UNIVERSIDADE FEDERAL DO RIO DE JANEIRO ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

Sistema de Controle Orçamentário WEB

Autor: _____

Gustavo Barreto Bergter

Orientador: _____

Prof. Antônio Cláudio Gómez de Sousa

Examinador: _____

Prof. Ricardo Rhomberg Martins

Examinador: _____

Prof. Aloysio de Castro Pinto Pedroza

DEL

Agosto de 2008

Resumo

O projeto tratado neste documento foi desenvolvido para cumprir o requisito de projeto final de curso de Engenharia Eletrônica e de Computação da Universidade Federal do Rio de Janeiro e tem por objetivo auxiliar o controle orçamentário de uma pequena ONG.

Nesse projeto foi implementado um sistema de controle orçamentário através do qual é possível a administração de diversos orçamentos simultâneos com a utilização de consultas e análises em um ambiente *Web* com interface simples e amigável. No sistema o usuário poderá controlar orçamentos indicando as fontes e os gastos, que são divididos em contas para uma melhor visualização e maior facilidade na administração dos valores.

Palavras-Chave:

- Sistemas de Informação
- Java
- Adobe Flex
- Sistema Orçamentário
- Banco de Dados
- Web
- Tecnologia Solidária

Glossário

JVM – Java Virtual Machine

IDE – Integrated Development Tool

JSP – Java Server Page

HTML – Hyper Text Transfer Protocol

XML – Extensible Markup Language

MVC – Model-View-Controller

MXML – linguagem criada pela Macromedia, hoje pertencente à Adobe, que é baseada em XML.

Orçamento - plano financeiro estratégico de uma administração para determinado exercício.

Conta Orçamentária – Divisão do orçamento em despesas de um mesmo gênero. Como, por exemplo, “despesas com manutenção”, “despesas com treinamento”.

Liberação – ato de liberar um montante de dinheiro para uma conta orçamentária. A partir da liberação o montante está disponível na conta.

Empenho – ato de alocar um montante de dinheiro de uma conta para um determinado fim. Esse montante empenhado só poderá ser utilizado para aquele fim.

Pagamento – ato de quitar parcial ou totalmente um empenho. Ao efetuar o pagamento o montante passa a ser um valor quitado.

Financiador – entidade que contribui financeiramente para o orçamento da organização

Fonte – junção de todos os financiadores que estão contribuindo para determinado projeto

Índice:

1- Introdução:

1.1 - Assunto:

Os estudos sobre orçamentos e o controle orçamentário sempre foram de grande importância tanto para grandes quanto pequenas empresas e até mesmo no âmbito pessoal, uma vez que é a partir desse controle que se torna possível a tomada de decisões corretas e são evitados problemas financeiros mais sérios.

Com isso temos visto um grande crescimento na busca de ferramentas que possibilitem que esse controle seja feito de uma forma simples e interativa, mas ao mesmo tempo tendo a capacidade de trabalhar com grandes quantidades de dados. Ao olharmos para o mercado podemos perceber que existem diversas opções de ferramentas que se encaixam nesse perfil, requerendo, no entanto, que um grande investimento seja feito, já que geralmente não são soluções baratas e nem sempre vão ser adequadas às necessidades do cliente.

Tendo isso em vista foi proposto o desenvolvimento de um sistema de controle orçamentário que fosse capaz de atender essas necessidades e oferecesse funcionalidades que pudessem ser utilizadas por diversos usuários. Partindo desse ponto inicial foram levantados os requisitos e então desenvolvido o sistema que será tratado nesse documento.

1.2 - Motivação

Duas motivações principais levaram à proposta e ao posterior desenvolvimento do projeto do Sistema de Controle Orçamentário. Uma delas foi a oportunidade de desenvolver um sistema completo que viria a ser útil para uma organização de economia solidária. Outra motivação foi a possibilidade de desenvolver um projeto integral de um *software*, desde o levantamento de requisitos, passando pela modelagem do sistema, pela implementação da base de dados, pela codificação dos algoritmos em Java e chegando até a elaboração da interface gráfica com usuário. A chance de realizar todas essas etapas e a importância dessa realização

para consolidar alguns conhecimentos adquiridos na faculdade foram, sem dúvida, de grande importância para o desenvolvimento do projeto.

1.3 - Objetivos

O objetivo do projeto é desenvolver um sistema para fazer o gerenciamento de orçamentos. O sistema permitirá o acompanhamento e atualização de diversos orçamentos simultaneamente, através de uma interface *Web*, simples e amigável.

No sistema proposto o orçamento será dividido em contas, criadas pelo usuário para agrupar despesas de um mesmo gênero, de modo a facilitar os lançamentos de dados orçamentários. Em cada conta poderão ser efetuadas as ações de liberação, empenho e pagamento, de forma que os dados da conta serão atualizados automaticamente após a execução de uma delas. Esses dados que irão ser calculados e mantidos atualizados são:

-valor previsto

-valor liberado

-valor empenhado

-valor quitado (pago)

-saldo atual (saldo inicial decrescido do valor já empenhado)

-saldo disponível (valor real disponível na conta, já liberado e ainda não empenhado)

O sistema a ser desenvolvido é independente, não necessitando de integração com qualquer outro sistema. Serão utilizadas ferramentas de código aberto e gratuitas que permitirão o desenvolvimento de um sistema independente de sistemas operacionais e navegadores de internet.

Na seqüência deste documento, falaremos sobre como transcorreu a elaboração do sistema. O capítulo dois irá tratar do ambiente de desenvolvimento e quais ferramentas foram utilizadas. O capítulo três trata da parte inicial do projeto, que engloba as partes de análise do problema e de modelagem do sistema para solucionar o mesmo. O capítulo quatro conta como

foi feita a implementação do sistema, mostrando sua arquitetura e a metodologia de desenvolvimento. O capítulo cinco expõe os resultados obtidos ao término do projeto e o capítulo seis mostra as conclusões que foram tiradas após fim da implementação e de todos os testes realizados sobre o projeto. No apêndice 1 é possível encontrar o manual do usuário, que irá mostrar como instalar o sistema e como utilizá-lo da forma correta.

2- Ambiente:

2.1- Linguagem de Programação (Java)

A linguagem de programação utilizada para a implementação do projeto foi a linguagem Java. A escolha se deveu a diversas características da linguagem, como versatilidade, eficiência e portabilidade entre plataformas. Além disso, a linguagem Java é atualmente uma das mais utilizadas no mundo, o que a torna uma linguagem bastante robusta, uma vez que está sempre passando por testes e recebendo atualizações e correções de *bugs* reportados por desenvolvedores.

Outra vantagem que decorre de sua ampla utilização diz respeito à sua documentação vasta e diversas fontes onde é possível pesquisar dúvidas e buscar recomendações e orientações para realizar uma melhor implementação.

Java é uma linguagem muito versátil, que vem sendo usada em diversos tipos de aplicações que variam desde programas para celulares até aplicações de grande porte para rodar em servidores. Essa versatilidade e portabilidade decorrem do fato de que as aplicações Java rodam em uma máquina virtual Java (JVM – *Java Virtual Machine*), o que as torna independentes do sistema operacional do ambiente em que estão sendo executadas.

Outro fator que influenciou na escolha da linguagem de programação a ser usada foi o fato de Java ser uma linguagem que permite a implementação orientada a objeto, o que facilita a utilização de uma metodologia orientada a objetos, que vem cada vez mais ganhando espaço no mundo do desenvolvimento de *software* por facilitar a manutenção do código e o desenvolvimento em equipe.

2.2- Plataforma de Desenvolvimento (Eclipse)

O Eclipse é um projeto de código aberto iniciado na IBM do Canadá que em 2001 foi continuado por um grupo de desenvolvedores culminando com a criação da Eclipse Foundation em 2004, que mantém o projeto até hoje.

O programa em si é uma Plataforma de Desenvolvimento Integrada (IDE), escrito totalmente em Java e que tem como principais objetivos prover recursos capazes de facilitar o trabalho dos desenvolvedores. Dentre esses temos ferramentas avançadas de *debug*, *deploy*, *refactor*, controle de versões além de outras que visam facilitar o desenvolvimento em equipe.

O Eclipse foi escolhido como plataforma de desenvolvimento a ser utilizada devido à sua grande aceitação tanto no meio acadêmico como profissional e por ter sido desenvolvida com o propósito inicial de ser uma IDE para o desenvolvimento em linguagem Java. Dessa forma, o Eclipse, em sua forma básica, consiste na Java Development Tools (JVT), que são as ferramentas básicas para o desenvolvimento Java.

No entanto seu grande ponto positivo se deve ao fato de ele ser baseado em *plugins*, o que o torna uma ferramenta de desenvolvimento muito versátil. Essa sua arquitetura aliada à sua grande utilização fez com que diversos *plugins* fossem desenvolvidos para as mais diversas funcionalidades. Dentre esses podemos citar alguns que facilitam o uso de ferramentas de versionamento e, no caso especifico desse projeto, um que permite a integração do servidor *Web* Apache Tomcat com o Eclipse.

Com a utilização dessa ferramenta o desenvolvimento de aplicações *Web* fica mais simples e, conseqüentemente mais produtivo, uma vez que tudo que é necessário ser feito para rodar e testar a aplicação pode ser executado de dentro do Eclipse. Entre essas tarefas necessárias podemos incluir a parada e reinicio do servidor Web, o *deploy* de classes e bibliotecas nas pastas corretas e a possibilidade de se debugar a aplicação.

2.3 - MySQL

Para o armazenamento e gerenciamento dos dados orçamentários e de usuários é necessária a utilização de um sistema gerenciador de banco de dados (SGBD). Para esse projeto foi decidido que seria utilizado o MySQL, que é um *software* de código aberto já bastante testado devido à sua popularidade, tendo assim uma estabilidade garantida.

O MySQL utiliza a linguagem SQL para acessar os dados persistidos no banco.

Um ponto positivo também levado em conta na escolha do *software* é o fato de ele oferecer *plugins* que fornecem uma interface gráfica simples e que facilita bastante a criação e administração do banco de dados. Entre esses *plugins* foram utilizados principalmente o MySQL Administrator e o Query Browser, que fornecem uma visão geral sobre o banco e uma interface para a interação com as tabelas diretamente em linguagem SQL, respectivamente.

2.4 - Apache Tomcat

O Apache Tomcat, ou simplesmente Tomcat, é um servidor Web livre para aplicações Java desenvolvido pela Apache Software Foundation, fazendo parte do projeto Apache Jakarta. É um servidor muito robusto, sendo utilizado inclusive em ambientes de produção e por isso, é oficialmente "divulgado" pela Sun Microsystems como sendo uma implementação de referência para as tecnologias *Java Servlets* e *Java Server Pages* (JSPs).

No caso desse projeto ele foi utilizado exatamente dessa forma, servindo como um servidor para receber as requisições através de JSPs e tratá-las com as funções Java rodando no *back-end*.

2.5 - Hibernate

Hibernate é uma biblioteca que permite o mapeamento de um modelo de banco de dados relacional utilizando uma sintaxe orientada a objeto e facilita a realização de *queries* na base de dados. Ela permite o desenvolvimento de classes persistentes utilizando conceitos de orientação a objeto, como associação, herança, polimorfismo, composição e coleção. No que diz respeito à utilização de *queries* ela permite que as mesmas sejam feitas em SQL ou HQL, sendo esta última uma extensão do SQL criada para Hibernate que possibilita a execução de *queries* sobre os objetos Java e não sobre as tabelas do banco, ou ainda através de uma API orientada a objetos. Dessa forma ele permite que haja uma separação de camadas entre o banco de dados (persistência) e as regras de negócio (código).

Hibernate utiliza arquivos XML para fazer o mapeamento das classes de persistência em relação ao banco de dados e com isso evita que o desenvolvedor tenha que se preocupar com a

conversão de dados Java para SQL, por exemplo. Além disso, simplifica o processo de acesso ao banco, gerando as chamadas SQL e tratando os resultados da forma adequada.

2.6 - Flex

Para o desenvolvimento da interface gráfica através da qual o usuário irá interagir com o sistema foi escolhido o *framework* Flex. Flex é um *framework* de código aberto relativamente novo, desenvolvido pela Adobe que permite o desenvolvimento de aplicações Web bastante amigáveis e interativas que rodam em praticamente todos os principais navegadores de internet e sistemas operacionais.

Projetos implementados em Flex consistem basicamente de arquivos escritos em MXML, que é uma linguagem declarativa baseada em XML, nos quais são descritos os *layouts* das interfaces e como elas devem responder a determinadas ações do usuário. Para tratar esses eventos disparados pelo usuário, os arquivos MXML geralmente chamam funções e métodos escritos em ActionScript, que é uma linguagem de programação orientada a objeto bastante versátil. Através da programação em ActionScript é possível criar lógicas e implementar algoritmos para tornar a aplicação Flex mais abrangente e interativa.

As aplicações Flex são executadas diretamente nos navegadores dos usuários, utilizando o Adobe Flash Player para isso. Isso permite que elas não sofram grandes alterações ou sejam muito afetadas por diferentes sistemas operacionais e navegadores de internet.

No caso desse projeto foi utilizado o Adobe Flex 3.

2.7 - Flex Builder

Para facilitar o desenvolvimento em Flex, a Adobe disponibiliza uma plataforma de desenvolvimento (IDE) chamada Flex Builder. Essa ferramenta é toda construída tendo como base a plataforma do Eclipse, que já foi detalhada anteriormente. A utilização do Flex Builder acelera o processo de forma muito semelhante ao Eclipse, facilitando a criação de dependências, agilizando o processo de *build,* entre outras vantagens. O Flex Builder é disponibilizado gratuitamente a estudantes, bastando para isso fazer a requisição de um código (*serial key*) na página da Adobe.

2.8 - Ant

Ao se realizar um projeto de um sistema de informação *web*, algumas vezes pode se tornar inconveniente, ou até mesmo improdutivo, o fato de ter que compilar o código e mover os arquivos binários para os seus devidos locais no servidor *Web*. Por isso foi utilizado nesse projeto a ferramenta Apache Ant, que auxilia na compilação e no *build* de projetos. Ela foi implementada em Java e é reconhecidamente mais adequada para a utilização em projetos desenvolvidos nessa mesma linguagem de programação. Ant utiliza XML para que o desenvolvedor possa descrever o que deve ser feito durante o *build* de um projeto.

3- Análise:

Para que o desenvolvimento do projeto pudesse transcorrer de forma tranqüila e com qualidade, foi necessário seguir um padrão de projeto de Engenharia de Software. Atualmente existem diversos padrões aceitos pela comunidade mundial e, no nosso caso, foi utilizado um padrão aplicável ao desenvolvimento orientado a objeto.

