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

ANDERSON VALTER GARCIA

Sistema de Gestão de Cooperativas de Reciclagem de Resíduos Sólidos

Prof. Me. André Fernando Rollwagen

PASSO FUNDO
2016

ANDERSON VALTER GARCIA

Sistema de Gestão de Cooperativas de Reciclagem de Resíduos Sólidos

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

Orientador: André Fernando Rollwagen

PASSO FUNDO

2016

ANDERSON VALTER GARCIA

SISTEMA DE GESTÃO DE COOPERATIVAS DE RECICLAGEM DE RESÍDUOS SÓLIDOS

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

Banca Examinadora:

_____

Prof. Me. André Fernando Rollwagen Orientador

_____

Prof. Me. Carmen Vera Scorsatto Convidada

_____

Prof. Me. Ricardo Vanni Dallasen Convidado

_____

Prof. Me. Adilso Nunes de Souza Coordenador

PASSO FUNDO

2016

RESUMO

A falta de sistemas que auxiliam na gestão de cooperativas de resíduos sólidos em Passo Fundo traz dificuldades aos gestores devido à falta de dados concisos para auxiliar nas tomadas de decisões em sua gestão. O sistema proposto tem por objetivo proporcionar uma ferramenta para auxiliar estes gestores em suas atividades nas cooperativas. Para o desenvolvimento deste sistema foram utilizadas várias tecnologias, entre elas pode-se destacar o PHP onde foi desenvolvida a lógica do sistema sobre a arquitetura MVC, o API *SBADMIN 2* que é o responsável pelo layout do sistema e o *plug in datatables*, encarregado pela comunicação entre cliente e servidor. Após o desenvolvimento do sistema o mesmo foi implantado, testado, utilizado e aprovado pelos gestores das cooperativas de Passo Fundo. Por fim o resultado desse trabalho foi um sistema WEB dinâmico e de fácil manutenibilidade devido a sua arquitetura lógica e a reutilização de código trazendo a oportunidade de expansão de funcionalidades ou facilidade de alteração.

Palavras-chave: Cooperativas, Sistema, PHP, MVC, WEB, *Datatables*, Manutenibilidade.

ABSTRACT

The lack of systems that assist in the management of solid waste cooperatives in Passo Fundo presents difficulties for managers due to the lack of concise data to aid decision making in their management. The proposed system aims to provide a tool to assist these managers in their activities in the cooperatives. For the development of this system, several technologies have been used, among them PHP can be highlighted where the system logic was developed on the MVC architecture, the SBADMIN 2 API that is responsible for the system layout and the plug in datatables, Communication between client and server. After the development of the system, it was implemented, tested, used and approved by the cooperative managers of Passo Fundo. Finally, the result of this work was a dynamic and easy-to-maintain WEB system due to its logical architecture and the re-use of code, bringing the opportunity to expand functionality or ease of change.

Keywords: Cooperative, System, PHP, MVC, WEB, datatables, Maintainability

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ANEXOA

LISTA DE ABREVIAÇÕES E DE SIGLAS

ABNT – Associação Brasileira de Normas Técnicas

ACID - Atomicity, Consistency, Isolation, Durability

AJAX - Asynchronous JavaScript e XML

CSS - Cascading Style Sheets

DOM - Document Object Model

HTML – HyperText Markup Language

IDE - Integrated Development Environment

IFSUL – Instituto Federal Sul-rio-grandense

IP – Internet Protocol

MVC - Model-view-controller

PDO – Preprocessor Data Objects

PDF - Portable Document Format

PHP – Hypertext Preprocessor

RS – Resíduos Sólidos

SGBD – Sistema de Gerenciamento de Banco de Dados

SQL – Structured Query Language

TCC – Trabalho de Conclusão de Curso

UML – Unified Modeling Language

W3C – World Wide Web Consortium

XML - eXtensible Markup Language

SUMÁRIO

1 **INTRODUÇÃO**

A dificuldade de se obter informações com qualidade e agilidade sem um sistema de informação pode ser percebida, por exemplo, quando a informação se limita a fonte de dados obtidos em relatórios manuais mal estruturados (PINHO, 2011). Portanto, tomadas de decisões precipitadas e equivocadas pela parte dos gestores, devido à análise de dados e informações de fontes ou relatórios duvidosos, incompletos ou até mesmo incorretos.

Um sistema de informação é capaz de armazenar um conjunto de elementos ou componentes inter-relacionados como coleta, manipulação e armazenamento de dados e informações para alcançar um objetivo (STAIR & REYNOLDS ,2011). Analisando esse conceito, é possível concluir que um sistema de gestão de resíduos é importante no auxílio dos gestores na busca de dados ou na tomada de decisões, trazendo vantagens, como diminuição de custos administrativos e acesso seguro à informação.

Os sistemas informatizados apontam, ainda, outros benefícios. Algumas dessas vantagens são mencionadas por Oliveira (2002) como: aprimoramento no acesso a informações; emissão de relatórios com maior precisão e menor espaço de tempo; tomada de decisões mais efetivas com acesso à informação de forma mais ágil e precisa; maior interação entre tomadores de decisão; melhora na atividade setorial; mais agilidade para enfrentar acontecimentos imprevistos; evolução e melhorias no serviço prestado, redução de custos operacionais; evolução motivacional dos envolvidos no processo administrativo; melhorias na organização e fluxo de informações e melhor controle organizacional para os gestores do sistema.

De acordo com Rollwagen (2013), foi constatado, através de contados com secretarias e envolvidos na gestão de RS de Passo Fundo, que não há um sistema de informação de apoio a gestão de cooperativas de reciclagem de RS. Tais informações são mantidas em relatórios e planilhas eletrônicas. Com base nas vantagens citadas sobre a utilização de um sistema de informação, justifica-se o desenvolvimento de um sistema de informação para gestão de cooperativas de recicladores de RS.

1.1 **Tema**

Desenvolvimento de um sistema para gerenciamento de cooperativas de reciclagem de resíduos sólidos.

1.2 **Problema**

Devido ao aumento populacional e ao consumo de produtos industrializados por parte dessa população, a quantidade de resíduos sólidos (RS) resultantes desses produtos cresceu consideravelmente nas últimas décadas, o que ocasionou muitos problemas sociais, econômicos e ambientais. Segundo Pinho (2011), a gestão desses RS envolve processos complexos, devido ao grande número de variáveis, e a qualidade de vida é influenciada pela forma na qual esses resíduos são gerenciados.

Segundo Engholm (2012), a não utilização de um software é bastante desvantajosa, já que pode acarretar alguns problemas, como a soma incorreta de custos, erros operacionais e perca de produtividade na empresa, além de "inconsistência de dados" (OLIVEIRA, 2002), o que pode resultar em graves enganos nas tomadas de decisões por parte dos gestores.

Entre os problemas contidos em uma cooperativa do tipo que não utiliza qualquer software de apoio para o gerenciamento de resíduos, segundo Oliveira (2002), pode-se destacar: a falta de consistência de dados, que não podem ser arquivados, por exemplo, com maior precisão; a não emissão de relatórios confiáveis para o auxílio nas tomadas de decisões por parte dos gestores; a dificuldade de acesso à informações que, geralmente, se encontram engavetadas ou até mesmo perdidas, desorganizadas e a indisponibilidade de um banco de dados que mantenha os dados recolhidos.

Com a inexistência de um sistema computadorizado, ou com a falta de softwares que satisfaçam as necessidades da gestão administrativa dessas cooperativas, o gerenciamento nesses espaços acontece mediante ações desarticuladas e imprecisas.

Assim, com todos esses problemas já relatados, observa-se a necessidade do desenvolvimento de um sistema de informação que auxilie e facilite o trabalho de gestão de RS nas cooperativas de recicladores.

1.3 Objetivos

1.3.1 Objetivo geral

Desenvolver um sistema de gerenciamento de acordo com as necessidades das cooperativas de reciclagem de resíduos sólidos de Passo Fundo.

1.3.2 Objetivos específicos

- Levantar os requisitos necessários para o desenvolvimento do sistema proposto;
- Analisar e desenvolver o sistema;
- Realizar os testes necessários para garantir a qualidade do software;
- Implantar o sistema em uma cooperativa de RS de Passo Fundo.
- Relatar o *feedback* dos gestores da cooperativa após a implantação e utilização do sistema.

2 REFERENCIAL TEÓRICO

Este capítulo faz um referencial teórico relacionado aos conceitos do que são RS e suas classificações, buscando esclarecer quais materiais serão gerenciados através do sistema proposto, bem como definições das tecnologias que serão utilizadas em seu desenvolvimento. Essas informações visam trazer melhor entendimento dos aspectos que devem ser priorizados no projeto.

2.1 Resíduos Sólidos: Definição e Classificação

O aumento da geração de resíduos sólidos está diretamente ligado ao aumento populacional, aos hábitos e costumes da população, à melhoria na renda familiar e ao consumo cada vez maior de produtos industrializados. A geração de RS, segundo Philippi JR (2005 apud TAGUCHI, 2010), é uma característica essencial ao cotidiano da sociedade.

Segundo a ABNT (Associação Brasileira de Normas Técnicas) os resíduos sólidos são definidos como:

> [...] Aqueles no estado sólido e semi-sólido, que resultam de atividades de origem industrial, doméstica, hospitalar, comercial, agrícola, de serviços e de varrição. Ficam incluídos nesta definição os lodos provenientes de sistemas de tratamento de água, aqueles gerados em equipamentos e instalações de controle de poluição, bem como determinados líquidos cujas particularidades tornem inviável o seu lançamento na rede pública de esgotos ou corpos de água, ou exijam para isso soluções técnica e economicamente inviáveis em face à melhor tecnologia disponível (2004a, p. 1).

As principais formas de classificação dos RS estão relacionadas à sua natureza de origem e pelo seu grau de periculosidade ao meio ambiente e à saúde pública. Taguchi (2010) destaca a importância de classificar esses resíduos para que a gestão pública se adapte ao procedimento de gerenciamento de acordo com as suas necessidades.

Na tabela 1 é apresentada a classificação de resíduos sólidos quanto ao risco de contaminação, segundo a ABNT (2004a).

Tabela 1: Classificação de resíduos sólidos quanto aos riscos de contaminação

CLASSIFICAÇÃO	PROPRIEDADES	EXEMPLOS
Classe I – Resíduos Perigosos	Inflamáveis, corrosividade, reatividade, patogenicidade e	Lixo Hospitalar, produtos químicos industriais, pilha,

	tóxicos.	bateria e pesticidas.
Classe II – Não-Inertes	Combustíveis, biodegradabilidade ou solubilidade em água.	Resto de alimentos, papel, palha de aço, agulhas, latas e fiação elétrica.
Classe III – Inertes	Não se degradam ou se degradam lentamente, muitos destes sendo recicláveis.	Tijolo, plástico, borracha, entulhos de demolição, pedras e areais retirados de escavações.

Fonte: Adaptado da ABNT(2004a)

A classificação dos resíduos sólidos de acordo com sua origem é baseada na Lei Federal n° 12.305 de 2010.

Na tabela 2, os resíduos sólidos são classificados conforme sua origem:

Tabela 2: Classificação de resíduos sólidos segundo sua origem

Classificação dos Resíduos	**Classificação dos Resíduos Origem**
Domiciliares	Atividades domésticas em residências urbanas.
Limpeza urbana	Varrição, limpeza de logradouros e vias públicas e outros serviços de limpeza urbana.
Sólidos urbanos	Resíduos domiciliares e de limpeza urbana
Estabelecimentos comerciais e prestadores de serviço	Resíduos de limpeza urbana, serviços públicos de saneamento básico, serviços de saúde, industriais, da construção civil e serviços de transporte.
Serviços públicos de saneamento básico	Gerados nessas atividades, e resíduos sólidos urbanos.
Industriais	Processo produtivo e instalações industriais.
Serviços de saúde	Serviços de saúde, conforme definido em regulamento ou em normas estabelecidas pelos órgãos do Sistema Nacional de Meio Ambiente e do Sistema Nacional de Vigilância Sanitária.
Construção civil	Construções, reformas, reparos e demolições de obras de construção civil, incluídos os resultantes da preparação e escavação de terrenos para obras civis.
Agrossilvopastoris	Atividades agropecuárias e silviculturas, incluídos os relacionados a insumos utilizados nessas atividades.
Serviços de transporte	Portos, aeroportos, terminais alfandegários, rodoviários e ferroviários e passagens de fronteira.

Mineração	Atividade de pesquisa, extração ou beneficiamento de minérios.

Fonte: ROLLWAGEN, 2013, p 23.

Para maior entendimento desses conceitos, consultar o trabalho de Rollwagen (2013) e os demais citados por ele.

2.2 Tecnologias e Softwares

Neste tópico serão abordadas as tecnologias que serão utilizadas no desenvolvimento do sistema.

2.2.1 PHP

O PHP (*Hypertext Preprocessor)* é uma linguagem de programação interpretada e especialmente interessante para desenvolvimento web que pode ser mesclada dentro do HTML. Sua versão mais atual encontra-se na versão 7.0.5, e tanto sua versão 5.5 quanto a 5.6 recebem constantes atualizações. Grande parte da documentação do PHP está traduzida para o português, o que facilita a sua utilização. Mais informações, assim como toda a documentação sobre a linguagem, podem ser consultadas no site oficial do PHP (PHP, 2016).

2.2.2 HTML 5

O HTML **(***HyperText Markup Language*) é uma linguagem de marcação utilizada para desenvolver páginas *web*, os *browsers* (navegadores) interpretam essas marcações e apresentam a página para o usuário conforme a interpretação dessas marcações. A responsável pelas especificações da linguagem é a W3C (*World Wide Web Consortium*), que também fornece toda a documentação necessária para a compreensão da linguagem, além de tutoriais explicativos (HTML, 2016).

2.2.3 JavaScript

O JavaScript foi criado por Brendan Eich em1995, sua última versão oficial foi lançada em junho de 2015(W3C). JavaScript é uma linguagem que roda ao lado do cliente, esse tipo de linguagem é interpretado automaticamente pelos navegadores, devido à uma codificação específica. Estes interpretadores estão disponibilizados para uso público, para que

usem conforme a sua necessidade. A Mozilla lançou duas versões do interpretador, *pider Monkey* escrita em C e *Rhino* escrita em Java (SILVA, 2010).

