INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

DOUGLAS KARCZESKI

DESENVOLVIMENTO DE UM SISTEMA GERADOR DE ORÇAMENTOS PARA A CONSTRUÇÃO CIVIL

Jorge Luis Boeira Bavaresco

PASSO FUNDO
2017

DOUGLAS KARCZESKI

DESENVOLVIMENTO DE UM SISTEMA GERADOR DE ORÇAMENTOS PARA A CONSTRUÇÃO CIVIL

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Jorge Luis Boeira Bavaresco

PASSO FUNDO

2017

DOUGLAS KARCZESKI

DESENVOLVIMENTO DE UM SISTEMA GERADOR DE ORÇAMENTOS PARA A CONSTRUÇÃO CIVIL

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Jorge Luis Boeira Bavaresco

_____

Jair José Ferronato

_____

Maikon Cismoski dos Santos

_____

Adilso Nunes de Souza

PASSO FUNDO
2017

DEDICATÓRIA

Aos meus pais,
pelo incentivo e pelo apoio constante.
À minha esposa,
pelo amor e compreensão nos momentos de dificuldade e ausência.
Aos professores,
pela dedicação e competência em transmitir seus conhecimentos.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais, Marlene Terezinha Leal e Enio Luiz Karczeski, pelo apoio, carinho e educação, que permitiram que eu chegasse até esse momento.

Agradeço a minha querida esposa, Gabriela Modler Karczeski, por contribuir com seu carinho, compreensão e motivação para que o objetivo fosse alcançado.

Agradeço ao professor Jorge Luís Boeira Bavaresco que com seu conhecimento e experiência, contribuiu imensamente na execução deste projeto.

Agradeço aos meus colegas pela amizade, pelas trocas de experiências e conhecimentos que contribuiriam com nossas trajetórias durante o curso.

Por fim, agradeço aos demais professores e funcionários do IFSUL por proporcionarem as condições necessárias para o que ensino e o aprendizado sejam de qualidade nesta instituição.

EPÍGRAFE

“A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê. ”

Arthur Schopenhauer

RESUMO

A área da construção civil possui várias demandas, sendo uma delas a simulação e geração de relatórios de orçamentos com mais agilidade. Para isso buscou-se o desenvolvimento de uma solução que seja acessível por meio de navegadores web, com o objetivo de otimizar a geração de orçamentos. Para a redução da margem de erro em relação aos orçamentos, buscou-se a partir do presente estudo, soluções para otimizar o processo de cálculo de insumos necessários para determinada obra, por intermédio de padrões de serviços por m². Com a implementação da aplicação, pode-se criar orçamentos com base nos dados de entrada selecionados pelo usuário, sendo necessário um banco de dados com todas as informações relacionadas aos insumos, possibilitando a geração de orçamentos e relatórios, os quais serão gerados com o auxílio da biblioteca *JasperReports*. O sistema web foi desenvolvido utilizando a plataforma Java EE, disponibilizando os recursos por meio de containers EJB e interface com *layout* responsivo, elaborado com o auxílio das bibliotecas *PrimeFaces* e *BootsFaces*. Também se utilizou a API JPA com o servidor de persistência *Hibernate*, o *framework* de componentes MVC (modelo-visão-controlador) *JavaServer Faces* baseado na linguagem Java, e o banco de dados objeto relacional *PostgreSQL*. Com o sistema web desenvolvido, constatou-se a sua contribuição efetiva na realização dos cálculos que definem os insumos necessários para determinados tipos de serviços, podendo-se visualizar os resultados ou gerar relatórios de forma ágil no momento que precisar.

Palavras-chave: Sistema web. Construção Civil. Java. Responsivo.

ABSTRACT

The construction sector has several demands, one of them being the simulation and generating reports of budgets with more agility. In order to do this, we sought to develop a solution that is accessible through web browsers, in order to optimize the generation of budgets. In order to reduce the margin of error in relation to budgets, we sought from the present study, solutions to optimize the process of calculating the inputs required for a particular work, by means of service standards per m². With the implementation of the application, you can create budgets based on the input data selected by the user, requiring a database with all the information related to the inputs, enabling the generation of budgets and reports, which will be generated with the aid From the JasperReports library. The web system was developed using the Java EE platform, providing the resources through EJB containers and interface with responsive layout, elaborated with the help of the PrimeFaces and BootsFaces libraries. We also used the JPA API with the Hibernate persistence server, the JavaServer Faces MVC (model-view-controller) component framework based on the Java language, and the PostgreSQL relational object database. With the developed web system, it was verified its effective contribution in the accomplishment of the calculations that define the necessary inputs for certain types of services, being able to visualize the results or generate reports of agile form in the moment that needs.

Keywords: Web system. Construction. Java. Responsive.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT - Associação Brasileira de Normas Técnicas

AJAX - *Asynchronous Javascript and XML*

API - *Application Programming Interface*

C ++ - *C plus plus*

CM - Centímetro

CRUD - *Create, Read, Update and Delete*

CSV - *Comma-separated values*

DAO - *Data Access Object*

E - Espessura

EJB - *Enterprise JavaBeans*

EPC - Equipamento de proteção coletivo

EPI - Equipamento de proteção individual

H - Hora

HTML - *Hypertext Markup Language*

IFSUL - Instituto Federal Sul-rio-grandense

JAVA EE - *Java Enterprise Edition*

JAVA ME - *Java Micro Edition*

JAVA SE - *Java Standard Edition*

JDBC - *Java Database Connectivity*

JPA - *Java Persistence API*

JPQL - *Java Persistence Query Language*

JSF - *JavaServer Faces*

JSP - *JavaServer Pages*

JTA - *Java Transaction API*

JVM - *Java Virtual Machine*

Kg - Quilograma

L - Litro

M - Metro

M² - Metro quadrado

M³ - Metro cúbico

MVC - *Model-View-Controller*

ORM - *Object-relational Mapping*

ODBC - *Open Database Connectivity*

ODT - *OpenDocument format*

PDF - *Portable Document Format*

PDL - *Page Description Language*

QRP - *Quick Report Preview*

RTF - *Rich Text Format*

SGBD - Sistema Gerenciador de Banco de Dados

SQL - *Structured Query Language*

TCC - Trabalho de Conclusão de Curso

TCPO - Tabela de composição de preços para orçamentos

TXT - Extensão Arquivo de texto.

UI - *User Interface*

UML - *Unified Modeling Language*

UN - Unidade

URL - *Uniform Resource Locator*

XHTML - *eXtensible Hypertext Markup Language*

XLS - Extensão Microsoft Excel.

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A tecnologia está cada vez mais presente no dia a dia das pessoas, para os mais diversos tipos de atividades, e com a evolução tecnológica também surgem novas formas de utilizar essa tecnologia em benefício dos trabalhadores da construção civil. Os engenheiros e construtores brasileiros ainda gastam muito tempo para realizar orçamentos, pois além de realizar o levantamento inicial que é imprescindível para qualquer obra, esses profissionais geralmente realizam os cálculos dos insumos necessários manualmente, demandando tempo e custo.

Em razão disso chegou-se a seguinte questão: Como otimizar a simulação e geração de orçamentos para a construção civil buscando reduzir a margem de erro por intermédio de padrões de serviços para a realização de orçamentos? Para contribuir na resolução desse problema, desenvolveu-se um sistema web, que é capaz de simular e gerar relatórios de orçamentos, a partir dos dados informados pelo usuário, orçamentos para reformas e construções de pequeno porte na área da construção civil.

Uma aplicação acessível por diversos tipos de dispositivos pode facilitar o processo de geração de um orçamento, auxiliando na realização de cálculos que definem os insumos necessários para determinados tipos de serviços, possibilitando ao cliente participar das simulações geradas e definir qual orçamento será executado.

Para desenvolver este trabalho foram efetuados estudos sobre as tecnologias de desenvolvimento para sistemas web com a linguagem de programação Java que se aplicariam melhor aos objetivos propostos. Posteriormente, foi realizada a modelagem do sistema para que se tornasse possível a visualização da estrutura a ser implementada, considerando as necessidades identificadas. No final, foi desenvolvida a aplicação web com as características definidas na modelagem, objetivando atender as necessidades apuradas.

1.1 OBJETIVOS

Nesta seção serão apresentados os objetivos deste trabalho.

1.1.1 Objetivo geral

Este projeto de pesquisa tem como objetivo desenvolver uma aplicação web para otimizar o processo de geração de orçamentos para a construção civil, utilizando padrões de serviços por m², com enfoque em obras de pequeno porte.

1.1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

a) Realizar a modelagem de uma aplicação, que esteja acessível através de navegadores de internet, que possibilite realizar orçamentos quando necessitar;
b) Auxiliar na resolução de cálculos, através da otimização do processo de simulação e geração de orçamentos para profissionais da construção civil.
c) Criar padrões de serviços na base de dados, para que se seja possível realizar os cálculos dos insumos necessários para determinada obra.
d) Realizar o desenvolvimento da aplicação modelada para a geração de orçamentos, utilizando os conceitos de programação orientada a objetos para aplicações responsivas.

2 REFERENCIAL TEÓRICO

Nesta seção serão apresentadas algumas tecnologias e metodologias que servirão de base para o desenvolvimento do sistema gerador de orçamentos baseado na linguagem Java.

2.1 ORÇAMENTO

Segundo Pieter e Vaart (apud SANTOS; JUNGLES, 2008), o orçamento de uma obra pode ser definido como uma estimativa ou previsão expressa em termos quantitativos físicos ou monetários que visa auxiliar o gerenciamento e a tomada de decisões seja para a empresa como um todo ou apenas para uma obra. Os quantitativos físicos referem-se, por exemplo, às quantidades de materiais de escritório, materiais de construção, horas de mão de obra, horas de equipamentos. Já os quantitativos monetários referem-se às receitas, despesas, custos, recebimentos e desembolsos.

Um dos fatores primordiais para um resultado lucrativo e o sucesso do construtor é uma orçamentação eficiente. Quando o orçamento é malfeito, fatalmente ocorrem imperfeições e possíveis frustrações de custo e prazo. Aliás, geralmente erra-se para menos, mas errar para mais tampouco é bom. (MATTOS, 2006, p. 22).

De acordo com Prado (*apud* SANTOS; JUNGLES, 2008*),* é comum a empresa de construção civil realizar um orçamento geral, por mais que seja por serviços e preços unitários, sem saber quando realmente determinado insumo ou serviço será efetivamente realizado dentro do canteiro de obras.

Conforme Gehbauer (*apud* SANTOS; JUNGLES, 2008), com o planejamento estruturado da obra, é possível realizar o orçamento operacional do empreendimento. Esse orçamento é vinculado ao planejamento do empreendimento, seguindo a mesma estrutura de codificação. A área de compras tem a função de cotar os materiais de acordo com os projetos do empreendimento.

2.1.1 Atributos do Orçamento

De acordo com Mattos (2006), os principais atributos do orçamento são aproximação, especificidade e temporalidade.

a) Aproximação

Por basear-se em previsões, todo orçamento é aproximado. O orçamento não tem que ser exato, porém preciso. Ao orçar uma obra, o orçamentista não pretende acertar integralmente o valor, mas não se desviar muito do que efetivamente irá custar.

A aproximação de um orçamento está embutida em diversos itens:

- ***Mão-de-obra:***
    - Produtividade das equipes - a produtividade afeta diretamente a composição de custo;
    - Encargos sociais e trabalhistas – o percentual de encargos que incidem sobre a mão-de-obra.
- ***Material:***
    - Preço dos insumos – os preços dos insumos orçados nem sempre serão os praticados na obra;
    - Impostos – os impostos embutidos no preço dos insumos podem variar durante a obra;
    - Perda – o percentual de perda e desperdício é arbitrado para cada insumo que entra no orçamento. Deve-se considerar as perdas na quantidade de materiais;
    - Reaproveitamento – consiste em quantas vezes o insumo poderá ser reutilizado.
- ***Equipamento:***
    - Custo horário – o custo horário depende de parâmetros de cálculo como, vida útil, custo de manutenção e operação etc.;
    - Produtividade – depende da disponibilidade mecânica (percentual de tempo em que o equipamento está em condições mecânicas de

ser utilizado) e do coeficiente de utilização (percentual do tempo disponível em que o equipamento efetivamente trabalha).

- ***Custos indiretos:***
    - Pessoal – salários e encargos sociais das equipes técnica, administrativa e de apoio;
    - Despesas gerais – contas de água, luz, telefone, aluguel de equipamentos gerais, seguros, fretes etc.;
    - Imprevistos – Retrabalho por causa da chuva, refazimento por má qualidade, danos causados por fenômenos naturais ou por terceiros, danos causados pela construtora a terceiros etc.

b) Especificidade

O orçamento para a construção de uma casa em uma cidade é diferente do orçamento de uma casa igual em outra cidade, não se pode falar em orçamento padronizado ou generalizado. Por mais que um orçamentista se baseie em algum trabalho anterior, é sempre necessário adaptá-lo à obra em questão.

c) Temporalidade

Um orçamento realizado tempos atrás não é válido hoje. Se, por exemplo, alguém orçou uma obra, mas a obra só vier a ser mobilizada um ano depois, é lógico perceber que alguns ajustes precisam ser feitos. Isso deve-se à:

- Flutuação no custo dos insumos ao longo do tempo;
- Criação ou alteração de impostos e encargos sociais e trabalhistas, tanto em espécie quanto em alíquota;
- Evolução dos métodos construtivos – surgimento de técnicas, materiais e equipamentos mais adequados;
- Diferenciação de cenários financeiros e gerenciais – terceirização, delegação de tarefas, condições de capital de giro, necessidade de empréstimo etc.

2.1.2 Composição de custo unitário

Conforme Mattos (2006) dá-se o nome de composição de custos ao processo de estabelecimento dos custos incorridos para a execução de um serviço ou atividade, individualizado por insumo e de acordo com certos requisitos pré-estabelecidos. A composição lista todos os insumos que entram na execução do serviço, com suas respectivas quantidades, e seus custos unitários e totais.

Segundo Mattos (2006), ela é constituída de cinco colunas:

- Insumo – é cada um dos itens de material, mão de obra e equipamento que entram na execução direta do serviço;
- Unidade – é a unidade medida do insumo. Quando se trata de material, pode ser Kg, m³, m², m, un, entre outras; para a mão de obra, a unidade pode ser hora ou m²; para equipamento, hora (de máquina);
- Índice – é a incidência de cada insumo na execução de uma unidade de serviço;
- Custo unitário – é o custo de aquisição ou emprego de uma unidade de insumo;
- Custo total – é o custo total do insumo na composição de custos unitários. É obtido pela multiplicação do índice pelo custo unitário. A somatória dessa coluna é o custo total unitário do serviço.

A Tabela 1 mostra um exemplo de composição de custo unitário:

Tabela 1 - Exemplo de composição de custo unitário

Insumo	Unidade	Índice	Custo unitário (R$)	Custo total (R$)
Ajudante de carpinteiro	H	1,20	4,20	5,04
Carpinteiro	H	1,20	6,90	8,28
Chapa compensada	m²	0,43	10,00	4,30
Desmoldante	L	0,10	7,00	0,70
Prego 18 x 27	kg	0,25	4,00	1,00
Pontalete 3” x 3”	M	2,00	2,00	4,00
Sarrafo 1” x 4”	M	1,53	1,00	1,53
Tábua 1” x 12”	M	1,40	5,00	7,00
Total				*31,85*

Fonte: Mattos, 2006, p.67

Essa formatação de composição de custos servirá de base para montar as tabelas do banco de dados do sistema, assim têm-se as referências de unidade para cada tipo de insumo, com isso pode-se gerar os cálculos relacionando as tabelas para gerar os relatórios para o usuário.

2.2 JAVA

Em 1991 a Sun Microsystems reconheceu o avanço significativo que os microprocessadores sofreram e financiaram um projeto de pesquisa corporativa que resultou em uma linguagem baseada em C++. No primeiro momento seu criador James Gosling chamou-a de Oak, em homenagem a uma árvore de carvalho vista pela sua janela na empresa, soube-se posteriormente que já existia uma linguagem de programação com esse nome. Então, na ocasião em que a equipe da Sun encontrou-se em uma cafeteria local, foi sugerido o nome Java, pois era nome da cidade de origem do café importado apreciado por eles (DEITEL e DEITEL, 2010).

O projeto de pesquisa da Sun passou por algumas complicações, pois o mercado não estava se desenvolvendo conforme o previsto. No entanto, em 1993, houve a explosão em popularidade da web, com isso o projeto da Sun teve nova vida, pois constatou-se o potencial de utilizar o Java para adicionar conteúdo dinâmico, como interatividade e animações às páginas web (DEITEL e DEITEL, 2010).

Em 1995, o Java foi anunciado formalmente pela Sun. No primeiro momento o Java chamou atenção pelo seu imenso interesse na web. Atualmente, o Java é utilizado para o desenvolvimento de aplicativos corporativos, essa linguagem de programação possui como objetivo, aprimorar a aplicabilidade de servidores web, fornecer aplicativos direcionados para o consumo popular entre tantos outros propósitos.

Conforme Oracle (2016), a linguagem de programação Java é uma linguagem de alto nível, que se destaca pelas seguintes características:

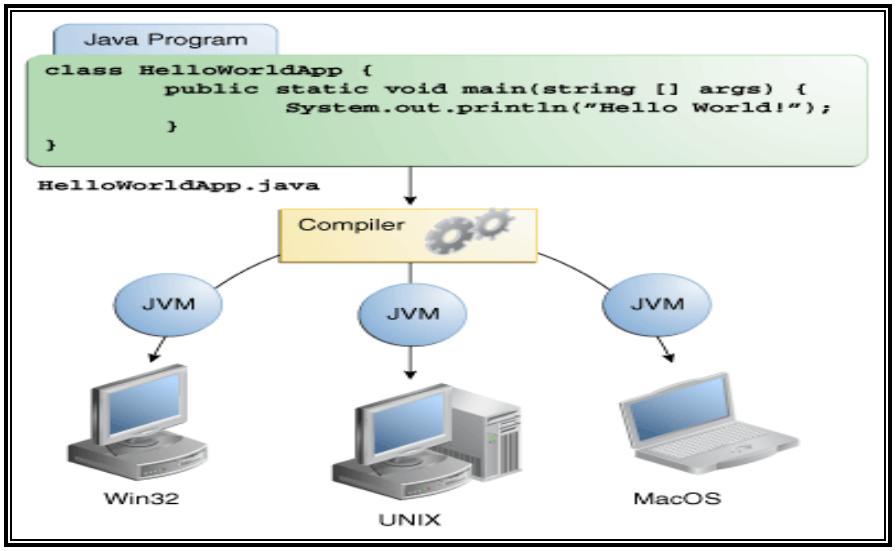
- Orientada a objeto;
- Distribuída;
- Multithreading;
- Dinâmica;

- Arquitetura Neutra;
- Portátil;
- Alta Performance;
- Robusto e seguro.

Na linguagem Java, os códigos fonte são escritos inicialmente em arquivos de texto simples que possuem a extensão *.java*. Esses arquivos gerados inicialmente serão compilados para a extensão *.class* pelo compilador *javac*. O arquivo *.class* não possui código nativo, possui *bytecodes*, a linguagem de máquina do *Java Virtual machine* (Java VM). Posteriormente o Java *launcher tool*, executa a aplicação com uma instância da JVM.