O primeiro passo para o desenvolvimento de um sistema é fazer uma análise do problema que esse sistema pretende solucionar, ou, ao menos, reduzir. Nessa fase são necessárias reuniões com o cliente para que seja possível obter dele o que realmente deverá ser implementado, definir o escopo exato do projeto e começar a pensar em uma possível modelagem do sistema.

Durante essa fase do projeto é de grande importância que o analista compreenda muito bem os desejos do cliente, uma vez que é a partir desse entendimento que vão ser desenvolvidos os modelos que virão a guiar a implementação do sistema propriamente dito.

No caso do sistema de controle orçamentário essa fase consistiu, em grande parte, na definição dos dados que deveriam ser tratados para que um orçamento pudesse ser representado de uma forma semelhante à que o cliente já utilizava anteriormente e também de forma que essa representação permitisse o tratamento de um orçamento da forma mais natural possível.

Além disso, devem ser definidos também os requisitos não funcionais que o sistema deverá respeitar. Requisitos não funcionais são as qualidades globais de um sistema. Eles não são funções que deverão ser realizadas pelo sistema, e sim comportamentos e restrições às quais o sistema deve obedecer. No caso do sistema de controle orçamentário, os requisitos não funcionais que foram levantados foram amigabilidade, segurança e manutenabilidade. Isso diz que o sistema deve ter uma interface simples para o usuário, de forma que ele consiga utilizar as funcionalidades de maneira natural e intuitiva. Além disso, por ser um sistema que lida com dinheiro, é importante que ele tenha segurança a fim de impedir que pessoas não autorizadas o utilizem. Outro requisito ao qual o sistema deverá obedecer é o de manutenabilidade, ou seja, deverá ser um sistema desenvolvido de tal forma que facilite a manutenção do mesmo no futuro.

Outra parte importante dessa etapa do projeto é a definição dos casos de uso do sistema. Essa definição permite entender como o sistema deve reagir a certos fatores externos, como a interação do usuário com o mesmo, por exemplo. Além disso, a definição dos casos de uso pode ajudar a detectar falhas na escolha dos dados que devem ser tratados, o que é de grande valia,

uma vez que nessa fase do projeto nada foi implementado ainda, tornando mais rápidas e fáceis as correções a serem feitas.

Esse processo de análise pode ser feito de duas maneiras distintas: partindo do problema como um todo (geral) e detalhando o processo até pequenas partes (particular) e da forma oposta, partindo de detalhes (particular) para o todo (geral). No caso do projeto do sistema de controle orçamentário foi utilizado o segundo método citado, partindo de detalhes para se chegar ao sistema como um todo. Essa escolha se deu, simplesmente, por questão de preferência pessoal, uma vez que os dois métodos poderiam gerar resultados positivos se executados de forma correta.

3.1 - Modelagem

A partir da análise feita no primeiro passo do projeto, se torna possível efetuar a modelagem do sistema. Isso é feito utilizando as definições obtidas pela análise, como a lista de casos de uso e os dados que devem ser tratados, que podem ser transformados em um modelo de classes.

3.1.1 – Casos de Uso:

Os casos de uso são todas as maneiras através das quais o sistema terá que interagir com o usuário ou com outros sistemas a partir de determinado estímulo. No sistema de controle orçamentário, esses casos de uso se restringem a estímulos vindos do usuário, uma vez que ele não interage com nenhum outro sistema. Foram definidos então os casos de uso que seguem.

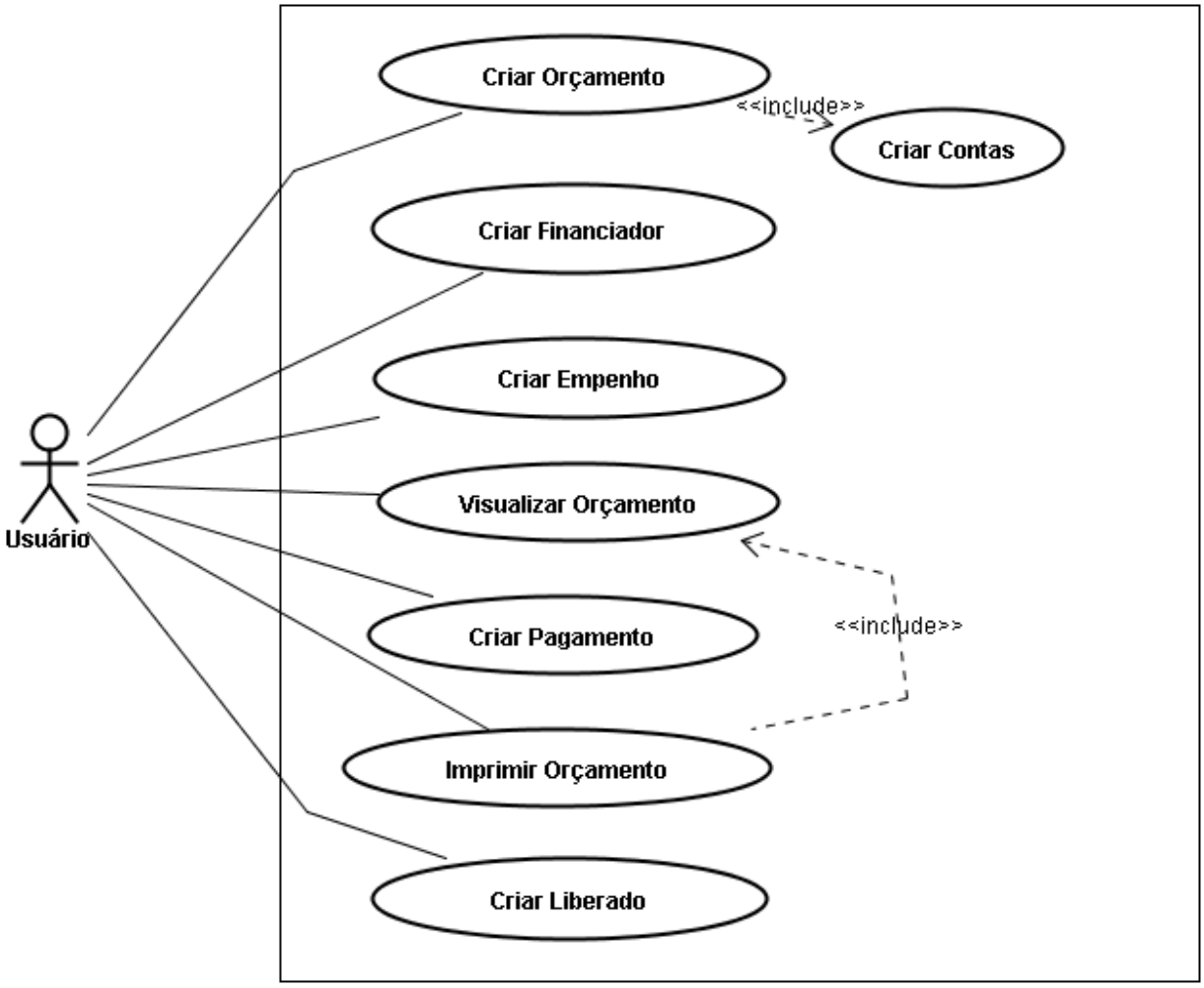


Figura 1 – Diagrama de Casos de Uso

- criar orçamento
- criar financiador
- criar empenho
- criar pagamento
- criar liberado
- visualizar orçamento
- imprimir visualização

Alguns dos casos de uso definidos acima podem ser divididos em blocos mais simples, tendo como objetivo modularizar o sistema, facilitando assim a sua compreensão. Isso agiliza a

etapa de codificação e também torna os testes mais pontuais e simples, além de aumentar a manutenabilidade do sistema quando este entrar em produção. Esse detalhamento dos casos de uso é feito abaixo.

a) Criar Orçamento

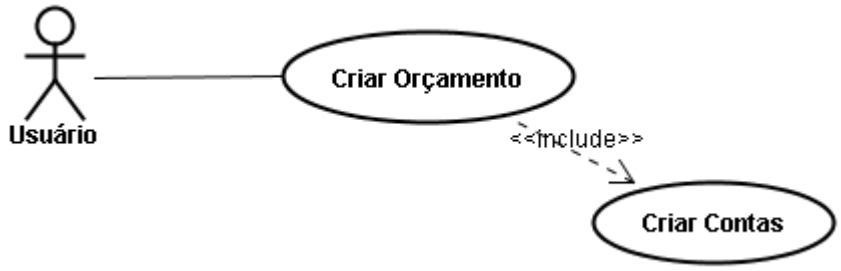


Figura 2 – Caso de Uso (Criar Orçamento)

- **Atores:** Usuário
- **Objetivo:** Permitir a criação de um novo orçamento no sistema
- **Descrição:** O usuário será apresentado a uma tela na qual ele deve inserir os dados básicos de identificação do orçamento (Nome, Descrição e Ano Corrente) e também a fonte à qual o orçamento está relacionado. Ao concluir essa etapa será apresentada uma nova interface que permitirá ao usuário a criação das contas que estarão presentes nesse orçamento. Nessa tela ele deverá inserir os dados de identificação da conta (Nome e Descrição)

b) Criar Financiador

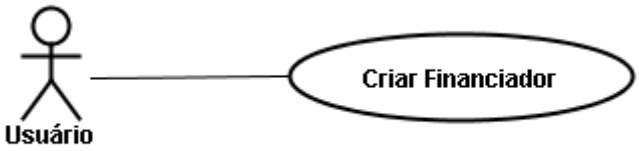


Figura 3 – Caso de Uso (Criar Financiador)

- **Atores:** Usuário
- **Objetivo:** Permitir a criação de um novo financiador no sistema. Esse financiador poderá fazer parte de nenhuma ou várias fontes
- **Descrição:** O usuário será apresentado a uma tela na qual ele deve inserir os dados de identificação do financiador. Ao concluir essa etapa o financiador será inserido no sistema e uma tela de sucesso será mostrada ao usuário

c) Criar Empenho

Figura 4 – Caso de Uso (Criar Empenho)

- **Atores:** Usuário
- **Objetivo:** Permitir a criação de um novo empenho relacionado a uma conta já existente no sistema
- **Descrição:** O usuário será apresentado a uma tela na qual ele deve selecionar o orçamento e a conta aos quais o empenho está relacionado e inserir tanto os dados de identificação do empenho (Nome e Descrição) como os dados financeiros do mesmo

(Valor). Ao concluir essa etapa o empenho será inserido no sistema, a conta à qual ele se refere atualizada e uma tela de sucesso será apresentada.

d) Criar Pagamento

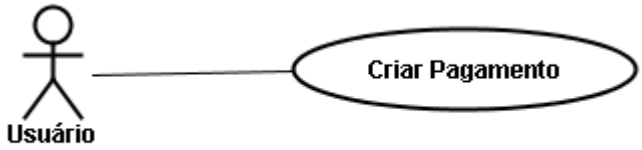


Figura 5 – Caso de Uso (Criar Pagamento)

- **Atores:** Usuário
- **Objetivo:** Permitir a criação de um novo pagamento relacionado a um empenho já existente no sistema
- **Descrição:** O usuário será apresentado a uma tela na qual ele deve selecionar o orçamento, a conta e o empenho aos quais o pagamento está relacionado e inserir tanto os dados de identificação do pagamento (Nome e Descrição) como os dados financeiros do mesmo (Valor e Beneficiário). Ao concluir essa etapa o pagamento será inserido no sistema, o empenho e a conta aos quais ele se refere serão atualizados e uma tela de sucesso será apresentada.

e) Criar Liberado

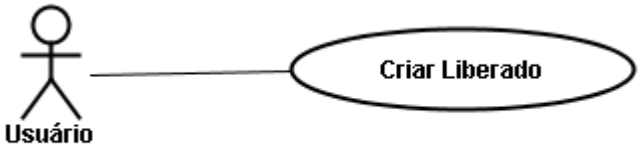


Figura 6 – Caso de Uso (Criar Liberado)

- **Atores:** Usuário
- **Objetivo:** Permitir a criação de um novo liberado relacionado a uma conta já existente no sistema
- **Descrição:** O usuário será apresentado a uma tela na qual ele deve selecionar o orçamento e a conta aos quais o liberado está relacionado e inserir tanto os dados de identificação do liberado (Nome e Descrição) como os dados financeiros do mesmo (Valor). Ao concluir essa etapa o liberado será inserido, a conta à qual ele se refere será atualizada e uma tela de sucesso será apresentada.

f) Visualizar Orçamento

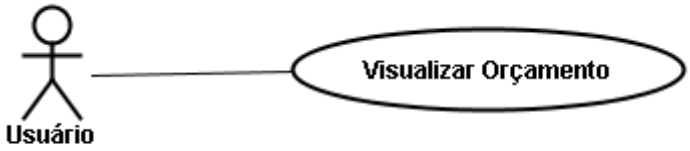


Figura 7 – Caso de Uso (Visualizar Orçamento)

- **Atores:** Usuário
- **Objetivo:** Permitir a visualização dos dados de um orçamento existente no sistema
- **Descrição:** O usuário será apresentado a uma tela na qual ele deve selecionar o orçamento que deseja visualizar. Ao concluir essa etapa será exibida uma tabela com

dados de todas as contas pertencentes àquele orçamento. Caso o usuário deseje obter detalhes sobre os empenhos ou liberados, ele deve selecionar a conta na tabela e duas novas tabelas serão exibidas com os dados dos empenhos referentes à conta selecionada. Caso ele ainda deseje visualizar os dados dos pagamentos referentes a algum empenho, ele deve repetir o procedimento selecionando dessa vez um dos empenhos na tabela. Com isso uma nova tabela é exibida mostrando os detalhes dos pagamentos pertencentes àquele empenho.

g) Imprimir Orçamento

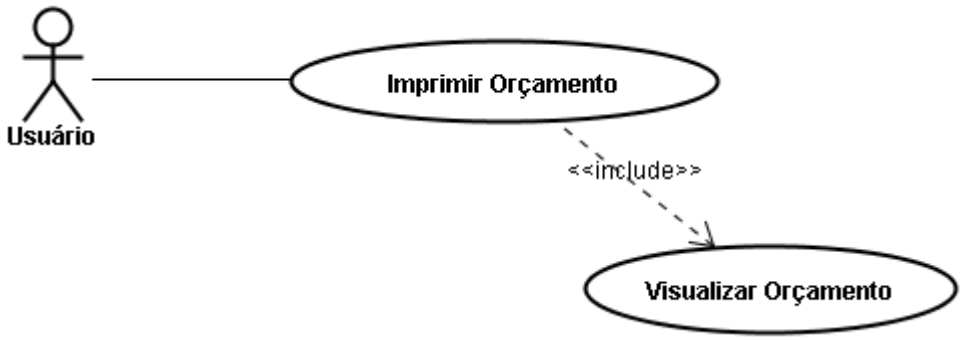


Figura 8 – Caso de Uso (Imprimir Orçamento)

- **Atores:** Usuário
- **Objetivo:** Permitir a impressão dos dados do orçamento a partir da tela de visualização dos mesmos
- **Descrição:** O usuário terá à sua disposição um botão na tela de visualização que permitirá que ele imprima os dados exibidos nas tabelas existentes na tela. Ao clicar no botão os dados serão enviados à impressora e posteriormente impressos.

