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

RICARDO BERTÉ

APLICATIVO WEB PARA CONTROLE DE AGENDA DE PROFISSIONAIS DA ÁREA DE SOFTWARE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2014

RICARDO BERTÉ

APLICATIVO WEB PARA CONTROLE DE AGENDA DE PROFISSIONAIS DA ÁREA DE SOFTWARE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2014

ATA Nº: **242**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **RICARDO BERTE.**

Às 20:05 hrs do dia 13 de agosto de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Soelaine Rodrigues Ascari (Convidada) e Andréia Scariot Beulke (Convidada), para avaliar o Trabalho de Diplomação do aluno Ricardo Berte, matrícula 993735, sob o título **Aplicativo Web para Controle de Agenda de Profissionais da Área de Software**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 21:20 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Soelaine Rodrigues Ascari, M.Sc.
Convidada

Profa. Andréia Scariot Beulke, Esp.
Convidada

Profa. Eliane Maria de Bortoli Fávero, M.Sc
Coordenadora de Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

BERTÉ, Ricardo. Aplicativo web para controle de agenda de profissionais da área de software. 2014. 54 f. Trabalho de conclusão de curso (graduação) do curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2014.

O processo de desenvolvimento de *software*, especialmente o que ocorre sob demanda, pode depender de muitas visitas ao cliente e/ou usuários. As entrevistas para a coleta de dados para definir os requisitos do ponto de vista do usuário, a coleta de modelos utilizados pelo sistema atual, quer seja manual ou automatizado, o entendimento de processos e regras de negócio, a personalização de funcionalidades e a manutenção podem ser dependentes de visitas *in loco* no ambiente de produção do cliente. Um sistema para o controle de agendamentos e de realização das visitas é útil tanto para os analistas e os outros profissionais que realizam as visitas quanto para a gerência da empresa de desenvolvimento de *software*. Esse sistema auxilia os profissionais que fazem visita aos clientes a organizar as suas atividades de maneira a otimizar o tempo e realizar o registro de horas para as métricas da empresa e pagamento do cliente. Assim, os gestores têm condições de saber efetivamente o tempo gasto com cada cliente e/ou projeto em atividades fora da empresa. A implementação de um aplicativo *web* para o controle desses agendamentos é o resultado da realização deste trabalho de conclusão de curso. Optou-se por um sistema *web* pela facilidade de acesso para aos seus usuários que podem estar na empresa ou em visitas a clientes.

Palavras-chave: Controle de agenda. PHP. Aplicativo web.

ABSTRACT

BERTÉ, Ricardo. Web application to control schedule of software development team. 2014. 55 f. Trabalho de conclusão de curso (graduação) do curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2014.

The process of software development, especially what happens on demand, can depend on many visits to the client and/or users. Interviews to collect data to define requirements from the point of view of the user, collection of models used by the current system, whether manual or automated, understanding of processes and business rules, customizing functionality and maintenance can be dependent on visits in loco in the production environment of the client. A system for controlling scheduling and carry out the visits is useful for analysts and other professionals who carry out visits and for company management software development. This system helps the professionals who visit clients to organize their activities so as to optimize the time and perform the registration of hours to the metrics of the company and client payment. Therefore, managers are able to learn effectively the time spent with each client and/or project activities outside the company. The implementation of a web application to control these schedules is the result of this term paper. It was chosen a web system for easy access for users who may be in the company or client visits.

Keywords: Schedule control. PHP. Web application.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE ABREVIATURAS E SIGLAS

DDI	Discagem Direta Internacional
CEP	Código de Endereçamento Postal
CGI	*Common Getway Interface*
CNPJ	Cadastro Nacional de Pessoa Jurídica
CPF	Cadastro de Pessoas Físicas
JSON	*JavaScript Object Notation*
HTML	*HyperText Markup Language*
MVC	*Model-View-Controller*
PHP	*PHP Hypertext Preprocessor*
RG	Registro Geral
RIA	*Rich Internet Application*
TI	Tecnologia de Informação
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais do trabalho, os seus objetivos e a justificativa.