A execução do Java não está diretamente relacionada com o Sistema Operacional, ele conversa diretamente com a *Java Virtual Machine* (JVM), possibilitando assim a portabilidade de seu código. O que for escrito em um sistema operacional Windows, irá rodar em um sistema operacional Linux. Esse processo cria uma independência do Sistema Operacional, dando ao desenvolvedor uma liberdade de desenvolver para múltiplas plataformas sem aquela preocupação de se o código irá funcionar corretamente. A *Virtual Machine* sim é desenvolvida em código nativo, pois ela comunica-se diretamente com o sistema operacional para que o programa Java funcione na máquina. A Figura 1 mostra o esquema de funcionamento da *Java virtual machine*:

Figura 1 - Esquema de Funcionamento da JVM



Fonte: ORACLE, 2016.

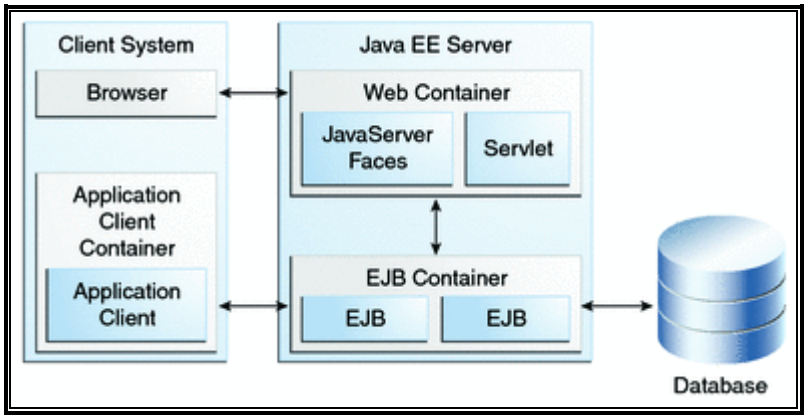
A linguagem Java é muito flexível, desta forma consegue ser multiplataforma. Identificando as necessidades distintas dos desenvolvedores, a Sun elaborou 3 especificações em conformidade com o tipo de plataforma: desktop (*Standard*), corporativa (*Enterprise*) e móvel (*Micro Edition*) (SAMPAIO, 2011).

Grande parte das aplicações constrói-se com a versão *Standard* (Java SE), esta versão dispõe de APIs para resolver inúmeros problemas de programação, desde redes até solicitações remotas. A versão *Enterprise* (Java EE), dispõe de especificações e containers (servidores) que hospedam soluções corporativas. Já a versão Micro (Java ME), dispõe de especificações, perfis de equipamentos e APIs para elaborar programas de plataformas embarcadas ou móveis (SAMPAIO, 2011).

2.3 JAVA ENTERPRISE EDITION

O conceito do *Java Enterprise Edition* (Java EE) é que desenvolvedor possa usufruir de toda a plataforma de serviços e APIs que já existe. O modelo de programação do Java EE é baseado em containers, proporcionando todos os serviços necessários para uma solução corporativa. Por consequência, o desenvolvedor dedica menos tempo para escrever o código, reduzindo o tempo para o desenvolvimento e os riscos do projeto (SAMPAIO, 2011). Na Figura 2 podem ser vistas as camadas do Java EE com separação por conceitos:

Figura 2 - Camadas com separação de conceitos Java EE



Fonte: ORACLE, 2013.

O modelo de containers fornece um ambiente perfeito para as aplicações corporativas, sendo necessário se preocupar somente com as funcionalidades almejadas, deixando os detalhes para os serviços e API do container (SAMPAIO, 2011).

2.4 JAVA PERSISTENCE API

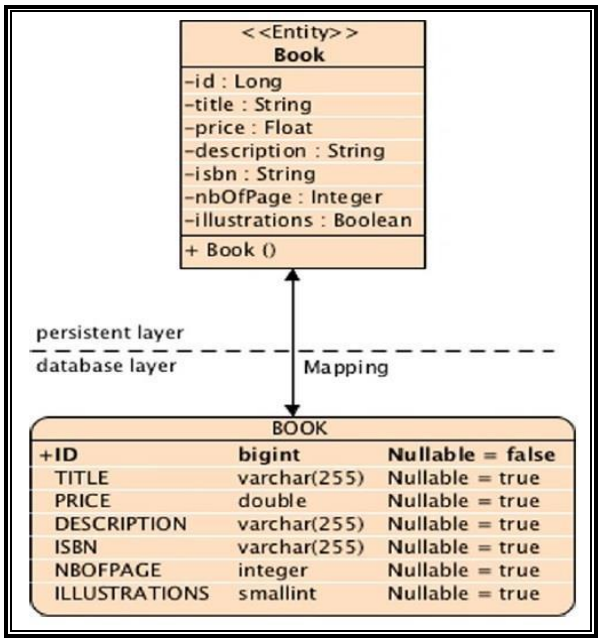
Segundo Gonçalves (2011), a *Java Persistence API* (JPA) foi criada para resolver problemas de persistência de dados no Java EE. Essa interface agrupa os modelos de orientação por objetos e relacional, JPA é uma abstração sobre o JDBC que se torna independente do SQL. Conforme o autor os principais componentes da JPA são:

- O mapeamento objeto relacional (ORM), que possui o objetivo de armazenar dados numa base de dados relacionais;
- Gerenciamento de entidades para executar operações com a base de dados (CRUD);
- A Linguagem de Consulta e Persistência em Java (JPQL), essa linguagem de consulta orientada por objetos viabiliza a recuperação de dados;
- Dispositivo de transação e bloqueio API (JTA), para o acesso de concorrente de dados;
- A inserção de *callbacks* e escutadores para atrelar a lógica funcional no ciclo de vida de um objeto persistente.

De acordo com Gonçalves (2011), a JPA possui métodos para criar correspondências entre objetos e tabelas. Desta forma, as classes, objetos e atributos podem ser mapeados para um banco de dados relacional que possui tabelas, linhas e colunas.

Este mapeamento pode ser visualizado na Figura 3:

Figura 3 - Sincronização dos dados



Fonte: GONCALVES, 2011.

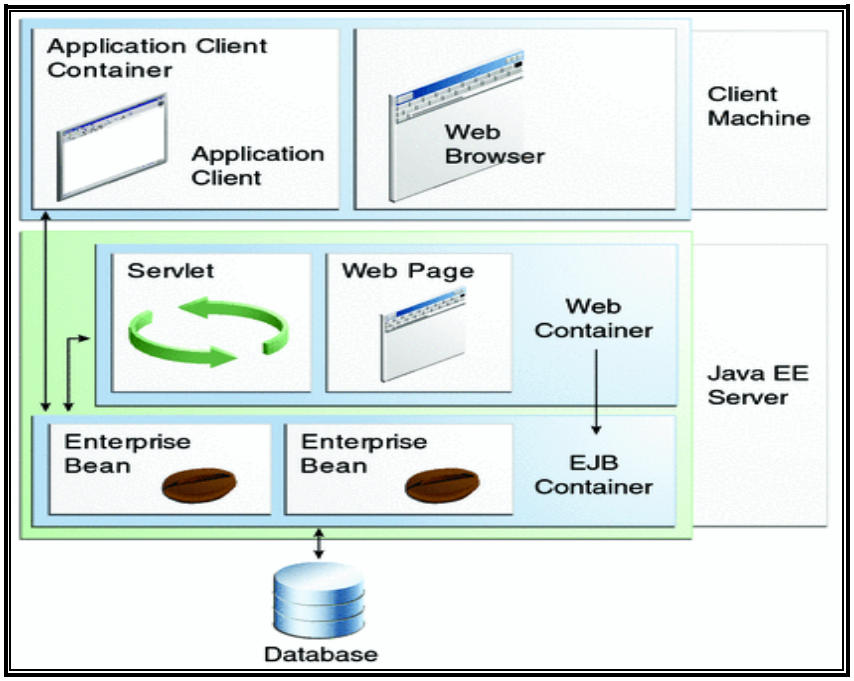
2.5 ENTERPRISE JAVABEANS

O *Enterprise JavaBeans* (EJB) é um componente servidor que executa um container para EJB do servidor de aplicação. Dessa forma, é classificado como um dos componentes mais importantes da plataforma Java EE seu principal objetivo é proporcionar um desenvolvimento ágil de aplicações Java baseadas em componentes, distribuídas, transacionais, seguras e portáveis (GONÇALVES, 2007).

Na obra de Gonçalves (2011), o autor afirma que os componentes EJB's encapsulam a lógica funcional protegendo as transações, a segurança e muitos outros serviços. Devido a facilidade que os EJB's possuem para integrar-se com várias ramificações das tecnologias Java SE e Java EE, por esse motivo são utilizados para a construção de camadas funcionais entre a camada de persistência e a camada de apresentação.

A Figura 4 mostra a arquitetura do Java EE server e seus containers:

Figura 4 - Java EE Server e seus Containers



Fonte: ORACLE, 2013.

- **Servidor Java EE**: Um servidor Java EE fornece containers EJB e web;
- **Enterprise JavaBeans (EJB)**: Gerencia a execução dos *beans* corporativos;
- **Conteiner da Web**: Gerencia a execução das páginas web, *servlets*, e alguns componentes EJB para aplicações Java EE;
- **Conteiner de aplicação do cliente**: Gerencia a execução de componentes na aplicação do cliente;
- **Conteiner de applets**: Gerencia a execução de *applets*. Consiste em um navegador Web e um Java *Plug-in* executando simultaneamente no cliente (ORACLE 2, 2013).

O modelo de programação dos EJB's é eficiente harmonizando simplicidade no seu uso com robustez, esse modelo de desenvolvimento permite a reutilização e escalabilidade do código, através de anotações utilizadas num *conteiner*. No EJB um *conteiner* é um ambiente de tempo de execução que disponibiliza serviços, sendo

eles, gerenciamento de transações, controle de concorrência, agregação e autorização de segurança. Com isso os desenvolvedores conseguem dedicar-se somente na implementação da lógica, durante o tempo que *conteiner* cuida das conexões técnicas (GONÇALVES, 2011).

Segundo Gonçalves (2011) existem os seguintes *beans* de sessão:

- Sem estado: Não possui nenhum estado de conversação para um cliente específico, podendo ser utilizada por qualquer cliente, podendo ser definida pela anotação @*Stateless*;
- Com estado: Possui o estado de conversação para um cliente específico, podendo ser definido usando a anotação @*Stateful*;
- Singular: Contém somente um bean de sessão, fornece facilidade ao acesso concorrente, podendo ser definido através da anotação @*Singleton*.

2.6 BEAN VALIDATION API

Conforme Red Hat (2013), realizar a validação dos dados de uma aplicação é um trabalho habitual para qualquer desenvolvedor. Para auxiliar na realização de validações dos dados, a plataforma Java possui a *Bean Validation API*, essa API é suportada por restrições na forma de anotações colocadas em um campo, método ou classe de um componente *JavaBeans*, como um *bean* gerenciado. Segundo Gonçalves (2013), na *Bean Validation API*, a implementação de uma restrição acontece por meio de uma anotação que pode validar o tipo de um dado, seu tamanho ou sua obrigatoriedade. Na Tabela 2 pode-se visualizar como funcionam algumas anotações e exemplos de implementação:

Tabela 2 - Restrições *Bean Validation API*

Anotações	Descrição	Exemplo
`@AssertFalse`	O valor do campo ou propriedade deve ser *false*.	`@AssertFalse` `Boolean isUnsupported;`
`@AssertTrue`	O valor do campo ou propriedade deve ser *true*.	`@AssertTrue` `Boolean isActive;`
`@Max`	O valor do campo ou propriedade deve ser um valor inteiro menor ou igual ao número no elemento de valor.	`@Max (10)` `Quantidade int;`
`@Min`	O valor do campo ou propriedade deve ser um valor inteiro maior ou igual ao número no elemento de valor.	`@Min (5)` `Quantidade int;`
`@NotNull`	O valor do campo ou da propriedade não deve ser nulo.	`@NotNull` `String username;`
`@Size`	O tamanho do campo ou da propriedade é avaliado e deve corresponder aos limites especificados.	`@Size (min = 2, max = 240)` `String briefMessage;`
`@Pattern`	O valor do campo ou propriedade deve corresponder a expressão regular definida no elemento regexp.	`@Pattern (regexp = "\\ (\\ d {3} \\) \\ d {3} - \\ d {4}")` `String phoneNumber;`

Fonte: ORACLE, 2014.

Como a tabela acima demonstra, também pode-se criar restrições personalizadas, onde o desenvolvedor irá definir as regras de validação dos dados por meio da anotação *@Pattern* (GONCALVES, 2013).

Gonçalves (2013) afirma que, ao construir uma restrição e validar uma determinada estrutura, é possível que se obtenham erros oriundos da validação dos dados. Para isso, a *Bean Validation API* possui mecanismos de gerencia de mensagens, que podem ser disparadas quando um dado não respeita a validação a que foi submetido. As mensagens de validação da API são úteis para informar o usuário de que ocorreu um erro ao validar determinada informação e apresentar ao mesmo a estrutura correta a ser inserida.

2.7 HIBERNATE

Para Gonçalves (2007), o *Hibernate* é uma solução íntegra para as adversidades no gerenciamento de dados persistentes em Java. O *Hibernate* é um framework responsável pelo mapeamento objeto/relacional (ORM) para o banco de dados, possui o objetivo de permitir que o desenvolvedor se concentre somente na lógica de negócio, para que isso seja possível, o desenvolvedor necessita seguir os padrões de desenvolvimento ao implementar a lógica de negócios e suas respectivas classes a persistir.

Conforme Bauer e King (2007), as aplicações precisam persistir os seus dados de alguma forma. Para esse fim, criou-se o *Hibernate* uma implementação de código aberto, para o serviço de ORM. Essa implementação surgiu como uma solução para persistir dados em sistemas de banco de dados orientado à objetos, além disso o *Hibernate* se adapta facilmente a qualquer aplicação, necessitando apenas pequenas modificações para a integração.

2.8 POSTGRESQL

Segundo Neto (2007), o *PostrgreSQL* é um SGBD – Sistema Gerenciador de Banco de Dados, que possui código aberto e implementa os padrões SQL ANSI 92, 96 e 99. O PostgreSQL é um sistema de alto desempenho, que possui a sua administração e utilização em projetos acessíveis.

Além das características citadas anteriormente esse SGBD destaca-se também por possibilitar a utilização de SQL, triggers e todas as funções para a programação e construção, sem deixar a desejar em relação aos SGBD's famosos

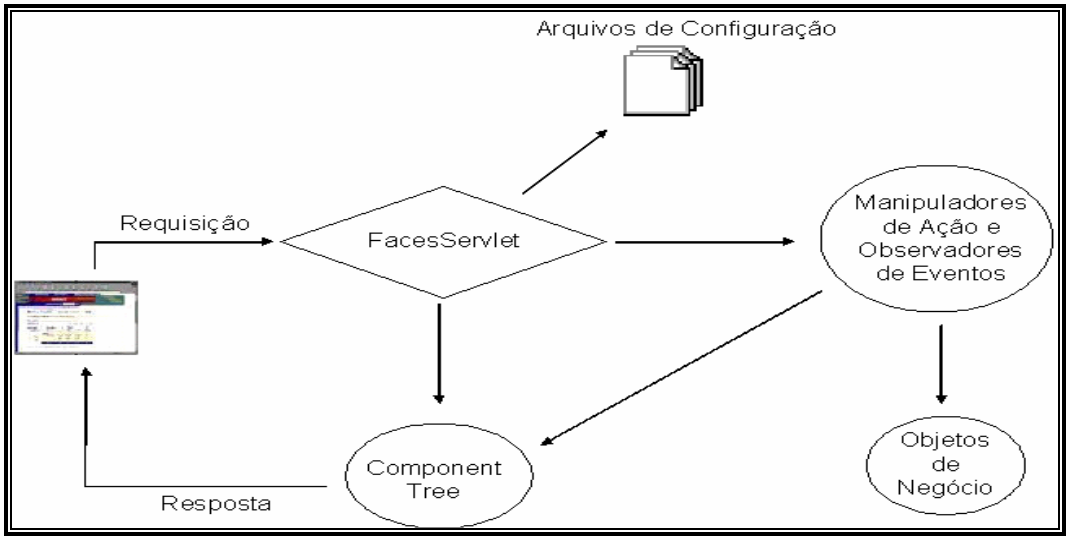
disponíveis no mercado (Oracle, InterBase, SQL Server, etc.), possui também "*Embedded SQL*" pré compilado. Sobre os drivers, dispõe dos drivers ODBC e JDBC para realizar a interface com ambiente e linguagens de programação, entre as mais relevantes temos: Borland Delphi, Borland C++, Perl, XML e Java (NETO, 2007).

De acordo com Silberschatz et al. (2012), o *PostgreSQL* é utilizado para implementar os mais diversos tipos de aplicações. O sistema por ser de código aberto está em plena evolução com mais de 1000 desenvolvedores na sua comunidade.

2.9 MODELO VISÃO CONTROLE

Conforme Gonçalves (2011), o padrão de projeto modelo-visão-controlador (MVC) é um padrão utilizado para separar a lógica funcional da interface de usuário. Com a utilização do MVC pode-se modificar tanto a interface quanto as regras de negócio sem que uma afete a outra, isso torna-se possível devido a separação de conceitos. No MVC, o modelo representa os dados da aplicação, a visualização refere-se à interface com o usuário, e o controlador gerencia a comunicação entre ambos. A Figura 5 mostra a arquitetura do *JavaServer Faces* baseada no modelo MVC:

Figura 5 - Arquitetura JSF baseada no modelo MVC



Fonte: PITANGA, 2017.

O modelo refere-se ao conteúdo, esse conteúdo é regularmente armazenado numa base de dados e apresentado na visualização. No JSF, o modelo pode consistir de beans de suporte, chamadas à EJB's e entidades da JPA. Já a visualização no JSF é a página XHTML, uma visualização fornece uma representação gráfica para determinado modelo. No caso do controlador, este possui a função de coletar, converter e validar os dados vindos das manipulações feitas pelo usuário na visualização, posteriormente chama a lógica funcional e elabora o conteúdo em XHTML (GONÇALVES, 2011).

2.10 JAVASERVER FACES

Segundo Deitel e Deitel (2010), o *JavaServer Faces* (JSF) é um *framework* de aplicativo web que facilita a construção da interface com o usuário e separa a apresentação da sua lógica de negócio. O JSF oferece um conjunto de componentes de interface com o usuário, também oferece duas bibliotecas de *tags* personalizados *JavaServer Pages* (JSP) para inserir esses componentes em uma página JSP. Ainda que os componentes padrão do JSF atendam satisfatoriamente à criação de aplicativos comuns, pode-se criar bibliotecas de componentes personalizadas ou então utilizar bibliotecas desenvolvidas por terceiros disponibilizadas em projetos de código-fonte aberto.

Na opinião Geary (2012), nos dias de hoje existe uma grande variedade de frameworks para o desenvolvimento de interfaces para os usuários de aplicações. O JSF é um *framework* baseado em componentes que permite criar interfaces de usuários em alto nível, tornando-se possível a reutilização de seus componentes.

Os principais componentes do JSF são:

- Conjunto de componentes *User Interface* (UI);
- Modelo de programação orientado à eventos;
- Modelo de componentes que possibilita que desenvolvedores autônomos forneçam seus componentes.

O JSF possibilita que os desenvolvedores se dediquem somente na lógica da aplicação, pois possui todo o código essencial para a manipulação de eventos e organização de componentes. Esse *framework* é a camada de visão padrão do Java EE (GEARY, 2012).

2.11 DESIGN RESPONSIVO

A ideia de Design responsivo baseia-se em criar uma interface com design flexível e adaptável, que se ajuste às características do navegador, do dispositivo e do contexto do usuário. Para que essa adaptação seja eficiente, uma página exibida em um dispositivo mobile não deve ser simplesmente menor, deve ser reestruturada, com isso exibir o conteúdo útil e priorizado (LOPES, 2012).