A partir desses casos de uso se torna mais fácil a definição dos métodos que deverão ser implementados na fase de codificação, o que permite uma maior produtividade nessa etapa do desenvolvimento.

3.1.2 Diagrama de Classes

Diagrama de Classes é um diagrama que mostra as classes de um sistema e seus relacionamentos, mostrando seus atributos e métodos, além de informar a cardinalidade das relações.

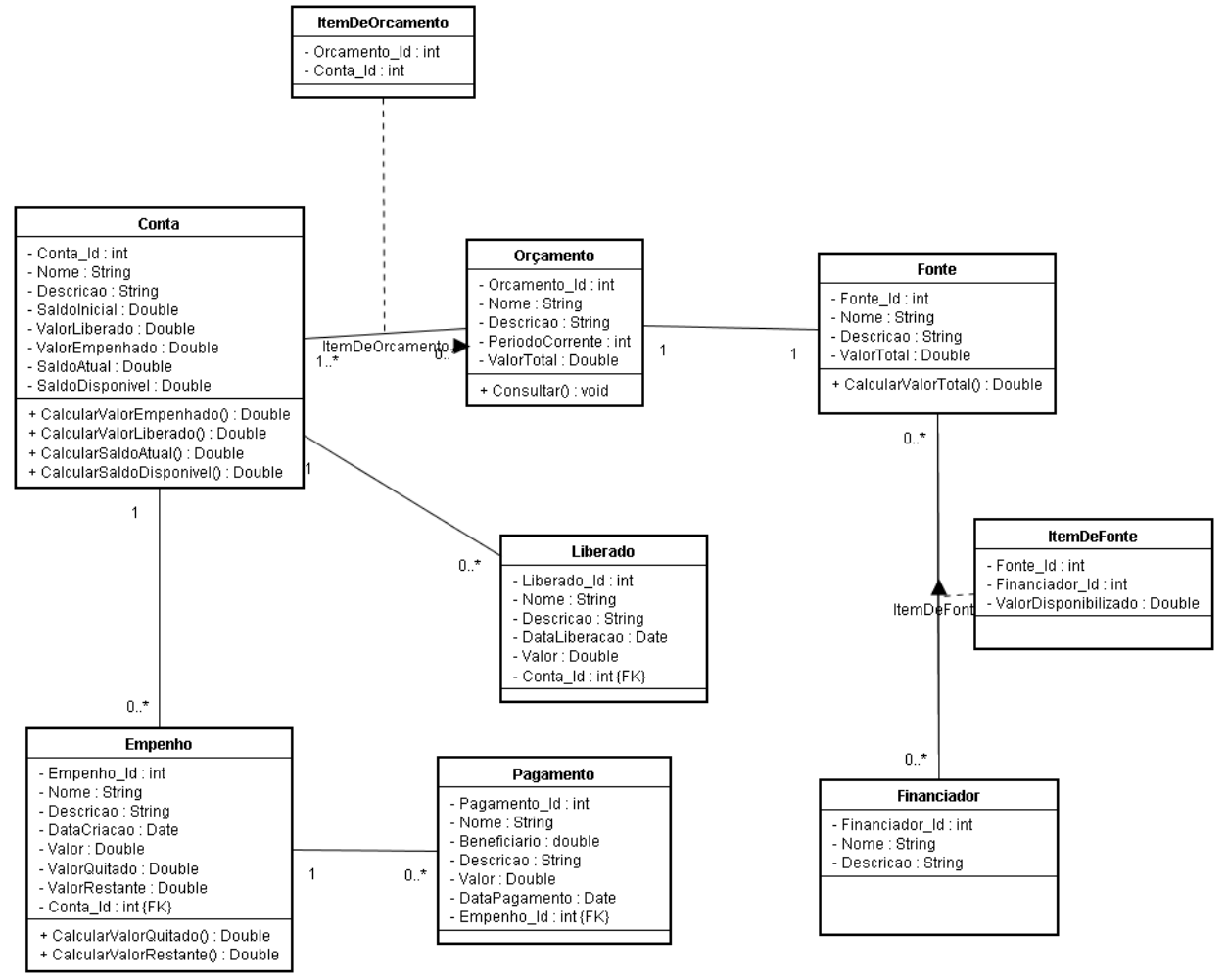


Figura 9 – Diagrama de Classes

As classes definidas no diagrama de classes seguem detalhadas abaixo.

a) *Orçamento* – classe que representa um orçamento, armazenando os dados de identificação do mesmo.

- Atributos
  - Orcamento_id – código que identifica unicamente o orçamento (gerado automaticamente pelo sistema)
  - Nome – nome dado ao orçamento pelo usuário
  - Descrição – texto curto com uma breve descrição do orçamento feita pelo usuário
  - PeríodoCorrente – ano ao qual o orçamento se refere
  - ValorTotal – valor total do orçamento calculado a partir da fonte do mesmo

b) *Fonte* – classe que representa a fonte financeira do orçamento, sendo formada por um ou mais financiadores

- Atributos
  - Fonte_id – código que identifica unicamente a fonte (gerado automaticamente pelo sistema)
  - Nome – nome dado à fonte pelo usuário
  - Descrição – texto curto com uma breve descrição da fonte feita pelo usuário
  - ValorTotal – valor total disponibilizado pela fonte

- Métodos
  - calcularValorTotal – método que calcula o valor total disponibilizado pela fonte, levando em consideração todos os financiadores presentes na mesma

c) *Financiador* – classe que representa um financiador de um orçamento, contendo seus dados de identificação

- Atributos
  - Financiador_id – código que identifica unicamente o financiador (gerado automaticamente pelo sistema)
  - Nome – nome do financiador
  - Descrição – texto curto com uma breve descrição do financiador feita pelo usuário

d) *Conta* – classe que representa uma conta de um orçamento, contendo seus dados de identificação e dados monetários da conta

- Atributos
  - Conta_id – código que identifica unicamente a conta (gerado automaticamente pelo sistema)
  - Nome – nome dado à conta pelo usuário
  - Descrição – texto curto com uma breve descrição da conta feita pelo usuário
  - SaldoInicial – valor que representa o saldo inicial estipulado pelo usuário para a conta
  - SaldoAtual – valor que representa o saldo atual da conta, sendo calculado subtraindo o valor já quitado do saldo inicial da conta
  - SaldoDisponível – valor que representa o saldo disponível da conta, sendo calculado subtraindo o valor empenhado do valor liberado da conta
  - ValorLiberado – valor que representa o montante liberado presente na conta, calculado a partir dos liberados relacionados à conta
  - ValorEmpenhado – valor que representa o montante empenhado da conta, calculado a partir dos empenho relacionados à conta

- Métodos
  - calcularValorEmpenhado – método que calcula o valor total empenhado na conta, levando em consideração todos os empenhos presentes na mesma
  - calcularValorLiberado – método que calcula o valor total liberado na conta, levando em consideração todos os liberados presentes na mesma
  - calcularSaldoAtual – método que calcula o saldo atual da conta, levando em consideração o saldo inicial dela e o valor já quitado presente na mesma
  - calcularSaldoDisponivel – método que calcula o saldo disponível na conta, levando em consideração os montantes liberados e empenhados presentes na mesma

e) *Liberado* – classe que representa um liberado de uma conta, contendo seus dados de identificação e dados monetários do mesmo. Um "Liberado" é um montante que está previsto no orçamento que se torna disponível para uso.

- Atributos
  - Liberado_id – código que identifica unicamente o liberado (gerado automaticamente pelo sistema)
  - Nome – nome dado ao liberado pelo usuário
  - Descrição – texto curto com uma breve descrição do liberado feita pelo usuário
  - DataLiberação – data que representa o dia em que o montante foi liberado
  - Valor – valor que representa o montante liberado
  - Conta_id – código que representa a conta à qual o liberado está relacionado

f) *Empenho* – classe que representa um empenho de uma conta, contendo seus dados de identificação e dados monetários do empenho. Um empenho é um montante que é destinado, ou seja, reservado para ser gasto em determinada necessidade. Ao se empenhar uma determinada quantia, essa só poderá ser utilizada para cobrir as despesas para as quais ela foi empenhada.

- Atributos
  - Empenho_id – código que identifica unicamente o empenho (gerado automaticamente pelo sistema)
  - Nome – nome dado ao empenho pelo usuário
  - Descrição – texto curto com uma breve descrição do empenho feita pelo usuário
  - DataCriacao – data que representa o dia de criação do empenho
  - Valor – valor que representa o montante empenhado
  - ValorQuitado – valor que representa o montante já quitado do valor empenhado, calculado a partir dos pagamentos relacionados ao empenho
  - ValorRestante – valor que representa o montante restante do empenho, calculado subtraindo o ValorQuitado do Valor do empenho
  - Conta_id – código que representa a conta à qual o empenho está relacionado

- Métodos
  - calcularValorQuitado – método que calcula o valor total já quitado do empenho, levando em consideração todos os pagamentos presentes no mesmo
  - calcularValorRestante – método que calcula o valor restante do empenho, levando em consideração o valor inicial do mesmo e o montante já quitado

g) *Pagamento* – classe que representa um pagamento de um empenho, contendo seus dados de identificação e dados monetários do pagamento.

- Atributos
  - Pagamento_id – código que identifica unicamente o pagamento (gerado automaticamente pelo sistema)
  - Nome – nome dado ao pagamento pelo usuário
  - Descrição – texto curto com uma breve descrição do pagamento feita pelo usuário
  - DataPagamento – data que representa o dia em que o pagamento foi realizado
  - Beneficiário – valor que representa o beneficiário do pagamento, ou seja, a pessoa física ou jurídica que recebe o montate quitado.
  - Valor – valor que representa o montante quitado
  - Empenho_id – código que representa o empenho ao qual o pagamento está relacionado

3.1.3 Diagrama de Seqüência

O próximo diagrama modelado foi o diagrama de sequência. Esse diagrama tem como principal objetivo mostrar a interações entre os objetos do sistema e a ordem em que essas interações acontecem. O diagrama de sequência ajuda a dar uma visão mais detalhada dos casos de uso, uma vez que mostra passo a passo a sequência de atividades que ocorrem enquanto uma funcionalidade está sendo executada.

No caso do nosso projeto vamo mostrar um diagrama de sequência genérico para todos os casos de uso, que pode ser observado na figura abaixo.

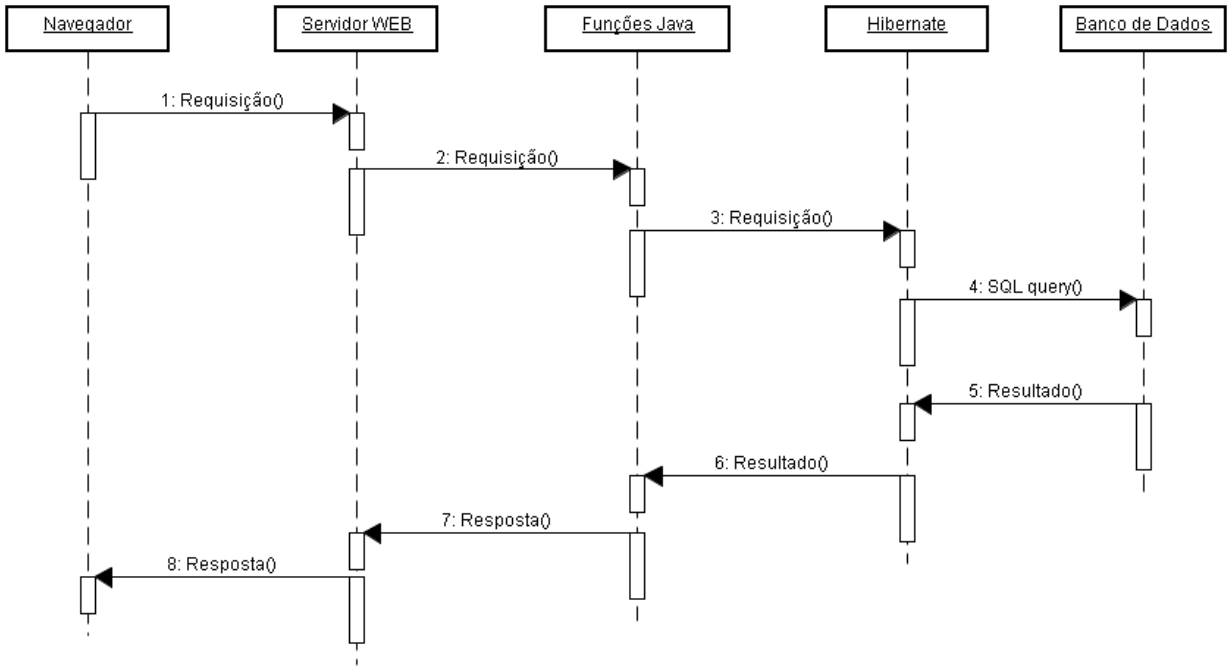


Figura 10 – Diagrama de Seqüência

A partir desse diagrama geral poderíamos desenhar um específico para cada caso de uso do sistema, especificando exatamente quais métodos seriam chamados, qual *query* seria executada e a resposta que seria recebida. Isso no entanto não foi necessário e causaria até uma repetição extrema de dados, uma vez que os casos de uso definidos anteriormente têm as mesmas estapas e relacionamentos entre os objetos.

4- Desenvolvimento

Após a realiazação da etapa de análise do problema e já tendo definido as classes, seus relacionamentos e todos os casos de uso do sistema, partimos para a implementação propriamente dita. Nessa fase utilizamos uma metodologia na qual dividimos o projeto em blocos e os desenvolvemos separadamente, efetuando testes isolados e deixando a integração para o final. A princípio o projeto foi dividido em três grandes blocos, que estão representados na figura 11: persistência, onde foi utilizado o software MySql para guardar os dados e onde foram implementadas classes em Java que representam as tabelas do banco, com a função de guardar temporariamente os dados a serem inseridos ou os que foram recuperados; interface, que foi implementada utilizando Flex e ActionScript; e o bloco de controle, com a implementação das funcionalidades e da integração das camadas de interface e banco de dados através de código escrito em linguagem Java, com a utilização de JSPs. Esse desenvolvimento foi feito dessa forma tomando como base a arquitetura MVC (Model-View-Controller). O principal objetivo de se densenvolver utilizando essa arquitetura é conseguir uma separação clara das camadas do projeto, possibilitando uma maior manutenabilidade do sistema e também facilitando possíveis alterações do projeto.