A documentação da linguagem pode ser encontrada no site oficial do W3C e, apesar de não possuir documentação oficial em português, somente em inglês, o site oferece vários tutoriais que auxiliam no aprendizado da linguagem.

2.2.4 PostgreSQL

PostgreSQL é um SGBD (Sistema de Gerenciamento de Banco de Dados), objeto relacional desenvolvido na Universidade de Califórnia em Berkeley. Em 1996, um grupo continuou seu desenvolvimento via e-mail, com código fonte pela internet. Hoje, tal SGDB é comparável aos disponíveis no mercado em questão de desempenho e confiança (Postgresql, 2016).

Entre os recursos e características do postgreSQL, Worsley e Drake (2002) e Milani (2008) destacam-se:

- Suporte a operações ACID (Atomicidade, Consistência, Isolamento e Durabilidade).
- Funcionalidade de criação de cópias de bases de dados entre servidores.
- Passivo de configuração de *cluster* para trabalhar de forma distribuída.
- Gerenciamento de várias conexões simultâneas através do recurso de *Multithreads*:
- Conexão segura com uso do suporte nativo SSL.

2.2.5 *Bootstrap*

Bootstrap é um *framework front-end* que proporciona um desenvolvimento mais rápido e fácil de sites *web* e aplicações *web* com o alinhamento responsivo. Ele é facilmente utilizado por desenvolvedores de qualquer nível e indicado para todo o tipo de projeto. O b*ootstrap* surgiu devido a problemas internos da Twitter de incompatibilidade de código entre os programadores. Mark Otto e Jacob Thornton solucionaram este problema apresentando o *bootstrap*, de onde surgiu a sua primeira versão estável e pronta (SILVA, 2015).

Além das facilidades de se utilizar o *framework*, um dos principais motivos técnicos que se leva a utilizar o *bootstrap* é um pré-processador chamado LESS, que gera as folhas de estilo do *bootstrap*, gerando, assim, muito mais flexibilidade e poder do que as folhas de estilos convencionais ou não processadas (SILVA, 2015).

2.2.6 CSS 3

CSS (*Cascading Style Sheets*), também conhecido como folha de estilo em cascata, é definida de forma precisa e simples em sua *homepage* na W3C que diz: Folha de estilo em cascata é um mecanismo simples para adicionar estilos (por exemplo: fontes, cores, espaçamentos) aos documentos web (SILVA, 2012).

Cabe ao CSS todas as funções de apresentação de um documento, cores de fontes, tamanhos de textos, posicionamentos e todo o aspecto visual de um documento.

2.2.7 FPDF

Esta ferramenta está disponível em seu site oficial, FPDF é uma classe PHP que permite gerar arquivos PDF com PHP puro sem a necessidade de utilizar bibliotecas específicas de PDF, sendo que sua única exigência é a utilização do PHP 5.1 ou superior.

Entre as vantagens do FPDF destacam-se:

- Escolha de unidade de medida para formato de páginas e margens.
- Manipulação do cabeçalho e rodapé.
- Quebra automática de páginas e de linhas.
- Texto justificado.
- Suporte a imagens (JPEG,PNG e GIF).
- Cores.
- Links.
- Compactação de página.

Toda a documentação do FPDF está disponível no site oficial, facilitando a manipulação da ferramenta (FPDF, 2016).

2.2.8 PDO

Segundo o site oficial do PHP, que contém o manual e a documentação do PDO (*PHP Data Objects*) – uma extensão do PHP – o PDO fornece uma interface leve e consistente para acessar banco de dados em PHP. De forma mais simplória, o PDO fornece uma camada de abstração no acesso ao banco de dados. O PDO não funciona em versões anteriores a 5.0 (PHP, 2016).

O PDO fornecerá ao sistema mais segurança contra injeções SQL (*Structured Query Language*) por que antes de enviar a query diretamente para o banco de dados, o PDO utiliza um método chamado *prepare* que apenas inicia a query, substituindo os valores que seriam passados para o banco de dados por interrogações(?), então, após iniciada as queries, esses valores são passados pelo método *bindParam.* Após informados os parâmetros, o método *execute* deve ser chamado para então o PDO realizar a comunicação com o bando de dados.

A documentação da ferramenta se encontra no site oficial do PHP, facilitando na manipulação e aprendizagem do PDO.

2.2.9 SB *ADMIN* 2

SB *ADMIN* 2 é um *template* com código fonte aberto e tem como base o tema administrador que foi construido utilizando recursos de interface do usuário *bootstrap*. O tema apresenta uma variedade de recursos de plugins jQuery e entre algumas de suas características estão (Sartbootstrap, 2016): Menu lateral responsivo com *dropdowns* multinível e classes ativas; Poucos arquivos para facilitar uma personalização mais profunda; Estilo dos painéis personalizáveis; Dois plugins jQuery gráficos poderosos, Flot Charts e Morris.js; Menu de navegação superior responsivo com itens de menu suspenso.

O *template* está disponível em seu site oficial ou no site do *github,* o fato de possuir código aberto auxilia na melhora continua do *template* pois vários programadores fazem melhorias na ferramenta e disponibilizam essas atualizações diretamente no *github*.

2.2.10 *DATATABLES*

Segundo o site oficial o *datatables* é definido como **“**um *plug-in* que utiliza a biblioteca jQuery do JavaScript. É uma ferramenta altamente flexível, com base nos fundamentos da optimização progressiva, e adicionará controles avançadas de interação a qualquer tabela HTML (Datatables, 2016):

Entre as características da ferramenta pode-se citar: Paginação, pesquisa instantânea e multi-coluna de ordenação; Suporte praticamente todas as fontes de dados como DOM, JavaScript, Ajax e processamentos do lado do servidor; Qualidade profissional apoiado por um conjunto de mais de 2900 testes de unidades; Extensas opções e um expressivo API; *Software* livre de código aberto.

3 METODOLOGIA

Neste tópico será abordado o levantamento de requisitos, as tecnologias e softwares que irão compor o sistema e um breve conceito sobre casos de uso e diagrama de classes, implementação e *feedback*.

3.1 Levantamento de Requisitos

A identificação e apresentação da atual estrutura e funcionamento das cooperativas de recicladores de Passo Fundo terá como base a metodologia utilizada por Rollewagen (2013) em seu trabalho de mestrado, que é composta por entrevistas. Tal material será complementado com pesquisa bibliográfica.

Para os dados coletados que não se fizerem satisfatórios, ou se mostrarem insuficientes para aplicação no projeto, será utilizada a técnica de levantamento de requisitos *role playing* (Leffingwell & Widrig, 2000), também conhecida como etnografia (Sommerville, 2011), muito útil para complementar, identificar ou transmitir informações.

A técnica de levantamento de requisitos etnografia é apresentada por Sommerville como:

> Uma técnica de observação que pode ser usada para compreender os processos operacionais e ajudar a extrair os requisitos de apoio para esses processos. Um analista faz uma imersão no ambiente de trabalho em que o sistema será usado. O trabalho do dia a dia é observado e são feitas anotações sobre as tarefas reais em que os participantes estão envolvidos. O valor da etnografia é que ela ajuda a descobrir requisitos implícitos do sistema que refletem as formas reais com que as pessoas trabalham, em vez de refletir, processos formais definidos pela organização (2011, p. 75).

Segundo o autor, esta técnica consiste na observação do cotidiano do usuário em seu ambiente de trabalho. Caso tal observação não se faça suficiente, o engenheiro de software pode, ainda, fazer o papel do cliente, passando a ser um usuário do sistema e conseguindo, portanto, identificar as necessidades e dificuldades na realização de determinada tarefa.

Será apresentado, ainda, um documento que especifique os requisitos funcionais e não funcionais do sistema de acordo com os dados levantados nas etapas de levantamento de requisitos anteriores.

Segundo Guedes (2011), os requisitos devem ser classificados em funcionais que dizem o que o sistema deve fazer na visão do cliente, ou seja, definem as funcionalidades do

sistema de informação; e não-funcionais são as consistências, condições, validações e restrições que devem ser aplicadas sobre suas funcionalidades.

Com dados devidamente analisados será possível identificar os requisitos funcionais e não funcionais do sistema, o que auxiliará no desenvolvimento prático do projeto.

3.2 Diagrama de Casos de Uso

Para uma melhor compreensão do funcionamento do sistema, será realizada uma análise dos dados coletados através dos levantamentos de requisitos, e a partir da análise destes dados será criado o diagrama de casos de uso do sistema.

Segundo Guedes, diagrama de caso de uso é definido como:

> o diagrama mais geral e informal da UML, utilizado normalmente nas fases de levantamento e análise de requisitos do sistema, embora venha a ser consultado durante todo o processo de modelagem e possa servir de base para outros diagramas. Apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma ideia geral de como o sistema irá se comportar (2011, p.30).

Através do diagrama de casos de uso será possível analisar e agrupar as funcionalidades e comportamento do sistema com a interação do usuário. Os principais conceitos ligados ao modelo em questão são: atores e casos de uso.

O ator representará o papel do usuário, a funcionalidade ou ação que o usuário desencadeará (GUEDES, 2011). Já o caso de uso é uma sequência de interações entre um sistema e agentes externos que utilizam esse sistema. O caso de uso deve definir, como o nome já diz, o uso da funcionalidade, mas sem revelar a estrutura ou comportamento interno desse sistema (BEZERRA, 2007).

3.3 Diagrama de Classes

Segundo Guedes (2014, p17) “o diagrama de classes tem por base a seguinte definição: a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos que cada classe tem, além de estabelecer como as classes se relacionam e trocam informações entre si”. Com a criação do diagrama de classes do projeto posposto, obter-se-ão informações sobre todos os objetos do modelo de forma visual, o que auxiliará na criação da base de dados da aplicação.

Para esta etapa que se realizará após os levantamentos de requisitos, a modelagem de dados é a primeira grande fase no projeto de banco de dados, e, uma das vantagens da utilização deste modelo é que se pode adiar escolha de SGDB, além da possibilidade dos usuários não especialistas de conseguirem entender melhor o sistema proposto de forma visual (SIEBRA, 2010).

3.4 Aplicação das Tecnologias e softwares

Várias linguagens trabalhando em conjunto serão utilizadas para desenvolver o sistema, que será programado utilizando PHP, responsável por todo o *back-end* do software. O *front-end,* ou seja, o layout do sistema será desenvolvimento com HTML, CSS e Bootstrap.

A comunicação com o servidor em PHP será realizada através de JavaScript, e as máscaras dos campos dos formulários vão ser desenvolvidas através de jQuery, que fornece ótimas funções para esse tipo de tarefa.

A base de dados será feita com o PostgreSQL, que é um software gratuito com confiabilidade e a segurança de grandes bancos pagos, como Oracle. A comunicação com a base de dados utilizará uma abstração em PHP, o PDO, funcionalidade que oferece segurança contra injeções SQL.

O sistema ainda deverá contar com emissões de relatórios, utilizando uma tecnologia gratuita disponível para estas finalidades, FPDF, que é código aberto e de fácil manuseio, o layout do arquivo PDF pode ser escrito puramente em PHP. O FPDF se encarregara de transformar as instruções em PHP em um documento PDF para o usuário.

3.5 Implantação

Após a conclusão do sistema, ele será implantado em uma cooperativa de Passo Fundo, e poderá ser instalado e hospedado em um servidor web, com acesso via internet. Caso a cooperativa não possua tal acessibilidade, o sistema poderá, ainda, ser instalado em uma máquina local. Depois de devidamente instalado, um funcionário da empresa receberá as devidas instruções para que o sistema possa ser executado na cooperativa, com a finalidade de se testar a aplicação.

3.6 *Feedback*

Para esta etapa que se realizará após a utilização do sistema por parte da cooperativa, será feita uma reunião para levantamento de um feedback sobre a utilização do sistema. O objetivo é que, nessa reunião, sejam abordadas questões como: o suprimento, ou não, das necessidades de gestão da cooperativa, as dificuldades encontradas em sua utilização e possíveis funcionalidades extras que tornariam esse sistema mais satisfatório para o andamento das atividades da cooperativa. Esta etapa é necessária para que o sistema seja aprimorado, ou seja, de forma a contemplar todas as funcionalidades da empresa.

4 RESULTADOS

Nesse tópico será abordado o desenvolvimento do sistema que envolve analise de requisitos, diagramas utilizados e as principais funcionalidades do sistema.

4.1 Análise de Requisitos

Com base nos requisitos levantados através de entrevistas e da técnica *role playing*, foi identificado que o sistema deve permitir cadastrar usuários que terão acesso ao sistema de acordo com seus próprios níveis de acesso. Tais níveis foram definidos da seguinte forma: o gerente responsável pelas cooperativas é o usuário nível 2, e é ele que poderá cadastrar a cooperativa, bem como cadastrar um usuário nível 1, que terá acesso ao sistema desta cooperativa e às funcionalidades do sistema, assim como o gerente.

Quanto à questão de confiabilidade, os usuários do sistema não poderão ser excluídos do banco de dados, apenas desativados, já que a manutenção desses dados usuários auxiliará aos gestores em possíveis análises. Tais dados são importantes para a manutenção de informações como: identidade dos usuários que fizeram alterações e inserções ou remoções de dados no sistema.

Não será permitido o acesso ao sistema sem um usuário devidamente cadastrado, já que apenas o gerente pode fazer esse trabalho de cadastramento. Os usuários que possuam cadastro no sistema, assim como os gerentes, poderão alterar seus dados quando desejarem, no entanto, todas as alterações ficarão catalogadas em um arquivo de registro.

Além dessas funcionalidades, o sistema precisará ainda de muitas outras. Serão elas: um módulo de gestão relacionado aos catadores autônomos, o qual registrará a hora trabalhada e salário dos colaboradores da cooperativa, além de cadastros de materiais recicláveis e registro de estoque. Um modelo de vendas que registre a venda dos materiais cadastrados no estoque da cooperativa para seus compradores, além de guardar os dados das vendas para futuras consultas e análises. O registro mensal das despesas e faturamento da cooperativa, através do cálculo das vendas realizadas, despesas e horas trabalhadas dos recicladores.