Com o crescimento da utilização de dispositivos móveis, que dispõem diferentes tamanhos de telas, o mercado web constatou a necessidade de elaborar versões específicas de seus sites para atender a este público. Primeiramente adotou-se a utilização de subdomínios, onde a detecção de um dispositivo móvel pelo site direcionava o usuário a um domínio especializado. Todavia, a criação de designs responsivos mostrou uma nova solução para esta problemática, onde o mesmo site, sem subdomínios, atende a todas as especificações de diferentes dispositivos, sem perder qualquer característica dos elementos visuais implementados na interface (ZEMEL, 2012). A Figura 9 apresenta como o mesmo sistema web, pode ser visualizado satisfatoriamente nos mais diversos dispositivos por uma interface que implementa *layout* responsivo.

Figura 9 - Exemplo com Interface Responsiva

Fonte: ZEMEL, 2012.

Petersson (2014) afirma que, páginas web que não implementam design responsivo acabam “presas” à detecção de dispositivo. Caso a detecção falhe, o usuário poderá visualizar uma página com *layout* inadequado ao seu dispositivo. A detecção do dispositivo é feita por meio de *media queries,* essa ferramenta permite a adaptação do *layout* sem redirecionar a navegação e os testes podem ocorrer em vários instantes, evitando que o usuário seja redirecionado incorretamente para uma página com layout incompatível.

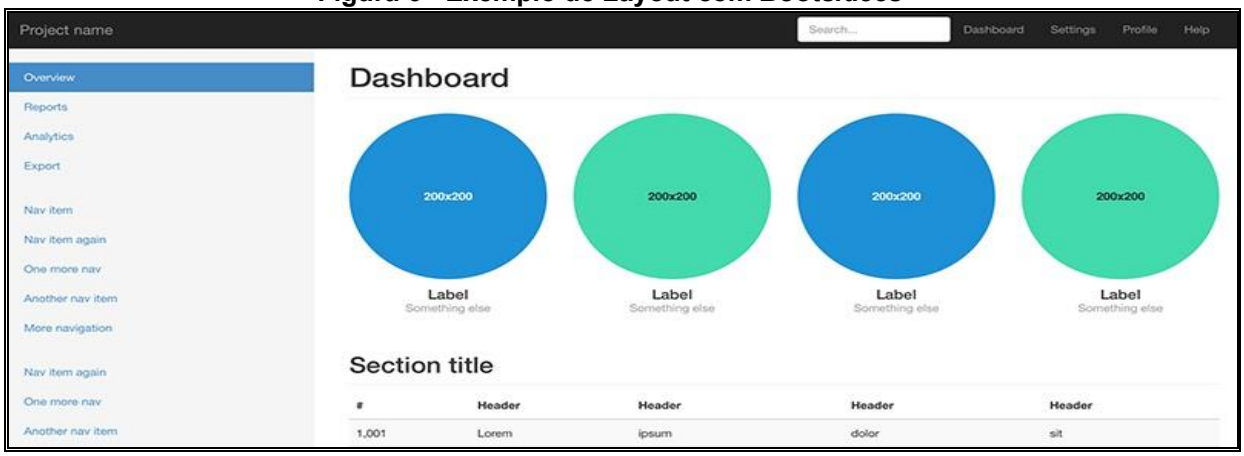
2.12 BIBLIOTECAS DE COMPONENTES

2.12.1 BootsFaces

O *BootsFaces* é uma biblioteca que agrega elementos visuais ao *JavaServer Faces*, baseia-se no estilo visual do *framework Bootstrap* (BOOTSFACES, 2017). O *Bootsfaces* é disponibilizado em um arquivo *.jar*, possui suporte à aplicação de AJAX e permite integração com outros *frameworks* como por exemplo o *PrimeFaces*, o *OmniFaces* e o *ButterFaces* (BOOTSFACES, 2017).

Com a biblioteca *BootsFaces* torna-se possível utilizar componentes previamente preparados para que a interface com o usuário se adapte automaticamente de acordo com o dispositivo utilizado, por exemplo: *desktop ou mobile,* pois, essa biblioteca permite o desenvolvimento de *layout* responsivo para as aplicações (BOOTSFACES, 2017). A Figura 6 apresenta um *template* construído a partir da biblioteca:

Figura 6 - Exemplo de Layout com *Bootsfaces*



Fonte: BOOTSFACES, 2017.

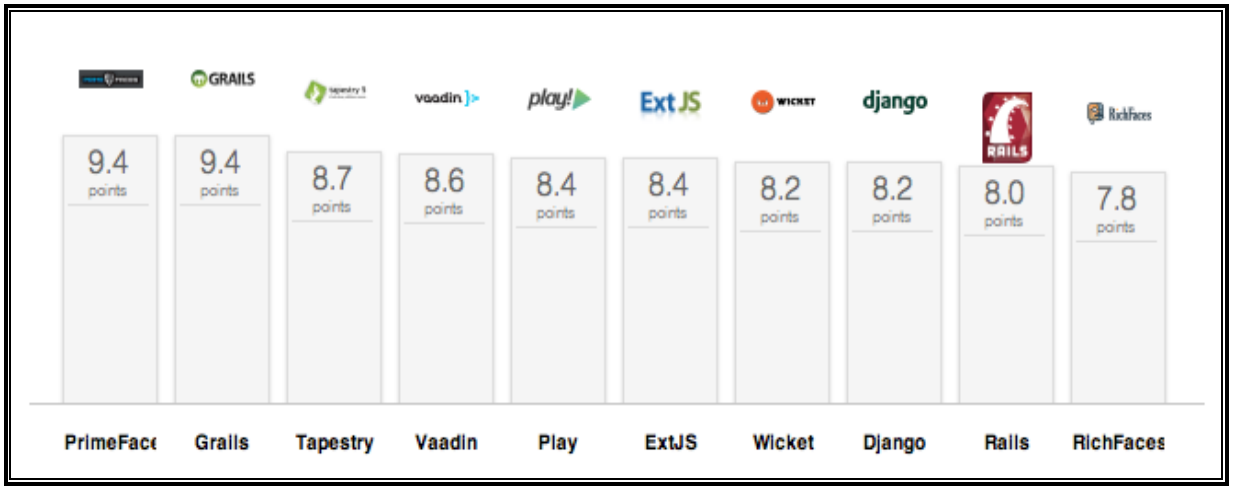
2.12.2 PrimeFaces

O *PrimeFaces* é uma biblioteca de componentes para interfaces gráficas leve e rica em recursos, utiliza-se em aplicações web em conjunto com JavaServer Faces. Essa biblioteca contém entre seus componentes inúmeros campos de entrada, botões, tabelas de dados, árvores, gráficos, diálogos, etc. (FARIA, 2013).

O *PrimeFaces* a exemplo do *BootsFaces* é disponibilizado por meio de um arquivo com a extensão *.jar,* dessa forma não necessita de dependências ou configurações adicionais para trabalhar. Os elementos dessa biblioteca compreendem funcionalidades de *Ajax* integrado por padrão, fundamentado na API de *Ajax* do *JavaServer Faces* (FARIA, 2013).

Com a biblioteca *PrimeFaces* torna-se menos complexa a tarefa de implementar uma interface gráfica para sistemas web baseados em JavaServer Faces, pois com a utilização de seus componentes abstrai-se a complexidade envolvida na criação e organização do design de uma interface com o usuário (PRIMEFACES, 2017). Conforme a DevRates, o *PrimeFaces* assumiu a liderança como o *framework* favorito para criação de interfaces pelos desenvolvedores pesquisados. A Figura 7 nos mostra os *frameworks* favoritos dos desenvolvedores:

Figura 7 - *Frameworks* favoritos dos desenvolvedores



Fonte: PRIMEFACES, 2017.

2.13 JASPER REPORTS

Gonçalves (2009) afirma que, a biblioteca *JasperReports* é uma ferramenta de código aberto para gerar relatórios baseado na linguagem Java. Essa ferramenta possibilita a geração de conteúdos de qualidade na tela, para a impressora, ou em diversos formatos bastante utilizados como PDF, HTML, XLS, RTF, ODT, CSV, TXT e XML. O *JasperReports* não é um dispositivo independente, por isso ele tem que ser integrado às aplicações Java, essa inclusão é feita através da inclusão da sua biblioteca no *classpath* da aplicação, essa biblioteca tem por objetivo inserir a capacidade de geração de relatórios para as aplicações Java.

Seguem as principais características da biblioteca *JasperReports*:

- Possui o layout para relatórios flexível;
- Pode apresentar os dados tanto textualmente quando graficamente;
- O desenvolvedor pode fornecer dados de inúmeras formas;
- Pode aceitar dados de múltiplas fontes de dados;
- Pode gerar marcas d'água;
- Pode gerar sub-relatórios;
- É capaz de exportar relatórios para uma variedade de formatos.

A biblioteca Java *JasperReports* possui todas as funcionalidades necessárias para a geração de relatórios do sistema proposto.

3 METODOLOGIA

Inicialmente foi realizada uma revisão bibliográfica para obter conhecimento em relação aos métodos de modelagem e das tecnologias utilizadas para o desenvolvimento.

Para obter os dados necessários ao desenvolvimento do sistema, foi realizada uma pesquisa aplicada junto a uma organização, da área da construção civil e uma análise nos aplicativos semelhantes.

3.1 ESTUDO DE CASO

O objetivo do trabalho proposto é o desenvolvimento de uma solução para gerar orçamentos para a construção civil. Essa aplicação será responsiva, ou seja, deverá se adequar a qualquer tamanho de tela. Nesta seção será apresentado o estudo do contexto atual, este estudo foi baseado na análise de três aplicações semelhantes existentes no mercado e dos métodos manuais ainda utilizados.

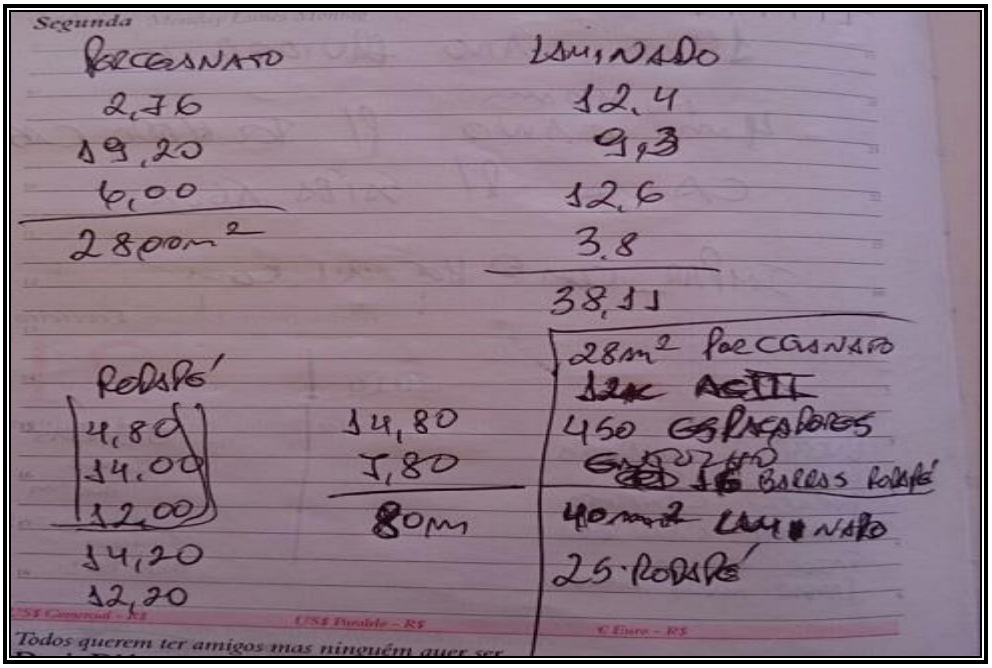
3.1.1 Contexto Atual

Nos dias de hoje é notável a mudança em relação aos métodos de trabalho devido aos avanços tecnológicos, pouco tempo atrás as tarefas eram realizadas manualmente através de máquinas de escrever ou caneta e papel. A tecnologia da informação atual nos permite trabalhar em qualquer lugar como se estivesse no escritório, com isso se ganha tempo e desempenho nas atividades. No contexto da construção civil, muitos construtores ainda fazem o levantamento no local e realizam os cálculos manualmente colocando seus resultados em planilhas eletrônicas ou até mesmo em planilhas manuais.

Nessas planilhas constam a descrição dos insumos necessários para a realização da obra e seus respectivos valores, seguem as formas geralmente utilizadas:

- Nas planilhas feitas à mão, o construtor vai até o local onde será realizada a obra, faz as medições necessárias para a obra em questão, logo após juntamente com o cliente, define-se os tipos de materiais que serão utilizados. Por exemplo, para a construção de uma parede, primeiramente é realizada a medição da área que pretendesse construir, com essa medição define-se a área em m², seguidamente o construtor e o cliente definem os tipos de materiais, para que finalmente o construtor possa realizar os cálculos dos insumos necessários, geralmente com o auxílio de uma calculadora. Com isso pode-se gerar uma planilha manual com os insumos e seus respectivos valores, uma cópia dessa planilha é entregue ao cliente. A Figura 8 mostra o levantamento inicial para gerar o orçamento posteriormente:

Figura 8 - Levantamento inicial do orçamento



Fonte: Do Autor

- Nas planilhas eletrônicas, o processo de levantamento e cálculos é igual ao das planilhas feitas à mão, porém muda-se a forma como os dados são preenchidos. As planilhas eletrônicas exigem o conhecimento da ferramenta.

Figura 9 mostra os dados do orçamento em uma planilha eletrônica:

Figura 9 - Dados do orçamento em planilha eletrônica

Muro lateral 70m² (chapisco e emboço interno e chapisco externo)
1100 tijolos = R$1485,00
6m³ areião = R$810,00
1,5m³ areia média = R$202,50
75sc cimento = R$2062,50
3m³ brita = R$186,00
12 tábuas 30 = R$384,00
24 ripas 7 = R$240,00
3kg arame= R$24,00
5kg prego= R$50,00
Aditivos= R$600,00
59kg aço 4.2= R$225,38
172kg aço 8.0= R$670,80
Total material= R$6.940,18
Mão de obra= R$6.600,00

Fonte: Do Autor

As duas abordagens citadas possuem limitações, pois não possuem nenhum tipo de padronização para a criação de orçamentos.

Com a finalidade de otimizar o processo de geração de orçamentos para a construção civil, surge a ideia do desenvolvimento de uma solução para padronizar esse processo, tornando-o eficiente e seguro.

3.1.2 Aplicativos semelhantes

Atualmente as pessoas buscam cada vez mais otimizar o seu tempo, para isso contam com aplicações que resolvam ou facilitem a realização das suas tarefas. Neste item da pesquisa será feita uma análise das aplicações semelhantes à solução proposta. Serão usados como base de comparação os três sistemas mais semelhantes encontrados, independente das suas plataformas.

Os sistemas semelhantes analisados são: SIG construtora, desenvolvido pela Nova Era Sistemas (NOVA ERA SISTEMAS, 2017); Suite Construção desenvolvido

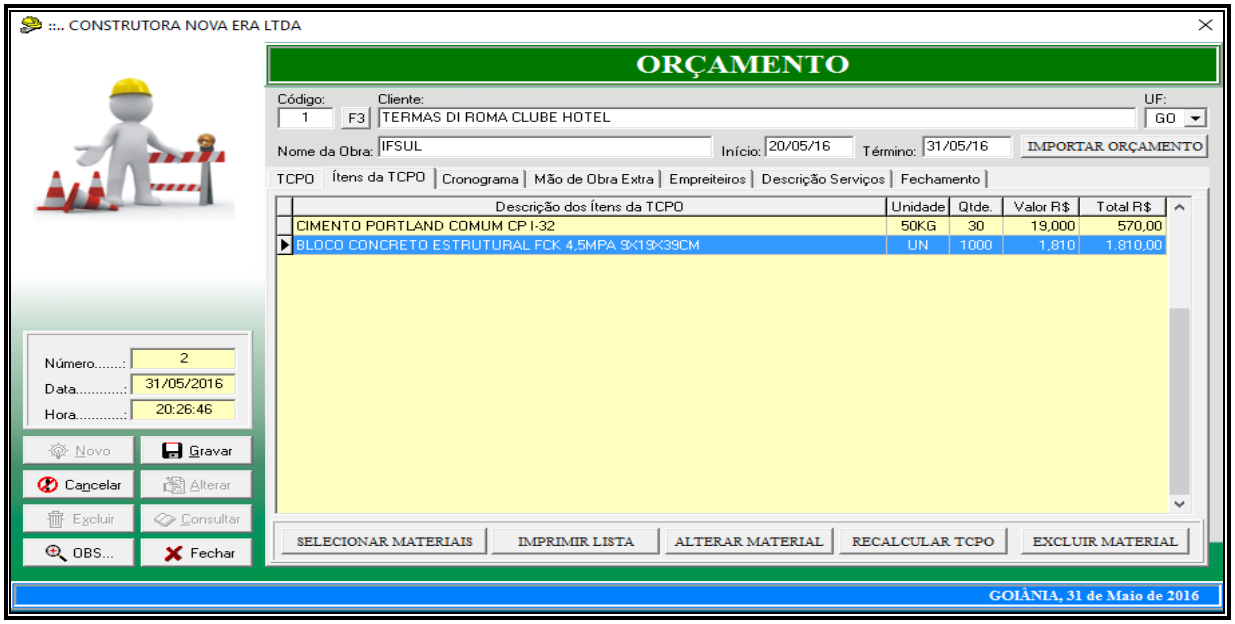
pela PowerDroid, com mais de 50 mil downloads (POWER DROID, 2017); Reforma Simples, desenvolvido pela Catraia Aplicativos, com mais de 10 mil downloads (CATRAIA APLICATIVOS, 2017).

O sistema desenvolvido pela Nova Era o SIG construtora é um sistema completo que possui todos os tipos de controle relacionados a uma obra, esse software só pode ser instalado em desktops ou notebooks, e é independente do seu sistema operacional. O SIG construtora é um dos programas mais utilizados por empresas de pequeno e médio porte na área da construção civil, entre suas principais características estão:

- Controle de acesso ao sistema;
- Cadastros de clientes, funcionários, fornecedores;
- Controle de EPI, EPC e ferramentas;
- Orçamento de obras;
- Controle de estoque;
- Controle de veículos;
- Controle de ponto diário de funcionários;
- Controle de locações de máquinas e equipamentos;
- Controle financeiro;
- Backup de dados;
- Geração de relatórios QRP;
- Pode ser acessado através da internet.

A Figura 10 mostra a tela de criação de orçamento da SIG:

Figura 10 - Tela criação de orçamento SIG



Fonte: NOVA ERA SISTEMAS, 2017

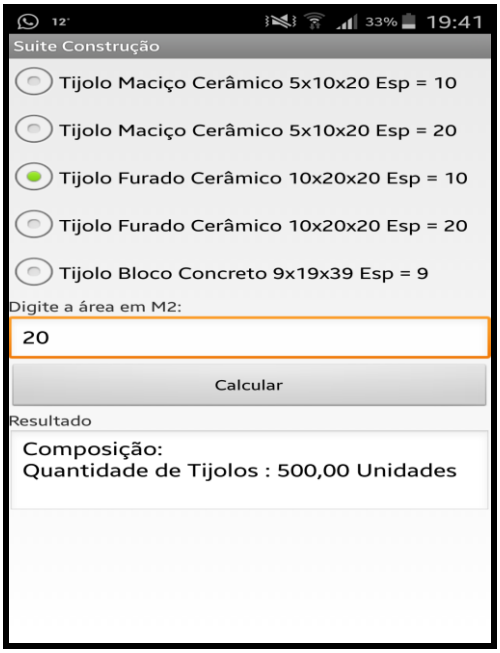
Esse software é instalado em um servidor local e pode ser acessado pela internet. O sistema não é gratuito, pode ser adquirido através de locação mensal ou compra definitiva.