A arquitetura MVC define que devem existir três tipos camadas no sistema: camada de modelo (*model)*, camada de controle (*controller*) e camada de interface (*view)*.

A camada de modelo (*model*) contém os dados e define a lógica para a manipulação dos mesmos, correspondendo no nosso sistema ao bloco “persistência”.

A camada de interface (*view*) é responsável por exibir informações para o usuário, sendo, portando, a interface gráfica do sistema, correspondendo ao nosso bloco “interface”.

Os objetos de controle (*controller*) são os responsáveis por fazer a integração entre os objetos de modelo e os objetos de interface. São eles que recebem os eventos gerados na camada *view* e respondem da forma adequada, que é definida pela regra de negócio do caso de uso específico. Podem requisitar alterações nos dados de persistência, como *insert* ou *update* em tabelas do banco. No nosso modelo é correspondente ao bloco “business”.

O modelo da arquitetura que foi utilizada no sistema de controle orçamentário segue abaixo.

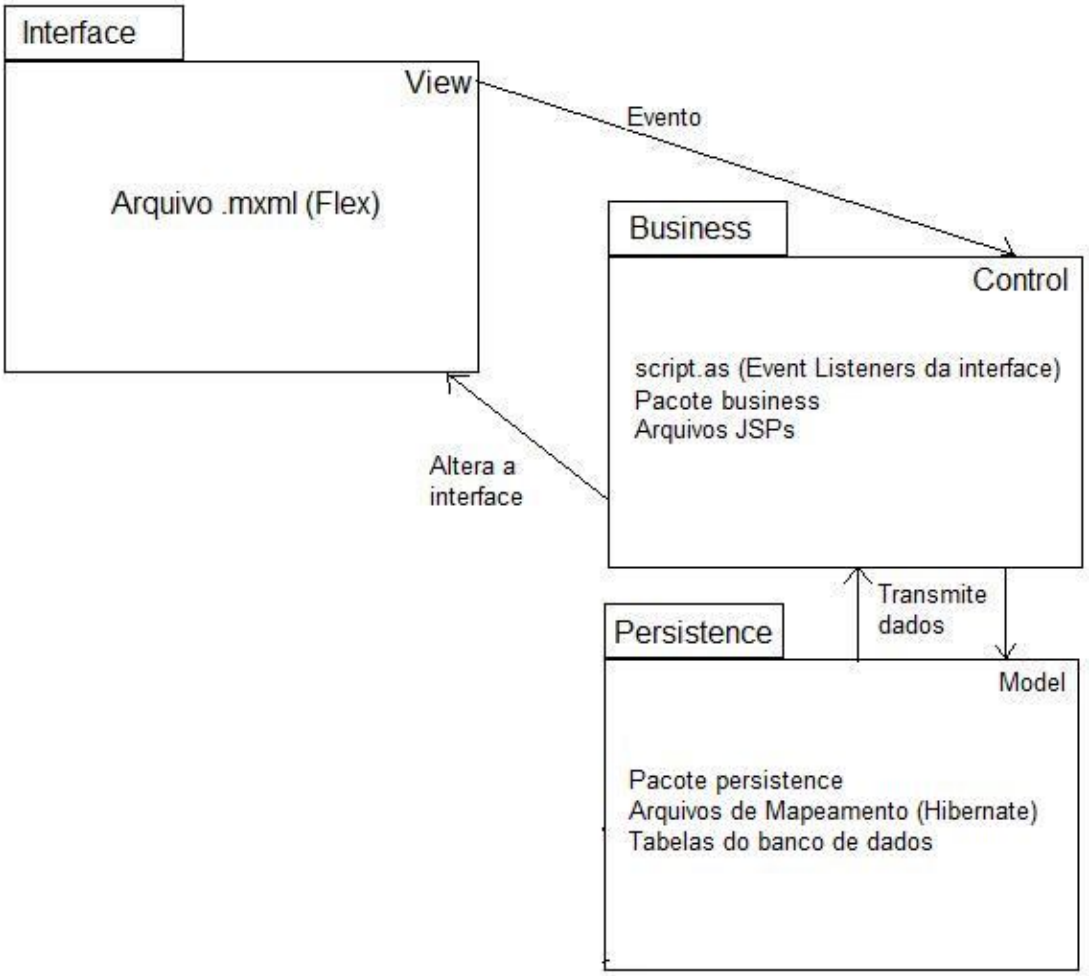


Figura 11 – Arquitetura do Sistema

O primeiro bloco desenvolvido foi o do banco de dados, que faz parte da camada de modelo do sistema (*model*) e é reponsável por fazer a persistência dos dados do sistema. Essa

parte da implementação consistiu simplesmente em criar as tabelas no banco de dados baseadas nos objetos definidos na etapa de análise. As tabelas e todas as operações realizadas nessa fase foram executadas utilizando um *plugin* do MySql que se chama MySql Administrator. Esse *plugin* disponibiliza uma interface gráfica para facilitar a manipulação de tabelas e outras funcionalidades do banco que não foram utilizadas nesse projeto. As tabelas que foram criadas seguem abaixo com as devidas descrições.

Orcamento:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
ORCAMENTO_ID	INTEGER	✔	✔	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico do orcamento
VALOR_TOTAL	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor total do orcamento vindo da fonte
ORCAMENTO_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome do orcamento
ORCAMENTO_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Descricao do orcamento
PERIODO_CORRENTE	INTEGER			☑ UNSIGNED ☐ ZEROFILL	NULL	Periodo corrente do orcamento

Figura 12 – Tabela ORCAMENTO

Fonte:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
FONTE_ID	INTEGER	✔	✔	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico da fonte
FONTE_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome da fonte
FONTE_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Descricao da fonte
VALOR_TOTAL	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor total disponibilado pela fonte vindo dos financiadores
ORCAMENTO_ID	INTEGER	✔		☑ UNSIGNED ☐ ZEROFILL	NULL	

Figura 13 – Tabela FONTE

Financiador:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
FINANCIADOR_ID	INTEGER	✔	✔	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico do financiador
FINANCIADOR_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome do financiador
FINANCIADOR_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Descricao do financiador

Figura 14 – Tabela FINANCIADOR

Conta:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
CONTA_ID	INTEGER	✔	✔	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico da conta
CONTA_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome da conta
CONTA_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Descricao da conta
SALDO_INICIAL	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Saldo inicial da conta
VALOR_LIBERADO	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Montante liberado presente na conta
VALOR_EMPENHADO	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Montante empenhado presente na conta
SALDO_ATUAL	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor calculado subtraindo o montante quitado do saldo inicial da conta
SALDO_DISPONIVEL	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor calculado subtraindo o montante emepenhado do liberado

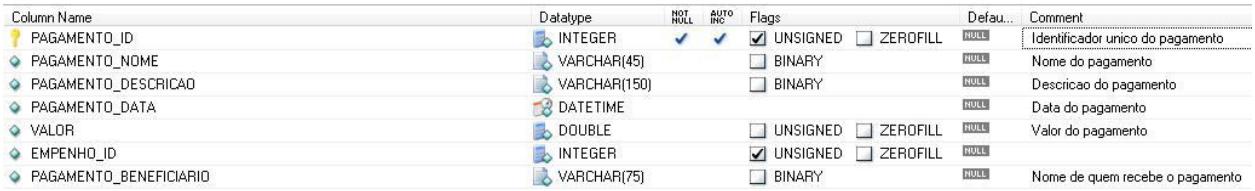
Figura 15 – Tabela CONTA

Empenho:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
EMPENHO_ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico do empenho
EMPENHO_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome do empenho
EMPENHO_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Descricao do empenho
VALOR	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor do empenho
VALOR_QUITADO	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor ja quitado do empenho
VALOR_DISPONIVEL	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor calculado subtraindo o valor quitado do valor do empenho
DATA_CRIACAO	DATETIME				NULL	Data de criacao do empenho
CONTA_ID	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico da conta a qual o empenho esta relacionado

Figura 16 – Tabela EMPENHO

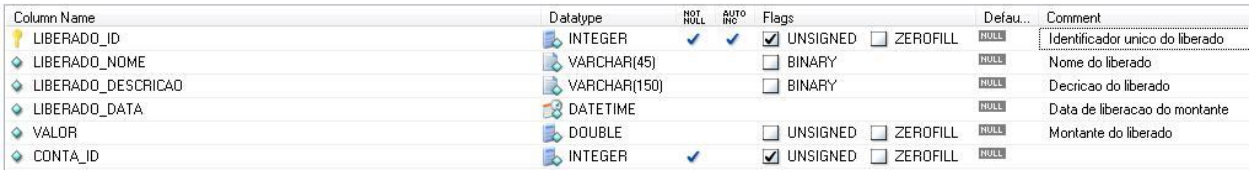
Pagamento:



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
PAGAMENTO_ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico do pagamento
PAGAMENTO_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome do pagamento
PAGAMENTO_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Descricao do pagamento
PAGAMENTO_DATA	DATETIME				NULL	Data do pagamento
VALOR	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Valor do pagamento
EMPENHO_ID	INTEGER			☑ UNSIGNED ☐ ZEROFILL	NULL	
PAGAMENTO_BENEFICIARIO	VARCHAR(75)			☐ BINARY	NULL	Nome de quem recebe o pagamento

Figura 17 – Tabela PAGAMENO

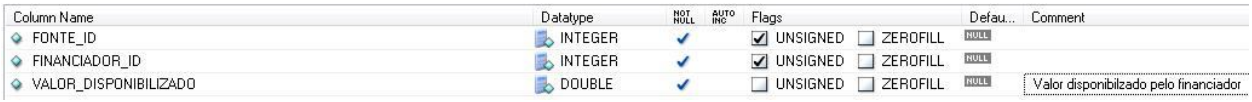
Liberado:



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
LIBERADO_ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	Identificador unico do liberado
LIBERADO_NOME	VARCHAR(45)			☐ BINARY	NULL	Nome do liberado
LIBERADO_DESCRICAO	VARCHAR(150)			☐ BINARY	NULL	Decricao do liberado
LIBERADO_DATA	DATETIME				NULL	Data de liberacao do montante
VALOR	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL	Montante do liberado
CONTA_ID	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	NULL	

Figura 18 – Tabela LIBERADO

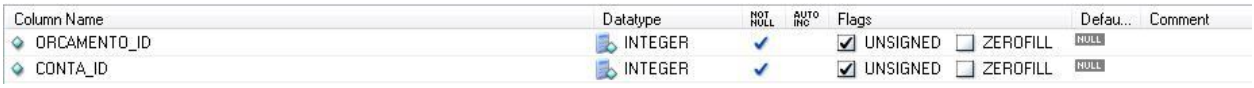
Fonte_Financiador:



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
FONTE_ID	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	NULL	
FINANCIADOR_ID	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	NULL	
VALOR_DISPONIBILIZADO	DOUBLE	✓		☐ UNSIGNED ☐ ZEROFILL	NULL	Valor disponibilzado pelo financiador

Figura 19 – Tabela FONTE_FINANCIADOR

Item_de_Orcamento:



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Defau...	Comment
ORCAMENTO_ID	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	NULL	
CONTA_ID	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	NULL	

Figura 20 – Tabela ITEM_DE_ORCAMENTO

Com o banco de dados já criado e em condições de receber informação, passamos ao desenvolvimento dos blocos implementados em Java. Esses blocos possuem objetos tanto da camada de modelo (*model*) quanto da camada de controle (*control*). A primeira parte da codificação feita foi relacionada à integração do código em Java com o banco de dados. Em

seguida partimos para o desenvolvimento da parte de controle do projeto, onde o código em Java precisa ser integrado à interface Flex do sistema.

Durante a primeira etapa de codificação Java citada acima foram implementadas as classes de domínio do problema no pacote "persistence", que são aquelas têm correspondência com a tabelas do banco de dados. Elas fazem parte da camada de modelo (*model*), uma vez que contém todos os métodos *getters and setters*, que são responsáveis pela manipulação dos dados da persistência. Essas classes são as que vão ser utilizadas tanto para inserir dados no banco quanto para recuperar os mesmos de lá. Isso é feito criando uma instância da classe que vai armazenar os dados que irão ser inseridos ou retirados do banco, sendo que eles são aí setados através dos métodos *getters* e *setters* que foram implementados para as classes. Portanto essas classes criadas têm relação direta com as tabelas do banco de dados e são as seguintes:

- *Orcamento.java*
- *Fonte.java*
- *Financiador.java*
- *Conta.java*
- *Empenho.java*
- *Liberado.java*
- *Pagamento.java*

Como foi usado Hibernate para fazer a integração do código Java com o banco de dados, foi necessária a criação de arquivos de mapeamento (*mapping files*) para todas essas classes. Esses arquivos de mapeamento são responsáveis por mapear os atributos da classe Java de acordo com as colunas e tabelas do banco de dados para permitir que o Hibernate seja capaz de salvar, atualizar e recuperar os dados corretamente para e do banco de dados. Nesses arquivos também são descritas as associações entre as classes, o que permite que o Hibernate seja capaz de entender a existência das relações de multiplicidade existentes no banco e até mesmo a existência de tabelas asssociativas em associações “n X m”. Os arquivos de mapeamento são de vital importância para o desenvolvimento do resto do projeto, uma vez que eles são fundamentais para que o Hibernate funcione de maneira correta e facilite a integração do código com o banco de dados. Por fazer o papel de manipulação dos dados do sistema, essa parte da codificação também faz parte do bloco de persistência do sistema. Os arquivos criados foram os seguintes:

- *Orcamento.hbm.xml*
- *Fonte. hbm.xml*
- *Financiador. hbm.xml*
- *Conta. hbm.xml*
- *Empenho. hbm.xml*
- *Liberado. hbm.xml*
- *Pagamento. hbm.xml*

Além dos arquivos de mapeamento o Hibernate necessita também de um arquivo de configuração, no qual é preciso inserir as informações sobre o banco de dados ao qual a conexão deve ser feita. Esses dados necessários são o endereçao do banco, o driver de conexão do banco e os dados para a autenticação do usuário no banco, além da enumeração de todos os arquivos de mapeamento sendo utilizados no projeto. Esse arquivo é chamado de *hibernate.cfg.xml*.

