorias, em que uma categoria sempre indique sua categoria pai. A Figura 36 mostra como ficou a tabela.

Figura 36 – Diagrama da tabela categoria.



Fonte: Elaborada pelo autor.

As informações básicas para esta tabela estão no código, descrição e usuário, onde cada vez que seja criado uma nova categoria, o usuário *logado* no sistema será identificado e salvo na tabela. O campo *categoria_pai* é responsável por montar a estrutura da árvore das categorias cadastradas, pois cada categoria pertencerá a uma categoria pai.

O campo fator será responsável por diferenciar as categorias que estejam abaixo das DESPESES ou RECEITAS. Dessa forma, o usuário informa o valor nominal da operação, e o que determinará se este valor é positivo ou negativo será o fator. Este terá valor 1 para RECEITA e -1 para DESPESA. No memento do cadastro de uma categoria, o valor do fator será sempre herdado da categoria acima.

O Quadro 4 mostra como ficou a classe Categoria com as anotações do *Hibernate*, assim, ela se tornará uma tabela no banco de dados. O primeiro atributo se refere a autor referência citada anteriormente, esse atributo poderá ser nulo, pois obrigatoriamente, o primeiro nível da árvore de categorias (RECEITAS e DESPESAS) não terá pai.

Além de referenciar a categoria pai, uma instancia de Categoria também deverá referenciar seus próprios filhos (atributo filhos). Utilizaremos um mapeamento *@OneToMany* para fazer esta carga da lista das categorias filhas. Como na maioria das vezes mostraremos a estrutura completa da árvore em tela, usaremos a anotação fetch igual a *FetchType.EARGE*, isso faz com que seja buscado uma carga imediata dos filhos no momento de uma consulta.

Também usamos a anotação *cascade = CascadeType.Remove*, essa configuração removerá os filhos e netos caso a categoria seja excluída. Está sendo utilizado o atributo *update = false*, o que garante que uma atualização em uma categoria não afete os seus filhos.

Quadro 4 – Modelo categoria.

```
@Entity
@Table
public class Categoria implements Serializable {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;


 @ManyToOne
   @JoinColumn(name = "categoria_pai", nullable = true, foreignKey =
@ForeignKey(name = "fk_categoria_categoria"))
   private Categoria pai;

   @ManyToOne
   @JoinColumn(foreignKey = @ForeignKey(name = "fk_categoria_usuario"))
   @OnDelete(action = OnDeleteAction.CASCADE)
   private Usuario usuario;

   private String descricao;

   private int fator;

   @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.REMOVE)
   @JoinColumn(name = "categoria_pai", updatable = false)
   @org.hibernate.annotations.OrderBy(clause = "descricao asc")
   private List<Categoria> filhos;

   //getters e setters omitidos
```

Fonte: Elaboração do autor.

A anotação *@OrderBy* é especifica do *Hibernate* e permite configurar a ordenação para a carga de seus filhos. Considere o valor para o atributo clause como se fosse exatamente o *order by* a ser utilizado em uma SQL padrão.

5.3.1.2 Regras de Negócios

As regras de negócios para categoria contêm algumas especificações, veremos alguns métodos relevante.

O Quadro 5 apresenta o método salvar categoria, o primeiro teste verifica se a categoria que está sendo salva tem um pai, sabemos que as categorias DESPESA e RECEITA não tem pai, pois estão no topo da hierarquia, por decisão de projeto juntamente com os representantes da empresa, ficou decidido que somente essas duas categorias ficariam no topo, quaisquer outra que fosse criada teria que ter um pai.

Ainda no método salvar, verificamos se o fator da categoria atual mudou, esse fator de multiplicação sempre será obtido da categoria pai na qual a categoria em questão foi criada. Ou seja, se a categoria for criada sob DESPESA, seu fator será -1, se for criada sob RECEITA, será 1.

A transferência do fator de pai para o filho é feita sempre antes de salvar a categoria. Pode ocorrer também, contudo, a alteração de um ramo de categorias RECEITS para DESPESES ou vice-versa. Neste caso, será preciso mudar o fator da categoria atual e de todos os seis filhos e netos. Como esse é um processo que pode ser pesado para o sistema, somente o re-

Quadro 5 – Salvar categoria.

```
public Categoria save(Categoria categoria) throws CategoriaException {

    Categoria pai = categoria.getPai();

    if(pai == null){
        String msg = "A Categoria " + categoria.getDescricao() +
                "deve ter um pai definido";
        throw new CategoriaException(msg);

    }

    boolean mudouFator = pai.getFator() != categoria.getFator();

    categoria.setFator(pai.getFator());

    categoria = this.categoriaRepository.saveOrUpdate(categoria);

    if(mudouFator){
        categoria = this.findById(categoria.getId());
        this.replicaFator(categoria,categoria.getFator());
    }

    return categoria;
}
```

Fonte: Elaboração do autor.

alizaremos se 'mudoufator' for true. Além disso, o objeto Categoria que recebemos não tem seus filhos carregados, pois veio do formulário, de modo que, antes de seguir adiante temos que carregar toda a sua estrutura.

Logo em seguida usamos o método '*replicaFator*' como mostra o Quadro 6, para repassar essa mudança de fator para todas as categorias da hierarquia abaixo da categoria alterada.

Temos ainda os outros métodos, como deletar, listar com filtros, listar todos, mas estes métodos não têm nenhuma regra de negócios especifica.

5.3.1.3 Visão

Nesta camada, integraremos o componente *Tree* do *PrimeFaces* e os recursos *Ajax* com toda a estrutura montada até agora.

Primeiramente vamos ver como ficou a classe *CategoriaController*.

O Quadro 7 mostra as declarações que serão usadas na página *categoria.xhtml*, sendo *TreeNode categoriasTree* a árvore de categorias, Categoria editada o objeto que alimentará o formulário e receberá a categoria selecionada na árvore, e *List categoriasSelect* a lista que alimentará a caixa de seleção de categorias pai. Além disso temos o *mostraEdicao* e *mostraExcluir*, pois os formulários de edição e exclusão só serão mostrados caso alguma categoria seja selecionada. Para que esses atributos possam ser acessados pela página, é preciso gerar os seus getters e setters.

Assim que o usuário selecionar a página, este controller se inicia e faz o carregamento da hierarquia como mostra o Quadro 8.

Quadro 6 – Método para replicar fator.

```
<persistence-unit name="ReiArthur">
   <!-- provedor/implementacao do JPA -->
   <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

   <properties>
      <!-- dados da conexao -->
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost/reiarthur" />
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password" value="root" />
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"
/>

      <!-- propriedades do hibernate -->
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect" />
      <property name="hibernate.connection.provider_class"
            value="org.hibernate.connection.C3P0ConnectionProvider" />

      <!-- controla multiplos acesso -->
      <property name="hibernate.c3p0.max_size" value="20" />
      <property name="hibernate.c3p0.min_size" value="5" />
      <property name="hibernate.c3p0.acquire_increment" value="1" />
      <property name="hibernate.c3p0.idle_test_period" value="300" />
      <property name="hibernate.c3p0.max_statements" value="50" />
      <property name="hibernate.c3p0.timeout" value="300" />
   </properties>
</persistence-unit>
```

Fonte: Elaboração do autor.

Quadro 7 – Declarações $CategoriaController$.

```
public class CategoriaController {

   private TreeNode categoriasTree;

   private TreeNode selectNode;

   private Categoria editada;

   private Usuario;

   private CategoriaService;

   private SecurityBean;

   private List<SelectItem> categoriasSelect;

   private boolean mostraEdicao = false;

   private boolean mostraExcluir = false;

   @Inject
   public CategoriaController(Categoria editada, CategoriaService, SecurityBean
securityBean) {
      this.editada = editada;
      this.categoriaService = categoriaService;
      this.securityBean = securityBean;
   }
```

Fonte: Elaboração do autor.

A anotação @*PostConstruct* indique que assim que o construtor da classe for executado, está função será executada, primeiramente é pegado o usuário logado no sistema, logo após são chamadas duas funções, a primeira, carregaCategoria, é encarregada de montar a estrutura

Quadro 8 – Iniciando o $controller$.

```
@PostConstruct
public void inicializa(){
    usuario = securityBean.getUsuerLogged().getUsuario();
    this.carregaCategorias();
    this.carregaCategoriasSelect();
}

private void carregaCategorias() {
    if(this.categoriasTree == null){
        List<Categoria> categorias = this.categoriaService.lista();
        this.categoriasTree = new DefaultTreeNode(null,null);
        this.montaDadosTree(this.categoriasTree,categorias);
    }
}
```

Fonte: Elaboração do autor.

hierárquica, ela recebe uma lista de categorias do banco de dados. Para transformar a estrutura hierárquica de *Categoria* em uma estrutura *TreeNode* foi construído o método *montadadosTree()* como mostra o Quadro 9.

Quadro 9 – Montando a hierarquia $TreeNode$.

```
private void montaDadosTree(TreeNode pai, List<Categoria> lista) {

    if(lista != null && lista.size() > 0){
        TreeNode filho;

        for(Categoria: lista){
            filho = new DefaultTreeNode(categoria,pai);
            this.montaDadosTree(filho,categoria.getFilhos());
        }
    }
}
```

Fonte: Elaboração do autor.

Este método tem um funcionamento recursivo para realizar essa operação, ou seja, o método percorre todas as categorias e chama a si mesmo cada vez que for necessário descer um nível na hierarquia.

Voltado ao Quadro 8, assim que a função *carregaCategorias* é finalizada, é chamada a função *carregaCategoriasSelectec*, Quadro 10.

Este método é semelhante ao *carregaCategorias*, porém este gera uma lista plana de categorias do banco de dados. Como o componente da caixa de seleção não mostra dados em estrutura de árvore, utilizamos o método *montaDadosSelect* par montar uma lista de *SelectItem*, como mostra o Quadro 11, esta lista será usada toda vez que se desejar criar uma nova categoria, assim será selecionado a categoria pai, ou simplesmente a edição de uma categoria.

Quadro 10 – Categorias Selecionadas.

```
private void carregaCategoriasSelect(){
   if(this.categoriasSelect == null){
      this.categoriasSelect = new ArrayList<>();
      List<Categoria> categorias = categoriaService.lista();
      this.montaDadosSelect(this.categoriasSelect,categorias,"");
   }
}
```

Fonte: Elaboração do autor.

Quadro 11 – Categorias para caixa de seleção.

```
private void montaDadosSelect(List<SelectItem> select, List<Categoria> categorias,
String prefixo) {

   SelectItem item;
   if(categorias != null){
      for(Categoria  : categorias){
         item = new SelectItem(categoria,prefixo + categoria.getDescricao());
         item.setEscape(false);
         select.add(item);
         this.montaDadosSelect(select,categoria.getFilhos(), prefixo);
      }
   }
}
```

Fonte: Elaboração do autor.

A função salvar é relativamente simples, como se vê no Quadro 12, primeiramente setamos o usuário que está editando ou criando uma nova e realizamos o salvamento, se for concluída ou não, o usuário irá recebe rum feedback.

Quadro 12 – Categorias para caixa de seleção.

```
public void salvar(){
   this.editada.setUsuario(usuario);
   try {
      categoriaService.save(this.editada);
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO,
MessageUtil.REGISTRO_SALVO,null));
   } catch (CategoriaException e) {
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO,
e.getMessage(),null));
   }

   this.limpaCampos();
   this.carregaCategorias();
   this.carregaCategoriasSelect();
}
```

Fonte: Elaboração do autor.

Na página *categoria.xhtml* temos o componente *Tree* do *Primefaces* declarado, como mostra o Quadro 13. O atributo value é exatamente a hierarquia que será mostrado(Figura 26),

treeNode é exibido o nome da categoria. No evento *ajax* e acionado assim que alguma categoria for selecionada, assim, a categoria é carregada para o objeto no *controller* e exibido o formulário de edição. Caso a categoria selecionada não tenha pai, ou seja, ela está no mais auto nível da hierarquia, o formulário não será renderizado, assim finaliza o *CRUD* de categorias.