O monitoramento de todas as funcionalidades do sistema também deve ser registrado, já que todas as ações do sistema devem ser guardadas para futuras auditorias, seja para análise ou para a verificação de possíveis falhas de segurança. O momento no qual um usuário acessa o sistema também deve ser registrado, através da data, hora e endereço IP (Internet Protocol) de acesso.

Para análise dos gestores da cooperativa, o sistema deverá emitir relatórios de estoque, usuários, vendas, clientes, recicladores, materiais, enfim, das principais informações de dados que o sistema contém, o que acarretará no acesso à informação confiável e ágil por parte dos gestores.

Os requisitos funcionais e não funcionais referentes ao sistema de informação proposto estão apresentados na tabela 3:

Tabela 3: Requisitos funcionais e não funcionais:

Requisitos Funcionais
1 – Cadastrar Usuários (gerentes): Informações: código, razão social, nome fantasia, CNPJ, endereço e telefone da empresa que trabalha, nome do usuário, telefone pessoal, e-mail, data de cadastro, data de desligamento, status de ativação.
2 – Cadastrar Cooperativas: Informações: código, razão social, nome fantasia, CNPJ, endereço e telefone, e-mail, data de cadastro, data de desligamento, status de ativação.
3 – Cadastrar Usuários do Sistema: (Sistema da cooperativa): Informações: código, cooperativa que trabalha, nome do usuário, telefone pessoal, e-mail, data de cadastro, data de desativação, status de ativação.
4 – Cadastrar Clientes: Informações: código, razão social, nome fantasia, CNPJ, endereço e telefone, e-mail, data de cadastro, data de desligamento, status de ativação.
5 – Cadastrar Despesas: (da cooperativa) Informações: código, descrição, cooperativa.
6 – Cadastrar Recicladores: (da cooperativa) Informações: código, nome, RG, CPF, data de nascimento, escolaridade, renda familiar, tipo de moradia, número de residentes, se recebe bolsa família, telefone, endereço, data de ingresso no projeto, data de desligamento, status de ativação.
7– Registar Salario: (dos recicladores) Informações: código, mês e valor liquido.
8– Registrar Horas Trabalhadas: (dos recicladores) Informações: código, mês, vale.
9– Cadastrar Materiais: (da cooperativa) Informações: código, descrição, estoque (em quilos), observações.
10 – Registrar Vendas: Informações: código, data da venda, valor total da venda, cooperativa, cliente.
11 – Registrar Item da Venda:

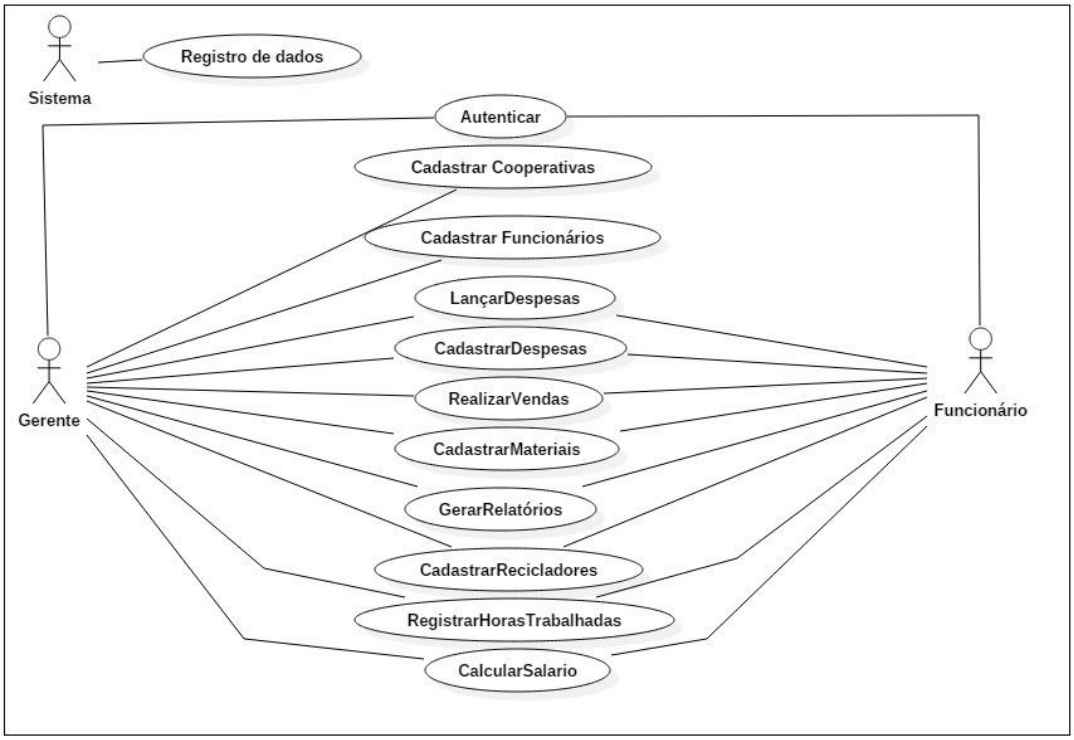
Informações: código, valor do quilo, peso vendido, valor do item, material.		
Requisitos Não Funcionais		
Nome	Restrição	Categoria
1 – Controle de Acesso	Apenas usuários devidamente cadastrados devem ter acesso ao sistema.	Segurança
3 – Compatibilidade com *browsers*	O sistema deve ser compatível com os três principais *browsers, Mozilla, Chrome* e *Internet Explorer*.	Portabilidade
4 – Facilidade operacional	Sistema de fácil manuseio, usuários com formação básico em computação deve conseguir utilizar o sistema.	Usabilidade
5 – Durabilidade dos dados	Mantém todas as informações registradas ou em banco de dados ou arquivos de registro.	Confiabilidade

Fonte: Elaborada pelo autor (2016)

4.2 Diagrama de Casos de Uso

O seguinte diagrama UML (*Unified Modeling Language*) tem por objetivo mostrar a interação do sistema com os usuários, lembrando que os registros dos usuários que estarão acessando ou fazendo manutenção no sistema, serão gerados pelo próprio sistema sem que o usuário perceba. A figura 1 representa o diagrama de casos de uso do sistema proposto:

Figura 1: Diagrama de Casos de Uso



Fonte: Elaborado pelo autor, 2016

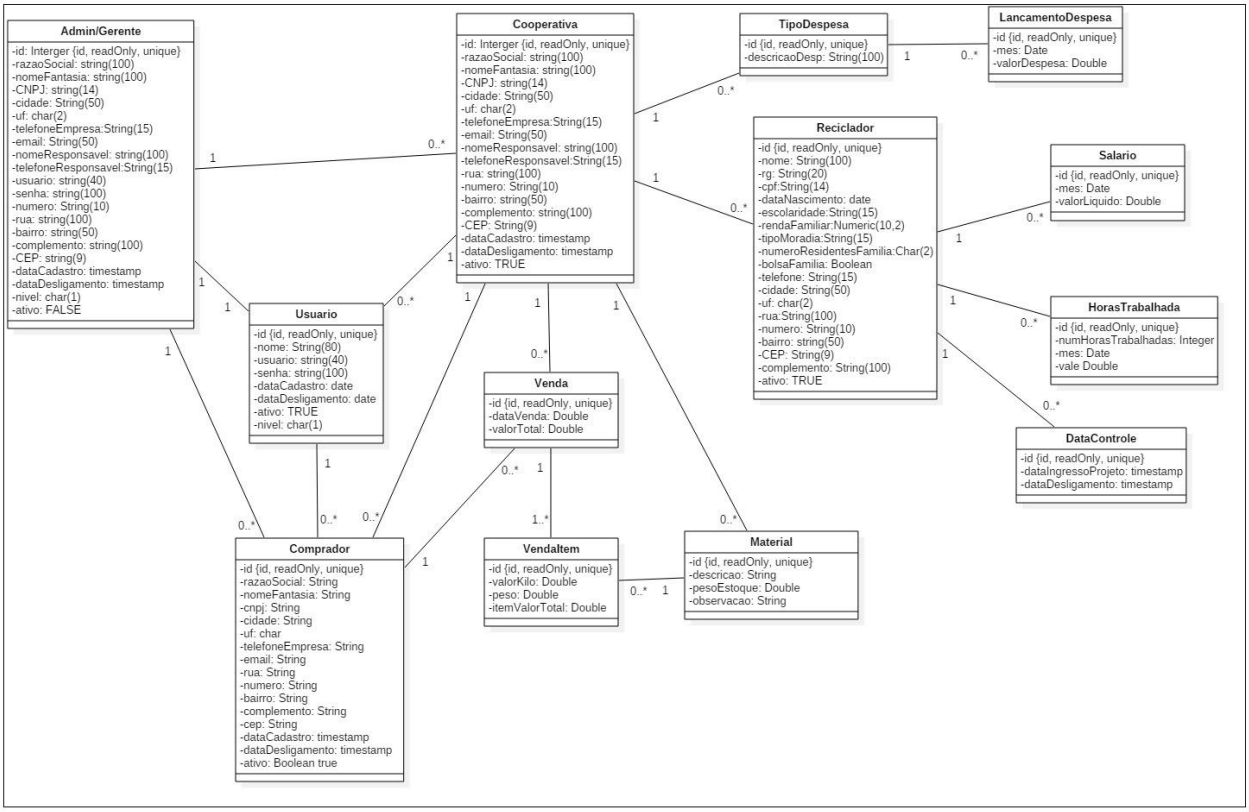
Explicando melhor, o sistema guardará, automaticamente, o registro de todas as alterações nele realizadas, bem como data, hora, endereço IP e usuário responsável. Apenas o Gerente poderá cadastrar as cooperativas, e, após isso, o gerente poderá cadastrar um usuário para o sistema desta cooperativa.

As funcionalidades como o cadastramento de recicladores, materiais, despesas, geração de relatórios, horas trabalhadas, lançamento de despesas e cálculo de salário serão realizadas tanto pelo funcionário quanto pelo usuário da cooperativa. Tanto o usuário poderá alterar dados que o gerente tenha cadastrado ou modificado, como vice e versa. Caso se faça necessário, essas alterações poderão ser consultadas no arquivo de registro.

4.3 Diagrama de Classes

O diagrama de classes visa mostrar com mais clareza o sistema proposto, com o objetivo de suprir as necessidades levantadas para o sistema. Após o levantamento e análise dos dados coletados, foi criado o diagrama de classes, o qual é apresentado na figura 2:

Figura 2: Diagrama de Classes



Fonte: Desenvolvido pelo autor, 2016

A classe “admin/Gerente” é a classe do administrador principal do sistema, nela está contido um atributo chamado “nível”, que servirá como base de controle para o sistema identificar quem pode realizar o quê no sistema. Esta classe se relaciona com a classe “cooperativa”, com o “usuário” e “comprador”, que devem ser cadastrados pelo gerente, com exceção do comprador que também pode ser cadastrado pelo usuário.

A classe “usuário”, refere-se ao usuário do sistema na cooperativa. Nessa classe existem dois atributos essenciais, um chamado “nível” e outro que é a chave estrangeira da cooperativa. Estes dois atributos são responsáveis por não deixar um usuário realizar alterações em outras cooperativas ou em tabelas que são de acesso a usuários de nível superior. Cada usuário apenas poderá ser cadastrado depois que uma cooperativa estiver, também, devidamente cadastrada no sistema.

A classe “comprador” armazena dados das empresas que compram os materiais recicláveis da cooperativa, ela se relaciona com a classe “venda” e com a “cooperativa”.

Já a classe “cooperativa” é a classe que possui ligação com todas as classes do sistema. Através de sua chave será possível controlar os recicladores que ela possui, assim como materiais e despesas, que em trabalho conjunto com o nível da sessão dos usuários, evitará que usuários não autorizados acessem informações de outras cooperativas cadastradas no

sistema.

A classe “tipo de despesas” é a classe responsável pelas despesas da cooperativa, como por exemplo, gastos com água, luz telefone, dentre outros. O lançamento dessas despesas será realizado na classe Lançamento de despesas, que solicita o mês e valor de tais gastos.

“Material” é a classe responsável pelos materiais recicláveis. Nela são cadastrados os produtos, informando o nome do material e sua quantidade em quilogramas assim que entrarem na cooperativa.

“Venda de Item” é um complemento de venda. Em conjunto, essas classes são responsáveis pelas vendas da cooperativa. É necessário informar os materiais que irão compor a venda e o comprador.

A classe “reciclador” é dos catadores que trabalham com a cooperativa, ela se relaciona com horas trabalhadas e com salário, que é calculado a partir da soma de todas as vendas realizada no mês da cooperativa, e subtração das despesas do respectivo mês. O valor final é dividido pelo total de horas trabalhadas de todos os recicladores cadastrados em suas respectivas cooperativas, obtendo assim o valor da hora trabalhada. Em seguida, basta multiplicar o valor da hora trabalhada com as próprias horas trabalhadas de cada reciclador.

4.4 Desenvolvimento

Nesta sessão serão apresentadas as principais etapas do desenvolvimento do sistema proposto, como arquitetura de desenvolvimento, ambiente e componentes considerados pontos importantes do sistema.

4.4.1 Ambiente de desenvolvimento

O sistema para cooperativas de reciclagem foi desenvolvido com foco para utilização WEB, ou seja, projetado para ser utilizado online, mais detalhes do desenvolvimento serão esclarecidos nos tópicos seguintes.