1.1 CONSIDERAÇÕES INICIAIS

A análise é uma das etapas, fases, atividades ou mesmo processos (fluxos de trabalho como definidos pelo Processo Unificado (BLAHA et al., 2006)), do ciclo de vida de *software*. As atividades relacionadas à análise podem incluir do levantamento (elicitação) dos requisitos do ponto de vista do usuário, à definição do problema e à modelagem da solução desse problema. Essa solução será implementada gerando o aplicativo ou sistema de *software*. Isso ocorre tanto para sistemas a serem implementados como para funcionalidades desenvolvidas como forma de personalização ou manutenção de sistemas para clientes específicos.

Um dos profissionais envolvidos na realização das atividades de ciclo de vida de *software* é o analista de sistemas. Analista é o profissional encarregado de sistematizar informações por meio do estudo de processos computacionais com o objetivo de encontrar a melhor e mais racional forma de processar a informação. O analista de sistemas desenvolve soluções que serão aplicadas pelo computador, baseadas nas conexões existentes entre o usuário, o programa e o equipamento (PACIEVITCH, 2013).

Uma das atividades iniciais da Engenharia de Requisitos é a elicitação dos requisitos (MAGELA, 2006). É comum que nessa atividade de levantamento ou descoberta de requisitos seja necessário realizar visitas a clientes e usuários do sistema. Essas visitas podem demandar um tempo considerável e, muitas vezes, é necessário deslocar-se para outras cidades e/ou estados para atendê-los. Assim, é importante que os analistas de uma empresa possuam um controle de agenda para que possam organizar-se para as visitas, do tempo necessário para realizá-las e coleta de dados para métricas e pagamento do cliente. Outras fases do ciclo de vida também podem demandar uma intensidade maior de visitas ao cliente com as fases de testes, de implantação, de transição entre o sistema atual e o novo e manutenção.

Considerando a necessidade de gerenciamento de visitas como exposto, foi realizada a implementação de um aplicativo para gerenciamento de agendas de profissionais da área de

software. O sistema foi desenvolvido para ambiente *web* por permitir acesso fácil. , pois o sistema pode ser executado por meio de um navegador e a qualquer hora e computador.

1.2 OBJETIVOS

A seguir estão o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

Implementar um aplicativo *web* para gerenciamento de agenda de profissionais da área de *software*.

1.2.2 Objetivos Específicos

- Possibilitar o controle de atividades e visitas para a direção da empresa e também para os profissionais que realizam visitas a clientes.
- Permitir o controle de horas dispensadas pelos profissionais de TI nas atividades desenvolvidas em visitas a clientes.
- Fornecer uma ferramenta para controlar a quantidade de horas cobráveis que cada profissional desenvolveu em determinado período de tempo e em cada projeto.

1.3 JUSTIFICATIVA

A justificativa de implementação de um aplicativo para gerenciamento de agenda de visitas a clientes tem como fundamentação a necessidade de empresas que desenvolvem vários projetos de *software* simultaneamente ou que possuem equipes com funções bem definidas. É importante que os profissionais dessas equipes ou empresas possam contar com o auxílio de um aplicativo que gerencie a agenda de visitas. Assim, é possível otimizar o tempo desses profissionais e facilitar o o setor de recursos humanos da empresa na realização da cobrança dos seus clientes. Assim, eles podem saber para quando a visita está agendada, o

objetivo e o tempo estimado, dentre outros. Podendo programar-se para as outras atividades que eles realizam na empresa.

Considerando que uma das funcionalidades essenciais do sistema é o controle de agenda de visitas a clientes e usuários do sistema, é importante que o sistema seja para *web*. Um sistema *web* facilita o acesso permitindo aos profissionais fazer registro do atendimento no próprio cliente. O sistema também auxiliará a registrar as horas utilizadas. As horas utilizadas são as horas despendidas no atendimento ao cliente. Esse atendimento é definido como agenda. Tanto o analista quanto os gerentes e gestores da empresa possuem acesso aos dados que podem ser utilizados em métricas de qualidade, cálculos de metas e de adicionais que são pagos aos profissionais por horas cobráveis realizadas.