Quadro 13 – Componente $tree$.

```
<p:tree id="treeCategorias"
      value="#{categoriaController.categoriasTree}"
      var="node" dynamic="false"
      cache="false" selectionMode="single"
      selection="#{categoriaController.selectNode}"
      animate="true" style="font-size: 18px">

   <p:ajax event="select" update=":edicao, :btnCategorias"
listener="#{categoriaController.selecionar}" />

   <p:treeNode>
      <h:outputText value="#{node.descricao}" />
   </p:treeNode>

</p:tree>
```

Fonte: Elaboração do autor.

5.3.2 Lançamentos

Nesta parte, o fluxo de caixa será de fato realizado, depois de termos construído toda a parte de categorias, iremos realizar os lançamentos no sistema.

5.3.2.1 Modelo

O modelo do lançamento é mais simples que o de categoria, como mostra o Quadro 14.

Como é realizado na categoria, o lançamento também está relacionado a tabela usuário para identificação de quem fez o lançamento, pelo fato de que vários usuários têm acesso ao sistema. Também temos a tabela categoria relacionada, a qual foi explicado a pouco, um atributo data com uma anotação *@Temporal(TemporalType.DATE)*, essa anotação diz que somente a data será salva no banco de dados, ignorando horas, minutos e segundos, uma descrição para o lançamento e o valor do mesmo.

5.3.2.2 Regras de Negócios e Visão

Para que o lançamento fosse exibido, foi criando somente uma página que contêm um componente do *PrimeFaces* chamado *dataTable*, este componente recebe uma lista de um objeto qualquer e a percorre exibindo os seus dados de acordo com o atributo desejado pelo desenvolvedor.

O *dataTable* criado foi para uma lista de lançamentos, mas como podemos ver no modelo (Quadro 14), a objeto não possui um atributo saldo, o qual terá de ser calculado em outra

Quadro 14 – Modelo lançamento.

```
@Entity
@Table
public class Lancamento implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JoinColumn(nullable = false, foreignKey = @ForeignKey(name =
"fk_lancamento_usuario"))
    private Usuario;

    @ManyToOne(fetch = FetchType.LAZY)
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JoinColumn(nullable = false, foreignKey = @ForeignKey(name =
"fk_lancamento_categoria"))
    private Categoria;

    @Temporal(TemporalType.DATE)
    private Date data;

    private String descricao;

    private BigDecimal valor;

    //getters e setters omitidos
```

Fonte: Elaboração do autor.

lista e exibido juntamente com os lançamentos. O Quadro 15 mostra as declarações que será utilizada pelo *LancamentoController*.

Quadro 15 – $LancamentoController$.

```
public class LancamentoController {

    private List<Lancamento> lista;

    private List<Double> saldos;

    private float saldoGeral;

    private Usuario usuario;

    private Lancamento editado;

    private SecurityBean securityBean;

    private LancamentoService lancamentoService;

    @Inject
    private LancamentoFilter filtro;

    @Inject
    public LancamentoController(Lancamento editado, SecurityBean securityBean,
LancamentoService lancamentoService) {
        this.editado = editado;
        this.securityBean = securityBean;
        this.lancamentoService = lancamentoService;

        filtro = new LancamentoFilter();
    }
```

Fonte: Elaboração do autor.

O primeiro atributo se refere a lista de lançamentos cadastradas no banco de dados, a

lista ‘saldos’ irá conter os saldos de acordo com a lista de lançamentos, ‘saldoGeral’ é o cálculo total do saldo do fluxo de caixa. Também temos o usuário logado no momento e um objeto Lançamento denominado editado, ele será usado toda vez que for criado um novo lançamento ou quando o usuário for editar um lançamento existente.

Como explicado anteriormente, o fator que irá calcular se um lançamento é de entrada ou saída está em sua categoria, para isso, teremos que recuperar a categoria do lançamento e verificar o seu fator, como mostra o Quadro 16.

Quadro 16 – Carrega lista de lançamentos.

```
public void carregaLista() {
   // para carregar ate 30 dias antes
   Calendar dataSaldo = new GregorianCalendar();
   dataSaldo.add(Calendar.MONTH, - 1);
   dataSaldo.add(Calendar.DAY_OF_MONTH, - 1);

   //recebe o saldo total
   this.saldoGeral = lancamentoService.saldo(dataSaldo.getTime());
   this.lista = lancamentoService.listar(filtro);

   Categoria categoria;
   double saldo = this.saldoGeral;
   this.saldos = new ArrayList<>();

   // calcula o saldo por lancamento
   for(Lancamento lancamento : lista){

      categoria = lancamento.getCategoria();
      saldo = saldo + (lancamento.getValor().floatValue() * categoria.getFator());
      this.saldos.add(saldo);

   }
}
```

Fonte: Elaboração do autor.

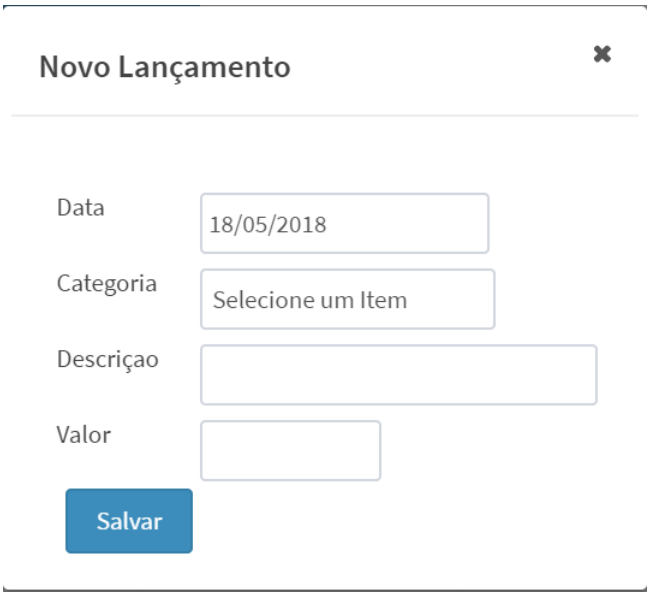
A classe *Calendar* utilizada é uma versão mais moderna do *java.util.Date*, a grande vantagem é que ela possui diversos métodos para manipulação de data, incluindo adição e subtração de segundos, anos, entre outros.

No método, temos uma data criada e utilizada internamente, dataSaldo. Por definição, determinamos que a tela de lançamentos só mostrará os lançamentos dos últimos 30 dias e os lançamentos futuros.

A variável saldoGeral por fim busca o saldo através da data passada como parâmetro e logo após é calculado o saldo por lançamento no comando for utilizando o fator de cada categoria. Os lançamentos podem ser conferidos na Figura 27. Caso seja preciso alterar a data de pesquisa, existe o filtro de datas, onde o usuário escolhe as datas que lhe interessarem.

Para salvar um novo lançamento, o usuário terá que selecionar a categoria que o lançamento pertence, Figura 37.

Figura 37 – Novo lançamento.



Fonte: Elaboração do autor.

A data, descrição e valor também são obrigatórios para que o lançamento possa ser salvo. O Quadro 17 mostra a função salvar, onde ocorrerá um erro se o usuário tentar salvar um lançamento com a categoria DESPESES ou RECEITAS selecionadas.

Quadro 17 – Salvando um lançamento.

```
public void salvar(){
   //usuario logado
   this.editado.setUsuario(usuario);

   //verifica se esta salvando com categorias do topo da arvore
   if(this.editado.getCategoria().getDescricao().equals("DESPESAS") ||
         this.editado.getCategoria().getDescricao().equals("RECEITAS")){
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR, "Selecione um
Tipo de Despesa ou Receita",null));
      return;
   }

   try {
      lancamentoService.save(this.editado);
      //limpa os dados
      this.novo();
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO,
MessageUtil.REGISTRO_SALVO,null));
      RequestContext context = RequestContext.getCurrentInstance();
      context.addCallbackParam("sucesso", true);
   } catch (LancamentoException e) {
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO,
e.getMessage(),null));
   }
}
```

Fonte: Elaboração do autor.

5.4 Reservas

Antes de mostrar como funciona as reservas, é importante explicar as regras de negócios para que uma reserva seja consolidada, regras essas obtidas com os representantes da empresa.

Em primeiro lugar, existem dois tipos de reservas, a reserva comum, que contêm valores para dias de semana e finais de semana, e a reserva por pacote, com valores fechados. Qualquer uma dessas reservas tem valores diferentes para cada aposento, apesar de que na empresa atual, os aposentos são divididos por andares (dois por andar) e os valores são iguais por andar, foi desenvolvido assim para que caso haja algum aumento de aposento em um local diferente, ele possa ser calculado separadamente.

Os pacotes têm os seus dias estabelecidos, caso haja alguma tentativa de reserva que pegue algum dia desse pacote, ou se o hóspede desejar ficar dias a mais do que estabelecido no mesmo, será exibido uma mensagem para o cliente pedindo para entrar em contato com os funcionários da empresa.

Qualquer reserva precisa de no mínimo 50% de entrada para ser confirmada, esse valor pode ser transferido ou depositado na conta bancária da empresa, o cliente tem até 24h para pagar este valor, tudo devidamente explicado no site desenvolvido.

A confirmação da reserva se dá quando o funcionário responsável pelas reservas recebe um e-mail com o comprovante de deposito ou transferência, assim ele confirma no sistema a reserva.

Caso a confirmação não aconteça dentre as 24h, foi usado um componente do *PrimeFaces* denominado *poll* que em um determinado período de tempo, determinado pelo desenvolvedor do sistema, irá acessar uma função, a mesma verifica no sistema se alguma reserva passou a data limite de deposito (campo na tabela de reserva) e que não esteja com o status de confirmada.

Assim, o sistema irá enviar um e-mail para o usuário constatando o ocorrido e cria uma notificação no sistema, o usuário por sua vez toma as decisões que forem cabíveis.

Existem várias etapas a serem verificadas para que uma reserva seja realizada pelo cliente, como verificar as datas, valores, aposentos, entre outros, essas verificações são ignoradas quando a reserva é realizada pelo usuário em uma página especifica do sistema. Isso ocorre pelo fato de acontecerem descontos ou até mesmo datas que estão em pacotes, mas poderão ser diminuídas ou aumentadas para o cliente em questão.

Mesmo assim, depois de serem confirmadas, quando outro cliente for realizar uma reserva, o sistema pegará essas reservas feitas pelo usuário como qualquer outra, barrando o quarto em uso, entre outros.

5.4.1 Modelo

O Quadro 18 e Quadro 19 exibem como ficou o modelo da reserva que será salvo como tabela.

Quadro 18 – Modelo reserva - parte 1.

```
public class Reserva implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(columnDefinition = "text")
    private String observacao;

    @ManyToOne
    @JoinColumn(name = "usuario_id")
    private Usuario usuario;

    @OneToOne(cascade = {CascadeType.ALL, CascadeType.REMOVE})
    @JoinColumn(name = "cliente_id")
    private Cliente cliente;

    @NotNull
    @ManyToMany
        @JoinTable(name = "reserva_aposento", joinColumns = @JoinColumn(name =
"reserva_id"), inverseJoinColumns = @JoinColumn(name = "aposento_id"))
    private List<Aposento> aposentos = new ArrayList<>();

    @ManyToOne
    @JoinColumn(name = "pacote_id")
    private Pacote pacote;

    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    private StatusReserva statusReserva;

    @NotNull
    @Column(precision = 10, scale = 2)
    private BigDecimal valorTotal = BigDecimal.ZERO;

    @Column(precision = 10, scale = 2)
    private BigDecimal valorTotalcomConsumo = BigDecimal.ZERO;

    @NotNull
    @Column(precision = 10, scale = 2)
    private BigDecimal valorEntrada = BigDecimal.ZERO;

    @NotNull
    @Column(precision = 10, scale = 2)
    private BigDecimal valorDepositado = BigDecimal.ZERO;
```

Fonte: Elaboração do autor.