Com os arquivos de mapeamento e as classes do domínio já criadas passamos a implementar as operações no banco, como *insert* e *update*. Essas classes foram criadas no pacote "business". Elas são responsáveis por fazer os acessos ao banco de dados, utilizando para isso métodos da biblioteca do Hibernate.

Com isso, podemos dizer que elas fazem parte do bloco de controle (*controller*) do sistema. Para cada classe do domínio foi criada uma classe para fazer o seu gerenciamento, contendo métodos estáticos públicos que permitem a integração com o banco de dados. Esses métodos oferecem todas as utilidades necessárias para que seja possível tratar os dados de forma simples e eficiente. As classes de gerenciamento são as seguintes:

- *OrcamentoManager.java*
- *FonteManager.java*
- *FinanciadorManager.java*
- *ContaManager.java*
- *EmpenhoManager.java*
- *LiberadoManager.java*
- *PagamentoManager.java*

Tendo todas essas classes criadas, a estrutra do projeto pode ser vista na figura a serguir (Fig 21).

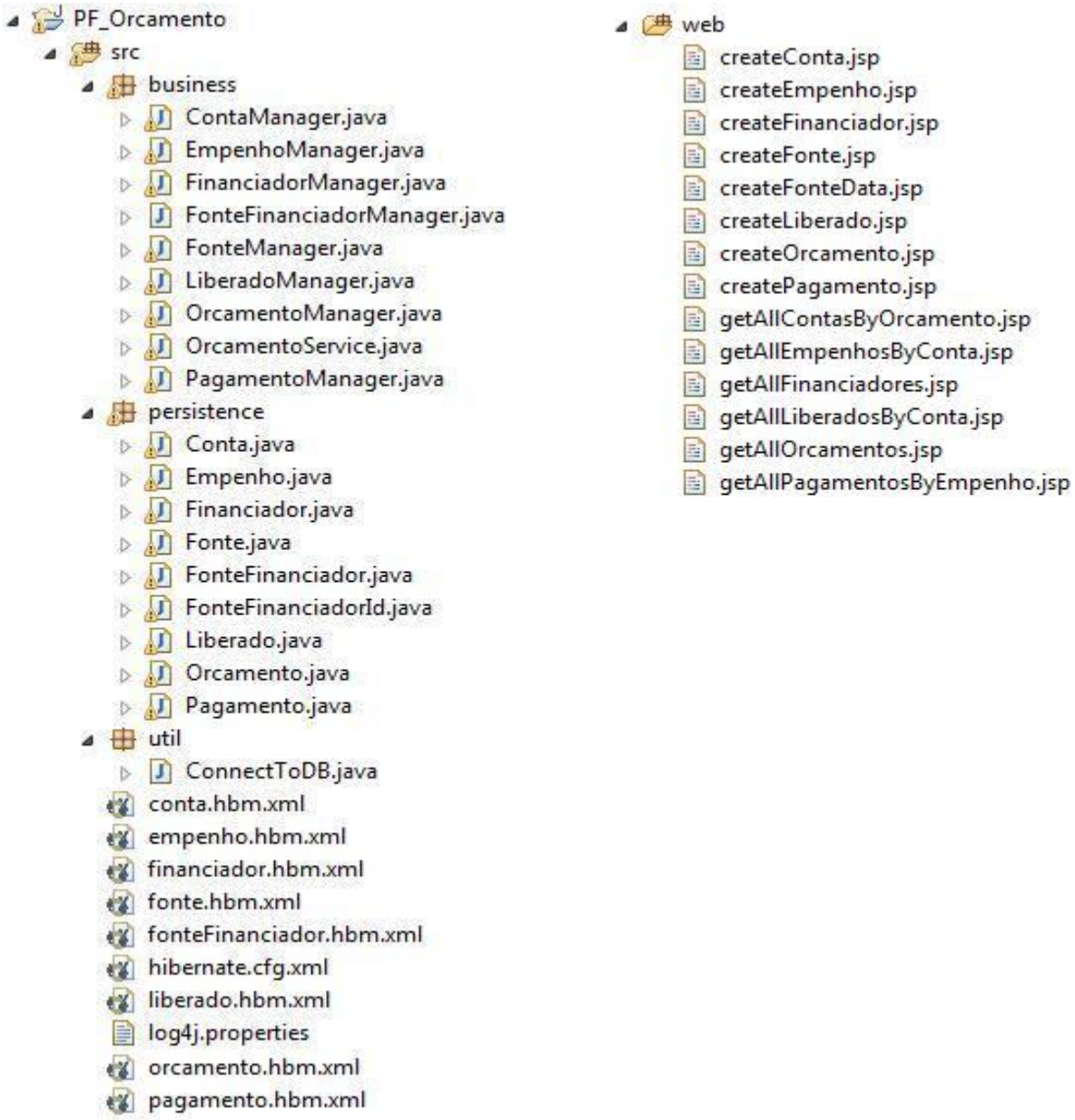


Figura 21 – Estrutura do Projeto

Durante o desenvolvimento de todos esses métodos eles foram testados individualmente. Isso foi feito para garantir que os módulos mais básicos do sistema estivessem funcionando perfeitamente quando métodos mais complexos precisassem utilizá-los e também por facilitar a

detecção de erros e, consequentemente, agilizar sua solução. Durante esses testes foram também testados os arquivos de mapeamento, uma vez que se houver erros neles o Hibernate não consegue realizar as operações de integração com o banco de dados. Esse período de testes serviu também para testar a classe de conexão ao banco criada anteriormente, que utiliza o driver de conexão do MySql para Java, o JDBC. Alguns métodos não puderam ser testados individualmente pois dependem de outros métodos mais simples, como o createEmpenho, por exemplo, que precisa chamar os métodos de atualizar conta. Esses métodos só foram testados quando os métodos que seriam por eles utilizados já haviam sido testados e estavam funcionando corretamente.

Após a implementação de todos as classes de negócio (business) e de testes extensivos para garantir que elas estavam funcionando conforme o esperado, passamos à etapa de testar as funcionalidades num sistema WEB.

Para isso precisamos criar um ambiente adequado para a aplicação. Isso foi feito criando um contexto para o projeto no Tomcat, o que é realizado gerando uma estrutura de pastas específica dentro do diretório “webapps” do Tomcat, como é mostrado na figura abaixo.

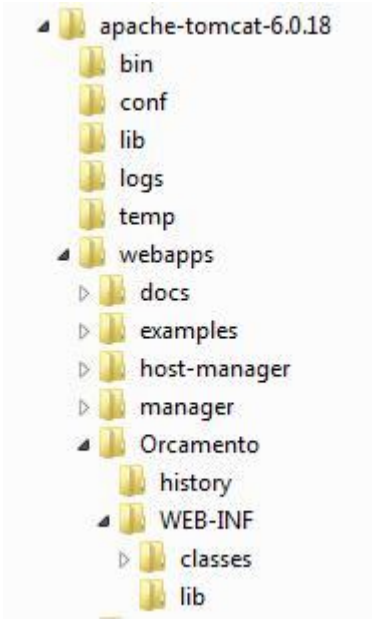


Figura 22 – Estrutura do Servidor WEB

Essa estrutura de pastas precisa ser criada para que o Tomcat seja capaz de executar a aplicação Web. Como pode ser observado na figura 22 foi criada uma pasta “Orcamento” dentro

de “webapps”. Essa pasta contém os arquivos que precisam ser visíveis para o navegador *web* do cliente, como JSPs, HTMLs, JavaScripts, imagens etc. No caso do Sistema de Controle Orçamentário os arquivos presentes nessa pasta são os HTMLs e SWFs (arquivos da interface Flex), AS (arquivo ActionScript responsável pela lógica presente na interface) e JSPs (são usados para chamar as classes de negócio passando os dados vindo da interface). Dentro da pasta “Orcamento” são criadas as pastas “classes” e “lib”. A pasta “lib” contém as bibliotecas que são usadas tanto pelas classes Java quanto pelos métodos implementados pela interface. Já a pasta “classes” contém os arquivos binários gerados na compilação do projeto Java e os arquivos de mapeamento do Hibernate (.hbm).

Tendo sido criado esse contexto para o projeto, implementamos um arquivo build.xml, utilizando a ferramenta Ant, que passou a ser responsável por fazer o build do projeto Java e mover os arquivos gerados para a pasta “classes”. Isso foi feito para facilitar o processo de se fazer build no projeto Java, uma vez que sem esse arquivo seria necessário mover “na mão” todos os arquivos binários gerados na compilação. No caso dos arquivos gerados na compilação da interface gráfica a solução foi mais simples. Como todos os arquivos devem ficar na pasta “Orcamento”, foi necessário apenas configurar no Flex Builder para que ele já criasse os arquivos binários nessa pasta.

Nessa etapa do desenvolvimento, como já tínhamos o contexto para rodar o projeto de forma Web, começamos a realizar novamente todos os testes das classes de negócio que já havíamos feito pelo Eclipse. A diferença dessa vez é que passamos a realizá-los de maneira *Web*.

Para que isso fosse possível, criamos uma interface gráfica simples para cada uma das classes de persistência. Essas interfaces tinham o objetivo de permitir que o métodos das classes de negócio fossem testados, logo elas continham campos para inserção dos dados necessários para que o teste de criação e atualização das classes de persitência fossem executados. Após a implementação dessas interfaces, foi necessário implementar algo que recebesse as requisições do navegador e as repassasse às classes de negócio. Para realizar essa integração utilizamos JSPs. Os JSPs implementados são bem simples, consistindo apenas de uma chamada para uma classe Java passando o método que deve ser executado e os parâmetros transmitidos pela interface. Os JSPs juntamente com essa classe (OrcamentoService) e com os métodos implementados em ActionScript, que serão detalhados mais à frente, também fazem parte do bloco de controle (*control*) do sistema, uma vez que eles são os reponsáveis por realizar a troca de dados entre a

interface (*view*) e a persistência (*model*). Para fazer essa transferência de dados da interface, passando pelo JSP e chegando às classes Java e também no sentido inverso, tivemos que serializar os dados. Para isso foi utilizada a tecnologia Json, que consiste na criação de Strings padronizadas, chamadas de "Json Strings". Essas *strings* podem ser transformadas em objetos Json e, utilizando métodos de alguma biblioteca, "corelib" no caso desse projeto, se torna bem fácil recuperar os dados transmitidos.

Foram efetuados então diversos testes simples e distintos por classe de persistência que permitiram a correção de erros e, quando finalizados, garantiram que todas as funcionalidades de integração "interface-java-banco de dados" estavam funcionando corretamente.

A partir desse ponto foi iniciada a etapa de implementação da interface gráfica, a qual também foi baseada nos casos de uso definidos durante a análise do projeto. Essa interface representa o bloco de visualização do sistema (*view*) e é responsável por exibir os dados para o usuário.

Inicialmente foi criada a interface de entrada do sistema, contendo um menu perincipal que permite ao usuário acessar as quatro principais funcionalidades do sistema: criar um novo orçamento, atualizar um orçamento já existente, visualizar um orçamento também já existente ou cadastrar um novo financiador. A partir desse ponto foram implementadas as interfaces necessárias para realizar cada caso de uso.

O primeiro caso de uso abordado foi o de criar um novo orçamento e, por isso, a primeira interface implementada foi a tela que permite ao usuário inserir os dados de identificação de um novo orçamento. Em seguida foram criadas as telas que dão ao usuário a possibilidade de criar a fonte financeira para o orçamento selecionando os financiadores já cadastrados no banco e também a tela que permite ao usuário cadastrar as contas que serão utilizadas pelo orçamento que está sendo criado.

Foram abordados depois os casos de uso de criação de empenhos, pagamentos e liberados, que, na interface, estão agrupados no módulo de atualização do orçamento. Na implementação dessa parte da interface foram reaproveitados os formulários que já haviam sido criados para os testes de integração da interface com o código Java e o banco de dados, o que agilizou bastante o processo.

Em seguida passamos à implementação da interface para permitir a utilização dos casos de uso de visualização de um orçamento e da impressão dos dados. Esses dois casos de uso

foram agrupados em apenas um módulo da interface pois estão muito relacionados. O usuário pode visualizar os dados na tela do navegador e, caso haja interesse, imprimi-los em papel através de uma opção presente na interface.

Tendo então implementado todos os módulos da interface que permitiriam a realização de todos os casos de uso, o passo seguinte consistiu em testar o sistema como um todo e analizar os resultados.

5- Resultados

Os resultados que serão demonstrados a seguir foram obtidos através de testes do sistema, uma vez que, infelizmente, ele ainda não foi implantado em um ambiente de produção. Ainda assim, são resultados válidos para que uma análise do sistema possa ser feita.

Para a realização dos testes finais do sistema foi utilizado um ambiente de teste similar ao que foi utilizado para o desenvolvimento, que já foi detalhado acima neste documento. Para a criação desse ambiente de testes foram seguidos os passos que estão descritos no manual do usuário (Apêndice 1) e tudo funcionou conforme previsto.

Os testes finais tiveram como objetivo simular a utilização completa do sistema, passando por todos os casos de uso e fazendo uso, consequentimente, de todas as telas da interface gráfica e os métodos Java. Esses testes não foram realizados pelos usuários finais do sistema, o que não permite uma análise definitiva sobre alguns aspectos do sistema, como amigabilidade da interface, por exemplo. Eles foram, entretanto, muito satisfatórios no que diz respeito aos aspectos técnicos do sistema, uma vez que todas operações foram executadas com sucesso e as funcionalidades implementadas funcionaram como deveriam.

Para analisar como um usuário final utilizaria o sistema foi feito também um teste simples com um usuário, que realizou algumas atividades no sistema, como criar um orçamento novo, atualizar alguns dados e visualizar os mesmos. Esse usuário conseguiu efetuar todas essas ações de modo intuitivo e sem grandes dificuldades, o que garantiu que a interface do sistema facilita a interação do usuário com o mesmo.

6- Conclusão

A realização deste projeto foi de grande valia para a consolidação de diversos temas estudados na faculdade, como, por exemplo, engenharia de *software*, banco de dados e desenvolvimento de algoritmos em linguagens de programação. Além disso ele possibilitou o aprendizado de diversas ferramentas utilizadas em desenvolvimento de projetos que são muito populares e de grande utilização no mercado, como Eclipse, Ant e Hibernate.

Tendo isso em mente podemos dizer que o desenvolvimento do projeto contribuiu de maneira significativa para agregar conhecimento, podendo vir a ser de grande ajuda na vida profissional.