O aplicativo implementado como resultado deste trabalho tem como base as necessidades e interesses de uma empresa específica, mas se aplicada para qualquer empresa de desenvolvimento de *software* nas quais os profissionais realizam visitas a clientes.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. Este é o primeiro e apresenta a introdução do trabalho. O Capítulo 2 contém o referencial teórico que está centrado desenvolvimento de aplicativos para *web*. No Capítulo 3 estão os materiais utilizados para modelagem e implementação do aplicativo e o método que contém as principais atividades realizadas para a modelagem e a implementação do sistema. O Capítulo 4 apresenta o sistema e exemplos dos códigos desenvolvidos. Por fim, no Capítulo 5, está a conclusão com as considerações finais.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho e está centrado em desenvolvimento de aplicações para *web* utilizando o padrão de arquitetura MVC (*Model-View-Controller*) e a linguagem PHP (*PHP Hypertext Preprocessor*).

2.1 DESENVOLVIMENTPO PARA WEB

O modelo cliente/servidor, sendo o computador servidor visto como o provedor da aplicação e o computador cliente como o usuário dessa aplicação, especialmente em aplicações para *web* trouxe diversas vantagens e facilidades. Dentre as quais estão a facilidade de manutenção pela instalação do aplicativo no servidor, a possibilidade de um cliente com poucos recursos e o uso de uma linguagem de marcação amplamente usada para compor a interface, a *HyperText Markup Language* (HTML).

Apesar das vantagens significativas, esse modelo pode exigir necessidade de grande tráfego de dados entre cliente e servidor. E em redes com baixa largura de banda ou com muitos clientes acessando o mesmo servidor, o tráfego de dados pode tornar-se lento para as necessidades dos usuários. Em relação à HTML, os recursos de interação que a mesma oferece tem se tornado limitados diante das necessidades e interesses dos usuários.

Uma forma de minimizar esses problemas é por meio do desenvolvimento de aplicações para *web* com interface rica, as denominadas *Rich Internet Application* (RIA) e o com o uso de recursos como *Asynchronous Javascript and XML* (AJAX). As RIAs se caracterizam pelos recursos de interatividade de interface que oferecem. Linguagens como PHP, por exemplo, possibilitam o desenvolvimento utilizando Ajax que oferece a possibilidade de processamento no cliente e comunicação assíncrona entre cliente e servidor. Minimizando, assim, o tráfego de rede, embora possa requer cliente com capacidade de processamento maior do que nas aplicações *web* tradicionais. Tradicionais se referem às páginas de hipertexto vinculadas por meio de *hiperlinks* e com formulários com componentes muito simples, baseados em *Common Getway Interface* (CGI) (CHO et al., 1997). Essas aplicações são desenvolvidas utilizando basicamente os recursos de HTML que oferece interface bastante limitada em termos de recursos de interação.

Os *scripts* definidos em linguagem PHP podem ser mesclados com o código HTML que define a interface de interação com o sistema. Existem diversas maneiras de separar os elementos de um programa visando facilitar a implementação, a manutenção, o reuso e mesmo o entendimento do código implementado, dentre outros. Uma dessas maneiras é pelo uso de conceitos de arquitetura de software e pelo uso de padrões de projetos. *Model-View-Controller* é um padrão de projetos que tem o objetivo de auxiliar na organização do código produzido, separando-o em elementos com finalidades ou funcionalidades semelhantes.

2.2 *MODEL-VIEW-CONTROLLER*

A arquitetura de um *software* pode ser vista como um conjunto de elementos para disponibilizar dados e controle em sistemas de software (BASS, CLEMENTS, KAZMAN, 2006). Esses elementos estão vinculados por conectores compondo módulos de software. Os conectores provêem um canal para vincular os componentes entre si. As conexões desempenham um papel significativo em sistemas distribuídos (MCHEICK, QI, 2011). Esses conectores podem ser desenvolvidos utilizando o *Model-View-Controller*.