Quadro 19 – Modelo reserva - parte 1.

```
    @NotNull
    @Column(precision = 10, scale = 2)
    private BigDecimal valorRestante = BigDecimal.ZERO;

    @Column(precision = 10, scale = 2)
    private BigDecimal valorConsumo = BigDecimal.ZERO;

    @Enumerated(EnumType.STRING)
    @Column(length = 20)
                    private FormaPagamento formaPagamentoEntrada =
FormaPagamento.DEPOSITO_BANCARIO;

    @Enumerated(EnumType.STRING)
    @Column( length = 20)
    private FormaPagamento formaPagamentoSaida;

    @Column(nullable = false)
    private int qtdHospede;

    @Column(nullable = false)
    private int qtdHospedeExtra;

    @Column(nullable = false)
    private int qtdAdulto;

    @Column(nullable = false)
    private int qtdCrianca;

    @NotNull
    @Column(nullable = false)
    private int qtdDias;

    @NotNull
    @Temporal(TemporalType.DATE)
    private Date dataReserva;

    @NotNull
    @Temporal(TemporalType.TIMESTAMP)
    private Date dataEntrada;

    @NotNull
    @Temporal(TemporalType.TIMESTAMP)
    private Date dataSaida;

    @NotNull
    @Temporal(TemporalType.DATE)
    private Date dataLimiteDeposito;

    @Column
    private int diasParaEntrada;
```

Fonte: Elaboração do autor.

São vários os atributos para que uma reserva seja confirmada, os principais são os valores que serão cobrados, valor total, consumo, entrada e restante, também vale ressaltar as datas de entrada, saída, de limite de deposito.

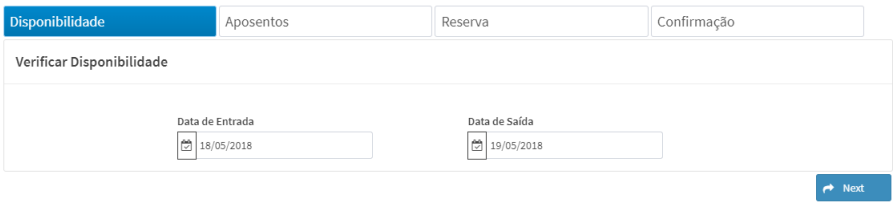
A classe cliente também é salva juntamente com a de reserva e temos uma lista de aposentos, a qual tem a anotação *@ManyToMany* onde o hóspede pode selecionar vários aposentos em uma reserva.

5.4.2 Realizando Reservas

As reservas pelos clientes podem ser realizadas de duas maneiras, online ou entrando em contato com o funcionário responsável. As reservas online seguem um passo a passo para serem confirmadas de fato, primeiramente o cliente escolhe a data de entrada e saída como

mostra a Figura 38.

Figura 38 – Pesquisa por data.

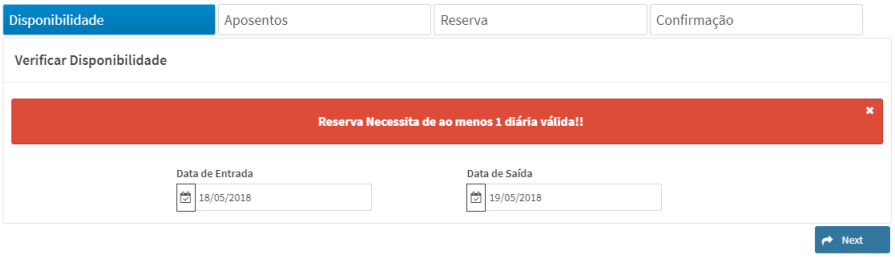


Fonte: Elaboração do autor.

Podem ocorrer três tipos de impedimentos para que uma reserva não consiga ser efetivada, não a quartos para a data selecionada, a data está pegando algum dia de um pacote cadastrado ou a data informada não contêm o mínimo de diárias necessárias.Quadro 20 mostra como é feito estás verificações.

Primeiramente é verifica se existe ao menos uma diária com as datas passadas, caso não haja, uma mensagem é mostrada ao cliente, como mostra a Figura 39.

Figura 39 – Erro com quantidade de diárias.



Fonte: Elaboração do autor.

O próximo teste é verificar se a data é exatamente um pacote, a verificação e feita pegando a data de entrada e data de saída da reserva e comparando com a data inicial e final do pacote, caso elas sejam iguais, uma variável indicando que é pacote recebe o valor true, caso não seja, é verificado se alguma data, entrada ou saída, está entre as datas de algum pacote cadastrado. Caso isso aconteça, outra mensagem de erro é mostrada ao usuário pedindo para entrar em contato com o funcionário responsável, Figura 40.

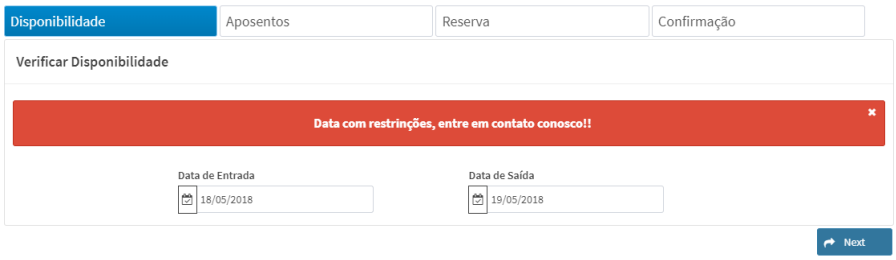
Outro erro é quando não existem aposentos livres, caso não haja, outro erro irá aparecer informando que não existe aposentos para a data selecionada, Figura 41.

Quadro 20 – Verificação de datas.

```
   if (!
reservaValidation.verificaDiariaMinima(reservaCliente.getDataEntrada(),reservaCliente.g
etDataSaida())){
        FacesContext.getCurrentInstance().addMessage(null,
              new FacesMessage(FacesMessage.SEVERITY_ERROR, "Reserva Necessita
de ao menos 1 diária válida!! ", null));
        return false;
   }

   //verifica se está dentro de algum pacote pre cadastrado ativo
   if(this.verificaPacoteIgual(entrada,saida)){
      //é um pacote especial
      isPacote = true;
   } else if(this.verificaPacoteEntreData(entrada,saida)){
      //verifica se alguma data está entre datas de pacote
      // se estiver, a reserva é feita somente com um funcionario
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR, "Data com
restrinções, entre em contato conosco!!", null));
        return false;
   }

   // verifica se existe aposentos disponiveis
   // recebe os aposentos que estao ocupados
   aposentosLivres = (ArrayList<Aposento>)
reservaService.verificaAposentosDisponiveis(entrada,saida);

   //nao existem  aposentos livres
   if(aposentosLivres == null){
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR,
ReservaException.getErroQtdaposentos(), null));
        return false;
   } else {

      //aposentosUteis contem os aposentos que podem ser selecionados
      aposentosUteis.clear();

      if(isPacote){
         this.calculaPacote(entrada,saida);
      }else{
         this.calculaReserva(entrada,saida);
      }
      return true;
   }
}
```

Fonte: Elaboração do autor.

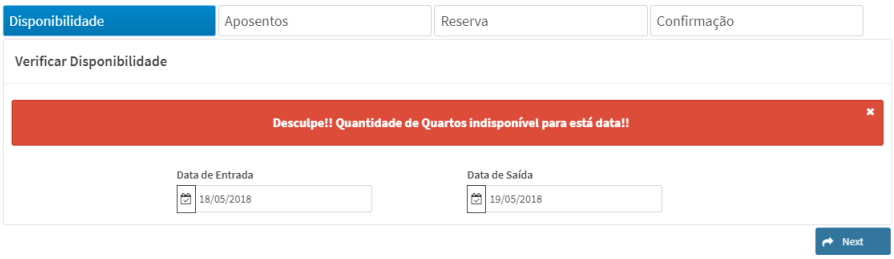
Figura 40 – Erro de datas entre pacotes.



Fonte: Elaboração do autor.

O Quadro 21 mostra como é realizada a pesquisa se existem aposentos livres. Primeiramente é recebido em uma lista, todas as reservas ativas entre a data selecionada pelo cliente, logo após é recebido em outra lista, todos os aposentos que estejam operantes. Caso a lista de reservas esteja vazia, é retornado todos os aposentos operantes.

Figura 41 – Erro de datas entre pacotes.



Fonte: Elaboração do autor.

Quadro 21 – Verificação de aposentos livres.

```
   //recebe todas as reservas que estão ativas ou em aguardo entre as datas do
paremtro
   ArrayList<Reserva> reservas = (ArrayList<Reserva>)
reservaRepository.reservasFinalizadasOuEmAguardoByData(entrada,saida);

   AposentoFilter aposentoFilter = new AposentoFilter();
   aposentoFilter.setStatusQuartoFilter(StatusQuarto.OPERANTE);


   ArrayList<Aposento> aposentos = (ArrayList<Aposento>)
aposentoService.listar(aposentoFilter);

   // não há reservas para esta data
   if(reservas.size() == 0){
      return aposentos;
   }

   // aposentos que podem ser utilizados
   ArrayList<Aposento> aposentosUteis = new ArrayList<>();

   boolean apLivre = true;
```

Fonte: Elaboração do autor.

Se existirem reservas na lista, é criada uma outra lista que irá receber os aposentos que podem ser usados, os aposentos livres são percorridos (Quadro 22) e verificados pelo *id* deles se estão sendo usados em alguma reserva recebida na lista, caso ele não esteja sendo usado em nenhuma reserva, ele é adicionado a lista de aposentos uteis.

Mas se ele estiver sendo utilizado, é verificado se a data de entrada do cliente é igual a data de saída da reserva, ou se a data de saída do cliente é igual a data de entrada da reserva, isso ocorre pelas regras de horários de entrada e saída da empresa, a entrada sempre ocorre as 14h00 e a saída ao 12h00, com isso, uma entrada pode ser na mesma data de saída da outra.

Depois de verificado os aposentos são retornados ao *controller*, após o retorna e calculado o valor de cada aposento com as diárias passada pelo cliente, caso seja um pacote, o Quadro 23 mostra como é realizado o cálculo.

É criado uma lista com o objeto chamado *AposentoLivre* que contém as informações

Quadro 22 – Percorredo Aposentos.

```
    //percorre todos os aposentos
   for (Aposento aposentoLivre: aposentos) {

      //verifica quais quartos serão utilizados
         for (Reserva reserva: reservas) {

         //percorre os aposentos que estão sendo utilizados pelas reservas
         for (Aposento aposentoReservas: reserva.getAposentos()) {
            //verifica se esta sendo utilizado
            if(aposentoLivre.getId() == aposentoReservas.getId()){

               entrada = setData(entrada);
               saida = setData(saida);
               Date saidaReserva = setData(reserva.getDataSaida());
               Date entradaReserva = setData(reserva.getDataEntrada());

               if(saida.compareTo(entradaReserva) == 0){
                  apLivre = true;
                  break;
               }
               if(entrada.compareTo(saidaReserva) == 0){
                  apLivre = true;
                  break;
               }
               apLivre = false;
               break;
            }
         }
      }
      if(apLivre){
         aposentosUteis.add(aposentoLivre);
      }
      apLivre = true;
   }
   if(aposentosUteis.size() == 0){
      return null;
   }
   return aposentosUteis;
}
```

Fonte: Elaboração do autor.

de diárias, aposento e valores para serem mostrado ao usuário, é recebido o pacote, pesquisado pelo banco de dados através das datas do cliente, com o comando for é percorrido os aposentos cadastrados no pacote e verificando se eles estão na lista de aposentos livres, os aposentos que estiverem neste pacote são adicionados ao objeto AposentoLivre com seus respectivos valores.