4.4.2 Ferramentas utilizadas no desenvolvimento

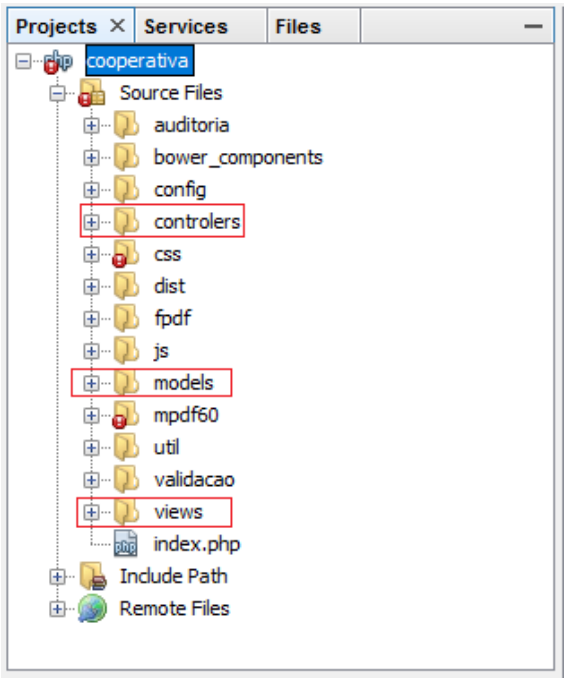
A IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado) utilizada para o desenvolvimento do software foi o NetBeans 8.1. Para o ambiente PHP foi utilizado o XAMPP v5.6.24, que traz uma fácil instalação e manuseio e consigo o

apache com o PHP na versão 5.6.24. Todo o desenvolvimento *front-end*, ou mais especificamente a parte do layout, é a adaptação do *template SB ADMIN 2* em conjunto com o plug-in *datatables* para realizar a comunicação entre *front-end* e *back-end* com auxílio de Ajax.

4.4.3 Padrão de arquitetura de software

Também conhecido como *desing pattern*, a estrutura utilizada para o desenvolvimento segue o padrão MVC (Model, View, Controller) que consiste em uma organização e divisão dos três principais componentes do sistema, onde o modelo consiste dos dados da aplicação e suas funções, as visões representam as saídas para o usuário e o controlador é responsável pela parte lógica do sistema, sistemática comumente abordada em qualquer framework de PHP. O projeto ficou estruturado como mostrado na figura 3:

Figura 3: Estrutura do projeto



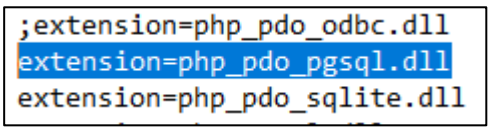
Fonte: do autor, 2016

4.4.4 Operações no banco de dados

Um dos pontos mais importante nesta aplicação, e em qualquer aplicação, é a conexão com o banco de dados, no sistema proposto foi utilizado uma extensão do PHP chamada PDO, que consiste em fornecer uma camada de abstração com o banco de dados.

O primeiro passo para se utilizar o PDO é ativar a extensão php_pdo_pgsql.dll no arquivo php.ini do Apache, como o banco de dados utilizado é o postgreSQL basta descomentar a linha como na figura 4:

Figura 4: Extensão PDO para postgreSQL



Fonte: do autor, 2016.

Foi criada uma classe chamada DataBase, que é responsável por realizar não só a conexão com o banco de dados desta aplicação, mas também todas as operações CRUD (*Create, Read, Update e Delete*) do sistema. Todas as classes do modelo estenderão a classe DataBase para poderem realizar as operações desejadas no banco de dados.

Para realizar a conexão com o bando de dados foi criado método *connect*, que será chamado sempre que uma classe requisitar uma operação. Neste método é criado um objeto PDO que retorna o a conexão com o banco de dados. O método é apresentado na figura 5:

Figura 5: Método *connect*

```
/* Faz a conexão com o banco de dados e retorna o metodo @this->conn com o conexão */
private function connect() {
    try {
        $this->conn = new PDO($this->getDBType() . ":host=" . $this->getHost() . ";port=" . $this->getPort()
                . ";dbname=" . $this->getDB(), $this->getUser(), $this->getPassword());
        $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    } catch (Exception $e) {
        echo 'Erro de conexão => '.$e->getMessage();
    }

    return ($this->conn);
}
```

Fonte: do autor, 2016.

Quando uma classe requisitar um *select*, a função *select*() é chamada, sendo necessário passar a query e seus valores por parâmetro, por exemplo, a *query 'select * from* vendas *where id* = ?', o valor que possui '?' deve ser passado em um *array* por parâmetro na chamada do metodo *select*. Método *select* na figura 6:

Figura 6: Método *select*

```
/* Método select que retorna um Value Object ou um array de objetos */
protected function select($sql, $params = null) {
    try {
        $query = $this->connect()->prepare($sql);
        $query->execute($params);

        $rs = $query->fetchAll(PDO::FETCH_OBJ);

    } catch (Exception $e) {
        echo 'Erro ao selecionar dados => '.$e->getMessage();
        $rs = false;
    } finally {
        $this->disconnect();
        return ($rs);
    }
}
```

Fonte: do autor, 2016

A função *connect* é chamada, então a função *prapare* recebe a *query* e a prepara para receber os parâmetros. Em seguida a função *disconnect* encerra a conexão com o banco de dados e então retorna um objeto contendo um *array* de objetos recuperados do banco de dados.

Os métodos *update* e *delete* são relativamente parecidos, tendo o mesmo comportamento base que o método *select*, a diferença é que tanto o método *update* quanto o método *delete* retornam o número de registros afetados. Comforne a figura 7 do método *update* e a figura 8 do método *delete*:

Figura 7: Método *update*

```
/* Método update que altera valores do banco de dados
 * e retorna o número de linhas afetadas */
protected function update($sql, $params = null) {
    try {
        $query = $this->connect()->prepare($sql);
        $query->execute($params);
        $rs = $query->rowCount();
    } catch (Exception $e) {
        echo 'Erro ao atualizar dados => '.$e->getMessage();
        $rs = false;
    }finally {
        $this->disconnect();
        return ($rs);
    }
}
```

Fonte: do autor, 2016

Figura 8: Método *delete*

```
/* Método delete que excluí valores do banco de dados
 * retorna o número de linhas afetadas */
protected function delete($sql, $params = null) {
    try {
        $query = $this->connect()->prepare($sql);
        $query->execute($params);
        $rs = $query->rowCount();
    } catch (Exception $e) {
        echo 'Erro ao excluir dados => '.$e->getMessage();
        $rs = false;
    }finally {
        $this->disconnect();
        return ($rs);
    }
}
```

Fonte: do autor, 2016

O método *insert*, responsável por salvar os registros no banco de dados também possui a mesma estrutura das demais operações, a diferença está que o seu retorno é o *id* do último registro inserido no banco de dados. Segue figura 9 do método *insert*:

Figura 9: Método *insert*

```
/* Método insert que insere valores no banco de dados
 * e retorna o último id inserido */
protected function insert($sql, $params = null) {
    try {
        $query = $this->connect()->prepare($sql);
        $query->execute($params);
        $rs = $query->fetch(PDO::FETCH_ASSOC);
        $rs = $rs['id'];
    } catch (Exception $e) {
        echo 'Erro ao inserir => '.$e->getMessage();
        $rs = false;
    }finally {
        $this->disconnect();
        return ($rs);
    }
}
```

Fonte: do autor, 2016

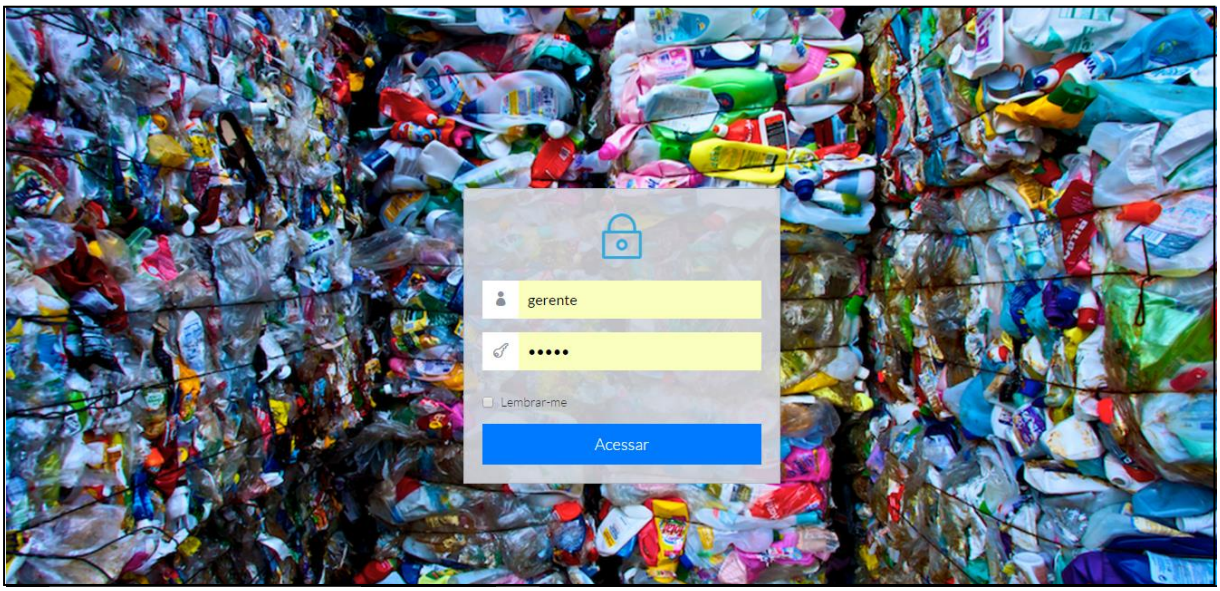
Caso ocorra algum erro durante alguma tentativa de realizar qualquer uma das operações no banco de dados, uma exceção é gerada e o retorno da função é falso.

4.4.5 *Login* e Registro

Para obter acesso ao sistema é necessário estar cadastrado no banco de dados como gerente ou usuário, lembrando que o gerente pode realizar todas as funcionalidades de um usuário comum, e o usuário não pode cadastrar cooperativas ou outros usuários.

Ao executar o sistema a tela de *login*, como mostrado a figura 10, é apresentada ao usuário, onde deve se informar o usuário ou *email* e a senha.n

Figura 10: Tela de *login*



Fonte: do autor, 2016

Ao clicar no botão “acessar”, uma função em *javascript* irá acessar o controlador de *login* através do *ajax*. No controlador é verificado se os campos não estão em branco, em seguida é criado uma instancia da classe *login*, que se encontra na camada de modelo. Ele é responsável por chamar o método que loga no sistema, que vai realizar uma query em *sql* buscando o usuário, caso a senha esteja incorreta a função retorna falso e o usuário será informado que a senha está incorreta. Caso os dados estejam todos corretos o usuário é registrado no sistema, criasse então uma sessão com os dados do usuário logado que em seguida é redirecionado para a tela principal do sistema.

As figuras 11, 12 e 13, contém os métodos logar, registrar usuários e a tela principal do sistema, respectivamente:

Figura 11: Método logar

```php
public function login($remember){
    // seleciona no banco o usuario
    $result = $this->select("SELECT * FROM admin_gerentes WHERE usuario = ? OR email = ? AND ativo = ?",
            array($this->getUser(),$this->getUser(),TRUE));
    if(empty($result)){// se não obtiver reultados retorna false
        return (false);
    }
    else if(!$this->verifyPass($this->getPass(), $result[0]->senha, 1)){ // verefica o pessowrd
        return (false);
    }
    if($remember){
        $this->remember($this->getUser(), $this->getPass());
    }


    $this->registerUser($result[0]->id,$result[0]->nome_responsavel, $result[0]->usuario, $result[0]->nivel);
    //registra o usuario e retorna true

    return (true);
}
```

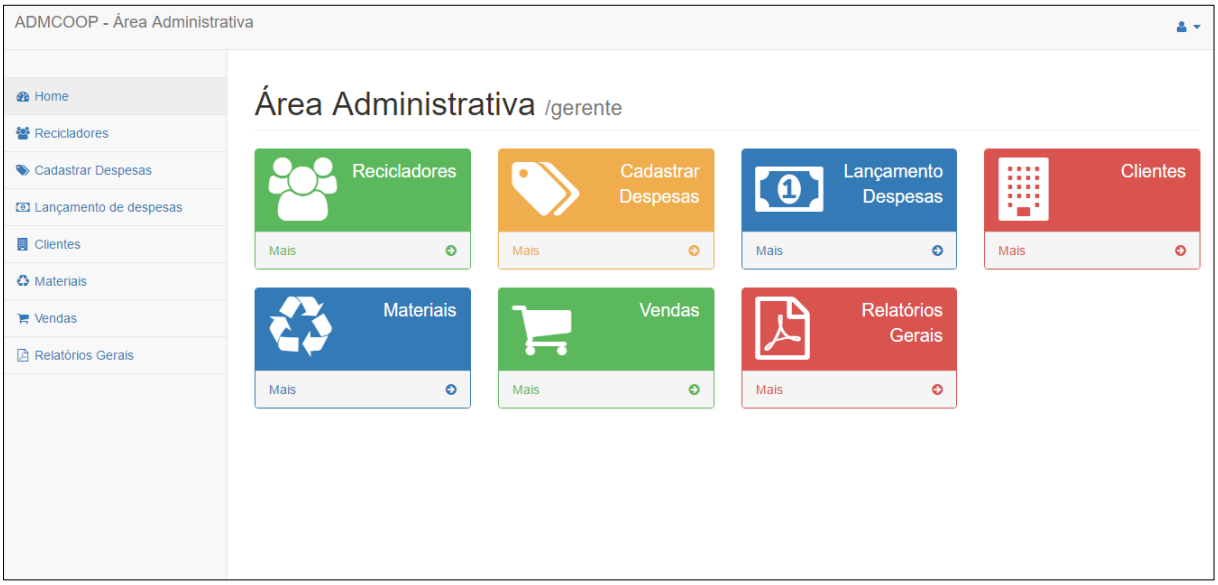
Fonte: do autor, 2016

Figura 12: Método registrar usuário

```php
private function registerUser($id,$name, $user, $nivel){
    session_start();
    $_SESSION['id']   = $id;
    $_SESSION['name']   = $name;
    $_SESSION['user']   = $user;
    $_SESSION['nivel'] = $nivel;
    $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
    session_regenerate_id();
    $this->log->register("([Sessao / ID: ".$nivel." / ".$id."] Novo Login de [" . $user . "]"
            . "[" .$_SERVER['REMOTE_ADDR']. "])", "LOGIN", "logLogin.txt");
}
```

Fonte: do autor, 2016

Figura 13: Tela principal do sistema



Fonte: do autor, 2016

Observando a figura 12, notasse ainda que o método *registerUser* está realizando o registro do usuário em um arquivo texto chamado logLogin.txt, ou seja, sempre que um usuário realizar um *login* no sistema, este registro será guardado contendo as informações da sessão e ainda o *IP*, data e hora do acesso como na figura 14:

Figura 14: Arquivo logLogin.txt

```
LOGIN] Mensagem: ([Sessao / ID: 2 / 2] Novo Login de [gerente][::1]) Data: Friday 16th of September 2016 07:20:46 PM
LOGIN] Mensagem: ([Sessao / ID: 1 / 1] Novo Login de [funcionario][::1]) Data: Friday 16th of September 2016 07:20:58 PM
LOGIN] Mensagem: ([Sessao / ID: 2 / 2] Novo Login de [gerente][::1]) Data: Tuesday 20th of September 2016 10:43:51 AM
LOGIN] Mensagem: ([Sessao / ID: 2 / 2] Novo Login de [gerente][::1]) Data: Tuesday 20th of September 2016 10:45:09 AM
LOGIN] Mensagem: ([Sessao / ID: 2 / 2] Novo Login de [gerente][::1]) Data: Tuesday 20th of September 2016 11:21:55 AM
LOGIN] Mensagem: ([Sessao / ID: 2 / 2] Novo Login de [gerente][::1]) Data: Wednesday 28th of September 2016 12:06:13 PM
```

Fonte: do autor, 2016

4.4.6 Visualização dos dados

Apresentada a tela principal ao usuário, este está autorizado a realizar as operações desejadas no sistema. Como exemplo simples, será detalhado a execução da funcionalidade cadastrar despesas. Como o desenvolvimento segue uma estrutura padronizada, as operações são realizas de forma muito semelhante, mudando basicamente o número de dados manipulados entre as funcionalidades especificas.