No aplicativo desenvolvido pela PowerDroid, o Suite Construção, pode-se obter a quantidade estimada de materiais por m² de construção, essa aplicação foi desenvolvida para a plataforma Android e pode ser baixada gratuitamente, entre as suas principais características destacam-se:

- Cálculos para concreto;
- Cálculos para assentar tijolos;
- Cálculo de tinta para pintura;
- Cálculo de quantidade de telha;
- Cálculo de quantidade de tijolos.

A Figura 11 mostra a tela de definições e resultados:

Figura 11 - Tela de definições e resultados Suite Construção



Fonte: POWER DROID, 2017

A aplicação Suite Construção não possui controle de acesso e geração de relatórios, ela pura e simplesmente realiza os cálculos de acordo com o valor digitado pelo usuário e mostra o resultado na tela, ou seja, a sua principal função é auxiliar o usuário nos cálculos, pois não possui nenhuma forma de armazenamento dos cálculos realizados.

No aplicativo desenvolvido pela Catraia aplicativos, o Reforma Simples, a exemplo da Suite Construção foi desenvolvido para a plataforma Android e pode ser baixado gratuitamente, com esse aplicativo também pode-se obter a quantidade estimada de materiais por m² de construção, porém o Reforma Simples é uma ferramenta mais completa, entre as suas principais funcionalidades estão:

- Cálculo para Infraestrutura e fundações;
- Cálculo para pisos;
- Cálculo para estrutura;
- Cálculo para paredes;
- Cálculo para acabamentos;
- Cálculo de cobertura.

A Figura 12 mostra a tela de definições e o relatório na tela:

Figura 12 - Telas de definições e relatório Reforma Simples

Fonte: CATRAIA APLICATIVOS, 2017

Assim como na aplicação Suite Construção, o aplicativo Reforma Simples não possui controle de acesso, realiza os cálculos de acordo com o valor digitado pelo usuário, no entanto o Reforma Simples possui ferramentas para gerar relatório, basta informar o e-mail de destino, no e-mail o relatório aparecerá de duas formas, uma como texto simples no corpo do e-mail, e outra como anexo em um arquivo XLS. A tabela 3 traz um comparativo entre os três sistemas analisados:

Tabela 3 - Tabela comparativa dos aplicativos analisados

Atributos analisados	SIG Construtora	Suite Construção	Reforma Simples
Controle de Acesso	Sim	Não	Não
Sistema Operacional	Windows	Android	Android
Armazenamento de dados	Sim	Não	Sim
Padrão de orçamento por m² de serviço	Não	Sim	Sim
Geração de relatórios	Sim, QRP	Não	Sim, e-mail e XLS

Fonte: Do Autor

3.1.3 Proposta da nova solução

Na solução proposta, o levantamento inicial para a realização do orçamento foi realizado da mesma forma que foi descrita no contexto atual, porém quanto à geração de orçamentos a solução desenvolvida possui diferenças significativas em relação aos aplicativos semelhantes analisados. Criou-se um sistema web independente da plataforma, com isso qualquer usuário autorizado e conectado à internet poderá acessar o sistema. O sistema SIG Construção até possui controle de acesso, porém o sistema é instalado localmente, tornando-o refém das plataformas para desktops, não sendo possível a utilização em dispositivos móveis, já nas aplicações para Android, Suite Construção e Reforma Simples não existe nenhuma forma de controle de acesso.

Na geração de orçamentos, o sistema desenvolvido possui uma tela para o preenchimento dos dados de acordo com a obra em questão. Criaram-se padrões de serviços por m², ou seja, o usuário irá informar o cliente, o tipo de serviço e a quantidade em m², consequentemente o sistema irá gerar o orçamento com os insumos necessários e seus respectivos valores, podendo gerar relatório disso quando precisar. O sistema SIG Construtora não possui esse padrão para gerar orçamento por m², o sistema gera orçamentos através da seleção dos insumos diretamente na sua TCPO (Tabela de composição de preços para orçamentos). Já os sistemas para Android, Suite construção e Reforma Simples possuem esse padrão, nessas aplicações o usuário seleciona o tipo de serviço e insere a quantidade em m² e o sistema gera a quantidade de insumos necessários, porém não possuem preços e não é possível gerar relatórios com qualidade, essas ferramentas somente auxiliam na realização dos cálculos.

Em referência à persistência dos dados, a solução desenvolvida irá manter seus dados no SGBD PostgreSQL, entretanto o sistema SIG construtora mantém os dados no seu servidor local, em relação à aplicação Suite Construção não há persistência dos dados ela somente mostra o resultado na tela, porém no aplicativo Reforma Simples os dados ficam salvos no aparelho onde foi instalado.

4 MODELAGEM DO SISTEMA

A partir de uma pesquisa aplicada, juntamente com uma empresa da construção civil e da análise de aplicações semelhantes, tornou-se possível a realização da modelagem da aplicação proposta. Esta solução possui o propósito de otimizar a geração de orçamentos para os profissionais da construção civil.

4.1 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Os requisitos funcionais definem claramente as funções e serviços do sistema. Os requisitos não funcionais definem as propriedades e restrições do sistema, com o objetivo de tornar o sistema seguro e com desempenho satisfatório.

Através do estudo de caso os requisitos funcionais e não funcionais foram definidos.

4.1.1 Requisitos Funcionais

- Manter Cidades;
- Manter Usuários;
- Manter Clientes;
- Manter Insumos;
- Manter Serviços;
- Manter Orçamentos;
- Calcular os insumos necessários para realizar determinada obra e gerar relatório;
- Usar um sistema de *login* e senha para identificar o usuário;
- Exibir relatório.

4.1.2 Requisitos Não Funcionais

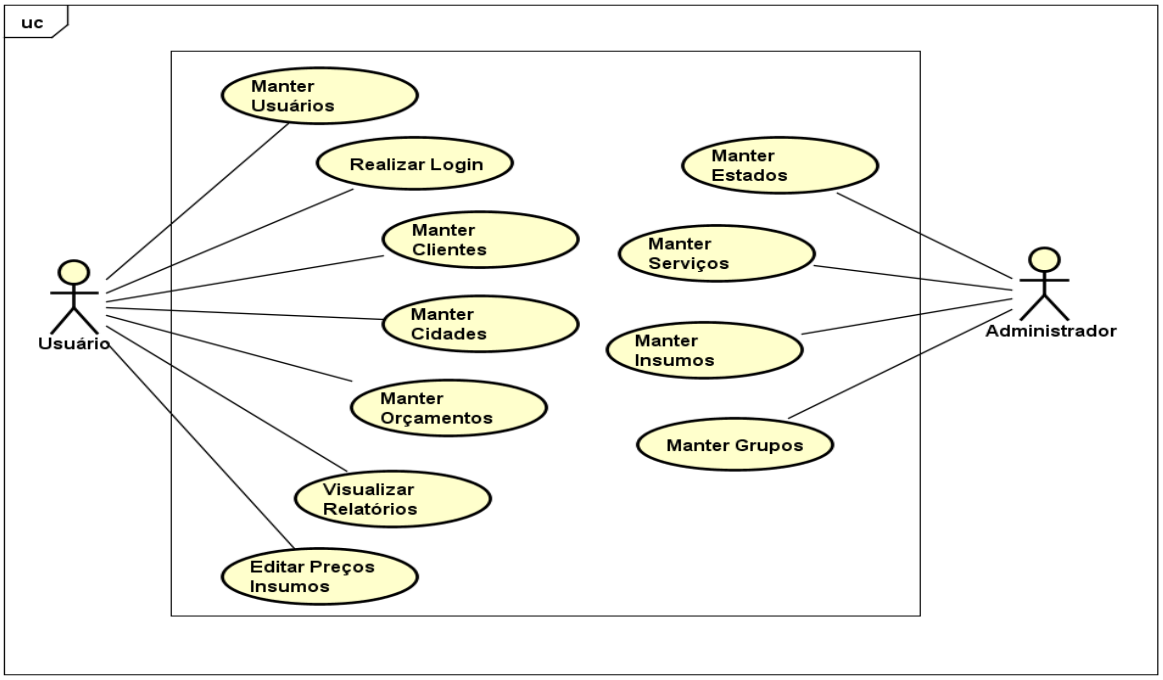
- A aplicação deve ser multiplataforma, deve poder ser utilizado em qualquer plataforma de *hardware* e de *software*;
- A interface da aplicação deve ser acessível e intuitiva;
- O sistema deve tratar acessos não autorizados;
- A persistência das informações deve ser implementada em um Sistema Gerenciador de Bando de Dados.

4.2 DIAGRAMA DE CASOS DE USO

O Diagrama de Casos de Uso tem o objetivo de auxiliar a comunicação entre os analistas e o cliente. Um diagrama de Caso de Uso descreve um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário, ou seja, a interação dessas funcionalidades com os usuários do mesmo sistema.

O diagrama apresentado na Figura 13 representa os requisitos levantados com base no estudo de caso realizado junto ao usuário:

Figura 13 - Diagrama de Casos de uso



Fonte: Do Autor

4.3 DESCRIÇÃO DOS CASOS DE USO

4.3.1 Caso de Uso Manter Usuários

Nesta descrição de caso de uso descreve-se o caso de uso manter usuários, devido à similaridade com os casos uso manter serviços, manter insumos, manter grupos, manter clientes e manter cidades, este caso de uso descreve genericamente todas as ações que serão executadas nas manutenções dos casos de uso citados.

Tabela 4 - Documentação do Caso de Uso Manter Usuários

Nome do caso de uso	**Manter Usuários**
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	
Resumo	Este caso de uso descreve os processos para criar, visualizar, atualizar e excluir os usuários do sistema.
Pré-Condições	Existência de um usuário novo a ser registrado.
Pós-Condições	Novo usuário criado.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Solicitar a criação de um novo usuário.	
	2. Mostrar a tela onde os dados do cadastro serão inseridos.
3. Informar os dados obrigatórios para efetivar o cadastro.	
	4. Persistir o novo usuário no SGBD.
Restrições/Validações/Regras de Negócio	1. Não pode existir um usuário cadastrado com o mesmo CPF.
	2. Para registrar o usuário no sistema, o mesmo deverá estar cadastrado na tabela de pessoa física.
Fluxo Alternativo I – Visualizar usuários cadastrados	
Ações do Ator	**Ações do Sistema**
1. Solicitar a visualização dos	

usuários cadastrados.	
	2. Mostrar a tela com os dados dos usuários cadastrados.
Fluxo Alternativo II – Atualizar os dados do usuário	
Ações do Ator	**Ações do Sistema**
1. Na tela de visualização clicar no botão alterar.	
	2. Mostrar a tela com os dados cadastrados, possibilitando a edição dos campos.
	3. Persistir os dados no SGBD.
Fluxo Alternativo III – Remover usuário	
Ações do Ator	**Ações do Sistema**
1. Na tela de visualização clicar no botão remover.	
	2. Remover o usuário do SGBD.
	3. Mostrar a tela com os usuários cadastrados atualizada.

4.3.2 Caso de Uso Realizar Login

Tabela 5 - Documentação do Caso de Uso Realizar Login

Nome do caso de uso	Realizar Login
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	
Resumo	Este caso de uso descreve os processos para o usuário entrar no sistema.
Pré-Condições	Estar devidamente cadastrado na tabela de usuários e conectado à internet.
Pós-Condições	Login realizado ou não
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Informar login e senha	
	2. Consultar banco de dados
	3. Se o usuário estiver devidamente cadastrado e informar os dados corretamente, é autorizado o acesso ao sistema. Se o usuário não estiver devidamente cadastrado ou não informar o

	login ou senha corretamente será mostrada uma mensagem de erro.
Restrições/Validações/Regras de Negócio	1. O usuário deverá estar devidamente cadastrado.
	2. O usuário deverá digitar os dados de login e senha corretamente.
	3. O preenchimento dos campos de login e senha são obrigatórios.
Fluxo Alternativo I – Usuário não cadastrado	
Ações do Ator	Ações do Sistema
1. Solicitar ao administrador o cadastro.	
Restrições/Validações/Regras de Negócio	1. Não pode haver nenhum usuário com o mesmo nome no sistema.

4.3.3 Caso de Uso Manter Orçamentos

Tabela 6 - Documentação do Caso de Uso Manter Orçamentos

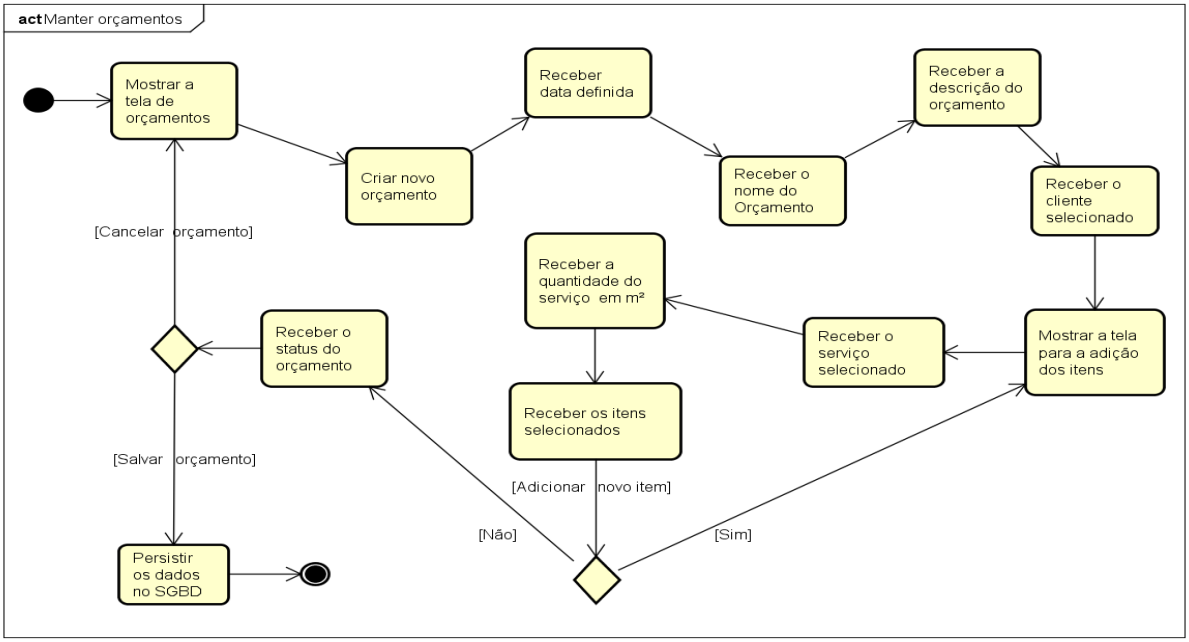
Nome do caso de uso	**Manter Orçamentos**
Caso de Uso Geral	
Ator Principal	Usuário
Atores Secundários	
Resumo	Este caso de uso descreve os processos para a geração de um orçamento.
Pré-Condições	Estar conectado ao sistema.
Pós-Condições	Orçamento gerado.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Solicitar a criação de um novo orçamento.	
	2. Mostrar a tela onde os dados do orçamento serão inseridos.
3. No campo Data definir a data da criação do orçamento.	
4. No campo Nome, inserir o nome da Obra da qual faz parte o orçamento.	
5. No campo Descrição, inserir observações sobre a obra.	
6. Selecionar o cliente.	

7. Na aba Novo Item, adicionar os serviços que farão parte do orçamento.	
8. No campo Status define-se a situação do orçamento, podendo ser “aberto” ou “aprovado”.	
	9.Persistir os dados gerados no SGBD.
Restrições/Validações/Regras de Negócio	1. O cliente deverá estar devidamente cadastrado no SGBD.
Fluxo Alternativo I – Cliente não cadastrado	
Ações do Ator	**Ações do Sistema**
1. Solicitar a criação de um novo cliente.	
	2. Mostrar a tela onde os dados do cliente serão inseridos.
Restrições/Validações/Regras de Negócio	1. Não pode haver nenhum cliente com o mesmo CPF.
Fluxo Alternativo II – Visualizar orçamentos cadastrados	
Ações do Ator	**Ações do Sistema**
1. Solicitar a visualização dos orçamentos cadastrados.	
	2. Mostrar a tela com os dados dos orçamentos cadastrados.
Fluxo Alternativo III – Atualizar os dados do Orçamento	
Ações do Ator	**Ações do Sistema**
1. Na tela de visualização clicar no botão alterar.	
	2. Mostrar a tela com os dados cadastrados, possibilitando a edição dos campos.
	3. Persistir os dados no SGBD.
Fluxo Alternativo IV – Remover Orçamento	
Ações do Ator	**Ações do Sistema**
1. Na tela de visualização clicar no botão remover.	
	2. Remover o orçamento do SGBD.
	3. Mostrar a tela com os orçamentos cadastrados atualizados.

4.4 DIAGRAMA DE ATIVIDADES

Um diagrama de atividade mostra um processo de negócios ou um software como um fluxo de trabalho por meio de uma série de ações. Esse tipo de diagrama possui muitas semelhanças com os antigos fluxogramas, com ele é possível representar a lógica de negócio e o fluxo de controle de uma aplicação. O diagrama de atividades na Figura 14 tem como objetivo demonstrar os fluxos para a geração de um orçamento:

Figura 14 - Diagrama de Atividades



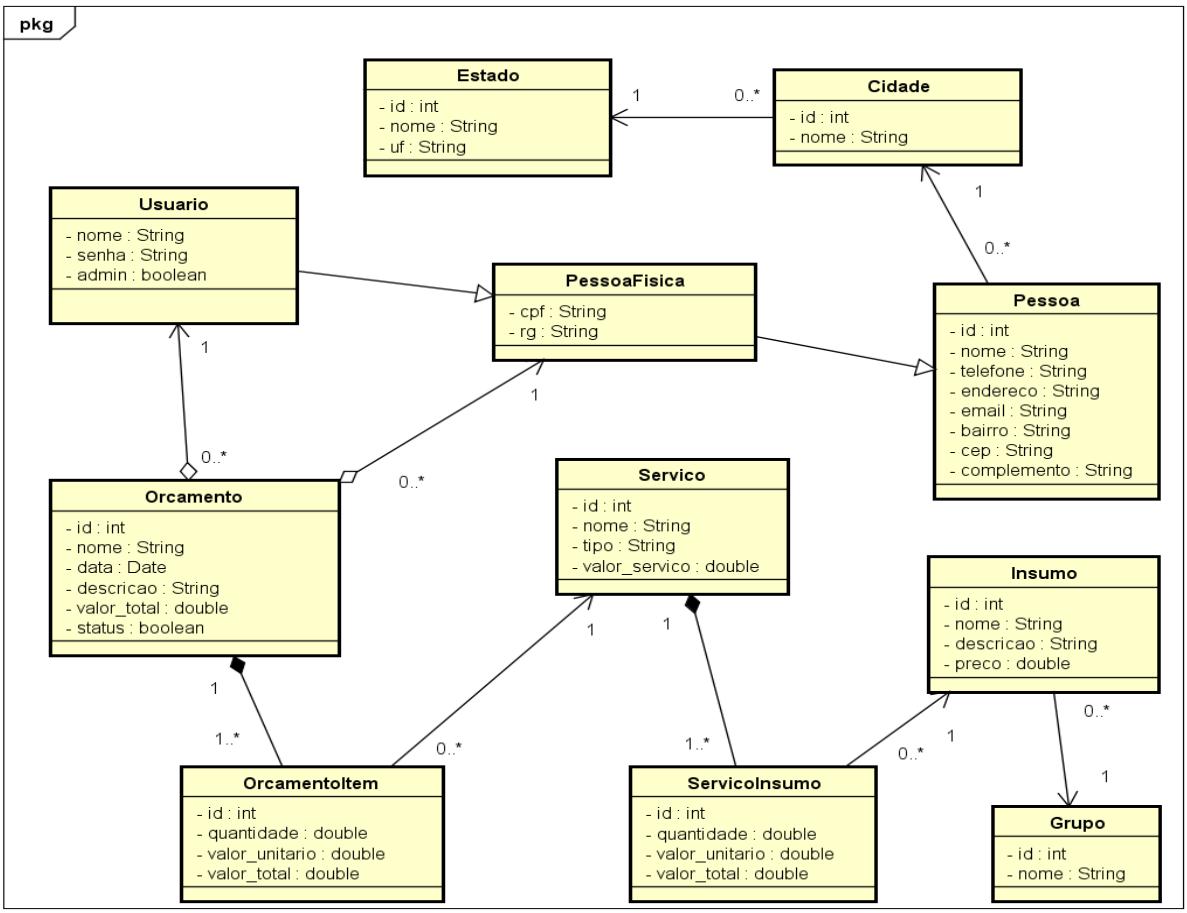
Fonte: Do Autor

O diagrama de atividades acima nos mostra o fluxo de ações necessárias para a geração de um orçamento. A partir da tela de orçamentos, ao usuário clicar no botão novo apresentam-se os campos para que o usuário os preencha de acordo com a obra em questão. Na tela para adicionar os itens do orçamento, adiciona-se o serviço previamente criado e define-se a quantidade em m² do serviço selecionado, posteriormente tem-se a opção de adicionar mais itens ao orçamento, assim que finalizada a adição dos itens pode-se definir os status do orçamento podendo ser definido como “aberto” ou “aprovado”. Por fim o usuário poderá salvar o orçamento, persistindo os dados no SGBD.