No que se refere ao sistema em si, podemos dizer que ele obteve resultados positivos, uma vez que atendeu aos pré-requisitos e funcionou da maneira desejada. Concluímos também que os métodos utilizados no desenvolvimento do mesmo funcionaram da maneira esperada, auxiliando bastante na implementação. Isso pode ser dito tanto para o método de implementação escolhido, que foi partir do desenvolvimento dos métodos mais simples até os mais complexos, quanto para a metodologia de testes, que seguiu o mesmo princípio e facilitou a detecção de erros e *bugs*, agilizando assim as suas soluções.

Bibliografia

http://livedocs.adobe.com/flex/3/ acessado em agosto/setembro de 2008

http://java.sun.com/javaee/5/docs/api/ acessado em agosto/setembro de 2008

http://www.exampledepot.com/ acessado em agosto/setembro de 2008

http://ant.apache.org/ acessado em agosto/setembro de 2008

http://tomcat.apache.org/ acessado em agosto/setembro de 2008

www.mikechambers.com/blog/ acessado em agosto/setembro de 2008

http://learn.adobe.com/wiki/display/Flex/Flex+and+Java acessado em agosto/setembro de 2008

http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html acessado em agosto/setembro de 2008

http://facestutorials.icefaces.org/tutorial/hibernate-tutorial.html acessado em agosto/setembro de 2008

http://www.json.org/ acessado em agosto/setembro de 2008

PRESSMAN, Roger S. - Software Engineering: A Practitioner's Approach, McGraw-Hill Science/Engineering/Math; 6 edition (2004)

BAUER, Christian & King, Gavin – Hibernate in Action, Manning Publications, 2004

Apêndice 1 – Manual do Usuário

1.1 - Introdução

Esse manual tem como objetivo principal servir de referência para que o usuário final do Sistema de Controle Orçamentário seja capaz de realizar todas as atividades disponibilizadas pelo sistema. Ele está dividido tendo como base os casos de uso e as telas da interface gráfica do sistema. Essa divisão visa facilitar a consulta por parte do usuário, permitindo que ele encontre facilmente a atividade sobre a qual deseja se informar.

1.2 - Instalação

Por ser um sistema *web*, só é necessária a instalação do mesmo no servidor. No entanto, antes da instalação do sistema de controle orçamentário é necessário verificar se o servidor possui os aplicativos necessários para o funcionamento correto do sistema. Esses aplicativos são os seguintes:

- Apache Tomcat versão 6 (***tomcat.apache**.org*)
- MySql Server 5.0 (www.**mysql**.com)

Com esses aplicativos instalados, podemos continuar com o processo de instalação do sistema de controle orçamentário. Para isso basta copiar o arquivo *orçamento.war* para o diretório "webapps" do Tomcat. O próximo passo é iniciar o servidor e ele irá realizar o *deploy* do sistema.

Quando o servidor já estiver *online*, devemos ir à pasta "webapps/Orcamento/WEB-INF/classes" e abrir o arquivo *hibernate.cfg.xml*. Nesse arquivo deverão ser alteradas as linhas que fazem referência ao banco de dados que será utilizado e que seguem abaixo.

```
<property name="connection.url">    endereço do banco </property>
<property name="connection.driver_class"> jdbcDriver  </property>
<property name="connection.username">     login </property>
<property name="connection.password">     senha </property>
<property name="dialect">     dialeto do sql (referente ao banco) </property>
```

O próximo passo é criar as tabelas no banco de dados e, para isso, basta executar o código em SQL que vem no arquivo “criarBanco”, o qual é empacotado junto do arquivo *.war*.

Após todos esses passos o sistema estará preparado para entrar em funcionamento.

1.3 - Funcionalidades

2.1) *Acessar o sistema*

Para acessar o sistema o usuário deve utilizar um navegador de internet e inserir o endereço do sistema (URL). Isso fará o navegador acessar a página principal do sistema, onde o usuário visualizará o menu principal.

2.2) *Menu principal*

O menu principal consiste de três botões (Fig 23) que levam o usuário às telas de criação de um novo orçamento, atualização de um orçamento já existente ou visualização de um orçamento também já existente.

Figura 23 – Menu Principal

2.3) *Criar Orçamento*

Para criar um novo orçamento o usuário deve selecionar a opção “Criar Orçamento” no menu principal. Isso o levará à tela de criação de orçamento, onde ele poderá preencher os campos necessários para a criação de um novo orçamento (Fig 24). Quando ele concluir essa etapa uma nova tela será exibida pedindo que o usuário crie a fonte relacionada ao novo orçamento. Ele deve preencher os campos necessários, incluir os financiadores e concluir a etapa. Tendo feito isso uma nova tela será exibida para que ele possa criar as contas que estarão presentes no novo orçamento. O procedimento de preencher os campos é repetido e, ao concluir a etapa o orçamento é criado no sistema.

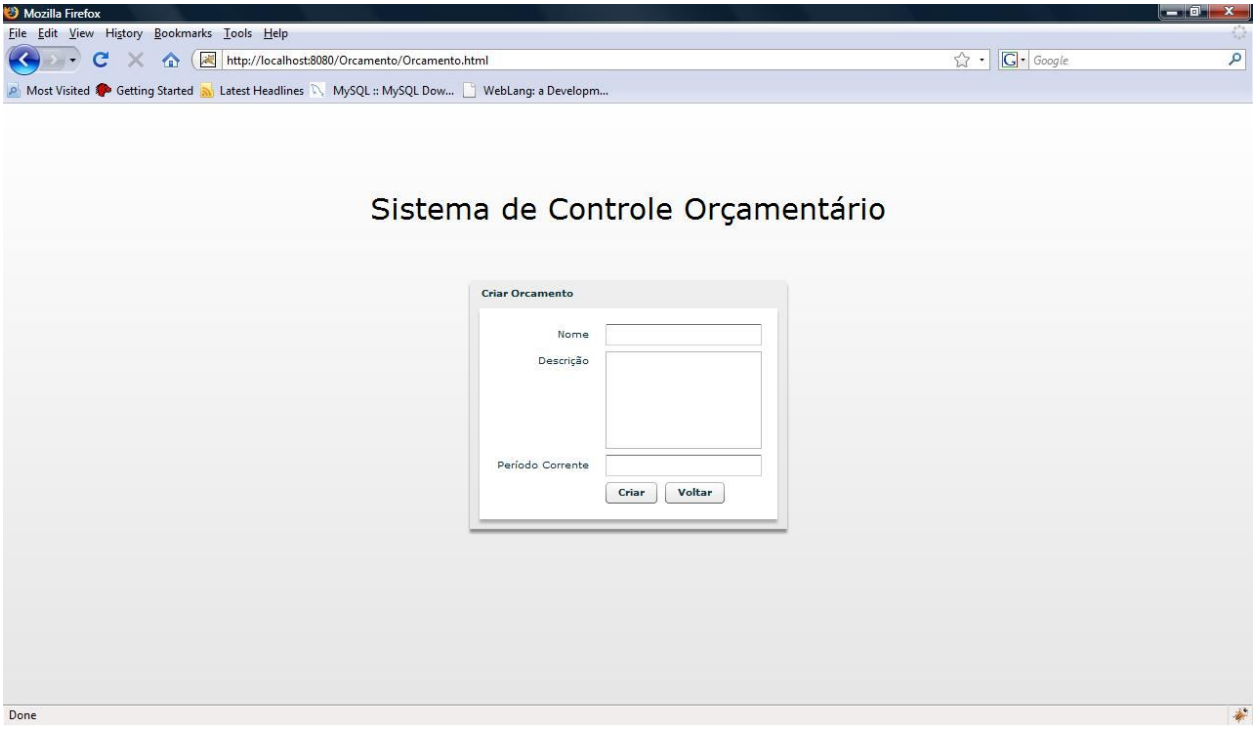


Figura 24 – Menu de criação de orçamento

2.4) *Atualizar orçamento*

Para criar um empenho o usuário deve selecionar a opção “Atualizar Orçamento” no menu principal. Isso o levará a tela de atualização de orçamentos já existentes no sistema. O usuário deve então selecionar qual orçamento ele deseja atualizar. Os dados do orçamento selecionado são exibidos na parte central da tela para que seja possível a identificação do mesmo.

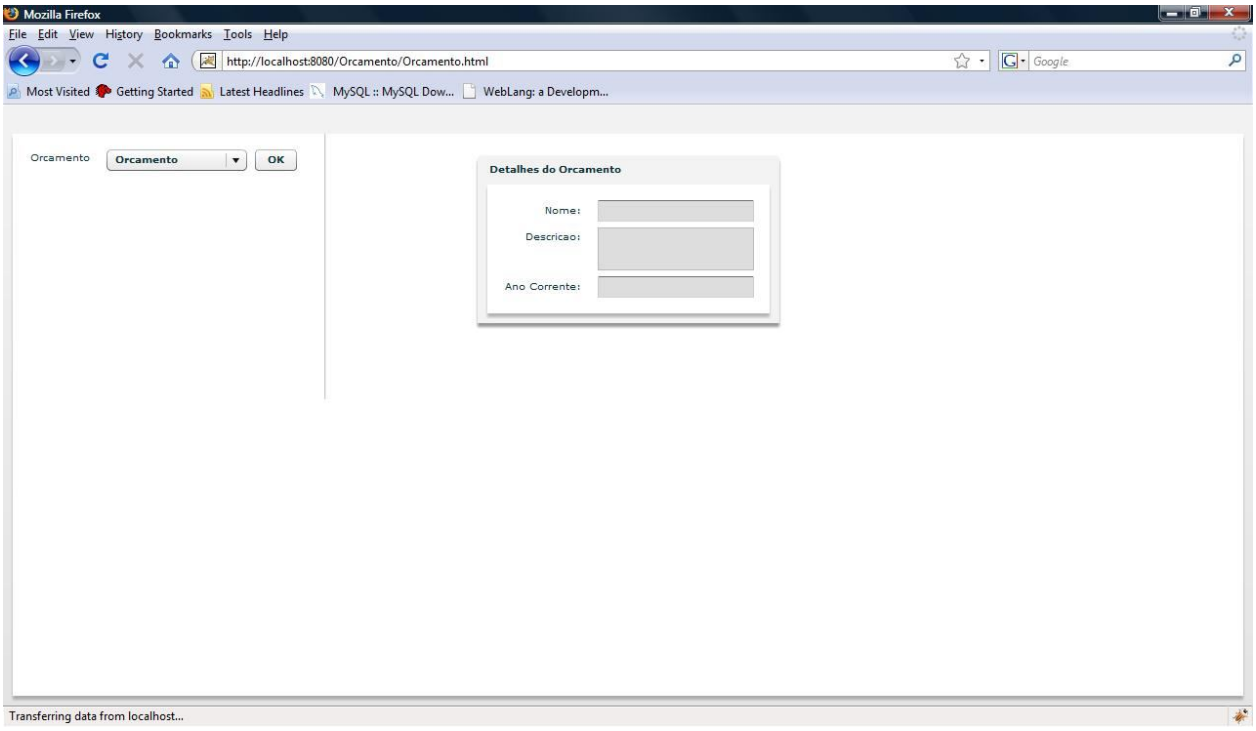


Figura 25 – Atualização de Orçamento

O usuário deve então se certificar que o orçamento selecionado é o correto e então confirmar clicando no botão “Ok”. Feito isso um novo seletor aparecerá, permitindo que o usuário escolha qual a conta do orçamento já selecionado ele deseja atualizar. Da mesma forma que na etapa anterior, os detalhes da conta são exibidos no centro da tela para permitir a identificação da mesma.

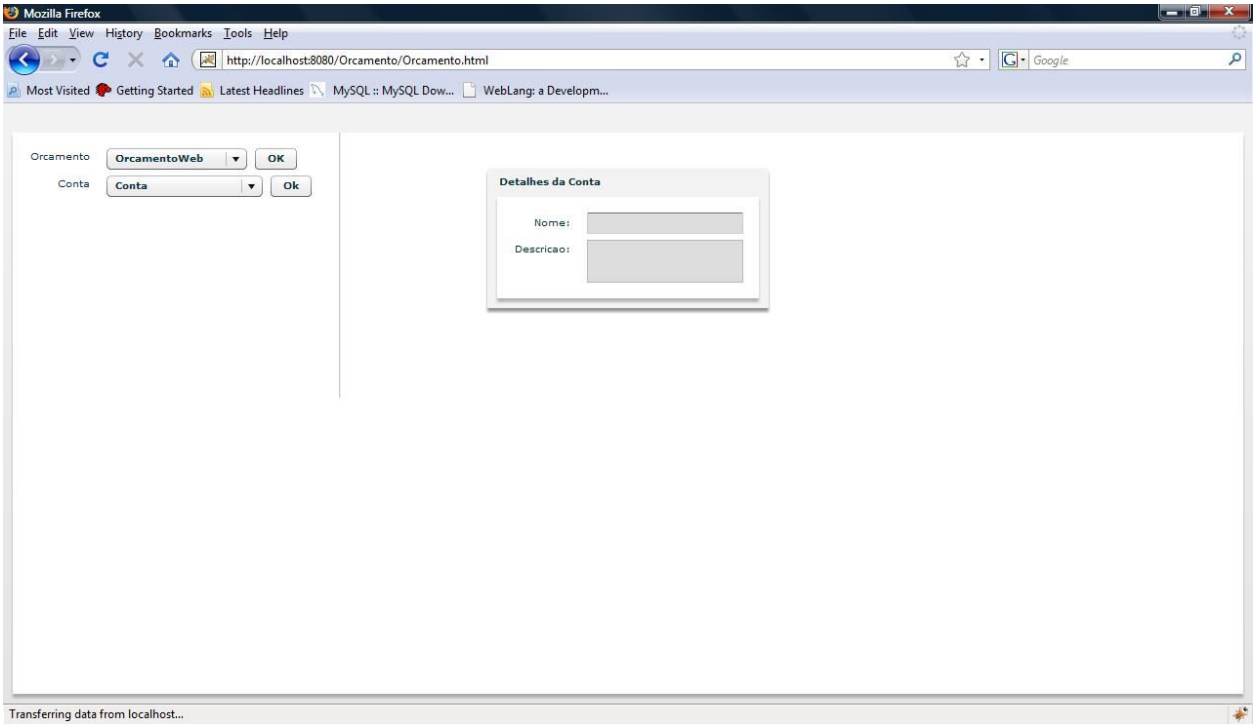


Figura 26 – Seleção de Orçamento (Atualização de Orçamento)

Da mesma forma como fez na etapa anterior o usuário deve se certificar que a conta selecionada é a que ele deseja atualizar e confirmar isso clicando no botão “Ok”. Tendo feito isso três opções aparecerão na tela, mostrando as atualizações que podem ser feitas na conta: *Criar Empenho, Criar Liberado e Criar Pagamento.*