Partindo da tela principal do sistema, ao se clicar em cadastrar despesas, seja no canto lateral esquerdo ou no menu personalizado em *boxes*, o usuário será direcionado para a página de manutenção dos cadastros de despesas, como segue na figura 15:

Figura 15: Tela Cadastrar Tipos de despesas

Fonte: do autor, 2016

Ao carregar a página, a primeira ação do sistema é mostrar todos os tipos de despesas que existem cadastradas no banco de dados. Para essa funcionalidade foi utilizado o *datatables* e *ajax*, conforme figura 16:

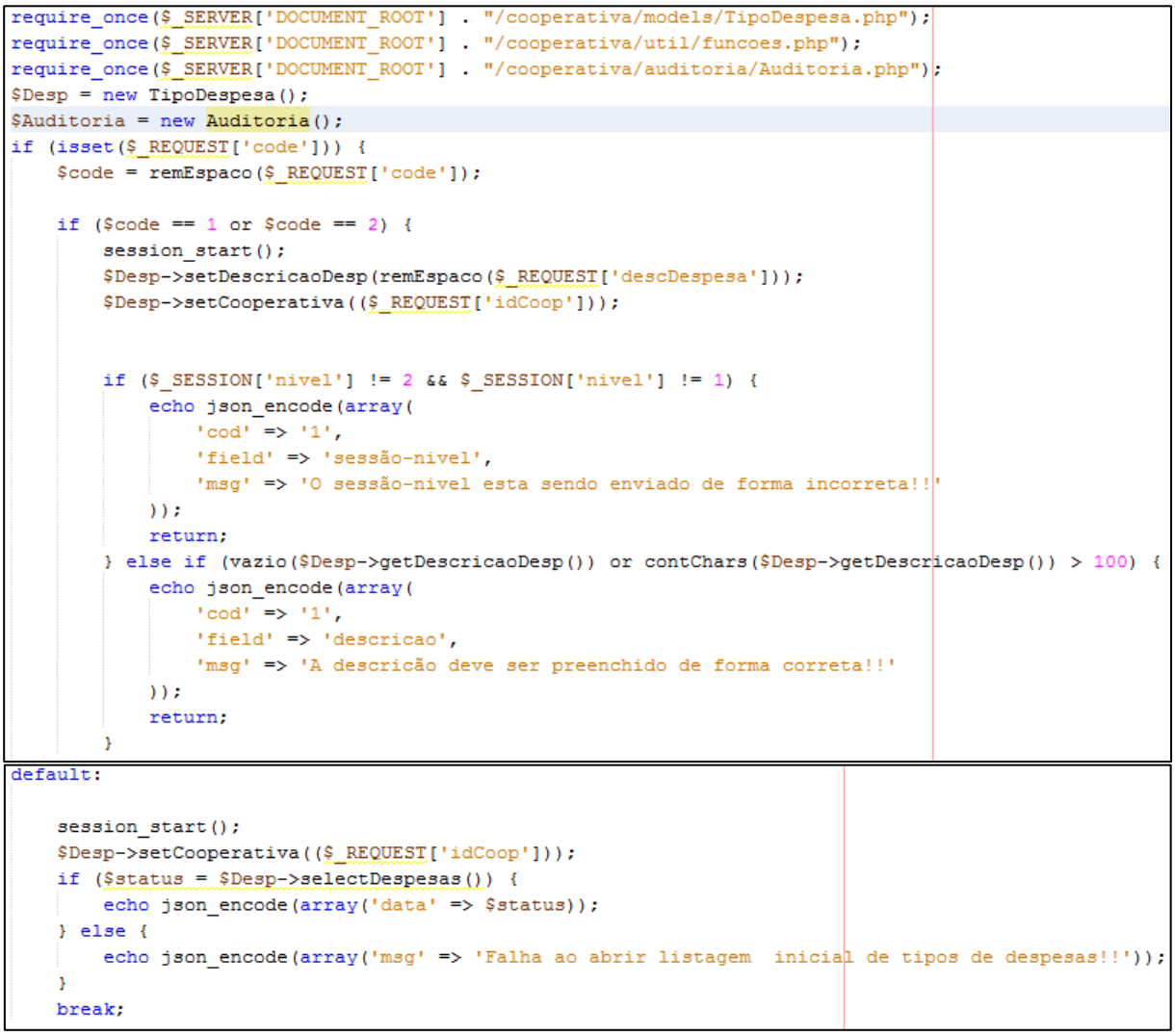
Figura 16: *Datatables* e *ajax*

```
$(document).ready(function () {
    $('#btnEditar, #btnDeletar').attr('disabled', 'disabled');
    var idCoop = ($('#idCoop').val()); //Preciso passar este parametro
    var table = $('#listaTipoDespesas').DataTable({
        "ajax": "../../controlers/TipoDespesaControle.php?code=0&idCoop=" + idCoop, // nessa URL
        "stateSave": "true",
        "language": {
            "sEmptyTable": "Nenhum registro encontrado",
            "sInfo": "Mostrando de _START_ até _END_ de _TOTAL_ registros",
            "sInfoEmpty": "Mostrando 0 até 0 de 0 registros",
            "sInfoFiltered": "(Filtrados de _MAX_ registros)",
            "sInfoThousands": ".",
            "sLengthMenu": "_MENU_ resultados por página",
            "oPaginate": {
                "sNext": "Próximo",
                "sPrevious": "Anterior",
                "sFirst": "Primeiro",
                "sLast": "Último"
            },
            "oAria": {
                "sSortAscending": ": Ordenar colunas de forma ascendente",
                "sSortDescending": ": Ordenar colunas de forma descendente"
            }
        },
        "columns": [
            {"data": "id"},
            {"data": "despesa_descricao"},
        ]
    });
```

Fonte: do autor, 2016

Assim que a página é carregada, a função da figura 16 é chamada, pois a função está sendo chamada na própria *view*, realizando então chamada por *GET* ao controlador passando o código da requisição e o *id* da cooperativa. O atributo *language* é a tradução do *datatables* e o atributo *columns* corresponde as colunas que serão exibidas ao usuário, os atributos após o campo *data* são os dados que o controlador ira retornar ao *datatables*. Para melhor compreensão do controlador, observa-se a figura 17:

Figura 17: Controlador do tipo de despesas



```php
require_once($_SERVER['DOCUMENT_ROOT'] . "/cooperativa/models/TipoDespesa.php");
require_once($_SERVER['DOCUMENT_ROOT'] . "/cooperativa/util/funcoes.php");
require_once($_SERVER['DOCUMENT_ROOT'] . "/cooperativa/auditoria/Auditoria.php");
$Desp = new TipoDespesa();
$Auditoria = new Auditoria();
if (isset($_REQUEST['code'])) {
    $code = remEspaco($_REQUEST['code']);

    if ($code == 1 or $code == 2) {
        session_start();
        $Desp->setDescricaoDesp(remEspaco($_REQUEST['descDespesa']));
        $Desp->setCooperativa(($_REQUEST['idCoop']));


        if ($_SESSION['nivel'] != 2 && $_SESSION['nivel'] != 1) {
            echo json_encode(array(
                'cod' => '1',
                'field' => 'sessão-nivel',
                'msg' => 'O sessão-nivel esta sendo enviado de forma incorreta!!'
            ));
            return;
        } else if (vazio($Desp->getDescricaoDesp()) or contChars($Desp->getDescricaoDesp()) > 100) {
            echo json_encode(array(
                'cod' => '1',
                'field' => 'descricao',
                'msg' => 'A descrição deve ser preenchido de forma correta!!'
            ));
            return;
        }
default:

    session_start();
    $Desp->setCooperativa(($_REQUEST['idCoop']));
    if ($status = $Desp->selectDespesas()) {
        echo json_encode(array('data' => $status));
    } else {
        echo json_encode(array('msg' => 'Falha ao abrir listagem  inicial de tipos de despesas!!'));
    }
    break;
```

Fonte: do autor, 2016

Primeiramente o controlador chama as classes necessárias através do *require* em seguida cria a instancia dessas classes para realizar as operações desejadas. A classe auditoria também é instanciada para que seja possível realizar os *logs* do sistema sobre as operações de *insert*, *update* e *delete*.

A variável *code* é requisitada, toda a requisição ao controlador deve ter um código de requisição chamado de *code*. Como opção de padronização foi adotado os valores 1 e 2 para *insert* e *update*, respectivamente, valor 3 para *delete*, valor 4 para uma seleção especifica por *id* e o valor *defautlt* para um *select* em todos os registros daquela manutenção especifica, neste caso, todos os registros de tipos de dados. Os valores de 1 a 4 e o *default* são controlados pela estrutura *switch*.

Nota-se neste caso específico que o valor passado para o controlador é o valor zero, caindo então na clausula *default* do *switch*, que faz uma chamada ao método selectDespesas que é simplesmente uma query em sql que irá retornar todos os dados requisitados. O retorno é em *json*, pelo fato que o *datatables* exige os dados retornados em *json* para poder manipula-los.

4.4.7 Inserção de Dados

Para adicionar novos registros no sistema, o usuário deve clicar no botão “novo” que pode ser visualizado na figura 15, que vai redirecionar o usuário a uma nova página, ou *view* a nível de sistema, contendo o formulário com os dados necessários para a inserção do novo registro, conforme a figura 18:

Figura 18: Formulário de tipos de despesas

Fonte: do autor, 2016

Nesta manutenção específica é necessário apenas informar a descrição da despesa, caso o usuário tente salvar a manutenção sem preencher o campo, não vai conseguir devido a atributo *required* no campo descrição.

Para a inserção e para atualização é um determinado registro, é sempre utilizado o mesmo formulário, o que vai diferenciar determinada ação é a existência ou não existência do atributo *id* no formulário.

Observando a figura 19, percebesse que um *if* está verificando se o atributo *id* está vazio, toda a inserção possui este *id* vazio, logo em seguida a variável *data* recebe todos os valores existentes preenchidos no formulário através de uma função chamada *serialize*.

Figura 19: Script do formulário de tipos de despesas

```
$('#form-Despesas').submit(function (e) {
    if ($('#id').val() == "") {
        var data = $(this).serialize();
        $('.form-group').each(function (index, el) {
            $(this).removeClass('has-error');
        });
        $('#formSuccess, #formError').fadeOut();
        $.ajax({
            url: '../../controlers/TipoDespesaControle.php?code=1',
            type: 'POST',
            dataType: 'json',
            timeout: 8000,
            data: data,
            beforeSend: function () {
                //console.log(data);
                $('#btn-cad-cliente').attr('disabled', 'disabled');
                $('#loading').css('display', 'inline-block');
            },
            complete: function (data) {
                // console.log("complete");
                $('#btn-cad-cliente').removeAttr('disabled');
                $('#loading').css('display', 'none');
            },
            success: function (data) {
                if (data.cod == 0) {
                    // console.log("sucesso");
                    $('#form-Despesas')[0].reset();
                    $('#formSuccess').delay(200).fadeIn('slow').html('<strong>' + data.msg + '</strong>')
                } else if (data.cod == 1) {
```

Fonte: do autor, 2016

Em seguida com ajax é realizada uma requisição ao controlador enviando o código de requisição code=1 e os dados do formulário por post, mais detalhes do controlador podem ser vistos na figura 20:

Figura 20: *Switch case* 1

```
switch ($code) {
    case 1:
        if (vazio($Desp->getCooperativa())) {
            echo json_encode(array(
                'cod' => '1',
                'field' => 'cooperativa',
                'msg' => 'A cooperativa deve ser preenchida!!'
            ));
            return;
        } else if ($status = $Desp->insertDespesa()) {
            $Auditoria->register("([COD" . $_SESSION['id'] . "] [NOME RESPONSAVEL:" . $_SESSION['name'] . "] "
                    . "Novo tipo de despesa criado por [" . $_SESSION['user'] . "]"
                    . "[Despesa criada: " . $Desp->getDescricaoDesp() . "])", "INSERT", "gerenciamento.txt");
            echo json_encode(array(
                'cod' => '0',
                'msg' => 'Tipo de despesa cadastrado com sucesso!'
            ));
        } else {
            echo json_encode(array(
                'cod' => '2',
                'msg' => 'Falha ao registrar novo Tipo de despesa!!'
            ));
        }
        break;
```

Fonte: do autor, 2016

O código de verificação passado para o controlador é o número 1, logo este cai no *case* 1 do *switch* que é responsável pelas inserções no sistema, basicamente o controlador está chamando o método insertDespesa, que pode ser visto na figura 21, e em seguida criando um registro de *log* no arquivo gerenciamento.txt, logo em seguida retorna um *array* em *json* contendo um código e uma mensagem, por questão de padronização no sistema, sempre que existe um retorno em *json* com *cod*=0, é por que a inserção foi realizada com sucesso e a mensagem é a mesma que o usuário vai visualizar.