4.5 DIAGRAMA DE CLASSES

O diagrama de classes UML descreve o objeto e informações de estruturas usadas pelo aplicativo internamente e comunicação com seus usuários. Ele descreve as informações sem referência a qualquer implementação específica. Com o propósito de persistir os dados, apresenta-se na Figura 15 o diagrama de classes do sistema:

Figura 15 - Diagrama de Classes



Fonte: Do Autor

Primeiramente tem-se a classe “Estado”, esta classe contém os atributos referentes ao cadastro de estados, como o nome do estado e a UF (unidade da federação), no caso da classe “Cidade”, esta classe possui os atributos pertencentes ao cadastro de cidades, ela é composta pelos atributos nome e pelo estado que a cidade pertence.

No caso da classe “Pessoa” apresentam-se os atributos relacionados ao cadastro de pessoas, essa superclasse dispõe dos dados necessários para a subclasse “PessoaFisica”. A subclasse “PessoaFisica” além de herdar os atributos da classe “Pessoa”, possui os atributos CPF (cadastro de pessoa física) e RG (registro geral). Por meio da classe “PessoaFisica” pode-se cadastrar os clientes e os usuários, no caso da classe “Usuario” ela herda todos os atributos relacionados à classe “Pessoa” e a classe “PessoaFisica” e pode-se definir o nome do usuário e a senha, esta classe também possui o atributo administrador, através desse atributo define-se se o usuário possuirá acesso de administrador ou usuário comum.

Com o objetivo de separar os insumos por grupos criou-se a classe “Grupo”, por exemplo: grupo tijolos, grupo areias, grupo cimentos, etc. Essa classe é composta pelo atributo nome, que se refere ao nome do grupo. Essa classe se faz necessária, pois facilita as consultas de insumos. Por conseguinte, tem-se a classe “Insumo”, essa classe contém os atributos referentes ao cadastro de insumos, além dos atributos nome, descrição, preço e possui o grupo do qual faz parte o insumo.

Para que seja possível definir o padrão do serviço por m², criou-se a classe “Servico”, essa classe é composta pelos atributos nome, tipo e valor do serviço, sendo que o atributo nome refere-se ao nome do serviço criado, por exemplo: Parede alvenaria tijolo furado, Parede alvenaria tijolo maciço, Parede bloco de concreto, etc. No caso do atributo tipo refere-se ao tipo de serviço criado, por exemplo: tipo parede, tipo piso, etc. No caso do atributo valor serviço, irá constar o valor da soma dos insumos que irão compor o serviço. Através da classe “ServicoInsumo” define-se a quantidade de insumos que farão parte do serviço por m².

Por último temos as classes “OrcamentoItem” e “Orçamento”, a classe “OrcamentoItem” é constituída pelos itens, no caso os serviços que irão compor o orçamento.

Consequentemente temos a classe “Orcamento”, essa classe possui os atributos referentes ao cadastro de orçamentos, apresenta os atributos nome, data, descrição, valor total e status, no status aceita-se a condição de orçamento “aberto” ou “aprovado”. Na classe “Orcamento” adicionam-se os itens que farão parte do mesmo, com o objetivo de gerar relatório com a discriminação dos insumos necessários e seus respectivos valores.

5 DESENVOLVIMENTO

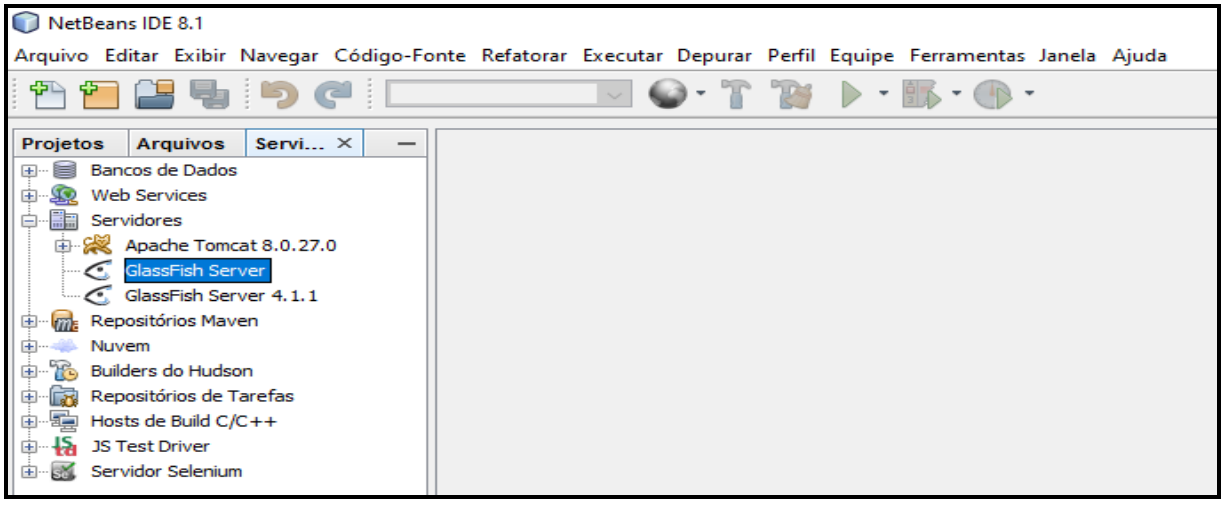
Nesta sessão serão descritos os procedimentos utilizados para o desenvolvimento da solução web, a estrutura em camadas da aplicação e os elementos computacionais envolvidos na sua operação.

5.1 AMBIENTE DE DESENVOLVIMENTO E RECURSOS

5.1.1 IDE e Servidores

Para desenvolver o sistema gerador de orçamentos para a construção civil, utilizou-se uma aplicação IDE e dois servidores, ambos foram necessários para a construção do código e administração o sistema. Como servidor da aplicação utilizou-se o *GlassFish Server* versão 4.0. Esse servidor mantido pela empresa Oracle, possui seu código fonte aberto e suporta todas as especificações da API *Java Enterprise Edition*, como por exemplo: suporte ao *Enterprise JavaBeans*, *Java Persistence API* e *Bean Validation API*, e essas três estruturas citadas foram utilizadas na implementação do projeto. A IDE adotada para implementar a aplicação foi o NetBeans IDE 8.1, ferramenta que traz consigo a possibilidade de incorporar o servidor *GlassFish*. A IDE NetBeans 8.1 e o Servidor *GlassFish* podem ser visualizados na Figura 16:

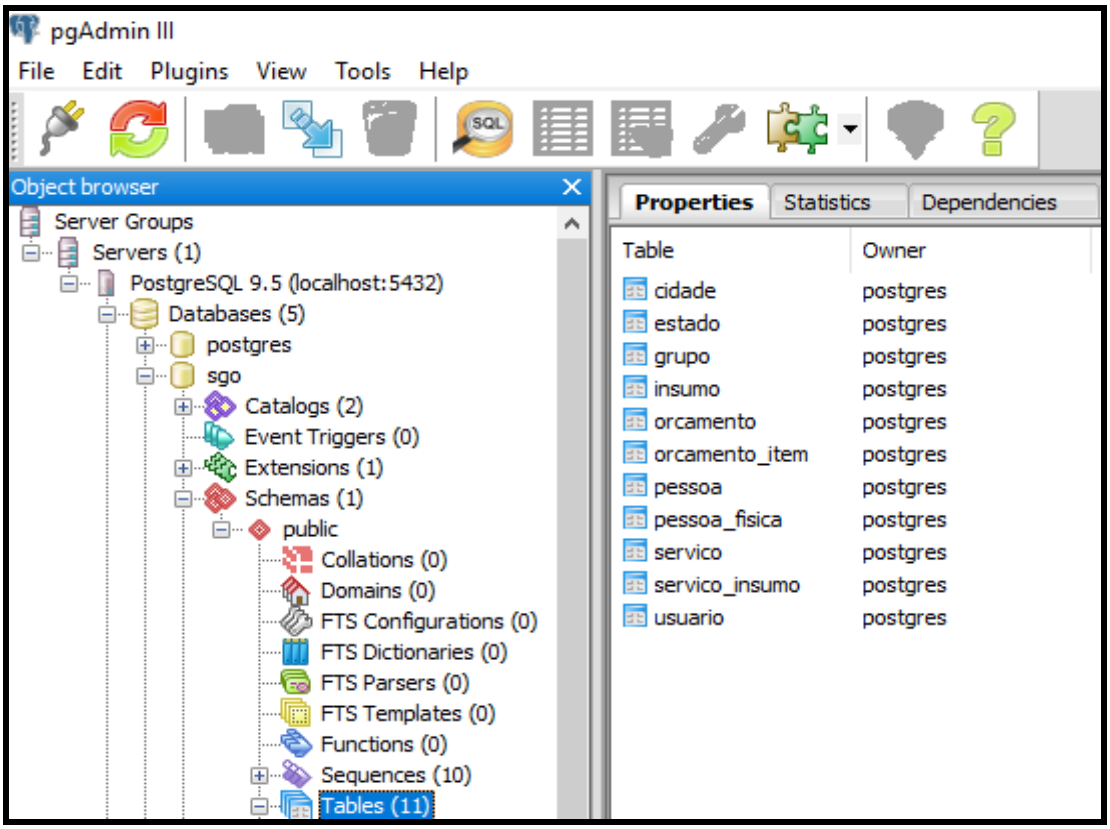
Figura 16 - IDE NetBeans e Servidor GlassFish Server



Fonte: ORACLE, 2017.

Para gerenciar e administrar o banco de dados, utilizou-se o Sistema gerenciador de Banco de Dados *PostgreSQL*. Por meio do servidor desse sistema gerenciador tornou-se possível guardar os dados e a comunicar-se com a aplicação web elaborada. Na Figura 17 é possível visualizar o banco de dados da aplicação através do *software* gráfico PGAdmin III:

Figura 17 - Banco de Dados da Aplicação web



Fonte: POSTGRESQL, 2017.

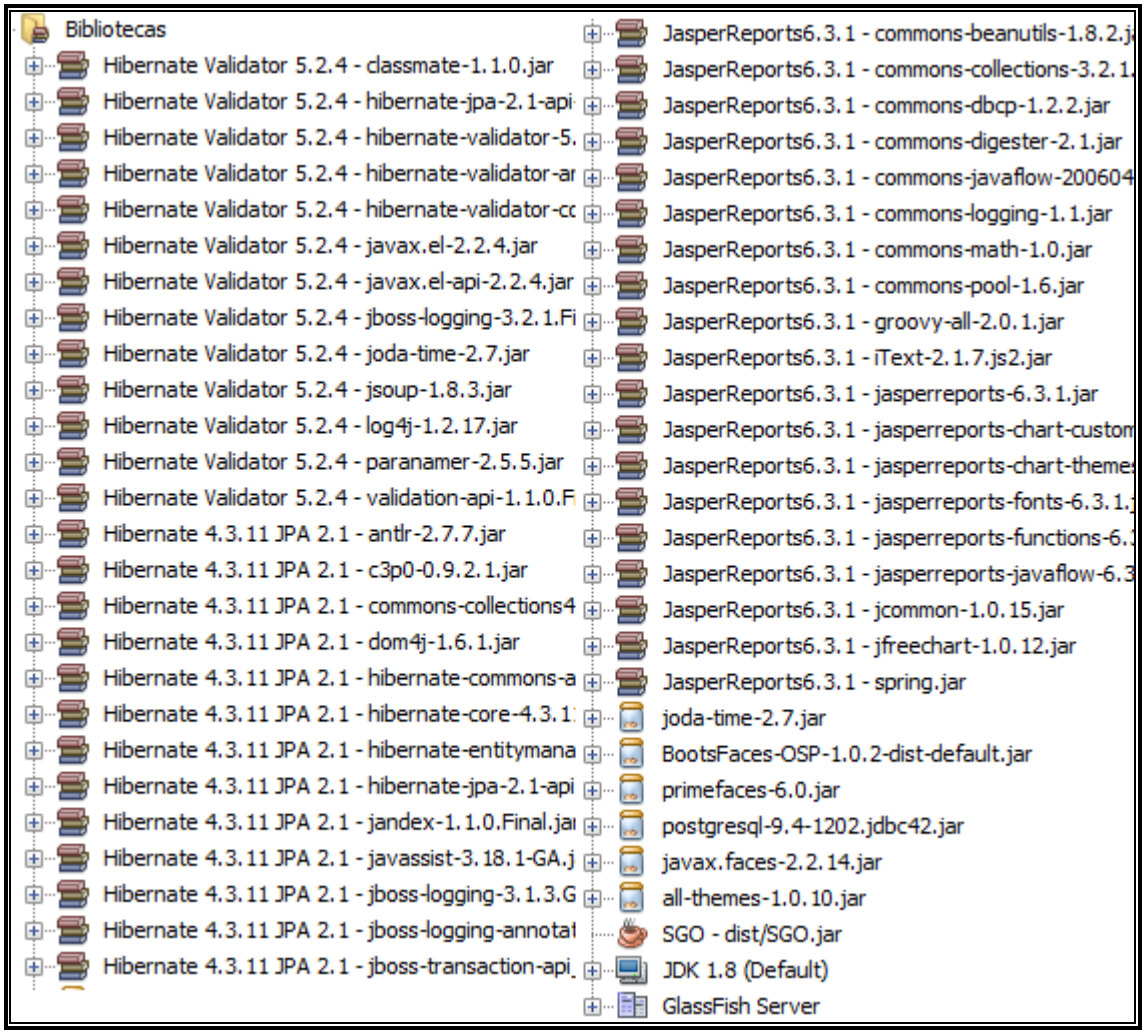
5.1.2 Bibliotecas e Frameworks

Para que se tornasse possível elaborar o sistema gerador de orçamentos, utilizaram-se os recursos viabilizados por frameworks e bibliotecas elaborados para a utilização em conjunto com o *JavaServer Faces*. Os recursos foram aplicados para auxiliar na criação da interface com o usuário, para validar e persistir os dados e tratar da comunicação com a aplicação, entre o banco de dados e o servidor. As bibliotecas empregadas na aplicação são:

- ***Biblioteca Hibernate Validator 5.2.4:*** A biblioteca *Hibernate Validator 5.2.4*, que implementa a *Bean Validation API 1.1*, foi usada para a validar os dados que integram o objeto persistido, impedindo improcedências na aplicação e seu banco de dados.
- ***Biblioteca Hibernate JPA 4.3.11 JPA 2:*** A biblioteca *Hibernate JPA* é uma biblioteca de persistência de dados que implementa a especificação *Java Persistence API* e foi aplicada como método para realizar o mapeamento objeto-relacional de objetos.
- ***Biblioteca JasperReports 6.3.1:*** A biblioteca *JasperReports* foi utilizada como solução para a geração de relatórios em Java, com esse *framework* tornou-se possível criar um modelo compatível com a aplicação elaborada, com o objetivo de gerar relatórios dos orçamentos em formato PDF (*Portable Document Format*).
- ***Biblioteca BootsFaces 1.0.2:*** A biblioteca *BootsFaces* é uma biblioteca de recursos gráficos para o JSF fundamentada no estilo visual da biblioteca *Bootstrap*. A finalidade de empregar essa biblioteca ao sistema consiste em que seus recursos são elaborados para a utilização em *layout* responsivo, sendo este um dos objetivos da aplicação web.
- ***Biblioteca PrimeFaces 6.0:*** A biblioteca *PrimeFaces* possui como sua principal funcionalidade, auxiliar na implementação *de* recursos gráficos, atualmente é o framework favorito dos desenvolvedores na construção de interfaces com JSF. Essa biblioteca foi utilizada no projeto em conjunto com a biblioteca *BootsFaces*.
- ***Driver JDBC PostgreSQL versão 9.4.1202:*** O *driver JDBC do PostgreSQL* dispõe de elementos para a conexão do sistema web com o banco de dados, possibilitando gerenciar e manipular os dados que compõe as tabelas da aplicação.

Na Figura 18 podem ser visualizadas as bibliotecas integrantes do sistema:

Figura 18 - Bibliotecas integrantes do sistema



Fonte: Do Autor

5.2 ESTRUTURA E LAYOUT DA APLICAÇÃO

Nesta sessão será detalhada a arquitetura em camadas da aplicação e os elementos que constituem a implementação do *layout* design responsivo.

5.2.1 Camada de Modelo

A camada de modelo do sistema é encarregada de manter as classes que determinarão os objetos instanciados pela aplicação. Os objetos instanciados foram

usados pela biblioteca *Java Persistence API* para a aplicação do mapeamento objeto-relacional para integrar-se ao banco de dados da solução desenvolvida.

Por intermédio da biblioteca *Java Persistence API*, criam-se as relações entre os objetos e as tabelas do banco de dados, essas relações são estabelecidas por meio de anotações nos objetos, com o objetivo de determinar as propriedades e as restrições para os dados das tabelas que serão arquivadas no banco de dados.