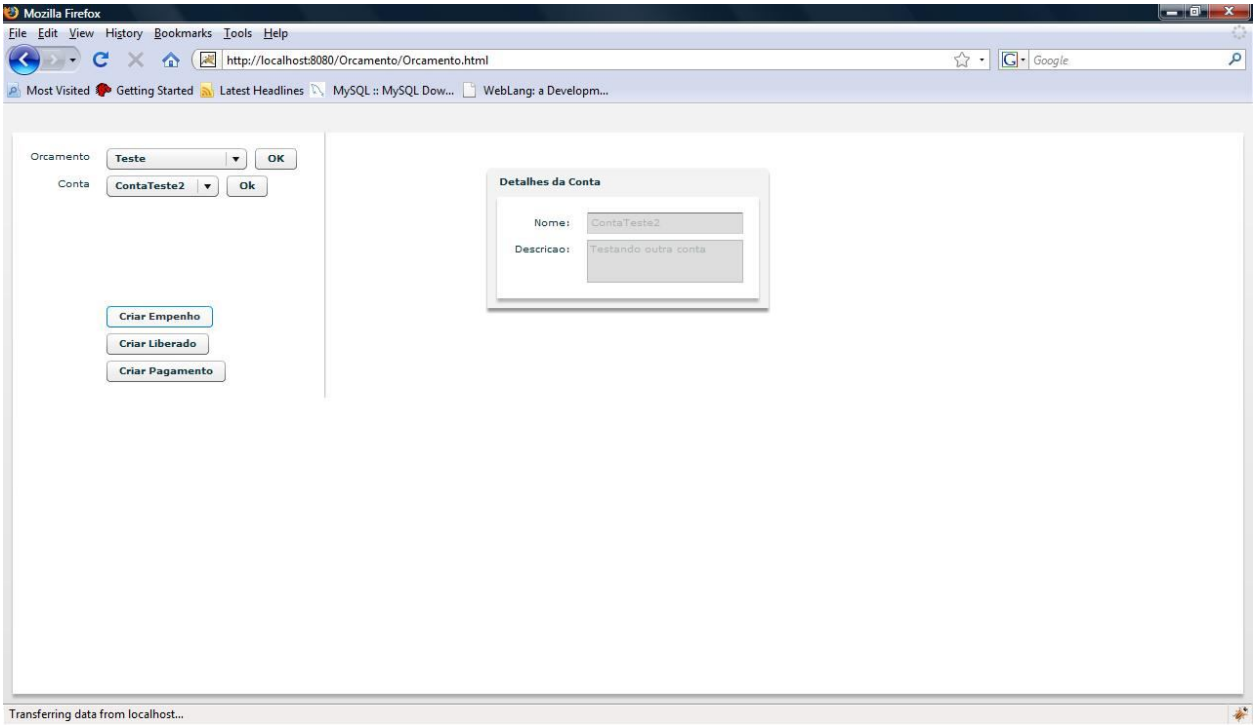


Figura 27 – Seleção de Conta (Atualização de Orçamento)

2.4.1) *Criar Empenho*

Se quiser criar um empenho o usuário deve selecionar a opção “Criar Empenho” clicando no botão com mesmo nome. Ao fazer isso será exibido um formulário no centro da tela onde o usuário pode preencher os campos com os dados do empenho a ser criado. Ele deve então confirmar a operação clicando no botão “Criar”. Com isso o novo empenho será criado e a conta à que ele se refere será atualizada automaticamente.

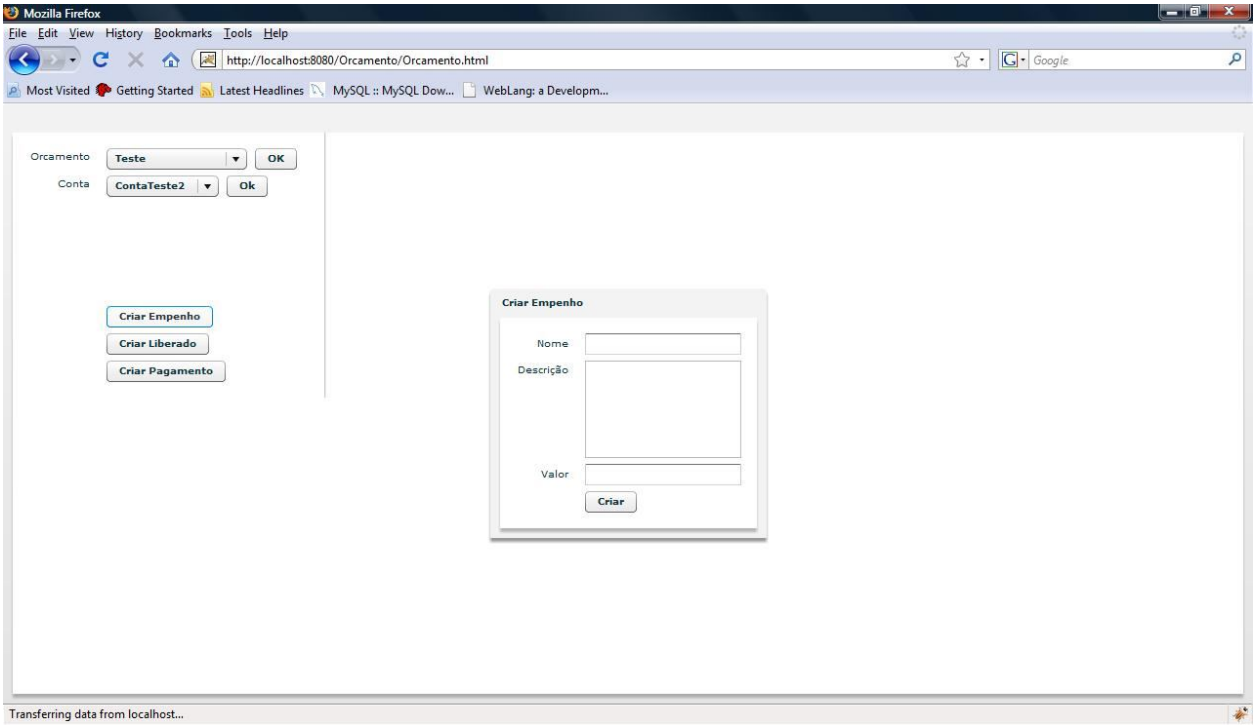


Figura 28 – Criação de Empenho (Atualização de Orçamento)

2.4.2) *Criar Liberado*

Se quiser criar um liberado o usuário deve selecionar a opção “Criar Liberado” clicando no botão com mesmo nome. Ao fazer isso será exibido um formulário no centro da tela onde o usuário pode preencher os campos com os dados do liberado a ser criado. Ele deve então confirmar a operação clicando no botão “Criar”. Com isso o novo liberado será criado e a conta à que ele se refere será atualizada automaticamente.

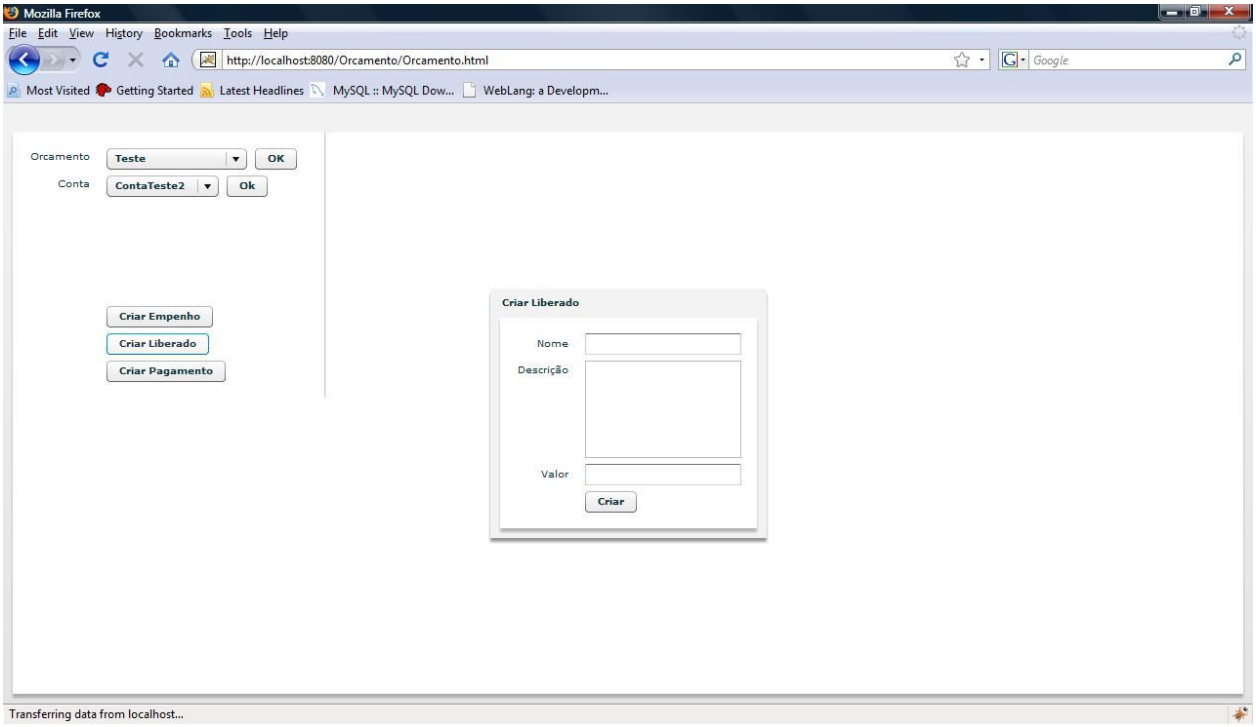


Figura 29 – Criação de Liberado (Atualização de Orçamento)

2.4.1) *Criar Pagamento*

Se quiser criar um pagamento o usuário deve selecionar a opção “Criar Pagamento” clicando no botão com mesmo nome. Ao fazer isso será exibido um formulário no centro da tela onde o usuário pode selecionar o empenho ao qual o pagamento se refere e preencher os campos com os dados do pagamento a ser criado. Ele deve então confirmar a operação clicando no botão “Criar”. Com isso o novo pagamento será criado e a conta e o empenho aos quais ele se refere serão atualizados automaticamente.

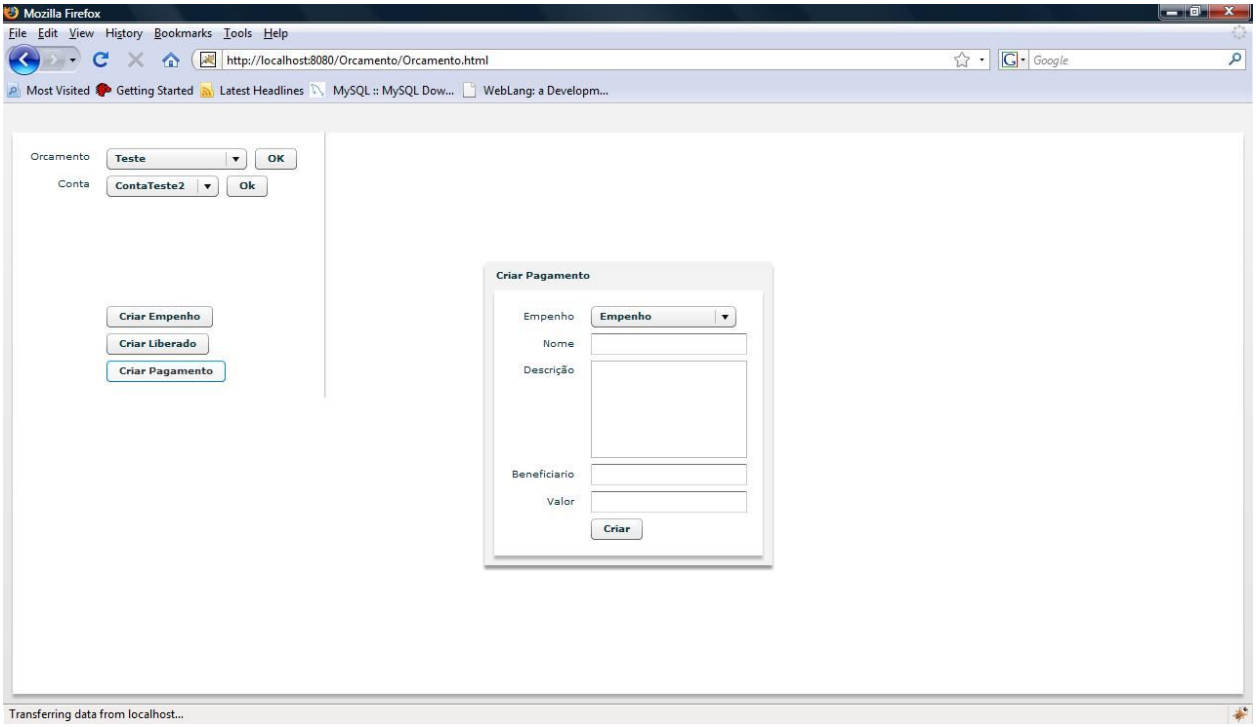


Figura 30 – Criação de Pagamento (Atualização de Orçamento)

2.5) *Visualizar orçamento*

Para visualizar um orçamento já existente o usuário deve selecionar a opção “Visualizar Orçamento” no menu principal. Isso o levará à tela de visualização de orçamento, onde ele deverá selecionar o orçamento que deseja visualizar.