A arquitetura MVC divide um sistema em três tipos de módulos (MCHEICK; QI, 2011; BURBECK, 1997):

a) Controle (*controller*) – o controle fica encarregado das entradas do usuário incluindo os eventos de mouse e teclado e de notificar o modelo por meio de eventos. O controle recebe entradas e inicializa a resposta por meio de chamadas a objetos do modelo. Um controle aceita entradas de um usuário e instrui o modelo a realizar ações baseadas nas entradas. O controle é implementado como um componente separado ou combinado com a visão.

b) Modelo (*model*) – o modelo lida com a lógica de negócio (o comportamento) e os dados da aplicação e é responsável pela atualização de informações na visão e por receber comandos do controle. Além disso, o modelo responde às requisições por informações sobre seu estados (requisições geralmente vindas da visão) e às instruções para mudar o estados (geralmente vindas do controle). Em sistemas orientados a eventos, o modelo notifica observadores (geralmente visões) quando a informação muda e assim eles podem atuar.

c) Visão (*view*) – a visão é responsável pela apresentação. É importante que a visão esteja separada da estrutura de dados. A visão transforma o modelo em uma forma adequada

para interação, que geralmente é um elemento de interface com o usuário. Múltiplas visões com propósitos distintos podem existir para um mesmo modelo de dados.

Uma arquitetura MVC clássica pode ser frequentemente aplicada a sistemas *desktop* que executam em um computador local. Os conectores entre os componentes são denominados por chamadas de métodos (ou funções). Esse tipo de MVC é também chamado de MVC baseado em método (QIU; PALLICKARA; UYAR, 2004). O diagrama da Figura 1 apresenta um esquema representativo dessa arquitetura clássica. Nessa figura os retângulos em cor cinza e as setas cheias representam a arquitetura clássica e os retângulos e as setas pontilhados são complementos indicados em Wang (2011).

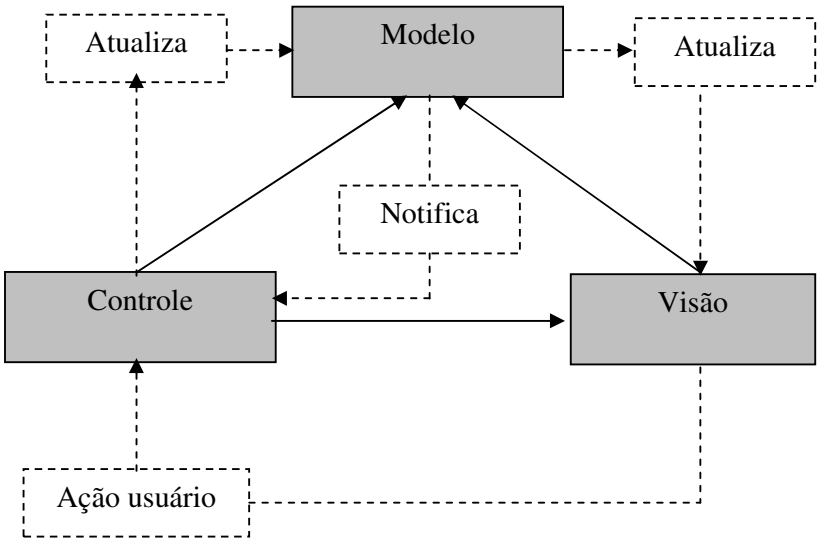

Figura 1 – Model-View-Controller
Fonte: baseado em Microsoft (2014).

2.3 PHP

PHP é um acrônimo recursivo para PHP Hypertext Preprocessor e é amplamente utilizada como uma linguagem de *script* (como programas interpretados) de propósito geral e é muito utilizada para o desenvolvimento de aplicações *web*. PHP é independente de plataforma e disponível como um módulo para uma ampla variedade de servidores *web* (SHEGALOV; WEIKUM, 2006).

PHP é uma linguagem de *script* que pode estar incorporada em documentos HTML na implementação de um servidor. Isso resulta em um modo típico de programação orientada a processos. Incorporar *scripts* PHP em documentos HTML agrupa código na camada de dados com as consultas ao banco de dados e o código da apresentação (WANG, 2011).

Essa estrutura tradicional de programação em PHP apresenta as seguintes vantagens (WANG, 2011; CHOLAKOV, 2008):

a) A estrutura do código é simples, fácil de compreender e de usar;
b) Código pode ser facilmente adicionado em uma página existente;
c) Programas PHP, páginas HTML e outros arquivos podem ser facilmente combinados em uma página, reduzindo requisições à página e melhorando a eficiência;
d) Independência de plataforma e portabilidade.