Caso não seja um pacote, o Quadro 24 mostra como é feito o cálculo que é parecido com o cálculo de pacotes, mas os valores adicionados ao *AposentoLivre* são calculados de acordo com os valores cadastrados para cada quarto, respeitando os valores de finais de semana e dia de semana. Esse cálculo é feito pela quantidade de diárias e pela data de entrada que sempre é

Quadro 23 – Calculando diárias com pacote.

```
private void calculaPacote(Date entrada, Date saida) {
   //recebe o pacote
   this.pacote = pacoteService.findByDatasAtivo(entrada,saida);

   AposentoLivre apUtil;
   //qtd de diarias
   int diarias = reservaValidation.quantidadeDeDiarias(entrada,saida);


   for (Aposento ap: aposentosLivres ) {

      // cria novo aposento para ser mostrado
      apUtil = new AposentoLivre();
      apUtil.setAposento(ap);
      apUtil.setQtdDiarias(diarias);

      //verifica o valor do pacote de acordo com o quarto
      for (PacoteValor pacoteValor: pacote.getPacoteValores()) {
         //o quarto livre pertence ao pacote
         if(ap.getId() == pacoteValor.getId()){
            apUtil.setValorParcial(pacoteValor.getTarifa());
         }
      }
      aposentosUteis.add(apUtil);
   }
}
```

Fonte: Elaboração do autor.

adicionada mais um dia para verificar em qual dia de semana está a data

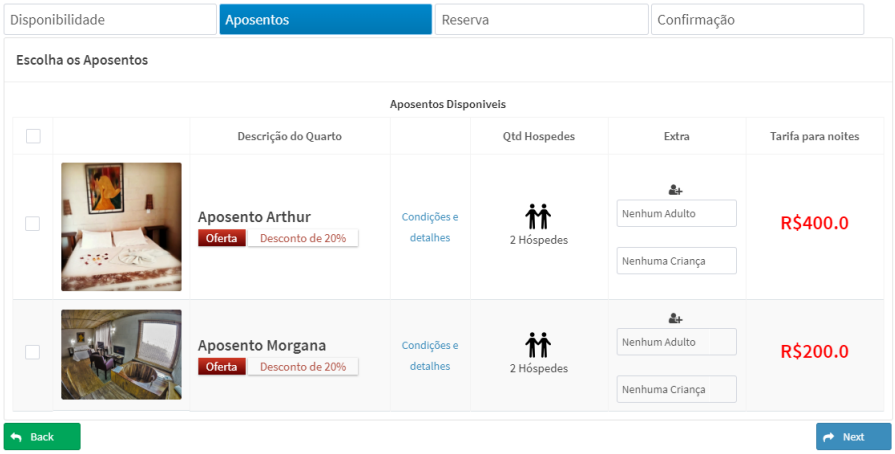
Quadro 24 – Calculando diárias comum.

```
private void calculaReserva(Date entrada, Date saida) {

   AposentoLivre apUtil;
   diarias = reservaValidation.quantidadeDeDiarias(entrada,saida);
   Date dataCalculada = entrada;

 for (Aposento ap: aposentosLivres ) {
      // cria novo aposento para ser mostrado
      apUtil = new AposentoLivre();
      apUtil.setAposento(ap);
      apUtil.setQtdDiarias(diarias);
      double valor = 0;

      //roda a quantidade de diarias
      for(int i = 0; i < diarias; i++) {
         //verifica se é um  final de semana
         if (reservaValidation.retornaFimdeSemana(dataCalculada)) {
            valor += ap.getTarifaPadraoDiariaF().doubleValue();
         } else {
            valor += ap.getTarifaPadraoDiariaC().doubleValue();
         }
         //incrementa mais um dia
         dataCalculada = reservaValidation.addDias(dataCalculada,1);
      }
      //adiciona o valor parcial
      apUtil.setValorParcial(BigDecimal.valueOf(valor));
      //adiciona a qtd de hospede padrao
      apUtil.setQtdTotalHospede(ap.getQtdHospedesPadrao());
      //preenche a lista de aposentos disponiveis
      aposentosUteis.add(apUtil);

   }
}
```

Fonte: Elaboração do autor.

Por fim, os aposentos são retornados ao cliente para serem selecionados, como mostra a Figura 42.

Figura 42 – Aposentos livres.



Fonte: Elaboração do autor.

Nesta etapa o cliente seleciona os aposentos desejados e a quantidade de adultos e/ou crianças extras, cada quarto tem a sua quantidade padrão de hóspedes e a quantidade limite que podem chegar, caso nenhum aposento seja selecionado ou a quantidade de hóspedes extras ultrapassem o limite, um erro é retornado ao cliente, o Quadro 25 mostra como é feito essa validação.

Quadro 25 – Verificando aposentos selecionados.

```
public boolean verificaValoresAposentos(){

   //verifica se algum aposento foi selecionado
   if(aposentosUteisSelecionados.size() == 0){
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR, "Nenhum
Aposento Selecionado", null));
      return false;
   }

   //verifica se algum selecionado passou dos limites de hospedes
   if(verificaLimiteHospede()){
      FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR, "Quarto não
suporta o Limite de Hospede Selecionado, consulte" +
                  " Condições e descrição", null));
      return false;
   }

   //recalcula os valores da hospedagem
   this.recalculaValores();

   //cria objeto reserva
   this.criaReserva();

   return true;
}
```

Fonte: Elaboração do autor.

Os primeiros testes são verificados se algum aposento foi selecionado ou se passaram

de limite dos hóspedes, o limite é calculado (Quadro 26) percorrendo os aposentos selecionados e somando a quantidade de criança e adultos extras mais a quantidade padrão, caso está soma ultrapasse o limite máximo do aposento (atributo da tabela aposento) é retornado o erro para o cliente.

Quadro 26 – Verificando limite de hóspedes.

```
public boolean verificaLimiteHospede(){
   //percorre os aposentos selecinados
   for (AposentoLivre apLivre: aposentosUteisSelecionados) {
      int adultosExtra = Integer.valueOf(apLivre.getQtdAdultoExtra());
      int criancasExtra = Integer.valueOf(apLivre.getQtdCriancaExtra());
      int total = adultosExtra + criancasExtra + 2;

      //verifica se as opções ultrapassaram os limites
      if(total > apLivre.getAposento().getQtdHospedesMaximo()){
         return true;
      }
   }

   return false;
}
```

Fonte: Elaboração do autor.

Após esta verificação, é calculado os novos valores, caso tenha hóspedes extras, o Quadro 27 mostra como é feito este cálculo.

Quadro 27 – Recalculando valores da hospedagem.

```
public void recalculaValores(){
   //percorre os aposentos selecinados
   for (AposentoLivre apLivre: aposentosUteisSelecionados) {
      //qtd de adultos e criancas extras selecionadas
      int adultosExtra = Integer.valueOf(apLivre.getQtdAdultoExtra());
      int criancasExtra = Integer.valueOf(apLivre.getQtdCriancaExtra());

      float valorExtra = 0;
      float valor = 0;
      if(isPacote){
         // percorre os aposentos do pacote
         for (PacoteValor pacoteValor: this.pacote.getPacoteValores()) {
            //verifica se o aposento selecionado esta no pacote
            if(pacoteValor.getAposento().getId() == apLivre.getAposento().getId()){
               valorExtra = (pacoteValor.getPorcentagemHospedeExtra() *
apLivre.getValorParcial().floatValue()  ) / 100;
            }
         }
      } else {
         valorExtra = (apLivre.getAposento().getPorcentagemHospedeExtra() *
apLivre.getValorParcial().floatValue()  ) / 100;
      }

      valor = apLivre.getValorParcial().floatValue();

      //calcula de acordo com a qtd Extra
      valor += (valorExtra * adultosExtra);
      valor += (valorExtra * criancasExtra);
      apLivre.setValorParcial(BigDecimal.valueOf(valor));
      apLivre.setQtdTotalHospedeExtra(adultosExtra + criancasExtra);

      //adiciona a qtd total de hospede
      apLivre.setQtdTotalHospede( apLivre.getQtdTotalHospede() + adultosExtra +
criancasExtra );
   }
}
```

Fonte: Elaboração do autor.

O recalculo é feito percorrendo os aposentos selecionados, os hóspedes extras são somados, é verificado se a reserva é um pacote, assim, são pegos os valores da porcentagem ca-

dastrada em cada aposento correspondente a hóspede extra, a porcentagem é calculada a partir do valor total da reserva até o momento atual.

Após o recalculo é criado a reserva do usuário passando as informações, como aposentos selecionados, quantidade de hóspede, de crianças e adultos, o valor total, valor de entrada e o restante a ser pago, Quadro 28 mostra como foi desenvolvido a função.

Quadro 28 – Recalculando valores da hospedagem.

```
public void criaReserva(){

    int qtdQuartos = aposentosUteisSelecionados.size();

    reservaCliente.setAposentos(new ArrayList<>());
    // aposentos selecionados
    for (AposentoLivre apLivre: aposentosUteisSelecionados) {
        adultosExtra += Integer.valueOf(apLivre.getQtdAdultoExtra());
        criancasExtra += Integer.valueOf(apLivre.getQtdCriancaExtra());
        totalExtra += apLivre.getQtdTotalHospedeExtra();
        total += adultosExtra + criancasExtra + 2;
        reservaCliente.getAposentos().add(apLivre.getAposento());
        valorTotalHospedagem += apLivre.getValorParcial().floatValue();
        reservaCliente.setQtdDias(apLivre.getQtdDiarias());
    }

//calcula quantidade de hospedes
    reservaCliente.setQtdCrianca(criancasExtra);
    reservaCliente.setQtdAdulto(adultosExtra + 2);
    reservaCliente.setQtdHospede(total);
    reservaCliente.setQtdHospedeExtra(totalExtra);

    // se for pacote
    reservaCliente.setValorTotal(BigDecimal.valueOf(valorTotalHospedagem));
    if(isPacote){
        reservaCliente.setPacote(pacote);
    }
    //calcula os valores
    reservaCliente.setStatusReserva(StatusReserva.ANALISE);
    reservaCliente.setValorEntrada(BigDecimal.valueOf((valorTotalHospedagem / 2)));
    reservaCliente.setValorRestante(BigDecimal.valueOf((valorTotalHospedagem / 2)));
}
```

Fonte: Elaboração do autor.

O próximo passo é pegar as informações do cliente, como mostra a Figura 43, todos os dados são obrigatórios, também é validado o CPF e e-mail do usuário, o qual receberá todas as informações da reserva, também é mostrado um resumo da reserva.

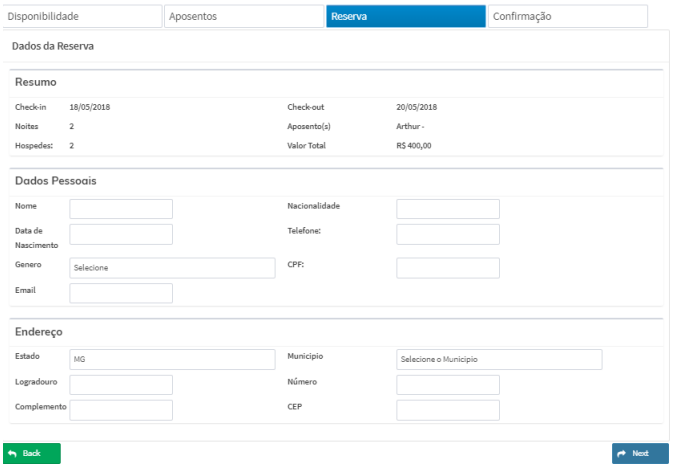
Depois da confirmação dos dados, é mostrado ao cliente um resumo total da reserva, Figura 44, com as datas de entrada e saída, aposento selecionado e valores.