No caso das classes, para que seja possível persisti-las, deve-se cumprir o padrão *JavaBeans*, com esse padrão define-se a forma de elaboração da classe para possibilitar sua persistência. O padrão *JavaBeans* consiste em implementar na classe a interface *Serializable* e possuir atributos encapsulados, viabilizando o manuseio dos atributos pelos métodos *getter* e *setter*. Um fragmento do código da classe *Orcamento* pode ser visualizado na Figura 19:

Figura 19 - Fragmento do código da classe *Orcamento*

```
@Entity
@Table(name = "orcamento")
public class Orcamento implements Serializable {
    @Id
    @SequenceGenerator(name = "seq_orcamento", sequenceName = "seq_orcamento_id",
            allocationSize = 1)
    @GeneratedValue(generator = "seq_orcamento", strategy = GenerationType.SEQUENCE)
    private Integer id;

    @NotBlank(message = "O nome deve ser informado")
    @Length(max = 50, message = "O nome não deve ultrapassar {max} caracteres")
    @Column(name = "nome", length = 50, nullable = false)
    private String nome;

    @NotNull(message = "A data deve ser informada")
    @Temporal(TemporalType.DATE)
    @Column(name = "data", nullable = false)
    private Calendar data;

    @Column(name = "descricao", columnDefinition = "text")
    private String descricao;
```

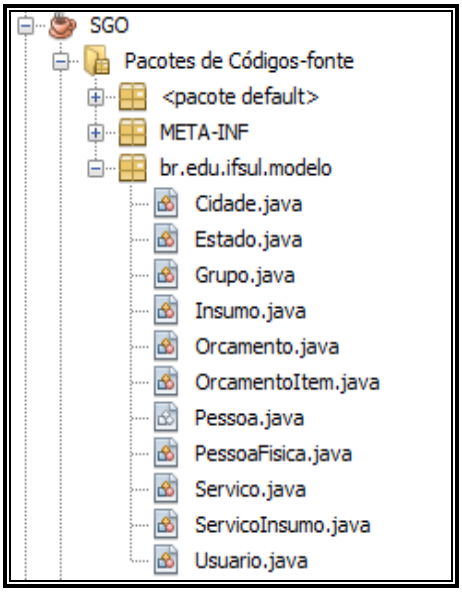
Fonte: Do Autor

Por intermédio das anotações que fazem parte do código, possibilita-se que a biblioteca *Java Persistence API* reconheça as especificidades dos objetos persistidos. Primeiramente tem-se na classe *Orcamento* a anotação *@Entity*, ela é responsável pela classificação da classe como uma entidade que representa uma tabela no banco de dados, o nome dessa tabela define-se pela anotação *@Table* (name = “nome_tabela”). Para representar as colunas da tabela para os atributos da entidade no banco de dados, usa-se a anotação *@Column*. Posteriormente tem-se a anotação *@Id*, com ela define-se o atributo identificador da tabela, para gerar o valor

do identificador automaticamente no banco de dados, utiliza-se a anotação *@GenerateValue*. A estratégia definida para a geração de valores é a geração sequencial, essa estratégia é determinada pela anotação *@GenerationType.SEQUENCE*, nessa anotação o gerador utiliza a anotação *@SequenceGenerator*. No caso dos relacionamentos entre as entidades, utiliza-se a anotação *@JoinColumn*, ela dispõe do nome da coluna referenciada pela chave estrangeira.

Para que seja possível validar os dados mantidos pelos atributos da entidade, aplicam-se anotações que examinam os dados reportados pela interface. A anotação *@NotNull* determina que não será aceito valor nulo para determinado atributo, no caso da anotação *@Length*, ela é determina o número mínimo e máximo de caracteres do atributo. A anotação *@NotBlank* possibilita a verificação de dados tipo *String*, não aceitando valores em branco, do mesmo modo que a anotação *@NotEmpty* que determina que não serão permitidos dados tipo *String* com conteúdo vazio. A anotação @temporal determina que o formato do atributo onde se aplica poderá somente aceitar dados referentes ao tempo. Se os requisitos definidos nos atributos não estiverem de acordo, é retornada uma mensagem personalizada por meio da propriedade *message*. A Figura 20 representa a organização da camada de modelo utilizada no projeto:

Figura 20 - Organização da Camada de Modelo do Projeto



Fonte: Do Autor

Um arquivo imprescindível para aplicar a JPA na aplicação é o arquivo *persistence.xml*. Esse arquivo possui os dados referentes a unidade de persistência do sistema e as configurações com a finalidade de efetivar a comunicação com o banco de dados. Faz parte desse arquivo, o elemento <*persistence-unit*>, nele insere-se o nome da unidade de persistência e o tipo de transição a ser utilizado, nesse caso foi definido o *transaction-type* como *Java Transaction API* (JTA). Por intermédio do elemento <*provider*>, definiu-se o Hibernate como provedor de persistência de dados. No caso do elemento <*jta-data-source*>, esse elemento é responsável por memorizar o diretório e o nome da fonte de dados em que foi aplicada a unidade de persistência. Seguidamente tem-se o elemento <*class*>, nesse elemento definem-se as classes que fazem parte do projeto. Posteriormente dispõe-se da propriedade <*property name="hibernate.hbm2ddl.auto" value="update"*/> com ela define-se o método automático de atualização das tabelas do banco de dados e a propriedade <*property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"*/> que é responsável pela determinação do "dialeto PostgreSQL" para ser o padrão paras as consultas SQL. Por intermédio da Figura 21, pode-se visualizar as configurações e as classes que fazem parte desse projeto:

Figura 21 – Conteúdo do arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <persistence-unit name="SGO-WEBPU" transaction-type="JTA">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>jdbc/sgo</jta-data-source>
        <class>br.edu.ifsul.modelo.Estado</class>
        <class>br.edu.ifsul.modelo.Cidade</class>
        <class>br.edu.ifsul.modelo.Grupo</class>
        <class>br.edu.ifsul.modelo.Insumo</class>
        <class>br.edu.ifsul.modelo.PessoaFisica</class>
        <class>br.edu.ifsul.modelo.ServicoInsumo</class>
        <class>br.edu.ifsul.modelo.Servico</class>
        <class>br.edu.ifsul.modelo.Usuario</class>
        <class>br.edu.ifsul.modelo.OrcamentoItem</class>
        <class>br.edu.ifsul.modelo.Orcamento</class>

        <properties>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
            <property name="hibernate.transaction.jta.platform" value="org.hibernate.service.jta.platform.internal.SunOneJtaPlati
            <property name="hibernate.classloading.use_current_tccl_as_parent" value="false"/>
        </properties>
    </persistence-unit>
</persistence>
```

Fonte - Do Autor

O arquivo responsável por manter os dados e colaborar na criação do *pool* de conexões e recursos JDBC no servidor é o *glassfish_resources.xml*, por meio dessas funções tornou-se possível criar a fonte de dados da aplicação web. O arquivo possui os dados referentes à autenticação por intermédio das propriedades de configuração para acessar o banco de dados, são elas: *serverName* (Nome do servidor utilizado), *portNumber* (Número da porta de conexão utilizada), *databaseName* (Nome da base de dados utilizada), *User* (Nome do usuário para entrar no sistema), *Password* (Senha do usuário para entrar no sistema), URL (Endereço para realizar a conexão), por fim tem-se a propriedade *driverClass* (Classe responsável pela conexão JDBC). No elemento <*jdbc-resource*> define-se o diretório JNDI da fonte de dados e o nome do *pool* de conexões por meio das propriedades *jndi-name* e *pool-name*, nessa ordem. Pode-se visualizar um trecho do código do arquivo *glassfish_resources.xml* na Figura 22:

Figura 22 - Trecho de código do arquivo glassfish_resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3
<resources>
        <property name="portNumber" value="5432"/>
        <property name="databaseName" value="sgo"/>
        <property name="User" value="postgres"/>
        <property name="Password" value="aluno"/>
        <property name="URL" value="jdbc:postgresql://localhost:5432/sgo"/>
        <property name="driverClass" value="org.postgresql.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/sgo" object-type="user"
                  pool-name="sgo_Pool"/>
</resources>
```

Fonte: Do Autor

A camada de modelo do sistema é encarregada de manter as classes que determinarão os objetos persistentes instanciados pela aplicação. As classes da modelagem da camada de persistência foram usadas pela biblioteca *Java Persistence API* para a aplicação do mapeamento objeto-relacional para integrar-se ao banco de dados da solução desenvolvida. Por intermédio da biblioteca *Java Persistence API*, criam-se as relações entre as classes e as tabelas do banco de dados. Essas relações são estabelecidas por meio de anotações nas classes com o

objetivo de determinar as propriedades e as restrições para os dados das tabelas que serão arquivadas no banco de dados.

No caso das classes persistentes, deve-se cumprir o padrão *JavaBeans*. O padrão *JavaBeans* consiste em implementar na classe a interface *Serializable,* possuir atributos encapsulados com a visibilidade privada, e possibilitar o manuseio dos atributos pelos métodos *getter* e *setter,* além de ter que possuir um construtor sem argumentos. Um fragmento do código da classe orçamento pode ser visualizado na Figura 23:

Figura 23 - Fragmento do código da classe *Orcamento*

```
@Entity
@Table(name = "orcamento")
public class Orcamento implements Serializable {
    @Id
    @SequenceGenerator(name = "seq_orcamento", sequenceName = "seq_orcamento_id",
            allocationSize = 1)
    @GeneratedValue(generator = "seq_orcamento", strategy = GenerationType.SEQUENCE)
    private Integer id;

    @NotBlank(message = "O nome deve ser informado")
    @Length(max = 50, message = "O nome não deve ultrapassar {max} caracteres")
    @Column(name = "nome", length = 50, nullable = false)
    private String nome;

    @NotNull(message = "A data deve ser informada")
    @Temporal(TemporalType.DATE)
    @Column(name = "data", nullable = false)
    private Calendar data;

    @Column(name = "descricao", columnDefinition = "text")
    private String descricao;
```

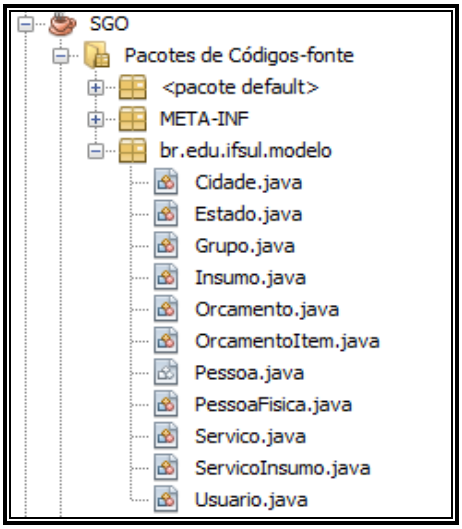
Fonte: Do Autor

Por intermédio das anotações que fazem parte do código, possibilita-se que a biblioteca *Java Persistence API* reconheça as especificidades dos objetos persistidos. Primeiramente tem-se na classe *Orcamento* a anotação *@Entity*, ela é responsável pela classificação da classe como uma entidade que representa uma tabela no banco de dados, o nome dessa tabela define-se pela anotação *@Table* (name = “nome_tabela”). Para representar as colunas da tabela para os atributos da entidade no banco de dados, usa-se a anotação *@Column*. Posteriormente tem-se a anotação *@Id*, com ela define-se o atributo identificador da tabela, para gerar o valor

do identificador automaticamente no banco de dados, utiliza-se a anotação *@GenerateValue*. A estratégia definida para a geração de valores é a geração sequencial, essa estratégia é determinada pela anotação *@GenerationType.SEQUENCE*, nessa anotação o gerador utiliza a anotação *@SequenceGenerator*. No caso dos relacionamentos entre as entidades, utiliza-se a anotação *@JoinColumn*, ela dispõe do nome da coluna referenciada pela chave estrangeira.

Para que seja possível validar os dados mantidos pelos atributos da entidade, aplicam-se anotações que examinam os dados reportados pela interface. A anotação *@NotNull* determina que não será aceito valor nulo para determinado atributo, no caso da anotação *@Length*, ela é determina o número mínimo e máximo de caracteres do atributo. A anotação *@NotBlank* possibilita a verificação de dados tipo *String*, não aceitando valores em branco, do mesmo modo que a anotação *@NotEmpty* que determina que não serão permitidos dados tipo *String* com conteúdo vazio. A anotação @temporal determina que o formato do atributo onde se aplica poderá somente aceitar dados referentes ao tempo. Se os requisitos definidos nos atributos não estiverem de acordo, é retornada uma mensagem personalizada por meio da propriedade *message*. A Figura 24 representa a organização da camada de modelo utilizada no projeto:

Figura 24 - Organização da Camada de Modelo do Projeto



Fonte: Do Autor

Um arquivo imprescindível para aplicar a JPA na aplicação é o arquivo *persistence.xml*. Esse arquivo possui os dados referentes a unidade de persistência do sistema e as configurações com a finalidade de efetivar a comunicação com o banco de dados. Faz parte desse arquivo, o elemento <*persistence-unit*>, nele insere-se o nome da unidade de persistência e o tipo de transição a ser utilizado, nesse caso foi definido o *transaction-type* como *Java Transaction API* (JTA). Por intermédio do elemento <*provider*>, definiu-se o Hibernate como provedor de persistência de dados. No caso do elemento <*jta-data-source*>, esse elemento é responsável por memorizar o diretório e o nome da fonte de dados em que foi aplicada a unidade de persistência. Seguidamente tem-se o elemento <*class*>, nesse elemento definem-se as classes que fazem parte do projeto. Posteriormente dispõe-se da propriedade <*property name="hibernate.hbm2ddl.auto" value="update"*/> com ela define-se o método automático de atualização das tabelas do banco de dados e a propriedade <*property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"*/> que é responsável pela determinação do “dialeto PostgreSQL” para ser o padrão paras as consultas SQL. Por intermédio da Figura 25, pode-se visualizar as configurações e as classes que fazem parte desse projeto:

Figura 25 – Conteúdo do arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <persistence-unit name="SGO-WEBPU" transaction-type="JTA">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>jdbc/sgo</jta-data-source>
        <class>br.edu.ifsul.modelo.Estado</class>
        <class>br.edu.ifsul.modelo.Cidade</class>
        <class>br.edu.ifsul.modelo.Grupo</class>
        <class>br.edu.ifsul.modelo.Insumo</class>
        <class>br.edu.ifsul.modelo.PessoaFisica</class>
        <class>br.edu.ifsul.modelo.ServicoInsumo</class>
        <class>br.edu.ifsul.modelo.Servico</class>
        <class>br.edu.ifsul.modelo.Usuario</class>
        <class>br.edu.ifsul.modelo.OrcamentoItem</class>
        <class>br.edu.ifsul.modelo.Orcamento</class>

        <properties>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
            <property name="hibernate.transaction.jta.platform" value="org.hibernate.service.jta.platform.internal.SunOneJtaPlat
            <property name="hibernate.classloading.use_current_tccl_as_parent" value="false"/>
        </properties>
    </persistence-unit>
</persistence>
```

Fonte: Do Autor

O arquivo responsável por manter os dados e colaborar na criação do *pool* de conexões e recursos JDBC no servidor é o *glassfish_resources.xml*, por meio dessas funções tornou-se possível criar a fonte de dados da aplicação web. O arquivo possui os dados referentes à autenticação por intermédio das propriedades de configuração para acessar o banco de dados, são elas: *serverName* (Nome do servidor utilizado), *portNumber* (Número da porta de conexão utilizada), *databaseName* (Nome da base de dados utilizada), *User* (Nome do usuário para entrar no sistema), *Password* (Senha do usuário para entrar no sistema), URL (Endereço para realizar a conexão), por fim tem-se a propriedade *driverClass* (Classe responsável pela conexão JDBC). No elemento <*jdbc-resource*> define-se o diretório JNDI da fonte de dados e o nome do *pool* de conexões por meio das propriedades *jndi-name* e *pool-name*, nessa ordem. Pode-se visualizar um trecho do código do arquivo *glassfish_resources.xml* na Figura 26:

Figura 26 - Trecho de código do arquivo glassfish_resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3
<resources>
        <property name="portNumber" value="5432"/>
        <property name="databaseName" value="sgo"/>
        <property name="User" value="postgres"/>
        <property name="Password" value="aluno"/>
        <property name="URL" value="jdbc:postgresql://localhost:5432/sgo"/>
        <property name="driverClass" value="org.postgresql.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/sgo" object-type="user"
                  pool-name="sgo_Pool"/>
</resources>
```

Fonte: Do Autor

5.2.1.1 Classes DAO

As classes DAO (*Data Access Object*) da aplicação web desenvolvida, são incumbidas de gerenciar a comunicabilidade entre o banco de dados, a unidade de persistência e os elementos do sistema que usufruem dos dados persistidos. Para tornar realizável a comunicação, as classes DAO empregam um gerenciador de entidades para acessar as classes da camada de modelo, esse gestor chama-se

EntityManager. Com ele pode-se manusear os dados por intermédio da unidade de persistência determinada pela anotação *@PersistenceContext*. O principal objetivo em viabilizar a comunicação entre a aplicação e o banco de dados é possibilitar as ações de inserir, visualizar, atualizar e excluir sobre os dados persistidos no banco. Na aplicação web geradora de orçamentos, pode-se reaproveitar o código da classe DAO genérica, que contém métodos pertinentes as operações de persistência sobre as classes da camada de modelo. A classe DAO genérica implementada no projeto, dispõe de recursos tais como, filtro e ordenação dos resultados da pesquisa, quantidade de registros por página a ser visualizada e paginação. Mediante essa classe genérica, possibilitou-se que as classes DAO distintas tivessem a sua codificação facilitada, devido à herança determinada entre as classes. Todas as classes da camada DAO do projeto implementam o padrão *JavaBeans*, por isso implementam a interface *Serializable*, as classes distintas receberam *@Stateful*, com a intenção de guardar o estado da sessão. Um trecho do código da classe DAOGenerico, pode ser visualizada na Figura 27:

Figura 27 - Trecho de código da classe DAOGenerico

```
public class DAOGenerico<T> implements Serializable {

    @PersistenceContext(unitName="SGO-WEBPU")
    private EntityManager em;
    private Class classePersistente;
    private String ordem = "id";
    private String filtro = "";
    private Integer maximoObjetos = 10;
    private Integer posicaoAtual = 0;
    private Integer totalObjetos = 0;
```

Fonte: Do Autor

No caso das classes DAO distintas, elas recebem como parâmetro a classe da camada de modelo a ser persistida. A classe citada possui a função de implementar os métodos oriundos da classe DAOGenerico. Pode-se também definir um padrão para ordenação dos resultados de pesquisa. Com o objetivo de atender as especificidades de cada classe do sistema, foram sobrescritos ou implementados os métodos da classe genérica.

A Figura 28 demonstra o código base para a implementação da classe OrcamentoDAO.

Figura 28 - Código base da classe OrcamentoDAO

```
@Stateful
public class OrcamentoDAO<T> extends DAOGenerico<Orcamento> implements Serializable {


    public OrcamentoDAO() {
        super();
        super.setClassePersistente(Orcamento.class);
        super.setOrdem("nome");
    }
```

Fonte: Do Autor

Para tornar possível a atualização dos preços dos insumos e das tabelas que possuem o insumo alterado, criou-se na classe InsumoDAO o método atualizarInsumo(), que recebe por parâmetro o objeto “in” do tipo Insumo. Primeiramente, criou-se uma lista com todos os “ServicoInsumo” (ServicoInsumo são os itens que compõe o serviço) que possuem o insumo com o preço alterado. Posteriormente, percorre-se a lista e atualiza-se o valor unitário e o valor total do item do “ServicoInsumo”. Em seguida, percorre-se novamente a lista com o objetivo de atualizar o valor total do “Servico” (Servico é onde cria-se o padrão de serviço por m²), para isso zerasse o valor do serviço e percorre-se os itens que compõe o serviço para pegar o novo valor total, por fim esse valor total atualizado por meio da variável “aux” é utilizado para atualizar o valor total do serviço.

Essa sequência de ações, que possui o objetivo de atualizar o preço de determinado insumo e todas as tabelas que contém esse insumo, é disparada quando o usuário editar o preço e clicar no botão salvar.