Figura 21: Método insertDespesa

```
public function insertDespesa(){
    $status = $this->insert("INSERT INTO tipo_despesas (despesa_descricao,cooperativa)"
    . " VALUES (?,?) RETURNING id",
    array($this->descricaoDesp,$this->cooperativa));
    return $status;
}
```

Fonte: do autor, 2016

A figura 22 mostra como fica registrada a operação de *insert* que foi realizada, as informações contidas no arquivo são o tipo de operação que neste caso é o insert, o código da sessão, o usuário responsável pela alteração, uma mensagem que contém a informação gravada no banco e por fim o dia e hora que a operação foi realizada.

Figura 22: Arquivo gerenciamento.txt

```
Tipo: [UPDATE] Mensagem: ([COD2] [NOME RESPONSAVEL:gerente]  Horas trabalhadas atualizada por [gerente]
Tipo: [UPDATE] Mensagem: ([COD2] [NOME RESPONSAVEL:gerente]  Horas trabalhadas atualizada por [gerente]
Tipo: [DELETE] Mensagem: ([COD2] [NOME RESPONSAVEL:gerente] Salario deletado por [gerente][ ID [20] | V
Tipo: [INSERT] Mensagem: ([COD2] [NOME RESPONSAVEL:gerente] Salario criado por [gerente][Referente ao m
Tipo: [INSERT] Mensagem: ([COD2] [NOME RESPONSAVEL:gerente] Novo tipo de despesa criado por [gerente]
[Despesa criada: luz]) Data: Saturday 15th of October 2016 03:51:39 PM
```

Fonte: do autor, 2016

4.4.8 Atualização de dados

Para atualizar um registro é necessário selecionar a linha onde se encontra e registro e clicar no botão editar, como na figura 23:

Figura 23: Item selecionado.txt

Fonte: do autor, 2016

Ao clicar no botão editar, é utilizada uma funcionalidade do *datatables* que pode utilizar um campo desejado da tabela que neste caso é a primeira coluna do item selecionado.

Por questões de padronização, neste sistema todos os itens exibidos pelo *datatables* possuem na primeira coluna o *id* do registro para que essas manipulações utilizando um registro único possam ser efetivadas. Analisando a figura 24 pode-se observar a variável idDesp recebendo o valor da primeira coluna da linha selecionada e em seguida redirecionando para a página que contém o formulário, só que desta vez passando um *id* e o código de requisição 4 na variável *code*.

Figura 24: Método do botão editar

```
$('#btnEditar').click(function (e) {
    var idDesp = table.$('tr.selected').children(':first-child').text();
    var idCoop = ($('#idCoop').val());
    e.preventDefault();
    location.href = 'FormTipoDespesa.php?code=4&id=' + idDesp + '&idCoop=' + idCoop;
});
```

Fonte: do autor, 2016

Logo que o formulário carregar é verificado se o valor *id* é diferente de vazio, caso seja, é feita uma requisição ajax ao controlador passando o code 4, seguindo a padronização adotada no sistema, vai realizar uma busca em um registro especifico sobre o *id* passado anteriormente pelo *datatables*. Figura 25 da função que faz a requisição:

Figura 25: Requisição *ajax code* 4

```
$(document).ready(function () {
    var id = $('#id').val();
    if (id != "") {
        $.ajax({
            url: '../../controlers/TipoDespesaControle.php?code=4',
            type: 'POST',
            dataType: 'json',
            timeout: 8000,
            data: {id: id},
            beforeSend: function () {
                //console.log(data);
            },
            complete: function (data) {
                // console.log("complete");
            },
            success: function (data) {

                $.each(data, function (index, valor) {
                    $('#descDespesa').val(valor.despesa_descricao);
                });
            },
            error: function (xhr, er) {
                if (er == "timeout") {
                    console.log("Erro de timeout " + er);
                } else {
                    console.log("Erro: " + xhr + " " + er);
                }
            }
        });
```

Fonte: do autor, 2016

No controlador será chamado na estrutura do *switch* o *case* de valor 4, conforme figura 26, que vai selecionar o registro pelo *id* especificado e retornar um *json* contendo os dados selecionados.

Figura 26: *Switch case* 4

```
case 4:
    session_start();
    $Desp->setId($_REQUEST['id']);

    if ($status = $Desp->selectDespesaById()) {
        echo json_encode($status);
    } else {
        echo json_encode(array(
            'cod' => '2',
            'msg' => 'Falha ao carregar o formulário!!'
        ));
    }
    break;
```

Fonte: do autor, 2016

Observe ainda na figura 25 que contém a requisição *ajax*, quando os dados são retornados pelo controlador com sucesso o *ajax* chama a função *success* que vai preencher o campo no formulário com os dados recebidos do controlador como na figura 27:

Figura 27: Formulário para editar tipo de despesas

Fonte: do autor, 2016

Analisando agora a figura 27 pode-se observar que os campos são carregados com a tela e preenchidos automaticamente mostrando *id* e a descrição, em uma edição de outra funcionalidade com mais dados a única diferença seria uma quantidade maior de campos.

Após carregado os campos do item selecionado o processo interno para realizar a atualização do registro é semelhante a uma inserção, a diferença é que agora é passado para o controle o código de verificação 2, que foi padronizado como o código responsável pelas operações de *update* no sistema. Para melhor compreensão observa-se a figura 28 do controlador que após receber o código de verificação 2 cai no *case* 2 do *switch*:

Figura 28: *Switch case* 2

```
case 2:
    $Desp->setId(expressaoChars("numero", remEspaco($_REQUEST['id'])));
    if ($result = $Desp->selectDespesaById()) {
        if ($status = $Desp->updateDespesa()) {

            $Auditoria->register("([COD" . $_SESSION['id'] . "] [NOME RESPONSAVEL:" . $_SESSION['name'] . "] "
                    . " Tipo de despesa atualizado por [" . $_SESSION['user'] . "]"
                    . "[Tipo de despesa [ID: " . $result[0]->id . "]: " . $result[0]->despesa_descricao . " "
                    . "atualizado para: " . $Desp->getDescricaoDesp() . "])", "UPDATE", "gerenciamento.txt");
            echo json_encode(array(
                'cod' => '0',
                'msg' => 'Tipo despesa editado com sucesso!'
            ));
        } else {
            echo json_encode(array(
                'cod' => '2',
                'msg' => 'Falha ao atualizar Tipo despesa!!'
            ));
        }
        break;
    }echo json_encode(array(
        'cod' => '2',
        'msg' => 'Falha ao recuperar o tipo despesa!!'
    ));
    break;
```

Fonte: do autor, 2016

O processo de atualização consiste em uma seleção do registro antes do mesmo ser atualizado que é guardado em uma variável, para que o sistema possa efetivar a auditoria do registro, em seguida o método updateDespesa, conforme figura 29, grava os dados no banco de dados, após a gravação é realizado o registro desta atualização no arquivo de gerenciamento.txt e retorna o código 0 com uma mensagem de sucesso para o usuário.

Figura 29: updateDespesa

```
public function updateDespesa(){
        $sql    = "UPDATE tipo_despesas SET despesa_descricao = ? WHERE id = ?";
        $params = array($this->descricaoDesp,$this->id);
    $status = $this->update($sql, $params);
    return $status;
}
```

Fonte: do autor, 2016

4.4.9 Exclusão de dados

Para deletar um registro de um determinada funcionalidade, assim como para editar, é necessário selecionar o item, o *datatables* vai pegar o primeiro registro da linha selecionada na tabela que é o *id* do item e enviar para o controlador passando o código de verificação de número 3, a única diferença entre este método e o de edição é que uma caixa de diálogo vai pedir a confirmação do usuário antes de realizar a operação de delete e outra caixa de diálogo vai trazer a mensagem de sucesso ou de falha sobre a operação. Na figura 30 pode ser visto o método do botão deletar:

Figura 30: Script do botão deletar

```
$('#btnDeletar').click(function (e) {
    var id = table.$('tr.selected').children(':first-child').text();
    var descricao = table.$('tr.selected').children(':nth-child(2)').text();
    bootbox.confirm("Você tem certeza que deseja deletar a despesa " + descricao + "?", function (result)
        if (result) {
            $.ajax({
                url: '../../controlers/TipoDespesaControle.php?code=3',
                type: 'POST',
                dataType: 'json',
                timeout: 8000,
                data: {id: id},
                success: function (data) {
                    if (data.cod == 0) {
                        // console.log("sucesso");
                        $('#btnEditar, #btnDeletar').attr('disabled', 'disabled');
                        bootbox.dialog({
                            message: data.msg,
                            buttons: {
                                main: {
                                    label: "Ok!",
                                    className: "btn-primary",
                                    callback: function () {
                                        location.reload();
                                    }
                                }
                            }
                        });
```

Fonte: do autor, 2016

No controlador é passado o código de requisição 3, que vai primeiramente selecionar o registro antes de delatar o mesmo para poder realizar a auditoria no arquivo de gerenciamento.txt para em seguida chamar o método deletar despesas, após o item ter sido deletado do banco de dados é criado o registro sobre a operação e retorna o código 0 e a mensagem para o usuário conforme mostra a figura 31:

Figura 31: Controlador do deletar

```
case 3:
    session_start();
    $Desp->setId(expressaoChars("numero", remEspaco($_REQUEST['id'])));
    if ($result = $Desp->selectDespesaById()) {
        if ($status = $Desp->deleteDespesas()) {
            $Auditoria->register("([COD" . $_SESSION['id'] . "] [NOME RESPONSAVEL:" . $_SESSION['name'] . "] "
                    . "Tipo de despesas deletado por [" . $_SESSION['user'] . "]"
                    . "[Tipo despesa deletada: " . $result[0]->despesa_descricao . "])-ID[" . $Desp->getId() . "]",
                    "DELETE", "gerenciamento.txt");
            echo json_encode(array(
                'cod' => '0',
                'msg' => 'sucesso ao deletar tipo de cadastro'
            ));
        } else {
            echo json_encode(array(
                'cod' => '2',
                'msg' => 'Falha ao deletar tipo de cadastro!!'
            ));
        }
        break;
    } else {
        echo json_encode(array(
            'cod' => '2',
            'msg' => 'Falha ao carregar descricao do tipo de despesa para a auditoria!'
        ));
        break;
    }
```

Fonte: do autor, 2016

4.4.10 Emissão de Relatórios

Foram criados 4 tipos relatórios no sistema utilizando a classe FPDF, 3 deles específicos e 1 geral. Um relatório de recibo de uma determinada venda, um relatório que mostra todos os recicladores cadastrados na cooperativa, um relatório que mostra os dados de um reciclador especifico e o relatório principal que mostra a receita, despesa, vendas, materiais vendidos e salários dos recicladores da cooperativa, ou seja, um relatório mensal contendo as informações gerais do sistema que é chamado de relatório de rateio.

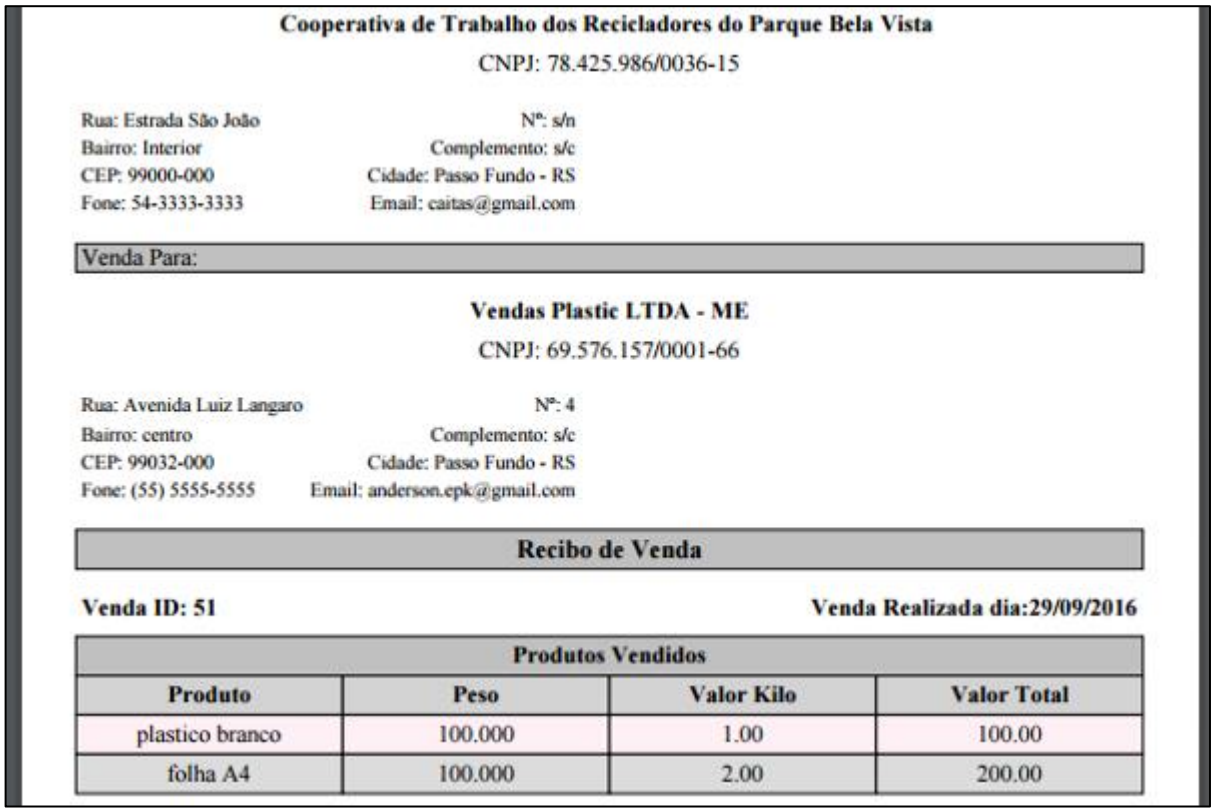
Os relatórios específicos de recibo de venda, e os outros dois de recicladores podem ser emitidos através de seus respectivos menus, já para o relatório de rateio foi criado uma tela especifica onde é possível fazer emissão das informações passando um período de tempo entre duas datas conforme mostra a figura 32:

Figura 32: Tela para emissão do relatório geral

Fonte: do autor, 2016

O relatório de recibos, conforme figura 33, mostra os dados de uma determinada venda que pode ser emitido após a finalização da venda e entregue ao cliente, ressalva-se que todos os dados contidos no relatório são fictícios e ilustrativos.