Também é exibido algumas informações de como ocorrerá a confirmação da reserva, para qual e-mail os dados serão enviados, entre outras que a empresa julgar necessárias e um campo para o cliente colocar alguma observação que desejar, Figura 45.

Assim que o cliente confirmar, a reserva é salva no banco de dados e é enviado o e-mail para o próprio com os dados e valores para deposito e também é enviado um e-mail para a empresa lhe avisando de que foi feito uima pre reserva, o cliente será redirecionado a uma tela confirmando a sua reserva, Figura 46.

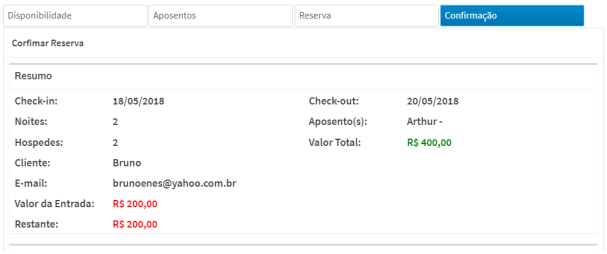
O Quadro 29 mostra algumas regras para que a reserva seja salva no banco de dados.

Figura 43 – Dados do cliente.



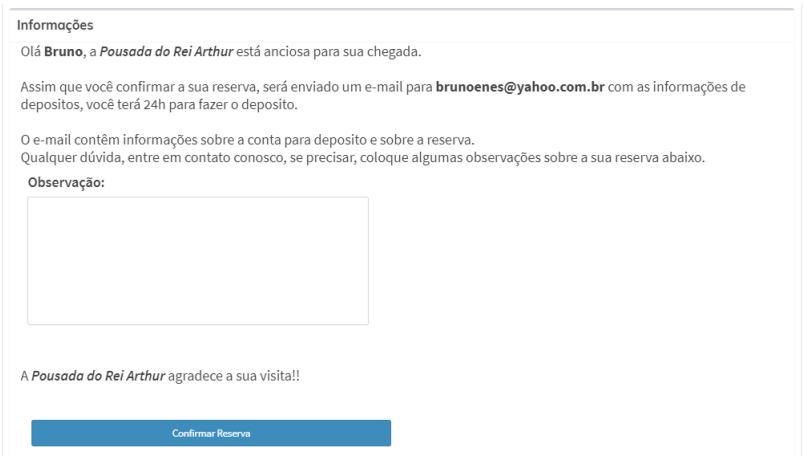
Fonte: Elaboração do autor.

Figura 44 – Resumo reserva.



Fonte: Elaboração do autor.

Figura 45 – Informações.



Fonte: Elaboração do autor.

Figura 46 – Informações.

Fonte: Elaboração do autor.

Quadro 29 – Salvando a reserva.

```
public Reserva saveOrUpdate(Reserva reserva) throws ReservaException {
   //prrenche os atributos da reserva

   // nova reserva
   if(reserva.getId() == null) {
      reserva.setDataReserva(new Date());
      //caso a reserva seja feita por um funcionario, já poderá vir com status
preenchido
      if((reserva.getStatusReserva() == null) ||
reserva.getStatusReserva().getDescricao().equals(StatusReserva.ANALISE.getDesc
ricao())) {
         reserva.setStatusReserva(StatusReserva.AGUARDANDO_PAGAMENTO);
      }
      reserva.getCliente().setDataRegistro(new Date());
      if(reserva.getFormaPagamentoEntrada() == null) {

reserva.setFormaPagamentoEntrada(FormaPagamento.DEPOSITO_BANCARIO);
      }
      reserva.setDataLimiteDeposito(acrescentaDia(reserva.getDataEntrada(),1));

      if (reserva.getUsuario() == null) {
         reserva.setUsuario(buscaAdmin());
      }

      // hora de check-in e check-out
      reserva = this.setHoras(reserva);
   }
   // qtd de dias para check-in
   reserva.setDiasParaEntrada(calculaDistanciaDias(reserva.getDataEntrada()));

   return reservaRepository.saveOrUpdate(reserva);
}
```

Fonte: Elaboração do autor.

Primeiramente é verificado se é uma nova reserva ou se está apenas atualizando, as for a atualização, nesta função só é alterado a quantidade de dias restante para o hóspede realizar o *check-in*.

Caso seja uma nova reserva, é setado a data que a reserva foi feita, adicionado o status de aguardando pagamento caso o status esteja vazio ou em analise, a data de registro do cliente

também é salva e alterado a forma de pagamento, caso a reserva venha do cliente.

No limite para data de deposito é adicionado mais um dia, como decido juntamente com a empresa, também é alterado as horas de entrada e saída, essa função é importante para que no mapa de reserva aparece o horário correto de entrada e saída.

5.4.3 Mapa Reservas

Para que o mapa de reservas fosse desenvolvido, foi escolhido um componente do *PrimeFaces* chamado *schedule*, este componente que é um calendário possui diversas funcionalidades e muitas usando requisições via *ajax*.

Para que as reservas fossem exibidas neste componente, foi preciso realizar uma conversão de objetos. O Quadro 30 mostra o código utilizado.

Quadro 30 – Carregando reservas para o calendário.

```
public void carregaListaReserva(){

   // não foi selecionado nenhum filtro, por padrão, esses serão carregados
   if((filter.getStatus() == null) || (filter.getStatus().length <= 0)) {
      StatusReserva[] lista = {StatusReserva.AGUARDANDO_PAGAMENTO,
StatusReserva.CONFIRMADA, StatusReserva.ANALISE};
      filter.setStatus(lista);
   }
   //limpa as listas
   reservas = new ArrayList<>();
   //recebe as reservas
   reservas = (ArrayList<Reserva>) reservaService.listar(filter);
   eventModel = new DefaultScheduleModel();
   for (Reserva reserva: reservas) {

      // se existir mais de 1 aposento na reserva é criado a qtd de reserva no mapa
      for(Aposento aposento: reserva.getAposentos()){

         eventoReserva = new DefaultScheduleEvent();

         ((DefaultScheduleEvent) eventoReserva).setTitle("Hospede : " +
               reserva.getCliente().getNome() + " Aposento: " + aposento.getNome());
         ((DefaultScheduleEvent) eventoReserva).setAllDay(false);
         ((DefaultScheduleEvent) eventoReserva).setEditable(true);
         ((DefaultScheduleEvent)
eventoReserva).setStartDate(reserva.getDataEntrada());
         ((DefaultScheduleEvent)
eventoReserva).setEndDate(reserva.getDataSaida());
         ((DefaultScheduleEvent)
eventoReserva).setDescription(String.valueOf(reserva.getId()));
         eventoReserva = this.atualizaStyle(eventoReserva,reserva);
eventModel.addEvent(eventoReserva);
      }
   }
}
```

Fonte: Elaboração do autor.

O primeiro teste ao carregar as reservas é verificas se existe algum filtro escolhido pelo usuário, caso não tenha, os filtros padrões da pesquisa são as reservas aguardando pagamento, em analise e confirmadas. Depois de montado o filtro recebemos a lista de reservas.

É criado uma lista de objeto do tipo *Schedule* para receber as reservas, logo após percorremos as reservas em um *for*, o *for* seguinte é para quando uma reserva conter mais um quarto escolhido pelo hóspede, caso isso ocorro, o mapa aparecerá duas reservas, mas com o mesmo nome de hóspede e número da reserva.

Quando percorrido, é criado um objeto *Schedule* para ser adicionado à lista do mesmo criado anteriormente, são preenchidos alguns eventos que este objeto nos proporciona, como o título que irá aparecer para o usuário, se o evento irá ocupar o dia todo, que recebe o valor falso, como mostrado anteriormente, as reservas têm horário de entrada e saída definidos, se a reserva pode ser editada, recebe o valor true, as datas de entrada e saída e uma descrição, na qual recebe o id da reserva que será usada mais adiante.

Logo após é chamado uma função para atualizar o estilo de cada evento, isso é importante para a identificação da reserva no calendário pelas cores, o comando que realiza a mudança é: *((DefaultScheduleEvent) event).setStyleClass(“”)*, dentro das aspas vai um css ou uma classe css já criada. Para este sistema, a função verifica qual o status da reserva, e cada status receberá uma cor pré-definida, está cor irá mudando de acordo com algum evento que o usuário faça, como confirmar a reserva, cancelar, ente outras.

Assim que o comando for finalizar, já temos o calendário pronto para ser exibido para o usuário. O Quadro 31 mostra o código da página que encontra o componente schedule.

Quadro 31 – Carregando reservas para o calendário.

```
<p:schedule draggable="false" ignoreTimezone="false"
        style="font-size: 13px"
        id="schedule"
        value="#{reservaMapaController.eventModel}"
        widgetVar="myschedule"
        locale="pt"
        resizable="false">

    <p:ajax event="eventSelect" listener="#{reservaMapaController.onEventSelect}"
        update=":detalhesReserva, :btnConfirmar, :labelDeposito, :valorDepositado"
        oncomplete="PF('detalhesReservaSelecionada').show()" />

</p:schedule>
```

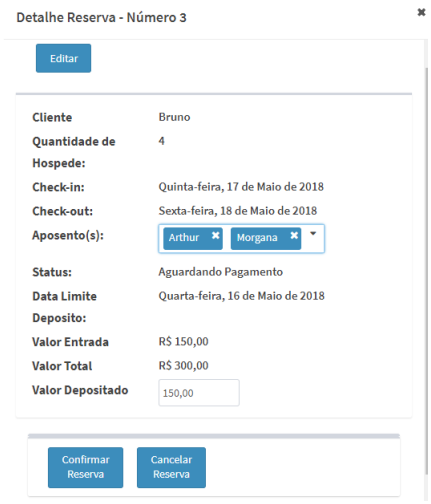
Fonte: Elaboração do autor.

Podemos observar que a lista criada pela classe *ReservaMapaController*, *eventModel* está sendo utilizada pelo *schedule*, outras propriedades também estão sedo utilizadas, como o *resizable = false*, o que significa que as reservas não poderão ser alterar pelo mouse, entre outras.

O evento *ajax* dentro do schedule indica que quando uma reserva for clicada pelo usuário, ele irá chamar uma função no *ReservaMapaController*, chamada *onEventSelect*, está função irá pegar o *id* da reserva que se encontra na descrição do evento, com o id, irá buscar a reserva no banco de dados e preenchê-la em um objeto Reserva, assim que sair desta função *ajax*, um poup up abrira com as informações da reserva e algumas opções que poderão ser feitas, como mostra a Figura 47.

Podemos confirmar, cancelar ou finalizar, ultima não aparece pelo fato da reserva não está confirmada. Para cancelar a reserva, a mesma tem seu status atualizada para CANCELADA

Figura 47 – *Poupup* Reserva.



Fonte: Elaboração do autor.

e logo após é enviado um e-mail para o cliente e a lista é carregada novamente para atualização do mapa, Quadro 32.

É importante firmar que quando uma reserva é cancelada, o fluxo de caixa não é alterado, pois, de acordo com a política da empresa o dinheiro pode ou não ser devolvido no momento do cancelamento, assim a movimentação no fluxo deve ser feita assim que o funcionário realmente retirar o dinheiro do caixa da empresa, logo o usuário do sistema faz um lançamento manual.

A confirmação da reserva passa por alguns passos, primeiramente é verificado se o seu status é realmente de aguardando pagamento, caso seja o seu status é atualizado como confirmada, também é *setado* o usuário que a confirmou.