Pode-se visualizar o código do método atualizarInsumo() na Figura 29:

Figura 29 - Código do método atualizaInsumo()

```
public void atualizarInsumo(Insumo in) throws Exception {
    Double aux = null; //váriável para pegar o novo valor do serviço
    //pegando todos os ServicoInsumo que possuem o Insumo
    List<ServicoInsumo> lista;
    lista = super.getEm().createQuery("from ServicoInsumo where insumo.id = '" + in.getId() + "'").getResultList();
    //Atualizando o valor total do ServicoInsumo
    for (ServicoInsumo si : lista) {
        si.setValorUnitario(in.getPreco());
        si.setValorTotal(si.getValorUnitario() * si.getQuantidade());
        super.getEm().merge(si);
    }
    //Atualizando o valorServico do Servico
    for (ServicoInsumo si : lista) {
        aux = 0.0;
        Servico s = si.getServico();
        s.setValorServico(0.0);
        for (ServicoInsumo si2 : s.getItens()) {
            aux += si2.getValorTotal();
        }
        s.setValorServico(aux);
        super.getEm().merge(s);
    }
}
```

Fonte: Do Autor

5.2.2 Conversores

As páginas web são compostas somente por textos, estes textos são interpretados pelo navegador e exibidos para o usuário. Devido a isso, torna-se necessária a conversão dos dados de quaisquer tipos da aplicação para o tipo *String*, para que seja possível exibi-los na interface com o usuário. Citando caso análogo, tendo-se dados tipo *Calendar*, que é utilizado para representar dados relacionados ao tempo, somente pode-se visualizar na tela com a conversão para *String*, pois os elementos gráficos da interface não manipulam dados tipos temporais.

No sistema gerador de orçamentos foram elaborados conversores para efetivar a conversão de *Strings* para objetos Java e de objetos Java para *Strings*, sendo que essa conversão torna-se possível devido a implementação da interface Converter, disponibilizado pelo pacote *javax.faces*. Por meio dos métodos abstratos implementados pela interface Converter, pode-se receber uma *String* por parâmetro

e retornar um objeto da classe determinada, por intermédio do método *getAsObject ()*. Na situação oposta, tem-se o método *getAsString()*, responsável por receber um objeto e retornar uma *String*. Para poder utilizar os conversores, emprega-se uma instância da classe *EntityManager*, com a finalidade de acessar os dados da camada de modelo. O código do conversor do objeto tipo *Orcamento* é apresentado na Figura 30:

Figura 30 - Conversor para objetos do tipo *Orcamento*

```
// converte da tela para o objeto
@Override
public Object getAsObject(FacesContext fc, UIComponent uic, String string) {
    if (string == null || string.equals("Selecione um registro")) {
        return null;
    }
    return em.find(Orcamento.class, Integer.parseInt(string));
}
// converte do objeto para a tela
@Override
public String getAsString(FacesContext fc, UIComponent uic, Object o) {
    if (o == null) {
        return null;
    }
    Orcamento obj = (Orcamento) o;
    return obj.getId().toString();
}
```

Fonte: Do Autor

5.2.3 Camada de Controle

No sistema gerador de orçamentos, a camada de controle é responsável pela comunicação entre os elementos da camada de persistência com a interface gráfica. Por meio dos métodos viabilizados pelos controladores da aplicação, a camada de visão é capaz de solicitar à camada DAO os dados transmitidos pelo usuário por intermédio da interface, com o propósito de manipular e persistir esses dados. No começo da classe de controle, determina-se o nome pelo qual a classe será solicitada, usou-se o atributo *value* da anotação *@Named* para definir o nome, essa anotação faz parte do pacote *javax.inject.name*.

Para definir o ciclo de vida do *ManagedBean* das classes de controle da aplicação, utilizou-se a anotação *@SessionScoped*, com isso o *bean* é mantido até a

sessão do usuário encerrar ou expirar. Os atributos que integram essas classes podem ser EJBs da camada DAO, dessa forma pode-se trabalhar com os dados provenientes do banco de dados. Entre as atividades pertinentes das classes de controle apresentam-se as funções de listar, inserir, editar e excluir dados, tal como a aplicação de paginação em listas e processos característicos como a inserção automatizada de dados nos campos. A Figura 31 apresenta um trecho do código do controlador controleOrcamento:

Figura 31 - Trecho de código do controlador controleOrcamento

```
@Named(value = "controleOrcamento")
@SessionScoped
public class ControleOrcamento implements Serializable {

    @EJB
    private OrcamentoDAO<Orcamento> dao;
    private Orcamento objeto;
    private Boolean editando;
    private Boolean novoItem;

    @EJB
    private UsuarioDAO<Usuario> daoUsuario;

    @EJB
    private PessoaFisicaDAO<PessoaFisica> daoPessoaFisica;

    @EJB
    private ServicoDAO<Servico> daoServico;
    private Boolean editandoOrcamentoItem;
    private OrcamentoItem item;
```

Fonte: Do Autor

Além das funções citadas anteriormente, com os controladores pode-se também criar métodos que auxiliam na execução de outras funções. Na aplicação desenvolvida tem-se como exemplo, o método *imprimirOrcamento()*, método que recebe como parâmetro o id do objeto que deseja-se gerar o relatório. Em seguida criou-se uma lista para adicionar o objeto recuperado, e posteriormente elaborou-se uma nova instância da Classe *HashMap* por meio do atributo *parametros*. Com isso possibilitou-se o mapeamento do objeto e após foram enviados os parâmetros para o método *imprimeRelatorio()*, método que está inserido no arquivo UtilRelatorios.java, classe que possibilita a geração dos relatórios de orçamento por intermédio da biblioteca *JasperReports*.

O código do método *imprimirOrcamento()* é apresentado na Figura 32:

Figura 32 - Código do método *imprimirOrcamento ()*

```
public void imprimeOrcamento(Integer id){
    try {
    objeto = dao.recuperar(id);
    List<Orcamento> listaOrcamento = new ArrayList<>();
    listaOrcamento.add(objeto);
    HashMap parametros = new HashMap();
    UtilRelatorios.imprimeRelatorio("relatorioOrcamento", parametros,
            listaOrcamento);
            } catch (Exception e) {
        Util.mensagemErro("Erro ao imprimir: " + Util.getMensagemErro(e));
    }
}
```

Fonte: Do Autor

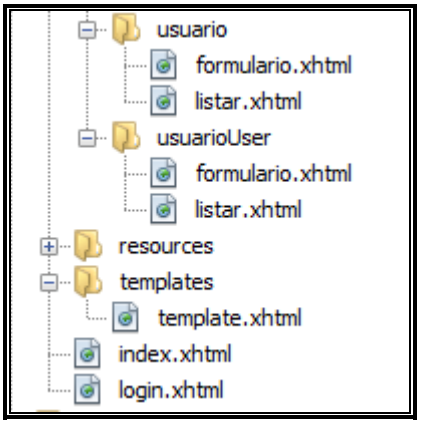
5.2.4 Camada de Visão

A camada de visão elaborada para o sistema gerador de orçamentos, possui a finalidade de se adaptar ao tamanho da tela de qualquer dispositivo, sendo que essa implementação torna-se possível, devido a utilização do design responsivo na configuração do *layout*. A interface gráfica da aplicação foi confeccionada com o *framework JavaServer Faces* (JSF) juntamente com as bibliotecas *PrimeFaces* e *BootsFaces*. Com essa forma de implementação simplifica-se a construção do layout das páginas, pois omite-se o desenvolvimento por meio das linguagens HTML e CSS, permitindo a utilização de componentes prontos disponibilizados pelas bibliotecas. Para a página de login e o corpo do sistema utilizou-se a biblioteca *BootsFaces*, no caso das tabelas e dos formulários utilizou-se a biblioteca *PrimeFaces*.

Para proporcionar a reutilização de código e simplificar a manutenção da aplicação, utilizou-se o componente *Facelets* do *JavaServer Faces*, esse componente é uma linguagem de descrição de páginas PDL (Page Description Language) criada especificamente para o JSF. Com a implementação desse componente JSF, insere-se o conteúdo dos arquivos responsáveis pelo *layout* do sistema no arquivo *template.xhtml*, esse arquivo possui tags correspondentes à

exibição do conteúdo na interface com o usuário. A Figura 33 apresenta a organização dos arquivos que fazem parte do *layout* da aplicação:

Figura 33- Organização da Camada de Visão do Projeto



Fonte: Do Autor

O arquivo utilizado como base para a implementação do *layout* é o *template.xhtml*, este arquivo contém locais editáveis que foram usados pelos arquivos que fazem parte do layout do sistema. O menu do sistema foi construído por intermédio do elemento *<b:dropMenu>*, no caso do local onde para onde foi destinado o conteúdo das páginas, este foi viabilizado por meio do elemento *<ui:insert>* que possui no seu atributo *name* o valor *conteudo*, esse local foi definido para que as páginas que compõe o layout do sistema possam mostrar as informações oriundas do banco de dados, assim como exibir as telas de edição das classes pertencentes à aplicação.

Um trecho do código do arquivo *template.xhtml* pode ser visualizado na Figura 34:

Figura 34 - Trecho de código do arquivo *template.xhtml*

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:b="http://bootsfaces.net/ui"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <f:view encoding="ISO-8859-1" contentType="text/html">
        <h:head>
            <title><ui:insert name="titulo">Gerador de Orçamentos</ui:insert></title>
            <h:outputStylesheet library="css" name="estilos.css"/>
        </h:head>
        <h:body style="background-color: #add">
            <b:container>
                <h:form id="formMenu" >
                    <b:navBar brand= "GERADOR DE ORÇAMENTOS">
                        <b:navbarLinks ...71 linhas />
                    </b:navBar>
                </h:form>
                <h:panelGroup>
                    <ui:insert name="conteudo">
                        CONTEUDO
                    </ui:insert>
                </h:panelGroup>
            </b:container>
        </h:body>
    </f:view>
</html>
```

Fonte: Do Autor

Para apresentar a tela inicial do sistema gerador de orçamentos, utilizou-se o arquivo *index.xhtml*, para isso empregaram-se elementos do JSF por intermédio de *facelets*, com o elemento *<ui:composition>* viabilizou-se a inserção no arquivo *template.xhtml*, por meio do elemento *<ui:define>* foram definidos o título e o conteúdo. A Figura 35 apresenta um fragmento do código do arquivo *index.xhtml*:

Figura 35 – Fragmento do código do arquivo *index.xhtml*

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="titulo">Gerador de Orçamentos</ui:define>
        <ui:define name="conteudo">
            <h3>Bem-Vindo ao Sistema!</h3>

        </ui:define>
    </ui:composition>
</html>
```

Fonte: Do Autor

Para controlar o acesso ao menu do sistema, utilizou-se o elemento *dropMenu* do *BootsFaces* no arquivo *template.xhtml*, o controle de acesso ao menu foi definido pela propriedade *rendered*, no caso se as condições definidas forem atendidas, o menu poderá ser visualizado. Com isso possibilita-se o acesso aos dados das classes do sistema, permitindo a manipulação desses dados. A aplicação dispõe de dois tipos de usuários, o usuário administrador, este possui acesso aos dados das classes e todas as ações do sistema, sem restrições. No caso do usuário comum, este poderá acessar somente os dados das classes referentes a geração dos insumos necessários para determinada obra, sendo este o principal objetivo da aplicação.

O acesso aos dados das classes do sistema ocorre por meio dos métodos elaborados nos seus respectivos controladores, por intermédio desses métodos, o usuário é redirecionado para a tela de manutenção desejada. Na Figura 36 pode-se visualizar um trecho do código referente ao menu com a permissão de administrador:

Figura 36 - Código referente ao menu do sistema

```
<b:dropMenu icon="list-alt" value="Cadastros"
            rendered="#{controleLogin.usuarioLogado != null
                        and(controleLogin.usuarioLogado.administrador == true)}">
    <b:navCommandLink icon="globe" value="Estados" ajax="false"
                      action="#{controleEstado.listar()}"/>
    <b:navCommandLink icon="globe" value="Cidades" ajax="false"
                      action="#{controleCidade.listar()}"/>
    <b:navCommandLink icon="user" value="Pessoas" ajax="false"
                      action="#{controlePessoaFisica.listar()}"/>
    <b:navCommandLink icon="user" value="Usuários" ajax="false"
                      action="#{controleUsuario.listar()}"/>
    <b:navCommandLink icon="list" value="Grupos" ajax="false"
                      action="#{controleGrupo.listar()}"/>
    <b:navCommandLink icon="usd" value="Insumos" ajax="false"
                      action="#{controleInsumo.listar()}"/>
    <b:navCommandLink icon="wrench" value="Serviços" ajax="false"
                      action="#{controleServico.listar()}"/>
    <b:navCommandLink icon="tasks" value="Orçamentos" ajax="false"
                      action="#{controleOrcamento.listar()}"/>
</b:dropMenu>
```

Fonte: Do Autor

Nas páginas de listagens de registros da aplicação web, pode-se selecionar a ordem de visualização dos registros por via de caixas de seleção, também possuem campos que possibilitam a aplicação de filtros relacionados aos registros e a quantidade de registros exibidos por página. Essas páginas de listagens, também possuem botões que proporcionam ações de criação, edição e exclusão de registros, no caso da lista de orçamentos também possui o botão de impressão, por meio desse botão pode-se visualizar e imprimir os dados do orçamento selecionado. Para viabilizar as ações citadas anteriormente, foram utilizados elementos *<b:commandButton>*, por intermédio desse elemento, possibilita-se o acesso aos métodos das camada de controle correspondentes à ação desejada. Por fim, as páginas de listagens também incluem botões de paginação, esses botões possibilitam a navegação entre as páginas. As requisições das listas são feitas por intermédio das classes DAO, pegando os dados e inserindo nos elementos *<p:column>* que fazem parte do elemento *<p:dataTable>*. Um fragmento do código do arquivo *listar.xhtml* da classe orçamento é apresentado na Figura 37:

Figura 37 - Fragmento de código do arquivo *listar.xhml*

```
<ui:define name="titulo">Manutenção de Orçamentos</ui:define>
<ui:define name="conteudo">
    <h:form id="formListagem">
        <h:panelGroup rendered="#{!controleOrcamento.editando}">
            <p:messages/>

            <p:commandButton value="Novo" icon="ui-icon-plus"
                             actionListener="#{controleOrcamento.novo()}"
                             update=":formEdicao :formListagem"/>
            <p:dataTable value="#{controleOrcamento.dao.listaObjetos}"
                         var="obj" reflow="true" id="listagem" >
                <f:facet ...20 linhas />
                <f:facet ...11 linhas />
                <p:column headerText="ID">
                    <p:outputLabel value="#{obj.id}"/>
                </p:column>
                <p:column headerText="Nome">
                    <p:outputLabel value="#{obj.nome}"/>
                </p:column>
                <p:column headerText="Data">
                    <p:outputLabel value="#{obj.data.time}">
                        <f:convertDateTime pattern="dd/MM/yyyy"/>
                    </p:outputLabel>
                </p:column>
```

Fonte: Do Autor

No caso das inclusões e edições de registros na lista, utilizou-se o arquivo *formulario.xhtml* para realizar essa ações, por intermédio dos formulários elaborados nesse arquivo pode-se incluir novos registros ou editar os dados de algum registro que já consta no sistema. Para a edição das informações nos campos do tipo texto empregou-se o elemento *<p:input>*, no caso dos campos do tipo seleção empregou-se o elemento *<p:selectOneMenu>*, no campo do tipo seleção, recupera-se registros provenientes dos relacionamentos entre determinadas tabelas do banco de dados. Para que se tornasse possível integrar os formulários de edição aos formulários do arquivo *listar.xhtml*, utilizou-se o elemento *<p:include>*. A Figura 38 apresenta um trecho de código do arquivo *formulário.xhtml* da classe *Orcamento*:

Figura 38 - Trecho de código do arquivo *formulario.xhtml*

```
    <p:outputLabel value="Edição de Orçamentos"/>
</f:facet>
<p:outputLabel for="txtID" value="ID"/>
<p:inputText id="txtID" value="#{controleOrcamento.objeto.id}"
             size="5" maxlength="5"/>
<p:outputLabel for="txtNome" value="Nome"/>
<p:inputText id="txtNome" value="#{controleOrcamento.objeto.nome}"
             size="50" maxlength="50"
             placeholder="Obrigatório"/>
<p:outputLabel value="Data" for="txtData"/>
<p:calendar ...6 linhas />
<p:outputLabel value="Cliente" for="selectPessoa"/>
<p:selectOneMenu id="selectPessoa"
                 value="#{controleOrcamento.objeto.pessoaFisica}">
    <f:converter converterId="converterPessoaFisica"/>
    <f:selectItem itemLabel="Selecione um registro"
                  noSelectionOption="true"/>
    <f:selectItems value="#{controleOrcamento.daoPessoaFisica.listaTodos}"
                   var="p" itemLabel="#{p.nome}"/>
</p:selectOneMenu>
```

Fonte: Do Autor

6 RESULTADOS

Com o objetivo de otimizar o processo de geração de orçamentos para a construção civil, criou-se um sistema web para auxiliar nessa tarefa. Com a conclusão do desenvolvimento do sistema gerador de orçamentos, tornou-se possível a geração de orçamentos.

Uma demonstração do processo de geração de um orçamento por meio do sistema desenvolvido será realizada e descrita nesta sessão. Inicialmente, tem-se a tela de login, por intermédio dessa tela pode-se entrar no sistema. Para efetivar o acesso, o usuário deverá estar devidamente cadastrado no banco de dados. Na Figura 39 apresenta-se a tela de *login* do sistema:

Figura 39 - Tela de *login* do sistema

Fonte: Do Autor

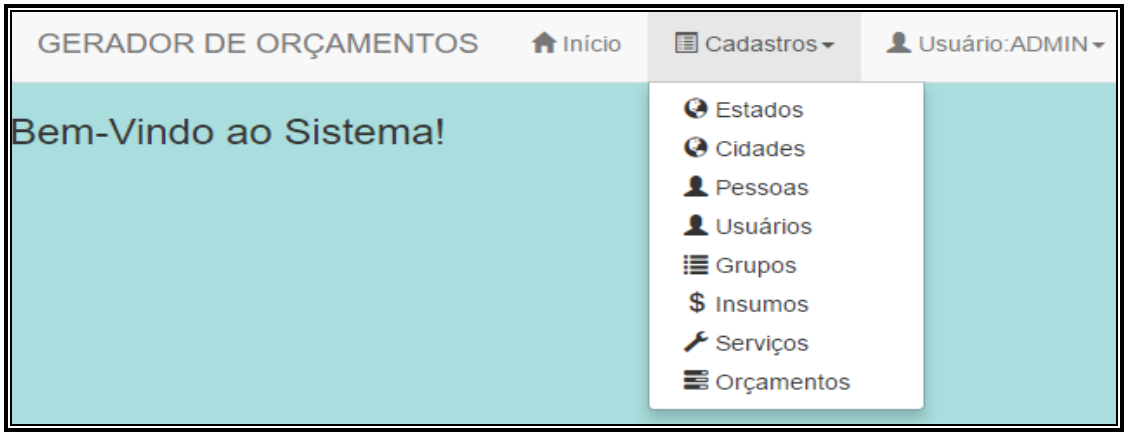
Logo após a validação de entrada para acessar o sistema, apresenta-se a tela de navegação da aplicação, ela é composta por *links* que possuem a função de voltar a tela inicial, acessar os cadastros e realizar o *logout*. Este poderá ser efetivado por meio da aba onde demonstra-se o usuário que está conectado ao sistema.

O sistema desenvolvido possui dois níveis de acesso, o usuário administrador e o usuário operacional. O usuário com nível de administrador pode executar qualquer ação na aplicação e os cadastros que competem somente ao administrador são: estados, grupos e serviços. No caso do cadastro de estados, não é necessário a disponibilização de edição para o usuário comum, pois é uma lista estática, com os

estados previamente cadastrados, o cadastro de grupos refere-se aos grupos de insumos, possui somente a função de facilitar as consultas e o cadastro de serviços deve ser acessado somente pelo administrador, pois é neste cadastro onde cria-se o padrão de serviço por m².