Figura 33: Relatório de recibo



Cooperativa de Trabalho dos Recicladores do Parque Bela Vista

CNPJ: 78.425.986/0036-15

Rua: Estrada São João — Nº: s/n
Bairro: Interior — Complemento: s/c
CEP: 99000-000 — Cidade: Passo Fundo - RS
Fone: 54-3333-3333 — Email: caitas@gmail.com

Venda Para:

Vendas Plastic LTDA - ME

CNPJ: 69.576.157/0001-66

Rua: Avenida Luiz Langaro — Nº: 4
Bairro: centro — Complemento: s/c
CEP: 99032-000 — Cidade: Passo Fundo - RS
Fone: (55) 5555-5555 — Email: anderson.epk@gmail.com

Recibo de Venda

Venda ID: 51 — **Venda Realizada dia:29/09/2016**

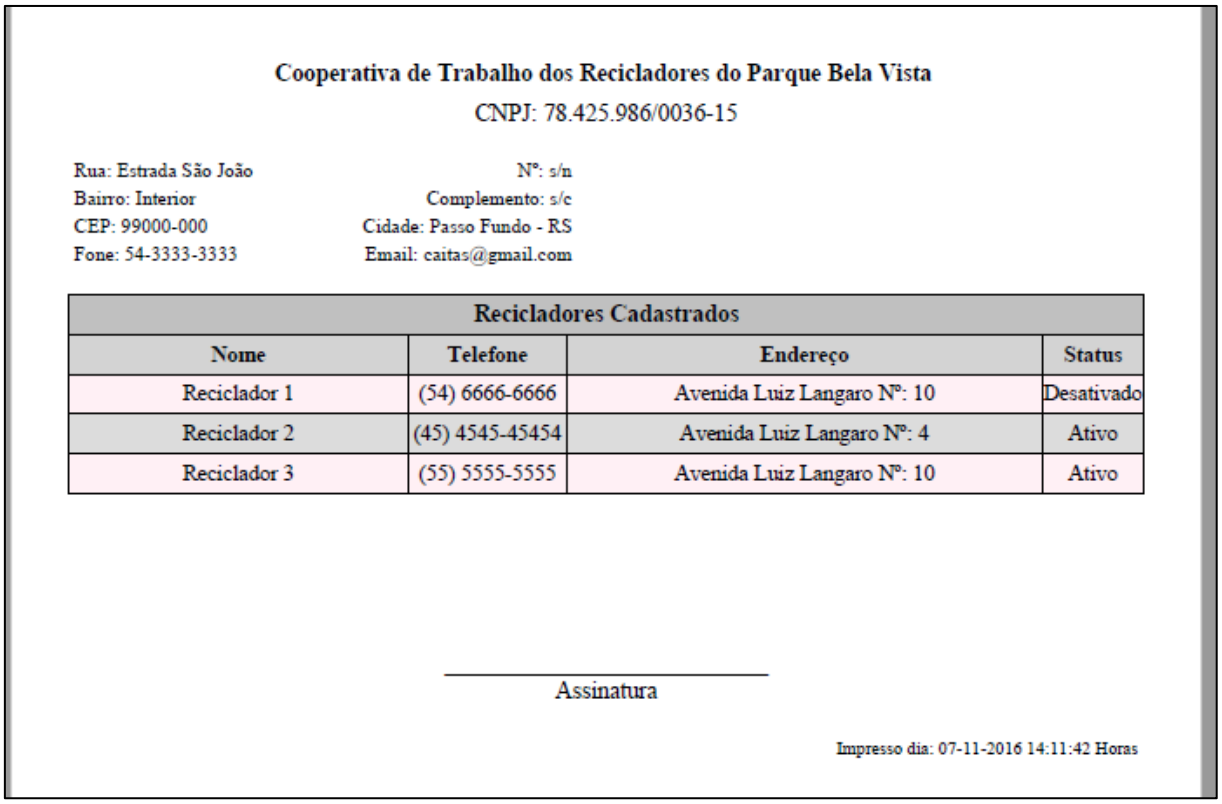
Produtos Vendidos

Produto	Peso	Valor Kilo	Valor Total
plastico branco	100.000	1.00	100.00
folha A4	100.000	2.00	200.00

Fonte: do autor, 2016

É possível gerar um relatório que contém os principais dados dos todos os recicladores de uma determinada cooperativa conforme a figura 34, ressalva-se que todos os dados contidos no relatório são fictícios e ilustrativos.

Figura 34: Relatório de todos os recicladores



Cooperativa de Trabalho dos Recicladores do Parque Bela Vista

CNPJ: 78.425.986/0036-15

Rua: Estrada São João — Nº: s/n
Bairro: Interior — Complemento: s/c
CEP: 99000-000 — Cidade: Passo Fundo - RS
Fone: 54-3333-3333 — Email: caitas@gmail.com

Recicladores Cadastrados			
Nome	**Telefone**	**Endereço**	**Status**
Reciclador 1	(54) 6666-6666	Avenida Luiz Langaro Nº: 10	Desativado
Reciclador 2	(45) 4545-45454	Avenida Luiz Langaro Nº: 4	Ativo
Reciclador 3	(55) 5555-5555	Avenida Luiz Langaro Nº: 10	Ativo

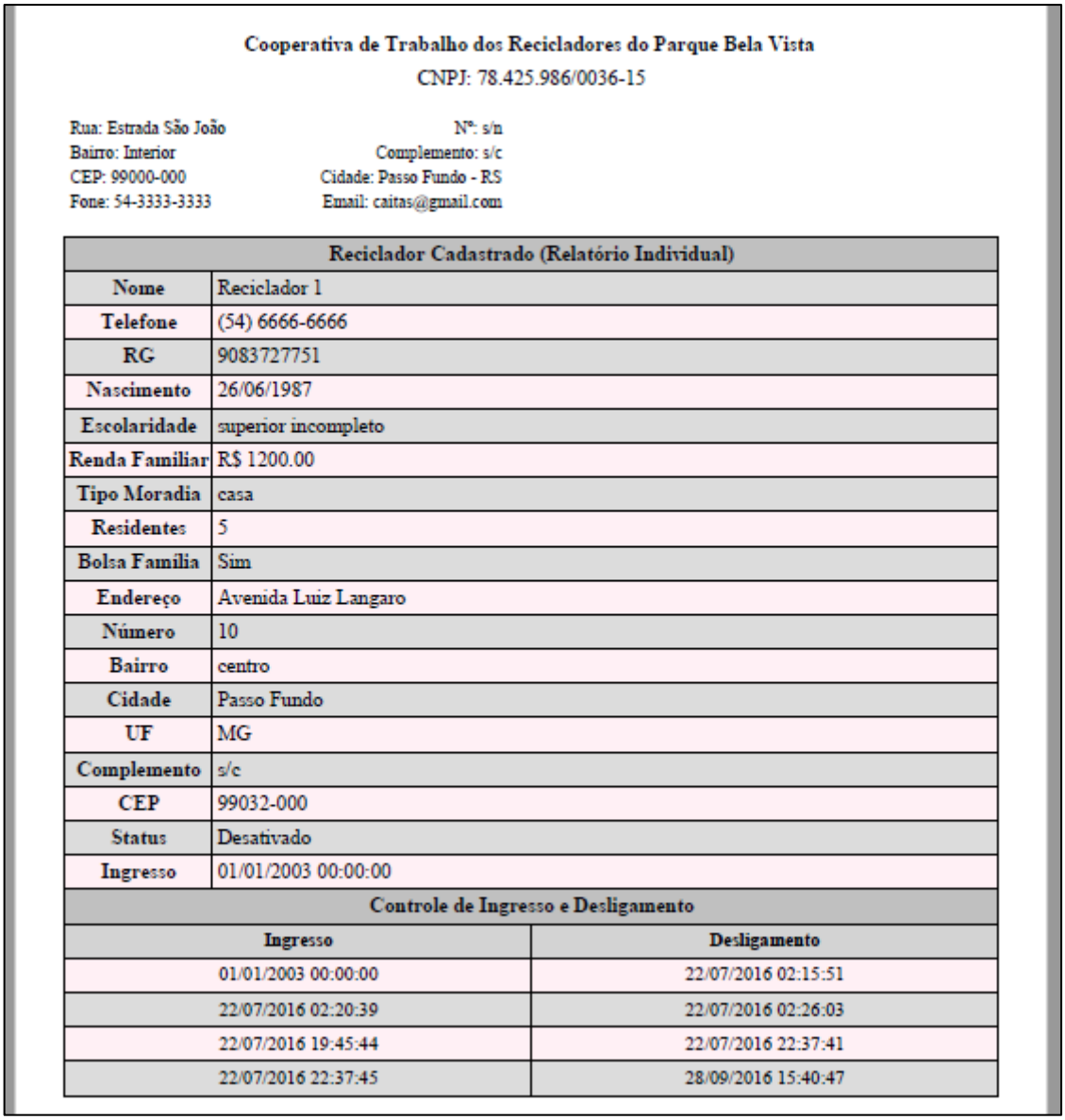
Assinatura

Impresso dia: 07-11-2016 14:11:42 Horas

Fonte: do autor, 2016

Foi implantado um relatório que mostra todos os dados de um reciclador em especifico além das datas relativas ao ingresso e desligamento do reciclador como na figura 35:

Figura 35: Relatório de um reciclador especifico



Cooperativa de Trabalho dos Recicladores do Parque Bela Vista

CNPJ: 78.425.986/0036-15

Rua: Estrada São João — Nº: s/n
Bairro: Interior — Complemento: s/c
CEP: 99000-000 — Cidade: Passo Fundo - RS
Fone: 54-3333-3333 — Email: caitas@gmail.com

Reciclador Cadastrado (Relatório Individual)	
Nome	Reciclador 1
Telefone	(54) 6666-6666
RG	9083727751
Nascimento	26/06/1987
Escolaridade	superior incompleto
Renda Familiar	R$ 1200.00
Tipo Moradia	casa
Residentes	5
Bolsa Familia	Sim
Endereço	Avenida Luiz Langaro
Número	10
Bairro	centro
Cidade	Passo Fundo
UF	MG
Complemento	s/c
CEP	99032-000
Status	Desativado
Ingresso	01/01/2003 00:00:00

Controle de Ingresso e Desligamento	
Ingresso	Desligamento
01/01/2003 00:00:00	22/07/2016 02:15:51
22/07/2016 02:20:39	22/07/2016 02:26:03
22/07/2016 19:45:44	22/07/2016 22:37:41
22/07/2016 22:37:45	28/09/2016 15:40:47

Fonte: do autor, 2016

O relatório de que contém um balanço do mês, chamado pelos gestores da cooperativa de relatório de rateio, que contem grande parte dos dados gerados pelo sistema, entre eles faturamento mensal da cooperativa, salários dos recicladores, materiais vendidos e despesas mensais conforme mostra figura 36:

Figura 36: Relatório de rateio

Cooperativa de Trabalho dos Recicladores do Parque Bela Vista

CNPJ: 78.425.986/0036-15

Rua: Estrada São João — Nº: s/n
Bairro: Interior — Complemento: s/c
CEP: 99000-000 — Cidade: Passo Fundo - RS
Fone: 54-3333-3333 — Email: caitas@gmail.com

Relatório de Rateio (Setembro/2016)	
Receita	R$ 4,225.00
Despesa	R$ 1,110.00
Valor Líquido	R$ 3,115.00
Valor Hora Trabalhada	R$ 10.74
Renda Média	R$ 651.80
Número de Associados	3

Setembro/2016					
Nome dos Associados	**Nº de Horas**	**Individual**	**INSS**	**Cota Parte**	**Líquido (Vale)**
Reciclador 1	80 Hs	R$ 859.20	R$ 50.00	R$ 50.00	R$ 759.20 (100.00)
Reciclador 2	80 Hs	R$ 859.20	R$ 50.00	R$ 50.00	R$ 759.20 (100.00)
Reciclador 3	50 Hs	R$ 537.00	R$ 50.00	R$ 50.00	R$ 437.00 (0.00)
Fundo de Investimento	80 Hs	R$ 859.20			R$ 859.20
Total	**290 Hs**		**R$ 150.00**	**R$ 150.00**	**R$ 2,814.60**

Pesagem / Vendas = Receitas			
Tipo Material	**Peso**	**Preço(Kg)**	**Total**
garrafa de vidro	50.000	R$ 1.50	R$ 75.00
folha A4	100.000	R$ 1.50	R$ 150.00
garrafas pet verde	100.000	R$ 20.00	R$ 2,000.00
plastico branco	500.000	R$ 4.00	R$ 2,000.00
Total	**750 Kg**		**R$ 4,225.00**

Gastos / Despesas		
Descrição	**Data**	**Valor**
luz	30/09/2016	R$ 1,000.00
telefone	30/09/2016	R$ 50.00
agua	30/09/2016	R$ 60.00
Total		**R$ 1,110.00**

Fonte: do autor, 2016

O cálculo realizado para se obter o salário de um reciclador consiste nos seguintes passos: primeiramente se obtém a receita liquida da cooperativa, subtraindo a receita bruta das despesas, para obter o valor da hora trabalhada se divide a receita bruta pelo número de horas trabalhadas de todos os recicladores, em seguida para obter o valor individual se multiplica o número de horas trabalhadas do reciclador pelo valor da hora trabalhada.

Observa-se ainda que existe um fundo de investimento na cooperativa, que é o salário do reciclador que obteve maior renda do mês.