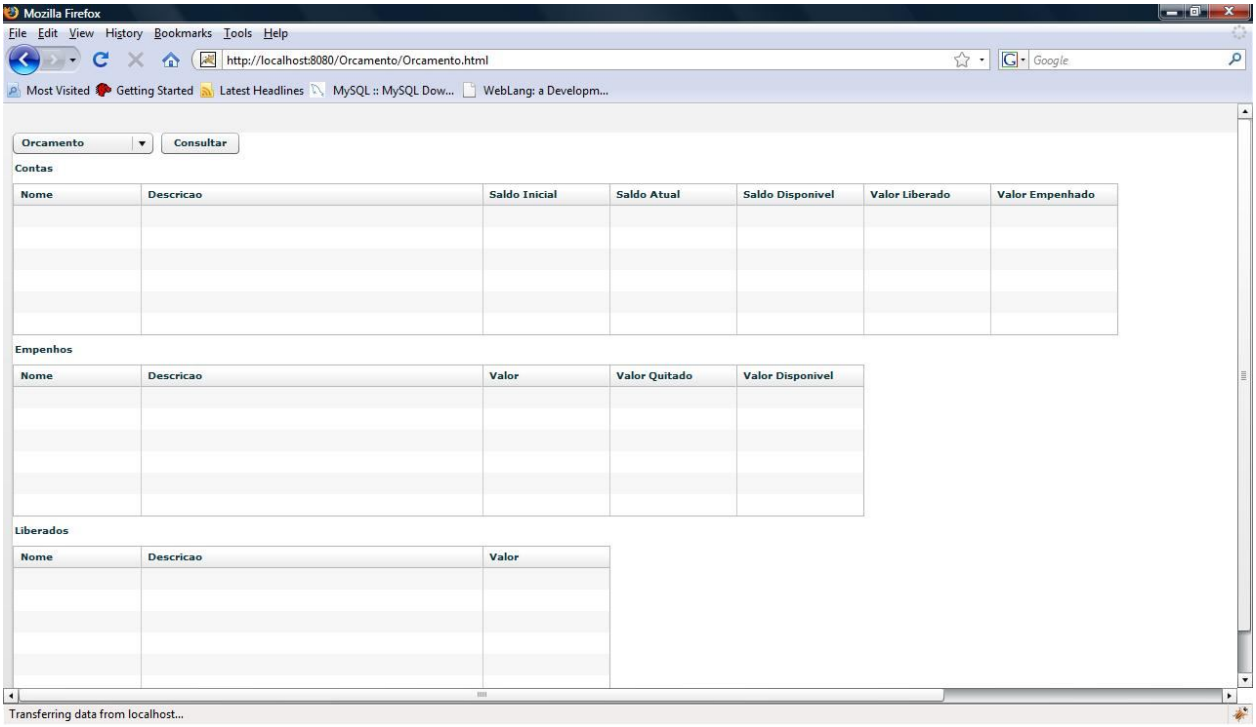


Figura 31 – Visualização de Orçamento

Ao fazer isso e confirmar a escolha do orçamento clicando no botão “Consultar”, os dados das contas presentes no respectivo orçamento serão mostrados na tabela referente às contas (primeira tabela).

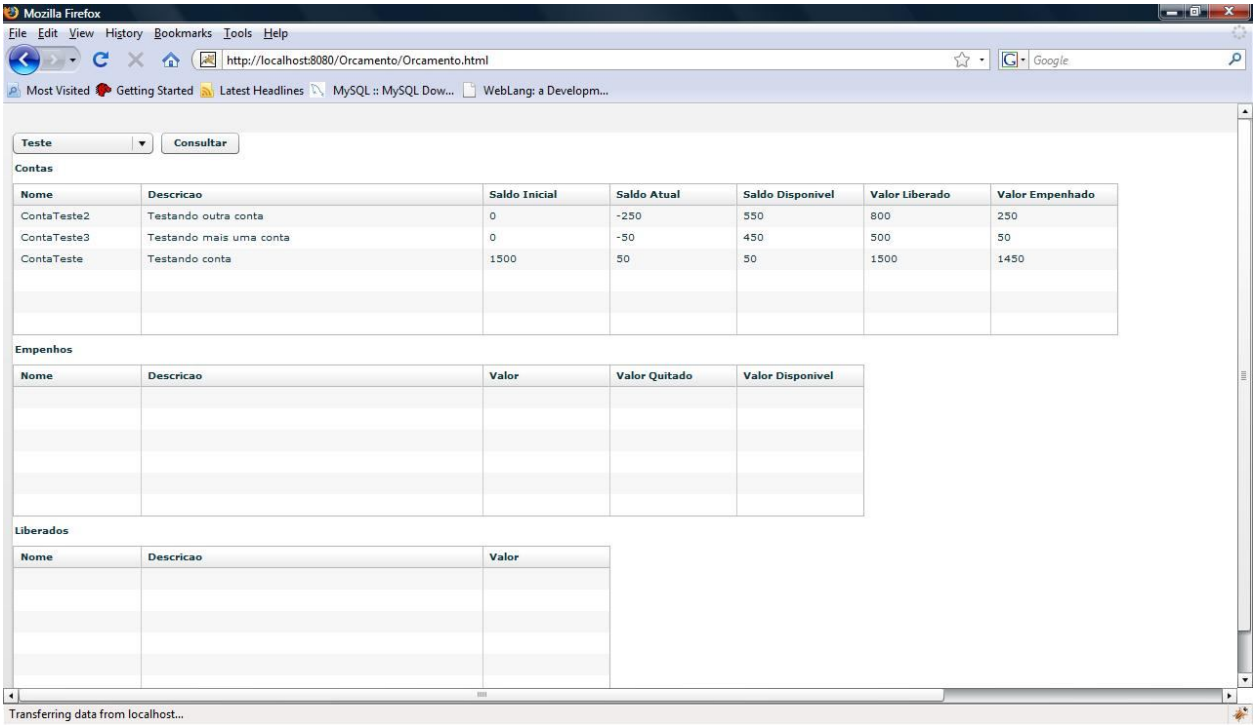


Figura 32 – Visualização de Orçamento

Caso o usuário deseje dados mais detalhados sobre os empenhos e os liberados de uma determinada conta ele deve clicar sobre uma das contas presentes na tabela. Isso fará o sistema carregar os dados dos empenhos e liberados da conta selecionada nas tabelas correspondentes.

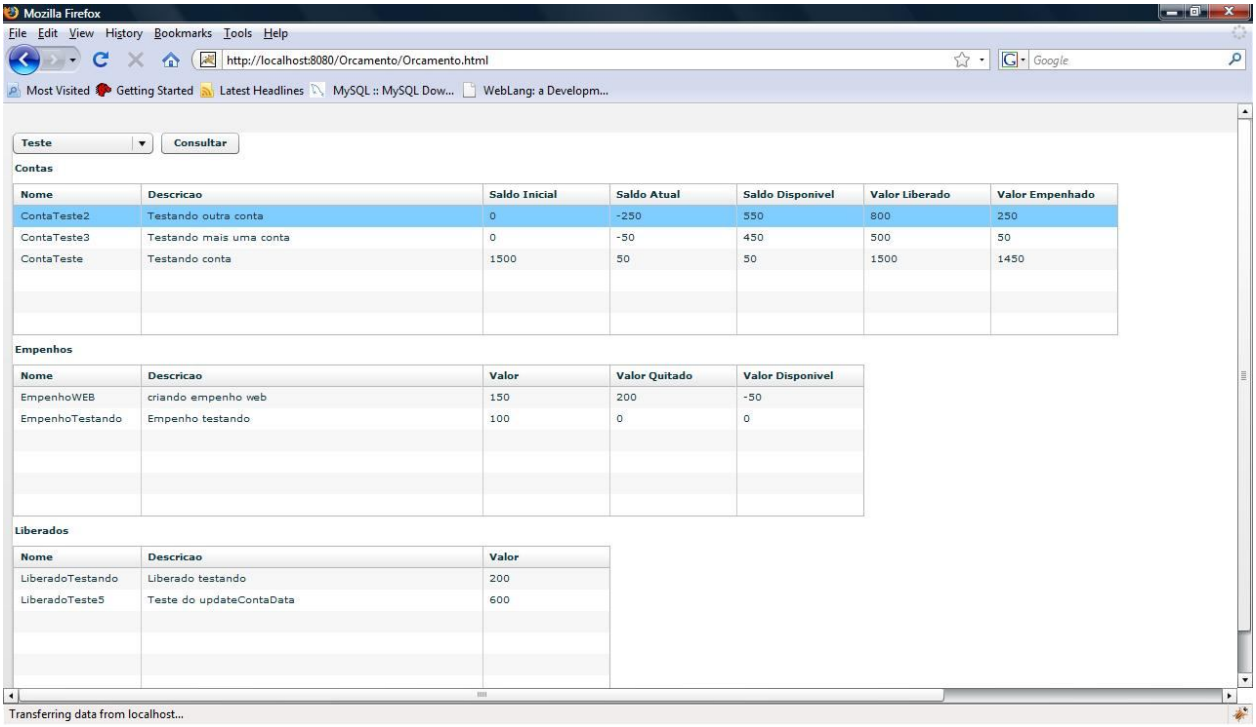


Figura 33 – Visualização de Orçamento

Ainda é possível visualizar os dados mais detalhados dos pagamentos de um determinado empenho, bastando para isso que o usuário clique em um dos empenhos exibidos na respectiva tabela, o que fará com que uma nova tabela seja exibida mostrando os dados de todos os pagamentos realizados no empenho selecionado.

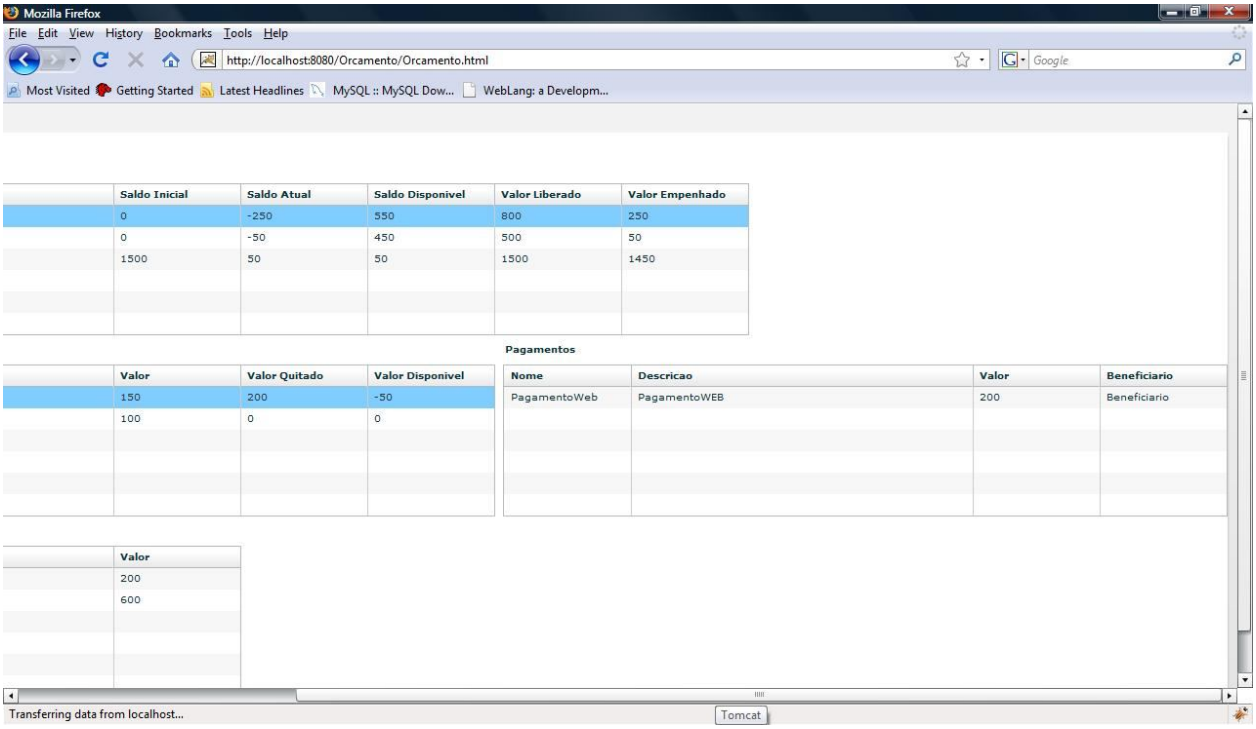


Figura 34 – Visualização de Orçamento

2.6) *Imprimir dados*

Para imprimir dados de um orçamento o usuário deve estar na tela de visualização do orçamento. Um botão localizado na parte inferior da tela (“Imprimir”) permite que o usuário imprima os dados que estão presentes na tela

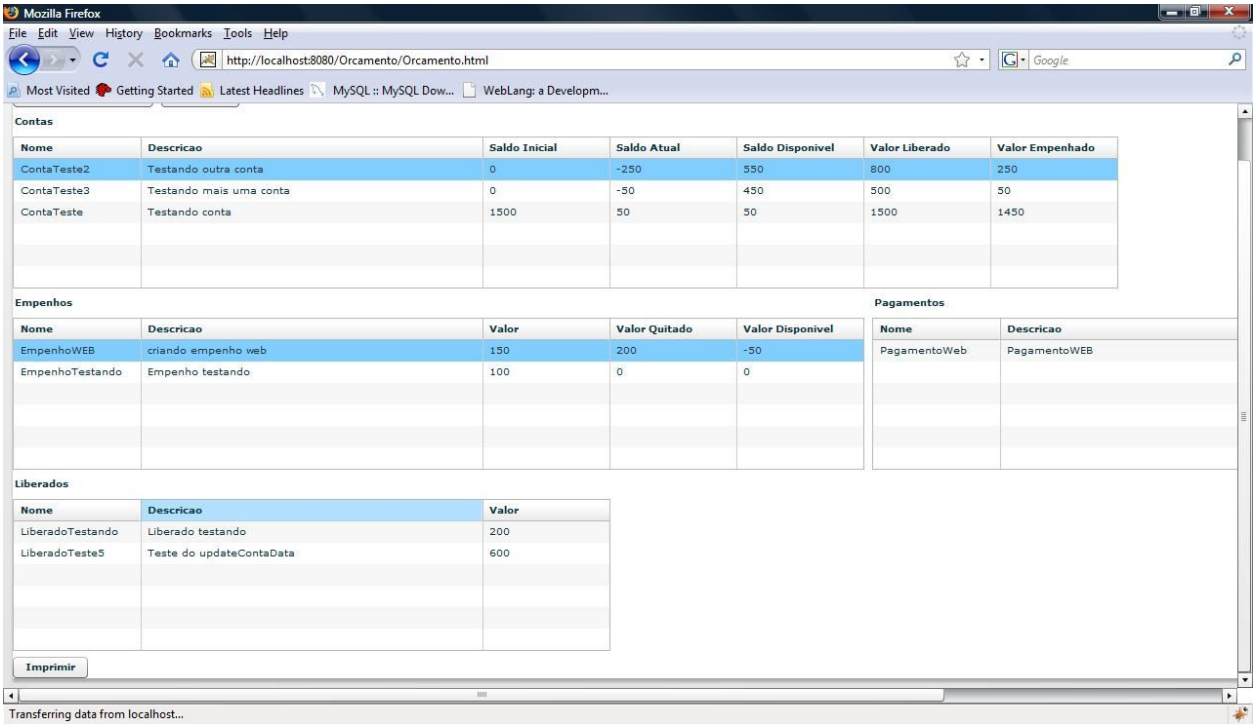


Figura 35 – Visualização de Orçamento