Após, é verificado se o valor depositado é o mesmo valor que foi sugerido para deposito no momento de realizar a reserva, como sabemos que é preciso um deposito de 50% do valor, a entrega somente é confirmada depois do comprovante ter chego ao usuário do sistema, mas o hóspede pode ter depositado um valor maior ou menor do que o recomendado, antes de confirmar é possível passar um novo valor, caso não seja passado, entende-se que o valor da entrada foi o valor designado no e-mail de reserva. Sabendo do valor, é calculado o valor restante da reserva a ser pago, como mostra o Quadro 33.

Depois de verificado, a reserva é atualizada no banco de dados, e é criado um lançamento com a categoria *Check-in* e inserido no fluxo de caixa, depois de inserido o fluxo, é enviado um e-mail ao hóspede confirmando a sua reserva e carregado a lista de reservas novamente para atualizar o calendário, Quadro 34, caso ocorra algum erro, o sistema envia uma mensagem para o usuário. Os botões Editar e Finalizar, o usuário e lavado a outras páginas com suas funcionalidades especificas.

Quadro 32 – Cancelar Reserva.

```
public void cancelarReserva(){

   FacesMessage message;

   reservaSelecionada.setStatusReserva(StatusReserva.CANCELADA);
   //usuario que confirmou a reserva
   reservaSelecionada.setUsuario(usuario);

   try {
      //atualiza
      reservaService.saveOrUpdate(reservaSelecionada);
      mailSend.sendMailClienteCancelamento(reservaSelecionada);
      message = new FacesMessage(FacesMessage.SEVERITY_INFO,
"Cancelada!!", MessageUtil.RESERVA_CANCELADA +
            " - A retirada do caixa terá que ser feita manualmente!!");

   } catch (ReservaException e) {
      message = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Erro!!",
MessageUtil.ERRO_RESERVA_CANCELADA);
   }

   addMessage(message);
   // atualiza a lista e o calendario
   this.carregaListaReserva();
}
```

Fonte: Elaboração do autor.

Quadro 33 – Confirma reserva parte 1.

```
if((reservaSelecionada.getStatusReserva() ==
StatusReserva.AGUARDANDO_PAGAMENTO) ||
      (reservaSelecionada.getStatusReserva() == StatusReserva.ANALISE)){
   //confirmada
   reservaSelecionada.setStatusReserva(StatusReserva.CONFIRMADA);
   //usuario que confirmou a reserva
   reservaSelecionada.setUsuario(usuario);
   //seta o valor do deposito
   // verifica se houve alguma alteração em relaçao ao valor passado pr e-mail
   if(reservaSelecionada.getValorDepositado().floatValue() <= 0){

reservaSelecionada.setValorDepositado(reservaSelecionada.getValorEntrada());
   } else {

reservaSelecionada.setValorEntrada(reservaSelecionada.getValorDepositado());
      reservaSelecionada.setValorRestante(BigDecimal.valueOf (
            (reservaSelecionada.getValorDepositado().floatValue() -
reservaSelecionada.getValorTotal().floatValue()))
      );
   }
```

Fonte: Elaboração do autor.

Quadro 34 – Confirma reserva parte 2.

```
try {
    //atualiza
    reservaService.saveOrUpdate(reservaSelecionada);
    message = new FacesMessage(FacesMessage.SEVERITY_INFO,
"Confirmada!!", MessageUtil.RESERVA_CONFIRMADA);

    //realiza um lançamento no fluxo de caixa
    lancamento = new Lancamento();
    lancamento.setUsuario(usuario);
    lancamento.setData(new Date());
    lancamento.setValor(reservaSelecionada.getValorDepositado());
    lancamento.setDescricao("Entrada referente Check-in a Reserva de número: " +
reservaSelecionada.getId() +
            ". Hospede: " + reservaSelecionada.getCliente().getNome());
    Categoria categoria = categoriaService.findByNome("Hospedagem Check-in");
    lancamento.setCategoria(categoria);

    lancamentoService.save(lancamento);

    mailSend.sendMailClienteConfirmacao(reservaSelecionada);

    mailSend.sendMailEmpresaConfirmacao(reservaSelecionada);

} catch (ReservaException e) {
    message = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Erro!!",
MessageUtil.RESERVA_ERRO);
} catch (LancamentoException e) {
    message = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Erro!!",
"Erro ao Realizar o Lança,ento no " +
            "Fluxo de Caixa, faça Manualmente!");
}
```

Fonte: Elaboração do autor.

5.4.3.1 Editar Reserva

Assim que o usuário é redirecionado a tela de editar reserva, são exibidas as informações da mesma com a possibilidade de aumentar a estadia do hóspede, para isso o usuário seleciona terá um componente com um calendário, nele ele irá selecionar quantos dias a mais o hóspede deseja ficar, como mostra a Figura 48, o valor será calculado de acordo com o valor do aposento ou o usuário também poderá, na hora de selecionar os dias a mais, passar um valor total desses dias, essa decisão foi tomada juntamente com os representantes da empresa.

Figura 48 – Adicionando diárias.

Fonte: Elaboração do autor.

O componente calendário para alterar a data da reserva é acionado com a data mínima

de saída da reserva, ou seja, não é possível selecionar nenhuma data anterior a data de saída.

Assim que confirmado os dias, e se for o caso o valor que será somado, o sistema calcula o total da reserva, como mostra o Quadro 35.

Quadro 35 – Recalculando valor da reserva.

```
public Reserva recalculaValores(Reserva reserva, BigDecimal valorExtra, int dias) {

    Date novaDataSaida = reservaValidation.addDias(reserva.getDataSaida(), dias);
    float valorDiariaExtra = valorExtra.floatValue();
    float total = 0;

    // nao houve alteração, valor sera calculado pela diaria comum do aposento
    if(valorDiariaExtra <= 0.0){
        total = this.calculaReserva(reserva,novaDataSaida,0,dias);
    } else {
        // os dias a mais serao calculados com um novo valor
        total = this.calculaReserva(reserva,novaDataSaida,valorDiariaExtra,dias);
    }
    // é feito a soma com o valor anterior
    reserva.setValorTotal( BigDecimal.valueOf(reserva.getValorTotal().floatValue() + total));

    // verifica se ja foi depositado o valor
if(reserva.getStatusReserva().getDescricao().equals(StatusReserva.CONFIRMADA.getDescr
icao())){
        //calcula o valor restante  da reserva
        float restante = reserva.getValorTotal().floatValue() -
reserva.getValorEntrada().floatValue();
        reserva.setValorRestante(BigDecimal.valueOf(restante));
    } else {
        // caso nao tenha sido depositado o valor, calcula uma nova entrada e envia um email
        reserva.setValorRestante( BigDecimal.valueOf(reserva.getValorTotal().floatValue() / 2));
        reserva.setValorEntrada( BigDecimal.valueOf(reserva.getValorTotal().floatValue() / 2));
        //envio de email
        mailSend.sendMailClienteDeposito(reserva);
    }
    //seta a nova data de saida
    reserva.setDataSaida(novaDataSaida);
    reserva = this.setHoras(reserva);
    // caso nao entre em nunhum, nao houve alteração
    return reserva;}
```

Fonte: Elaboração do autor.

A função para recalcular recebe a reserva em questão, a quantidade de dias extras e o valor que seja adicionado caso haja algum, primeiramente é calculado a nova data de saída e logo após é verificado se houve algum valor passado pelo usuário e enviado para uma outra função que será explicada a seguir.

Depois de ter retornado o valor recalculado, é feito uma soma com o valor com o valor total, depois é verificado se a reserva já estava confirmada, pois se já estiver, significa que o valor de entrada já foi depositado, então esse novo valor será pago no *check-in*, mas se a reserva não esteja confirmada, o valor total é dividido e enviado um novo e-mail para o hóspede com um novo valor de entrada. A reserva é retornada peara o *controller* e uma mensagem de sucesso é exibida para o funcionário.

O Quadro 36 nos mostra como é feito de fato a cálculo de dias de reservas, o cálculo é realizado por um comando for que que é percorrido pela quantidade de diárias pegando o valor de diária do aposento, é passado a data de entrada e sempre verificado se a data é um dia de semana ou final de semana, pois como já vimos, os valores podem ser diferentes, está data sempre é recalculada aumentado um dia até a quantidade de diárias, depois é retornado o valor total.

Quadro 36 – Calculando reservas por dia.

```
private float calculaReserva(Reserva reserva, Date novaDataSaida, float extra, int
diarias) {
    Date dataCalculada = reserva.getDataSaida();
    int contadorDias = 0;
    if(diarias == 0){
        contadorDias = reserva.getQtdDias();
    } else {
        contadorDias = diarias;
    }
    float valor = 0;
    for (Aposento ap : reserva.getAposentos()) {
        //roda a quantidade de diarias
        for (int i = 0; i < contadorDias; i++) {

            //verifica se é um  final de semana
            if (reservaValidation.retornaFimdeSemana(dataCalculada)) {
                if(extra == 0)
                    valor += ap.getTarifaPadraoDiariaF().floatValue();
                else
                    valor += extra;
            } else {
                if(extra == 0)
                    valor += ap.getTarifaPadraoDiariaC().floatValue();
                else
                    valor += extra;
            }
            //incrementa mais um dia
            dataCalculada = reservaValidation.addDias(dataCalculada, 1);
        }
    }
    return valor;
}
```

Fonte: Elaboração do autor.

5.4.3.2 Finalizar Reserva

Ao finalizar uma reserva, o usuário terá na tela todas as informações da mesma, antes de finalizar, o usuário poderá adicionar o valor total do consumo do hóspede, o sistema atual não está preparado para separar o consumo e nem fazer o controle de estoque, isso ficará para um trabalho futuro, mas o valor total pode ser adicionado a reserva e também ao fluxo de caixa.

Se o usuário desejar adicionar o consumo, um *poup up* se abrirá para que o valor seja adicionado, Figura 49.

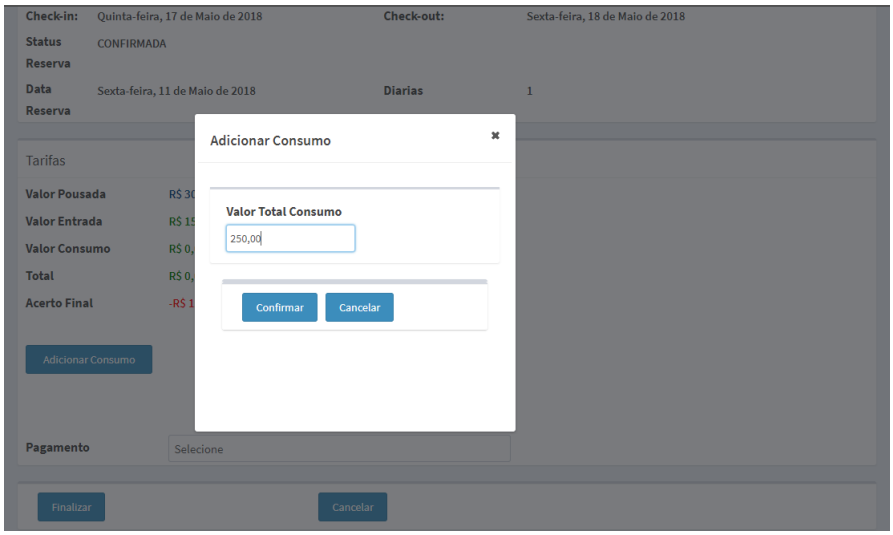
Assim que adicionado, é feito um cálculo para saber o valor restante que o hóspede deverá pagar, como mostra o Quadro 37.

Primeiramente, o tributo de valor do consumo da reserva é adicionado, depois é calculado o valor total da reserva já com o consumo, o valor restante a ser pago é calculado subtraindo o total com consumo menos o valor de entrada já pago, assim é mostrado na tela para o usuário um resumo com todos os valores e o restante a ser pago pelo hóspede, autoref{resumoValores}.