Já no caso do usuário com nível operacional, este pode acessar somente os cadastros mais funcionais do sistema, por exemplo: cadastro de cidades, pessoas, usuários, insumos e orçamentos. Para a criação de pessoas e cidades não possui nenhuma restrição, no cadastro de usuários, o usuário operacional não pode criar um usuário administrador, somente outro usuário operacional. No caso do cadastro de insumos, o usuário operacional poderá alterar somente os preços, essa funcionalidade foi desenvolvida com o objetivo de tornar o sistema mais flexível, pois sabe-se da grande variação de preços de acordo com a região do país onde a aplicação poderá ser utilizada. A tela de navegação está ilustrada na Figura 40:

Figura 40 - Tela de navegação do sistema

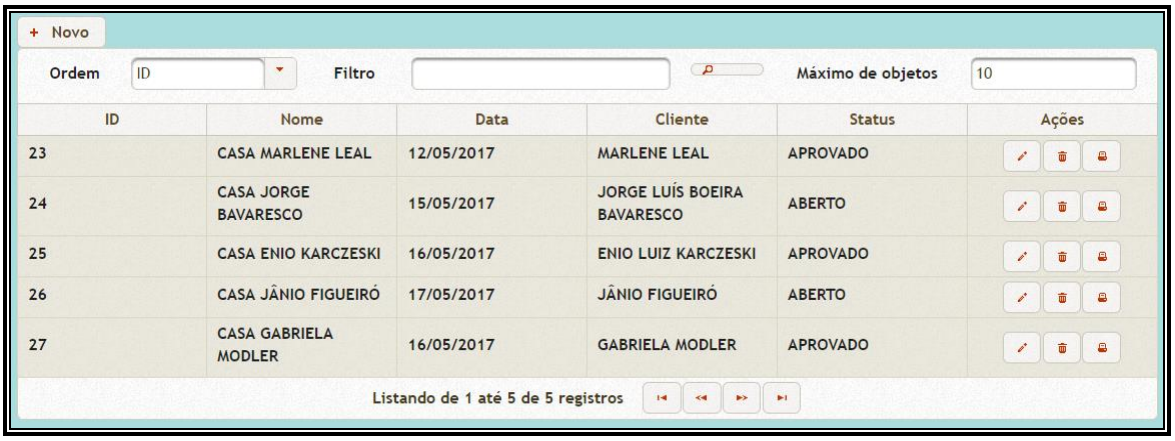


Fonte: Do Autor

Em seguida demonstra-se a lista dos orçamentos cadastrados, essa tela é constituída pelos registros dos orçamentos, por intermédio dessa tela pode-se criar um novo orçamento, alterar ou visualizar detalhes do orçamento, excluir o orçamento e gerar relatório dos insumos necessários para a realização dos serviços definidos no orçamento. Na tela de listagem de orçamentos, também pode-se definir a ordem em que os orçamentos serão listados, filtrar registros e definir o número de registros visualizados por página.

A Figura 41 apresenta a tela de listagem dos orçamentos:

Figura 41 - Tela de listagem dos orçamentos



Fonte: Do Autor

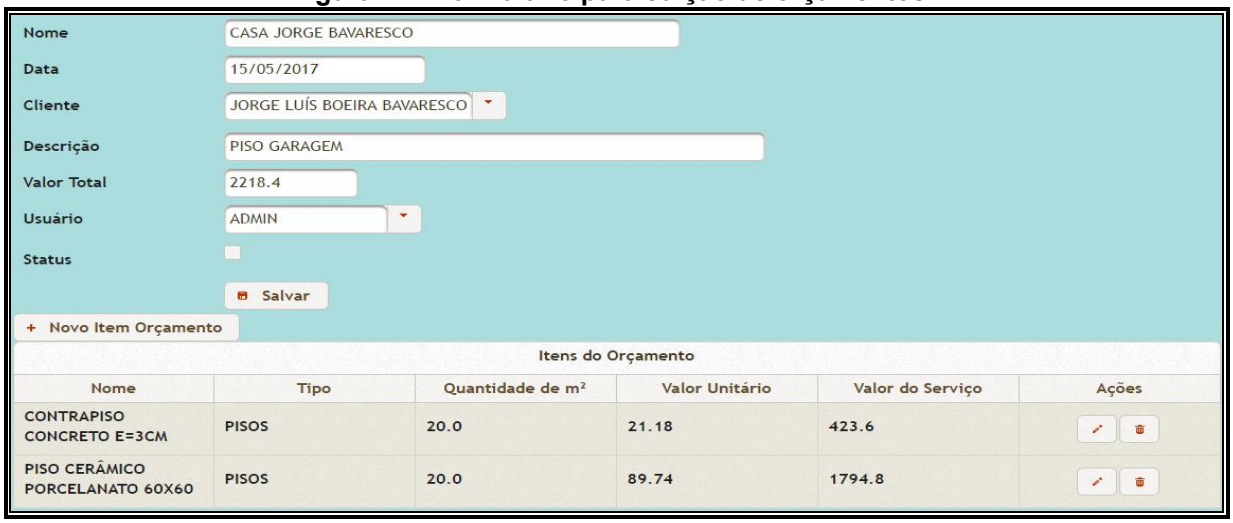
Ao clicar no botão novo ou editar será aberto o formulário para preenchimento. No caso do botão novo, o formulário apresenta-se em branco, já no caso do botão de edição o formulário será apresentado com os dados provenientes do banco de dados para o orçamento selecionado, conforme pode-se visualizar na Figura 42. Se o usuário optar pela exclusão do orçamento a ação será executada e posteriormente a tela será atualizada com os registros que permanecem cadastrados.

Na tela de formulário, pode-se preencher os dados referentes ao orçamento como, por exemplo: nome da obra, data do orçamento, cliente, descrição da obra, usuário responsável pela criação do orçamento e o status. No caso do cliente, este deve estar registrado por meio do cadastro de pessoas. O *checkbox* de status define se a obra foi aprovada ou não, ou seja, se for preenchido refere-se que a obra possui o status de aprovada, no caso de o usuário não preencher o *checkbox* a obra receberá o status de orçamento aberto, ou seja, ainda não foi aprovado. O formulário também possui o campo valor total, este campo não pode ser editado, somente recebe o valor total dos serviços desejados pelo cliente.

Posteriormente, tem-se o botão “Novo Item Orçamento”, ao clicar nele, apresenta-se a Figura 42, nessa tela definem-se as quantidades em m² e os serviços que irão compor o orçamento, após o preenchimento desses campos, o sistema irá recuperar o valor unitário do serviço e consequentemente realizar a

multiplicação pela quantidade definida pelo usuário, com isso gera-se o valor total do serviço. Por fim, calcula-se a soma dos serviços registrados, o resultado do cálculo será inserido no valor total do orçamento. Com as quantidades e serviços definidos finaliza-se o processo de criação do orçamento clicando no botão salvar, ao realizar essa ação o usuário será redirecionado para a página de listagem dos orçamentos cadastrados. As Figuras 42 e 43 apresentam respectivamente o formulário para a edição de orçamentos e o formulário para adição de itens ao orçamento:

Figura 42 - Formulário para edição de orçamentos



Fonte: Do Autor

Figura 43 - Formulário para adição de itens

Fonte: Do Autor

Na listagem de orçamentos cadastrados, pode-se a qualquer momento gerar relatório com os insumos necessários para a execução do orçamento previamente realizado, para executar essa ação necessita-se somente clicar no botão imprimir do orçamento que deseja-se obter o relatório, o relatório pode ser somente visualizado ou baixado para o dispositivo onde está sendo realizado o acesso. A Figura 44 apresenta o relatório gerado de um orçamento.

Figura 44 - Relatório de orçamento gerado

ORÇAMENTO

CLIENTE: JORGE LUÍS BOEIRA BAVARESCO DATA: 15/05/2017

OBRA: CASA JORGE BAVARESCO VALOR OBRA: R$ 2218,40

DESCRIÇÃO: PISO GARAGEM STATUS: ABERTO

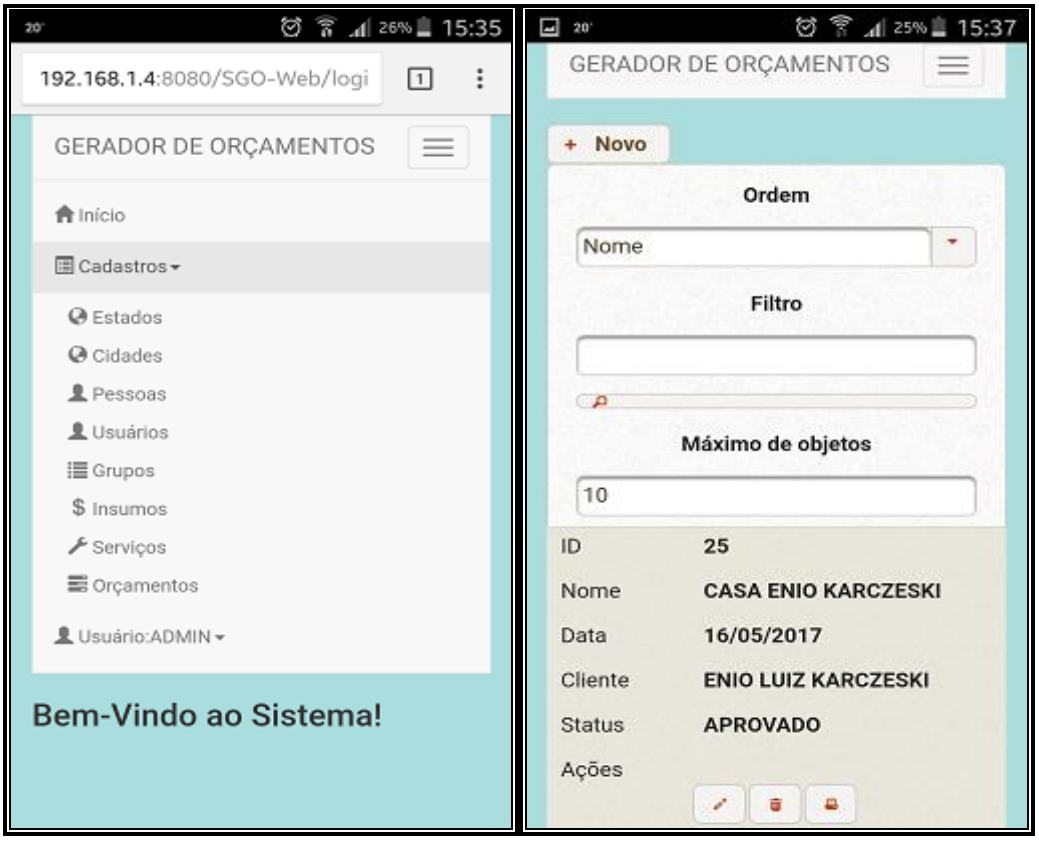
LISTA DE SERVIÇOS	QUANTIDADE	VALOR UNITÁRIO	VALOR TOTAL
CONTRAPISO CONCRETO E=3CM	20.0	R$ 21,18	R$ 423,60
INSUMOS NECESSÁRIOS			
CIMENTO VOTORAM 50	4,00	R$ 26,90	R$ 107,60
MÃO DE OBRA CONTRAPISO	20,00	R$ 14,90	R$ 298,00
AREIA MÉDIA	0,20	R$ 89,90	R$ 18,00
PISO CERÂMICO PORCELANATO 60X60	20.0	R$ 89,74	R$ 1794,80
INSUMOS NECESSÁRIOS			
ARGAMASSA AC-I QUARTZOLIT INTERNO	8,60	R$ 7,90	R$ 68,00
REJUNTE ÁREAS SECAS	2,80	R$ 3,90	R$ 11,00
MÃO DE OBRA PISO PORCELANATO	20,00	R$ 29,90	R$ 598,00
PISO PORCELANATO RETIFICADO 60X60	22,40	R$ 49,90	R$ 1117,80

Fonte: Do Autor

Para testar a eficiência do método de design responsivo, realizaram-se testes em dispositivos com telas de diversos tamanhos, a Figura 45 representa o sistema aplicando a renderização do *layout,* ou seja, reorganizando os elementos que compõe a interface com o usuário, de acordo com o espaço disponível na tela de 5 polegadas do smartphone onde realizaram-se os testes.

Nas imagens abaixo podem ser visualizadas respectivamente o menu e a listagem na tela de 5 polegadas do *smartphone*.

Figura 45 – Menu e listagem em *Smartphone* 5 polegadas



Fonte: Do Autor

Durante o processo de desenvolvimento do sistema gerador de orçamentos, procurou-se a resolução de todas as questões relacionadas ao estudo de caso referentes à geração de orçamentos para determinadas obras da construção civil.

7 CONSIDERAÇÕES FINAIS

Considerando a concorrência atual na área da construção civil, o desenvolvimento de uma ferramenta, com o objetivo de otimizar determinados processos no âmbito da gestão de tempo e custos que envolvem uma obra, pode ser visto como um diferencial em relação a seus concorrentes.

O presente estudo teve como objetivo o desenvolvimento de um sistema gerador de orçamentos para a construção civil. Ao realizar testes em uma empresa da área da construção civil, constatamos que a utilização do sistema desenvolvido otimizou o processo de geração de orçamentos para os profissionais, isso deve-se principalmente ao método de padronização de serviços por m² utilizado na solução desenvolvida.

Sendo assim, a aplicação apresentada buscou supriu as necessidades identificadas no processo de geração de orçamentos, para isso buscaram-se dados de acordo com as normas da ABNT (Associação Brasileira de Normas Técnicas), que regem a construção civil. Destaca-se que as atividades de levantamento de requisitos e coleta de dados, foram realizadas por intermédio de entrevistas com profissionais certificados e pesquisas. Seguidamente realizou-se a modelagem de acordo com o estudo de caso realizado anteriormente. Por fim, realizou-se o desenvolvimento da solução web, contendo métodos específicos para realizar as operações necessárias ao objetivo de facilitar a geração dos insumos necessários para a execução de determinadas obras.

Deste modo, o desenvolvimento da aplicação deu-se pela integração de variadas tecnologias, tais como: a plataforma Java EE, que disponibiliza recursos através de containers EJB e possibilita a criação de interfaces com *layout* responsivo, o layout do sistema foi elaborado com o auxílio das bibliotecas *PrimeFaces* e *BootsFaces*. Também se utilizou a API JPA com o servidor de persistência *Hibernate*, o *framework* de componentes MVC baseado na linguagem Java, o *JavaServer Faces* e o banco de dados objeto relacional *PostgreSQL*.

Por fim, as funcionalidades propostas no presente estudo, foram implementadas com sucesso. Por meio de testes, juntamente com uma empresa da construção civil, constatou-se que o processo de geração de orçamentos por intermédio da aplicação, tornou-se eficiente, pois ao invés de realizar os cálculos

manualmente ou por meio de planilhas eletrônicas, o sistema gerador de orçamentos realiza os cálculos dos insumos necessários para a execução de determinada obra somente com a definição do tipo do serviço e a quantidade desejada em m², com isso otimizando o processo de geração de orçamentos.

8 REFERÊNCIAS

BOOTSFACES. **Dashboard**. Disponível em< http://showcase.bootsfaces.net/Examples/dashboard.jsf> Acesso em: 13 abr. 2017.

_____. **BootsFaces Showcase and Documentation**. Disponível em< https://showcase.bootsfaces.net/> Acesso em: 13 abr. 2017.

CATRAIA APLICATIVOS. **Reforma Simples.** Disponível em: < https://play.google.com/store/apps/details?id=com.uninorte.orcafacil >. Acesso em: 23 jun. 2017.

DEITEL, Paul; DEITEL, Harvey. **Java: como programar**. São Paulo: Pearson Prentice Hall, 2010.

DUARTE, Lucas Silva. **Desenvolvimento de planilha eletrônica para orçamento de obras de pequeno porte**. 2012. 86f. Monografia (Especialização em Gerenciamento de Obras) – Departamento Acadêmico De Construção Civil, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

GONCALVES, Antonio. **Beginning Java EE 7**. Apress, 2013.

_____. **Introdução à Plataforma JavaTM EE 6 com GlassFishTM 3**. Rio de Janeiro: Ciência Moderna, 2011.

GONÇALVES, Edson. **Desenvolvendo Aplicações Web com JSP, Servlets, JavaServer Faces, Hibernate, EJB3 Persistence e Ajax**. Rio de Janeiro: Ciência Moderna, 2007.

_____. **Desenvolvendo Relatórios Profissionais com iReport™ para Netbeans IDE**. Rio de Janeiro: Ciência Moderna, 2009.

LOPES, Sérgio. **A Web Mobile: Programe para um mundo de muitos dispositivos.** Casa do Código, 2012.

ORACLE. **Java Platform, Enterprise Edition: The Java EE Tutorial, 2104.** Disponível em: < https://docs.oracle.com/javaee/7/tutorial/bean-validation001.htm>. Acesso em: 12 abr. 2017.

_____. **NetBeans IDE**. Disponível em: <https://netbeans.org/downloads/8.1/>. Acesso em: 23 jun. 2017.

_____. **The Java EE 6 Tutorial, 2013**. Disponível em: < http://docs.oracle.com/javaee/6/tutorial/doc/bnabo.html>. Acesso em: 24 mai. 2016.

_____. _____. Disponível em: < http://docs.oracle.com/javaee/6/tutorial/doc/bnacj.html>. Acesso em: 23 jun. 2017.

_____. **The Java™ Tutorials**. Disponível em: < https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>. Acesso em: 22 abr. 2016.

MATTOS, Aldo Dória. **Como preparar orçamentos de obras**: dicas para orçamentistas, estudos de caso, exemplos. São Paulo: PINI, 2006.

NETO, Álvaro Pereira. **PostgreSQL: Técnicas Avançadas: Versões Open Source 7.x e 8.x: Soluções para Desenvolvedores e Administradores de Banco de Dados**. São Paulo: Érica, 2007.

NOVA ERA SISTEMAS. **SIG Construtora**. Disponível em < http://www.programaparaconstrutora.com.br/>. Acesso em 23 jun. 2017.

PETERSON, Clarissa. **Learning Responsive Web Design: a begginer's guide**. O'Reilly, 2014.

PITANGA, Talita. **JavaServer Faces: A mais nova tecnologia Java para desenvolvimento WEB.** Disponível em < http://www.cin.ufpe.br/~jvwr/JSF/jsf.pdf>. Acesso em 23 jun. 2017.

POSTGRESQL. **pgAdmin III**. Disponível em <https://www.postgresql.org/download/>. Acesso em 23 jun. 2017.

POWER DROID. **Suite Construção.** Disponível em < https://play.google.com/store/apps/details?id=br.suite.construcao>. Acesso em 23 jun. 2017.

PRIMEFACES. **Why PrimeFaces**. Disponível em < http://www.primefaces.org/whyprimefaces>. Acesso em 14 abr. 2017.

RED HAT. **Bean Validation Specification,2013**. Disponível em: < http://beanvalidation.org/1.1/spec/#d0e168>. Acesso em: 12 abr. 2017.

SAMPAIO, Cleuton. **Java Enterprise Edition 6-Desenvolvendo Aplicações Corporativas**. Brasport, 2011.

SANTOS, Adriana de Paula Lacerda; JUNGLES, Antonio Edésio. **Como gerenciar as compras de materiais na construção civil: diretrizes para implantação da compra proativa**. São Paulo: PINI, 2008.

SILBERSCHATZ, Abraham et al.; **Sistema de Banco de Dados**. Rio de Janeiro: Elsevier, 2012.

ZEMEL, Tárcio. **Web Design Responsivo: Páginas adaptáveis para todos os dispositivos.** Casa do Código, 2012.