4.5 Implantação

Primeiramente foi realizada uma visita ao Cáritas [1], órgão responsável pelas cooperativas de Passo Fundo, onde foi apresentado o sistema e sua funcionalidades. Neste estágio algumas alterações essências foram requisitadas, como a inclusão de relatórios dos recicladores individual e em grupo por cooperativas, mas a principal delas foi a emissão do relatório de rateio, que contém quase todas as informações do sistema que é emitido mensalmente e utilizado para prestar contas ao município.

Após as alterações terem sido realizadas, foi instalado o sistema localmente em um computador do Cáritas, um funcionário foi treinado e instruído de como utilizar o sistema. No treinamento foram executadas todas as funcionalidades do sistema pelo funcionário, acompanhado pelo desenvolvedor do sistema. Não houveram dificuldades por parte do funcionário para entender o funcionamento do sistema que conseguiu utilizar de todas as funcionalidades do sistema logo nas primeiras tentativas.

4.6 *Feedback*

Após a utilização do sistema por parte dos responsáveis pelas cooperativas, foi realizado um questionário com 10 questões, este o qual foi enviado ao gestor das cooperativas via e-mail com um link para o formulário criado com a ferramenta da Google chamada Google Form, as questões se encontram no anexo 1.

Foi questionado se as funcionalidades que estavam presentes satisfaziam as necessidades de gestão de cooperativas, a resposta foi positiva e que as funcionalidades estavam em acordo com as suas atividades. Questões de usabilidade também foram realizadas como dificuldades de utilizar o sistema, se o layout estava agradável ou ainda se houve travamentos ou erros enquanto o sistema era utilizado, o gestor gostou da aparência dos layouts e ressaltou que não teve muitas dificuldades para utilizar o sistema e que não ocorreram erros e a velocidade de resposta do sistema se encontrava normal.

Também foi questionado a satisfação do gestor enquanto utilizava o sistema e se os relatórios gerados emitidos estavam de acordo, a resposta foi positiva, mas que ainda eram necessários alguns ajustes nos relatórios além da ressalva onde o gestor afirma que o sistema pode trazer ganhos em sua gestão.

[1] A Cáritas é uma organização da Igreja Católica, que abrange vários níveis: comunitários, paroquiais, (arqui) diocesanos, regionais, nacionais, continentais e, até mesmo, internacionais. Sua principal missão é estar presente na comunidade e, a partir da realidade onde está inserida, buscar respostas às suas necessidades. (Caritaspf, 2016)

Analisando o feedback é possível concluir, de acordo com o gestor que utilizou o sistema, que o software atingiu um patamar satisfatório, destacando ainda a vontade do gestor de utilizar o sistema nas suas atividades.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo desenvolver um sistema de gerenciamento de acordo com as necessidades das cooperativas de reciclagem de resíduos sólidos de Passo Fundo. O sistema foi desenvolvido sobre uma base solida de levantamento de requisitos além de uma modelagem inicial que buscou suprir todas as necessidades de gestão que uma cooperativa poderia vir a ter.

Importante também ressaltar os recursos trabalhando em conjunto levaram ao desenvolvimento dentro do tempo previsto e que com o feedback da gestão pode-se concluir que os objetivos foram alcançados.

Além das funcionalidades do sistema, todas podem ser auditadas posteriormente devido processo de criação de *logs* existe no sistema, nenhuma operação que afeta o banco de dados acontece sem que esta seja registrada em um arquivo de log contendo data, hora, usuário, *ip* e as informações alteradas, trazendo aos gestores uma segurança extra aos seus dados, cujo qual foi um ponto importante sinalizado por eles. A questão de segurança também está presente no sistema, já que toda a estrutura que faz a conexão com o bando de dados foi elaborada sobre a base sólida do PDO, oferecendo segurança extra em relação ao *sql injection*.

Ainda foi implantado o sistema para que os gestores pudessem utilizá-lo, em um primeiro momento foi feita uma apresentação do sistema e em seguida foi realizada a implantação e treinamento para que estes pudessem utilizar e avaliar o sistema em operação. O feedback obtido sobre a da utilização do sistema foi positivo em vários aspectos como funcionalidade e usabilidade, destaca-se o interesse dos gestores em continuar a utilizar o sistema em suas atividades diárias.

O sistema possui boa manutenibilidade devido ao processo MVC e a reutilização de código, pois a lógica central do sistema é simples de entender permitindo futuramente adicionar novas funcionalidades sem que estas afetem qualquer parte do sistema atual, já que cada classe é separada e organizada, cada uma com a sua funcionalidade bem definida e clara.

Tendo em vista as necessidades de gestão das cooperativas e a vontade dos gestores em utilizar o sistema de forma efetiva, pode-se pensar como trabalhos futuros na implantação do sistema na WEB, em adquiri um servidor WEB próprio e configurá-lo ou um serviço de hospedagem profissional, deste modo o sistema seria utilizado em um ambiente real e ainda, se necessário, realizar uma futura expansão do sistema para atender mais funcionalidades e necessidades dos gestores das cooperativas.

6 REFERÊNCIAS

____. Lei nº 12.305 de 02 de agosto 2010. Institui a Política Nacional de Resíduos Sólidos. Presidência da República. Brasília, DF, 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT, NBR 8419: Apresentação De Projetos De Aterros Sanitários De Resíduos Sólidos Urbanos. Rio de Janeiro: 1992.

BRASIL, Constituição (1988). Constituição da República Federativa do Brasil. Brasília, DF: Senado, 1988.

CARITAS PF – Disponível em: http://caritaspf.com.br/quem_somos.html. Acesso em: 15 julho. 2016.

DATATABLES – Datatables. Disponível em: https://datatables.net/. Acesso em: 29 maio. 2016.

ENGHOLM, H. Jr. Softwares Por Que Usar, Novatec editora, 2012

FPDF: FPDF Library: Disponível em: http://www.fpdf.org/. Acesso em: 29 maio. 2016.

GUEDES, G. T. A. UML 2: Uma Abordagem Prática. 2ª ed. São Paulo: Novatec editora, 2011.

GUEDES, G. T. A. 2:Guia Prático 2ª ed. São Paulo: Novatec editora, 2014.

HTML – The World's Largest Web Developer Site (W3C). Disponível em: http://www.w3schools.com/html/. Acesso em: 29 maio. 2016.

KALIL, R. Redes de Infraestrutura Urbana: Gestão De Programas e Projetos Na Cidade De Passo Fundo de 2005 a 2007. 2008. 185 f. Dissertação (Mestrado em Engenharia) - Universidade de Passo Fundo - Passo Fundo, 2008.

LEFFINGWELL, D.; WIDRIG, D. Managing Software Requirements: A Unified Approach. Addison-Wesley, 2000.

MILANI, André. PostgreSQL-Guia do Programador. Novatec Editora, 2008.

MILANI, André. Programando para iPhone e iPad: Aprenda a Construir Aplicativos Para o IOS. Novatec Editora, 2012.

OLIVEIRA, D. P. R. Sistemas de Informações Gerenciais: Estratégicas Táticas Operacionais. 12ª Ed. São Paulo: Atlas, 2008.

OLIVEIRA, F. de O. Sistemas de Informação: Um Enfoque Gerencial Inserido No Contexto Empresarial e Tecnológico. 3ª ed. São Paulo: Érica, 2002.

PHP – Prefácio. Disponível em: http://php.net/manual/pt_BR/preface.php/. Acesso em: 29 maio. 2016.

Postgresql – About. Disponível em: https://www.postgresql.org/about/. Acesso em: 29 maio. 2016.

PINHO, P. M. Avaliação Dos Planos Municipais De gestão Integrada De Resíduos Sólidos Urbanos Na Amazônia Brasileira. 2011. 249 f. Tese (Doutorado em Ciência Ambiental). Universidade de São Paulo. São Paulo, 2011.

ROLLWAGEN, F. A. Protótipo De Um Sistema De Informação Para gestão Pública Integrada de Resíduos. Passo Fundo.2013.

SB ADMIN 2 – Start Bootstrap. Disponível em: https://startbootstrap.com/template-overviews/sb-admin-2/. Acesso em: 16 setembro. 2016.

SIEBRA, A. S. Banco de Dados. Recife, 2010. Disponível em:<http://jcinfoweb.xpg.uol.com.br/site/banco/livro_banco_dados.pdf>. Acesso em: 29 maio. 2016.

SILVA, M. S. JavaScript Guia do Programador. 2010. São Paulo: Novatec Editora, 2010.

SILVA, M. S. CSS3: Desenvolva Aplicações Web Profissionais Com Uso Dos Poderosos Recursos de Estilização das CSS3 / Maurício Samy Silva; [tradução Rafael Zanolli]. -- São Paulo: Novatec Editora, 2012.

SILVA, M. S. Bootstrap 3.3.5 - Aprenda a Usar o Framework Bootstrap para Criar Layouts CSS Complexos e Responsivos São Paulo: Novatec Editora, 2015.

SOMMERVILLE, I. Engenharia de Software. 9º ed. São Paulo: Pearson Prentice Hall, 2011.

STAIR, R. M.; REYNOLDS, G. W. Princípios de Sistemas de Informações. 9º ed. São Paulo: Cengage Learning, 2011

TAGUCHI, R. L. Gestão Integrada de Resíduos Sólidos Urbanos Domiciliares Com Uso Do Balanced Scorecard. 2010. 178 f. Dissertação (Mestrado em Ciências). Universidade de São Paulo. Ribeirão Preto, 2010;

WORSLEY, John; DRAKE, Joshua D. Practical PostgreSQL. "O'Reilly Media, Inc.", 2002.

ANEXOS

ANEXO 1: Questionário enviado para os gestores

As respostas não podem ser editadas

Questionário Sistema de gestão de Cooperativas

Responda as questões desse formulário de acordo com o seu uso do sistema de gestão de Cooperativas.

O sistema satisfaz as necessidades de gestão das cooperativas?

- (●) Sim
- ( ) Não
- ( ) Satisfaz Parcialmente

Na sua opinião, as telas do sistema estão visualmente agradáveis?

- (●) Sim
- ( ) Não
- ( ) Sim, mas mudaria alguns detalhes

Qual foi o seu grau de dificuldade para utilizar o sistema?

- ( ) Facil
- (●) Médio
- ( ) Dificil

Na sua opinião, você achou o sistema rápido?

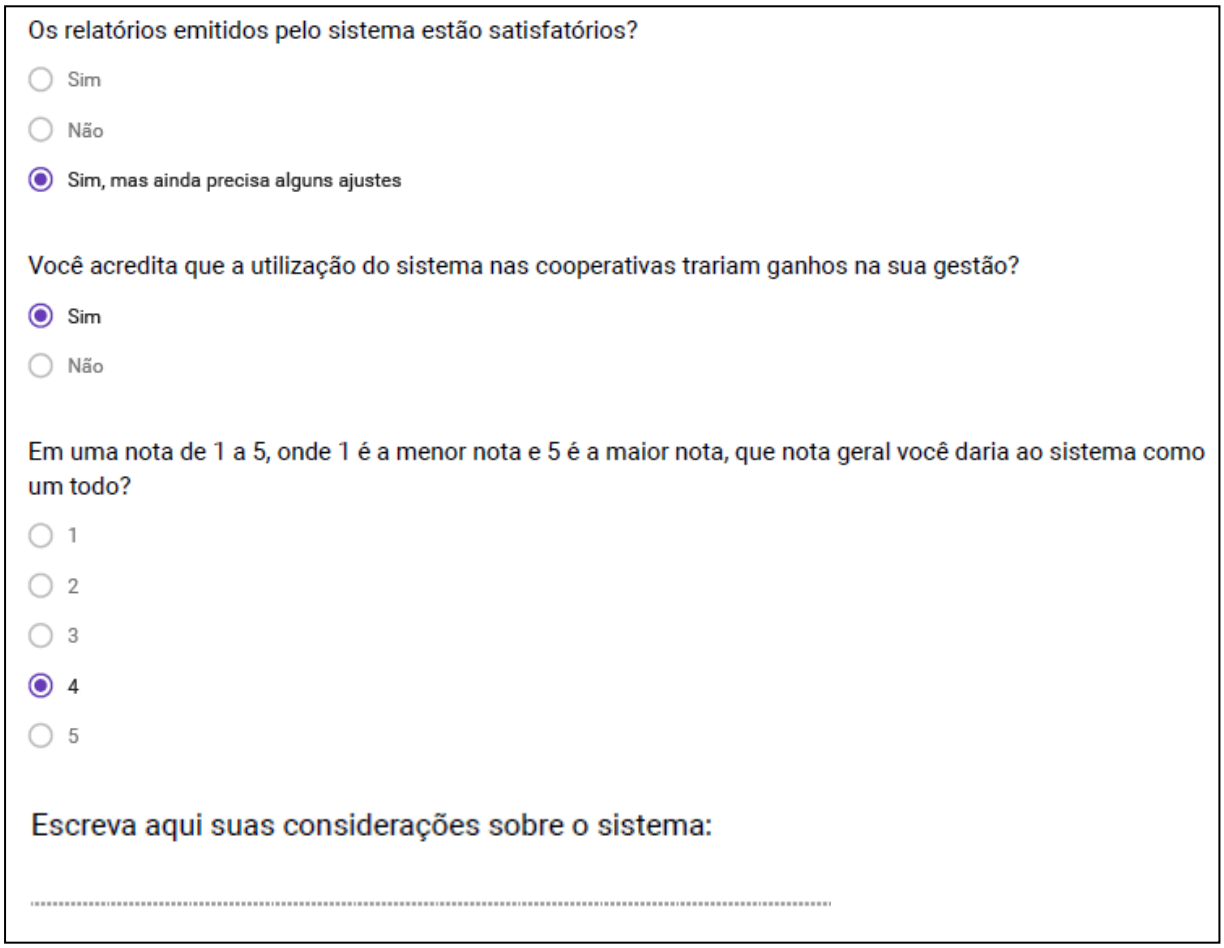
- ( ) Sim
- ( ) Não
- (●) Normal

O sistema travou ou se comportou de forma estranha durante a sua utilização?

- ( ) Sim
- (●) Não

Você gostaria de continuar utilizando o sistema para as suas atividades de gestão das cooperativas?

- (●) Sim
- ( ) Não



Fonte: do autor, 2016