Depois de tudo calculado, o usuário seleciona a forma de pagamento e confirma o fechamento da reserva, o Quadro 38 mostra como a função de finalizar foi desenvolvida.

Primeiramente é verifica se a reserva já estava finalizada (Quadro 38), caso não, é alterado o status para finalizada e salva no banco de dados, caso não haja erro ao salvar, é verifica se poderá lançar o valor no fluxo de caixa, caso a forma de pagamento seja dinheiro, o valor restante pago é lançado ao fluxo de caixa (Quadro 39), caso a forma de pagamento seja outra,

Figura 49 – Adicionando consumo.



Fonte: Elaboração do autor.

Quadro 37 – Calculando o consumo.

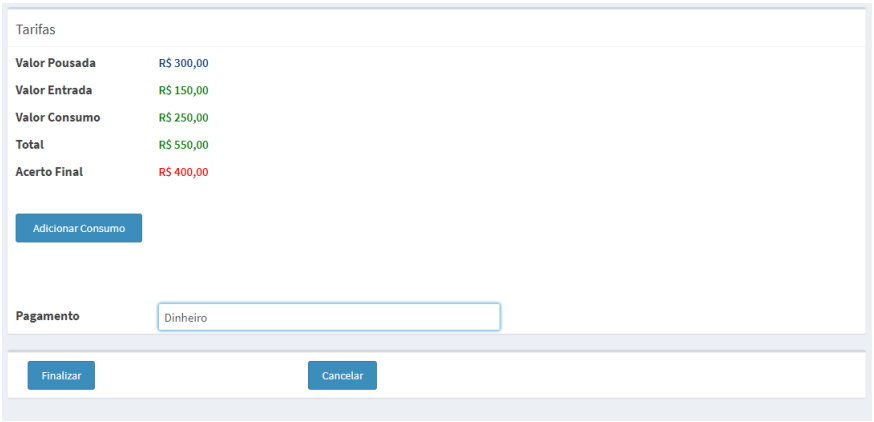
```
public void calcularConsumo(){

    float valorRestante = reservaSelecionada.getValorRestante().floatValue();
    float total = 0;
    // valor do consumo
    reservaSelecionada.setValorConsumo(BigDecimal.valueOf(
reservaSelecionada.getValorConsumo().floatValue() + consumo  ));

    //valor total com consumo
    total = reservaSelecionada.getValorTotal().floatValue() +
reservaSelecionada.getValorConsumo().floatValue();
    reservaSelecionada.setValorTotalcomConsumo(BigDecimal.valueOf(total));

    //valor restante
    valorRestante = total - reservaSelecionada.getValorEntrada().floatValue();
    reservaSelecionada.setValorRestante(BigDecimal.valueOf(valorRestante));

    consumo = 0;
}
```

Fonte: Elaboração do autor.

o lançamento terá que ser feito pelo usuário, esta decisão foi tomada juntamente com os representantes da empresa. O mesmo acontece com o consumo, caso haja algum consumo, também é lançado ao fluxo de caixa (Quadro 40).

Uma mensagem é mostrada ao usuário confirmando todas as operações, como mostra a Figura 51.

Figura 50 – Resumo dos valores.



Fonte: Elaboração do autor.

Quadro 38 – Finalizando a reserva - parte 1.

```
if (reservaSelecionada.getStatusReserva().getDescricao()
      .equals(StatusReserva.FINALIZADA.getDescricao())) {

   FacesContext.getCurrentInstance().addMessage(null,

         new FacesMessage(FacesMessage.SEVERITY_ERROR,

                  "Reserva já está Finalizada!", null));
```

Fonte: Elaboração do autor.

Quadro 39 – Finalizando a reserva - parte 2.

```
//realiza um lançamento no fluxo de caixa
Lancamento lancamento = new Lancamento();

Categoria categoria = new Categoria();

//verifica se houve pagamento de check-out

float valorRestante = reservaSelecionada.getValorTotal().floatValue()

      - reservaSelecionada.getValorDepositado().floatValue();

if (valorRestante > 0) {

   lancamento.setUsuario(usuario);

   lancamento.setData(new Date());

   lancamento.setValor(BigDecimal.valueOf(valorRestante));

   lancamento.setDescricao("Entrada referente Check-out a Reserva de número: "

         + reservaSelecionada.getId() +

         ". Hospede: " + reservaSelecionada.getCliente().getNome());

   categoria = categoriaService.findByNome("Hospedagem Check-out");

   lancamento.setCategoria(categoria);

   try {

      lancamentoService.save(lancamento);

      FacesContext.getCurrentInstance().addMessage(null,

            new FacesMessage(FacesMessage.SEVERITY_INFO,

                  "Valor da hospedagem lançado ao fluxo de caixa com sucesso!!",

                  null));

   } catch (LancamentoException e) {

      FacesContext.getCurrentInstance().addMessage(null,

            new FacesMessage(FacesMessage.SEVERITY_ERROR,

                  "Erro ao lançar valor da hospedagem ao fluxo de caixa!!",

                  null));
```

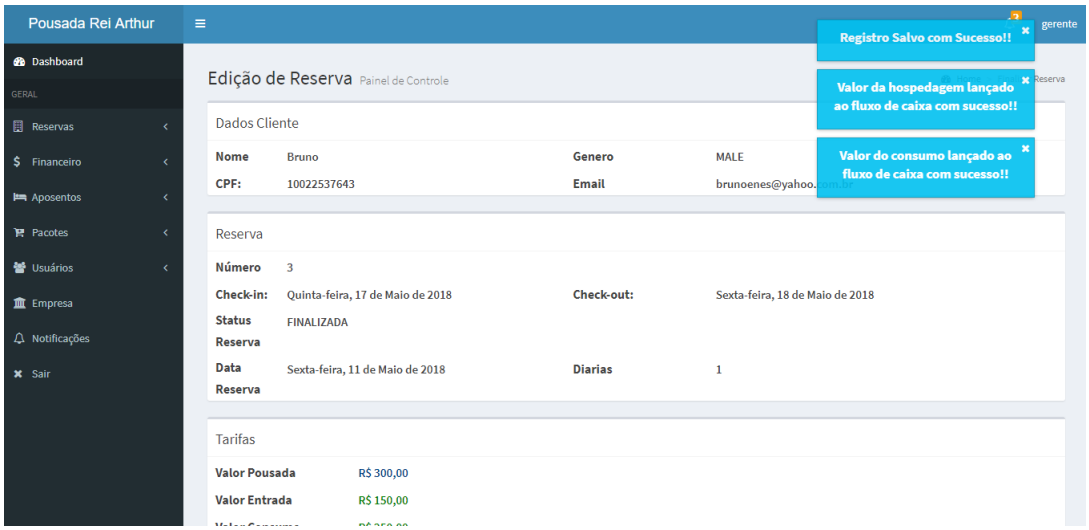
Fonte: Elaboração do autor.

Quadro 40 – Finalizando a reserva - parte 3.

```
//verifica se houve consumo
if (reservaSelecionada.getValorConsumo().floatValue() > 0) {
    lancamento = new Lancamento();
    lancamento.setUsuario(usuario);
    lancamento.setData(new Date());
    lancamento.setValor(reservaSelecionada.getValorConsumo());
    lancamento.setDescricao("Entrada referente ao Consumo da Reserva de número:
"
            + reservaSelecionada.getId() +
            ". Hospede: " + reservaSelecionada.getCliente().getNome());
    categoria = categoriaService.findByNome("Consumo");
    lancamento.setCategoria(categoria);
    try {
        lancamentoService.save(lancamento);
        FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage(FacesMessage.SEVERITY_INFO,
                        "Valor do consumo lançado ao fluxo de caixa com sucesso!!",
                        null));
    } catch (LancamentoException e) {
        FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage(FacesMessage.SEVERITY_ERROR,
                        "Erro ao lançar valor do consumo fluxo de caixa!!",
                        null));
    }
```

Fonte: Elaboração do autor.

Figura 51 – Confirmação de reserva finalizada.



Fonte: Elaboração do autor.

5.5 Envio de E-mails

A comunicação entre a empresa e o hóspede, quando é realizada a reserva pelo site, é realizada por envios de e-mails, o sistema os envia em divesas situações. A primeira e uma das mais importantes é quamdo o hóspede realiza uma pré-reserva no site, o sistema recebe a mesma e envia um e-mail para o hóspede contendo os dados bancários da empresa para o depósito, assim como o valor que precisa ser depósitado e um resumo de sua reserva. Nesta mesma etapa é enviado um e-mail para a empresa avisando sobre a pré-reserva realizada.

Quando o funcionário recebe a confirmação de pagamento da pré-reserva, ele terá que confirma-lá no sistema, assim que estpa confirmação é feita, também é enviado um e-mail para o hóspede lhe avisando sobre a confirmação, o mesmo acontece quando uma reserva é cancelada.

De acordo com as regras da empresa, uma cliente tem até vinte e quatro horas para realizar o pagamento de sua pré-reserva, o sistema verifica a data na qual a reserva foi feita, caso já se tenha passado às vinte e quantro horas para o pagamento e a reserva em quustão não esteja confirmada, é enviado um e-mail para o funcioário responsável para que o mesmo possa tomar as atitudes necessárias, seja ela cancelando a reserva ou entreando em contato com o cliente.

5.6 *Design* Responsivo

O *design* responsivo tem chamado bastante atenção para as empresas que utilizam o meio Web pelo alto crescimento de acesso à internet por diferentes dispositivos e não somente pelo computador, pensando nisso, o site desenvolvido neste projeto utilizou o *framework Bootstrap*.

O *Bootstrap* é uma estrutura HTML, CSS e *JavaScript* gratuita e popular para o desenvolvimento de sites responsivos que priorizam os dispositivos móveis ("*mobile-first*"). A estrutura inclui modelos CSS e HTML responsivos para botões, tabelas, navegação, carrosséis de imagens e outros elementos que você pode usar na sua página da web.

Estão disponíveis alguns *plug-ins JavaScript* opcionais, que permitem que mesmo os desenvolvedores com conhecimento básico de programação desenvolvam sites excelentes e responsivos.

O *Bootstrap* funciona com um sistema de *grids* (grades) para posicionar os elementos na página. Esse mecanismo funciona como uma espécie de tabela abstrata, e é responsivo (*responsive*), orientado a dispositivos móveis (*mobile first*) e se ajusta de acordo com a tela (*fluid*), quando ela muda de tamanho ou de orientação.

Dizer que o *Bootstrap* é *Mobile First* significa que o *framework* assume, inicialmente, que a tela é de um dispositivo móvel, com tamanho pequeno. Assim, ele adapta todos os conteú-

dos para o tamanho menor. Depois, ele verifica o tamanho real da tela e vai ajustando os itens para que fiquem posicionados corretamente, conforme o tamanho e a resolução.

As *grids* do *Bootstrap* são divididas por tamanhos diferentes que cada div pode obter, a Figura 52 exibe como é o sistema de grids.

Figura 52 – *Grids* do *Bootstrap*.

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Fonte: Disponível em: < https://getbootstrap.com/docs/4.0/layout/grid > acesso em 18/04/2018.

Usando estes conceitos, o site foi desenvolvido e suporta o acesso de qualquer tamanha de tela, ele irá se ajustar ao tamanha necessário, a Figura 53 mostra o site sendo acessado de um computador de 15', abaixo temos a Figura 54, onde é acessado de um celular.

Figura 53 – Acesso de um computador.

Fonte: Elaborada pelo autor.

Figura 54 – Acesso de um celular, $design$ responsivo.

Fonte: Elaborada pelo autor.

6 TRABALHOS FUTUROS

Muitas são as possibilidades para trabalhos futuros neste sistema, sugere-se o controle de estoque dos produtos vendidos pela empresa, onde há o consumo dos hóspedes, também é possível deixar o *Website* mais dinâmico, fazendo com que o usuário consiga alterar todo o conteúdo das páginas sem a ajuda de um desenvolvedor, um bom exemplo seria o *WordPress* [1] onde o mesmo possibilita ao usúario adicionar textos, imagens entre outros em quaisquer página html, até mesmo criar novas páginas.

Outra parte importante é a integração do sistema com aplicativos móveis, criando um aplicativo para que o usuário possa controlar e visualizar tanto o fluxo de caixa quanto o controle de estoque, verificando as reservas que estão cadastradas, podendo alterá-las do mesmo jeito que o sistema permite atualmente.

A criação de um sistema para o hóspede também é uma sugestão interessante, onde o mesmo, depois de sua reserva confirmada, poderia acessar e verificar suas informações e até mesmo realizar algum pedido de consumo sem sair do quarto.

[1] <https://br.wordpress.com/>

7 CONSIDERAÇÕES FINAIS

Este projeto teórico e prático realizado durante o desenvolvimento do trabalho de conclusão de curso atendeu aos requisitos que foram levantados com base na modelagem do sistema, além de atender todos os objetivos específicos propostos do trabalho.

O sistema desenvolvido é capaz de atender grande parte da necessidade do sistema hoteleiro que é (*i*) o fluxo de caixa, (*ii*) o controle de reservas online, (*iii*) *Website* responsivo entre outras funcionalidades implementadas. Como já citado, o sistema será de grande valia tendo em vista que nos dias atuais a empresa não tem nenhuma tecnologia diferenciada que a auxilie nestes quesitos.

Foi realizada a validação por funcionários da empresa, os quais afirmaram que o produto irá melhorar o atendimento dando agilidade a tais processos. Além disso, o sistema será aprimorado em possíveis trabalhos futuros, tendo em vista a realização de novas implementações e/ou correções conforme citado.

Durante a realização do projeto, foi adquirido um vasto aprendizado sobre as tecnologias descritas e colocado em prática o conhecimento obtido na graduação possibilitando e acrescentando em minha formação e que também será de grande valia em minha futura atuação como profissional.

Referências

ALEXANDER, C. *An Introduction for Object-Oriented Designers*. 1997. Acesso: 11 de abril de 2018. Disponível em: <http://g.oswego.edu/dl/ca/ca/ca.html>. Citado na página 27.

ALMEIDA, A. *Entenda os MVCs e os frameworks Action e Component Based*. 2012. Acesso: 12 de abril de 2018. Disponível em: <http://blog.caelum.com.br/entenda-os-mvcs-e-os-frameworks-action-e-component-based/>. Citado na página 30.

BALLEM, M. *Consultas com Hibernate e a API Criteria Parte I*. 2017. Acesso: 13 de abril de 2018. Disponível em: <http://www.mballem.com/post/consultas-com-hibernate-e-a-api-criteria-parte-i/?i=3>. Citado na página 36.

BITZ. *Software de gestão para hospedagem*: O segredo do sucesso! 2018. Acesso: 12 de maio de 2018. Disponível em: <http://www.bitzsoftwares.com.br/software-de-gestao-para-hospedagem/>. Citado na página 21.

BOOCH JAMES RUMBAUGH, I. J. G. *UML*: guia do usuário. [S.l.: s.n.], 2005. v. 9ł Reimpressão. Citado na página 53.

BUENO, K. J. *O que é JSF (Java Server Faces)?* 2015. Acesso: 12 de maio de 2018. Disponível em: <http://fabrica.ms.senac.br/2013/06/o-que-e-jsf-java-server-faces/>. Citado na página 29.

CAELUM. *O que é Java EE?* 2015. Acesso: 11 de abril de 2018. Disponível em: <https://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/>. Citado na página 25.

CALçADO, P. *Curso Spring Framework*. 2008. Acesso: 19 de abril de 2018. Disponível em: <http://blog.flexdev.com.br/wp-content/uploads/spring/apostila-spring.pdf>. Citado na página 37.

CHHATPAR, A. *Apache Geronimo e o Spring Framework, Parte 1*. 2006. Disponível em: <http://www.ibm.com/developerworks/br/library/os-ag-springframe1/authors.html>. Citado na página 37.

DEVMEDIA. *Desenvolvendo com Hibernate*. 2009. Acesso: 13 de abril de 2018. Disponível em: <https://www.devmedia.com.br/desenvolvendo-com-hibernate/14756>. Citado na página 34.

DEVMEDIA. *Uso de PrimeFaces no Desenvolvimento de aplicações ricas para web*. 2012. Acesso: 11 de abril de 2018. Disponível em: <https://www.devmedia.com.br/desenvolvendo-aplicacoes-ria-com-primefaces-revista-easy-java-magazine-24-parte-2/26400>. Citado na página 39.

DEVMEDIA. *Introduzindo o servidor de aplicação Apache Tomcat*. 2013. Acesso: 02 de maio de 2018. Disponível em: <https://www.devmedia.com.br/introduzindo-o-servidor-de-aplicacao-apache-tomcat/27939>. Citado 2 vezes nas páginas 32 e 33.

DEVMEDIA. *Introdução ao Padrão MVC*. 2013. Acesso: 11 de abril de 2018. Disponível em: <https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>. Citado 2 vezes nas páginas 27 e 28.

DEVMEDIA. *Java EE*: Entendendo a plataforma. 2014. Acesso: 10 de abril de 2018. Disponível em: <https://www.devmedia.com.br/java-ee-entendendo-a-plataforma/30195>. Citado 2 vezes nas páginas 25 e 26.

DEVMEDIA. *Java Server Faces (JSF)*: Desenvolvimento de aplicações web. 2014. Acesso: 11 de abril de 2018. Disponível em: <https://www.devmedia.com.br/java-server-faces-jsf-desenvolvimento-de-aplicacoes-web/31026>. Citado na página 30.

DEVMEDIA. *Introdução ao Processo Unificado*. 2016. Acesso: 02 de maio de 2018. Disponível em: <https://www.devmedia.com.br/introducao-ao-processo-unificado/3931>. Citado 2 vezes nas páginas 50 e 51.

ESJUG. *Tutorial Hibernate Básico*. 2014. Acesso: 18 de abril de 2018. Disponível em: <https://inf.ufes.br/~vsouza/files/TutorialHibernateBasico.pdf>. Citado na página 36.

FEITOSA, D. B. *Visão geral sobre Primefaces*. 2010. Sergipe. Acesso: 10 de abril de 2018. Disponível em: <http://williamgamers.wordpress.com/2012/06/04/visao-geral-sobre-primefaces/>. Citado na página 39.

GOMES, Y. M. P. *Java na Web com JSF, Spring, Hibernate e Netbeans 6*: de universitários a desenvolvedores. Rio de Janeiro: [s.n.], 2008. Citado na página 30.

JOHSON, R. et A. *Spring Reference*. 2011. Acesso: 22 de abril de 2018. Disponível em: <http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/htmlsingle/spring-framework-reference.html>. Citado na página 37.

KIOSKEA. *Hibernate Primeira Parte*: Apresentação. 2013. Acesso: 18 de abril de 2018. Disponível em: <http://pt.kioskea.net/faq/10525-hibernate-primeira-parte-apresentacao>. Citado na página 34.

MARCOTTE, E. *Responsive Web Design*. 2010. Acesso: 22 de abril de 2018. Disponível em: <http://alistapart.com/article/responsive-web-design>. Citado na página 41.

PERES, L. U. *Empreendimentos Fractional*. 2018. Acesso: 05 de maio de 2018. Disponível em: <http://www.revistahotelnews.com.br/portal/opiniao.php?> Citado na página 21.

PISA, P. *O que é e como usar o MySQL?* 2012. Acesso: 05 de maio de 2018. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html>. Citado na página 49.

PRIMEFACES. *Primefaces*. 2018. Acesso: 10 de abril de 2018. Disponível em: <http://www.primefaces.org/>. Citado na página 40.

REGISTRO.BR. *Estatísticas*. 2018. Acesso: 05 de maio de 2018. Disponível em: <https://registro.br/estatisticas.html>. Citado na página 21.

ROBERT, M. P. W. *Spring Security 3.1*. 2012. Acesso: 19 de abril de 2018. Disponível em: <http://www.springsource.org/spring-security>. Citado na página 38.

SAUVé, J. P. *Persistência Usando Hibernate*. 2000. Acesso: 18 de abril de 2018. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/hibernate/hibernate.htm>. Citado na página 34.

SCARIONI, C. *Pro Spring Security*. [s.n.], 2013. Acesso: 18 de abril de 2018. Disponível em: <https://books.google.com.br/books?id=VIqInfAXFaoC&printsec=frontcover&dq=spring+security&hl=pt-BR&sa=X&ei=JNUMUtb2JIfk9gSo8oDIAQ&ved=0CD8Q6AEwAg#v=onepage&q=spring%20security&f=false>. Citado na página 37.

SCHITINI, I. et A. *Spring Framework*. 2011. Acesso: 18 de abril de 2018. Disponível em: <http://kenai.com/projects/pos-sistemasjava-jf/sources/pos-java-ufjf-2009-2011/content/02Daves/SpringFramework.doc?rev=48>. Citado na página 37.

SILVA, M. S. Web design responsivo. In: ____. [S.l.]: São Paulo, 2014. cap. Capítulo 1: Introdução ao Design Responsivo, p. 27 a 47. Citado na página 40.

SILVA, M. S. *Web Design Responsivo*. [S.l.]: São Paulo, 2014. Citado 2 vezes nas páginas 41 e 42.

SILVA, P. R. da. *Dinâmica local alterada pelo turismo no Distrito de Lavras Novas, Ouro Preto-MG*. Viçosa: Universidade Federal de Viçosa, 2013. Citado na página 21.

SOUZA, J. W. D. T. P. Desenvolvimento de aplicação web com framework javaserver faces e hibernate. 2014. Acesso: 14 de abril de 2018. Disponível em: <http://web.unipar.br/~seinpar/2014/artigos/graduacao/Tiago_Peres_Souza.pdf>. Citado 3 vezes nas páginas 30, 31 e 33.

TARGETWARE. *IntelliJ IDEA*. 2017. Acesso: 05 de maio de 2018. Disponível em: <http://www.software.com.br/p/intellij-idea>. Citado 2 vezes nas páginas 48 e 49.

TECHSOUP. *Microsoft Visio*: A melhor ferramenta para criação de diagramas. 2017. Acesso: 05 de maio de 2018. Disponível em: <https://www.techsoupbrasil.org.br/node/11863>. Citado na página 47.

TECNOLOGY, P. *Why PrimeFaces?* 2014. Acesso: 10 de abril de 2018. Disponível em: <http://www.primefaces.org/whyprimefaces.html>. Citado na página 39.

VENTURINI, K. R. d. C. M. D. *Desenvolvimento web utilizando Primefaces*. 2011. Paranavaí. Acesso: 10 de abril de 2018. Disponível em: <http://web.unipar.br/~seinpar/artigos/Danilo-Venturini.pdf>. Citado na página 39.

VESPA, T. G. *MySQL Workbench*. 2010. Acesso: 05 de maio de 2018. Disponível em: <https://thiagovespa.com.br/blog/2010/09/18/mysql-workbench/>. Citado na página 49.

VUKOTIC, A. M. Pro spring 2.5. In: ____. [S.l.]: São Paulo-SP, 2009. p. 3–5 e 38. Citado na página 37.

WALLS, C. Spring in action. In: ____. [S.l.]: CT USA, 2008. p. 5–6 e 278. Citado na página 38.

ZEMEL, T. *Web Design Responsivo*: páginas adaptáveis para todos os dispositivos. [S.l.]: São Paulo, 2013. Citado na página 44.