Contudo a estrutura tradicional de PHP apresenta também desvantagens (WANG, 2011; CHOLAKOV, 2008):

a) A liberdade para escrever o código pode resultar em dificuldade de leitura e gerenciamento do código;
b) O modelo em cascata e desenvolvimento rápido usado em estrutura PHP tradicional é propício à criação de código que pode ser difícil de reusar;
c) A não necessidade de definição explícita de tipo de dado na declaração de variáveis pode não ser uma boa prática de programação;
d) A declaração de variáveis globais que podem tornar os sistemas vulneráveis.

Contudo, MVC resolve alguns desses problemas e juntamente PHP apresenta as seguintes vantagens (WANG, 2011): O código PHP e a código HTML são separados de forma a tornar a estrutura do código mais fácil de entender e tornam a depuração do código mais adequada; módulos e classes facilitam a manutenção de código; a ênfase em abstração de dados é conveniente para o gerenciamento de projetos.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados no desenvolvimento da modelagem e implementação parcial do aplicativo obtido como resultado deste trabalho. A implementação foi realizada com o objetivo de estudo das tecnologias.

3.1 MATERIAIS

Para a modelagem e a implementação do sistema foram utilizadas as ferramentas e tecnologias apresentadas no Quadro 1.

Tecnologia /Ferramenta	**Versão**	**Descrição de uso**
Astah Community	6.2.1	Para modelagem do diagrama com a visão geral do sistema e diagrama de casos de uso.
Case Studio 2	2.25.0	Para modelagem do diagrama de entidades e relacionamentos do banco de dados.
PHP	5.3.4	Como linguagem de programação.
HTML	5.0	Para o desenvolvimento da interface *web* do aplicativo.
Apache	2.2.17	Como servidor *web*.
MySQL Admin		Para sistema gerenciador de banco de dados.
MYSQL	5.5	Como banco de dados.
JQuery	1.8.2	Para a implementação da interface *web*.
JSON		Para formatação de dados.
Ajax		Para implementar processamento do cliente permitindo assincronismo de mensagens entre cliente e servidor.

Quadro 1 – Ferramentas e tecnologias utilizadas

3.2 MÉTODO

As etapas para a modelagem e o estudo das tecnologias utilizadas na implementação do sistema foram realizadas como trabalho de estágio do autor deste trabalho. . Nesse trabalho, a ênfase incide na implementação do sistema.

Para a realização do trabalho foram definidas as seguintes etapas:

a) Levantamento dos requisitos

Os requisitos foram definidos como trabalho de estágio do autor deste trabalho. Neste trabalho de conclusão de curso esses requisitos foram complementados e avaliados.

b) Análise e projeto

A modelagem foi revisada visando realizar ajustes e complementos. A modelagem é apresentada neste texto com o objetivo de facilitar o entendimento dos requisitos do aplicativo.

c) Implementação

A implementação foi realizada utilizando as tecnologias apresentadas na Seção 3.1.

d) Testes

Os testes foram informais e com o objetivo de verificar erros de código e se os requisitos do usuário haviam sido atendidos.

4 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho que é a implementação de um aplicativo *web* para controle de agenda de profissionais de informática.

4.1 ESCOPO DO SISTEMA

O sistema para controle de agendas de profissionais da área de *software* tem como objetivo controlar as agendas dos profissionais de TI e permitir o controle de horas e adicionais de pagamento. O sistema permite que os funcionários do departamento financeiro e os diretores da empresa controlem as metas de seus colaboradores e possibilita maior agilidade no processo de cálculo de valores adicionais que serão pagos aos seus funcionários. Os funcionários também terão maior controle sobre o que já desenvolveram em determinado período de tempo e quanto isso influenciará positivamente em sua renda, gerando maior motivação e consequentemente melhor produtividade, desde que as informações sejam bem aproveitadas pela direção da empresa.

Para que o sistema ofereça um controle correto das agendas é necessário categorizar os colaboradores por função, sejam analistas de implantação, analistas de sistemas, desenvolvedores ou profissionais de Tecnologia de Informação (TI). Essa classificação facilitará no momento da escolha de um profissional disponível para o trabalho. Para cadastrar um funcionário é necessário informar o nome, a função, a data de nascimento, o Cadastro de Pessoas Físicas (CPF), o Registro Geral (RG), o endereço, o local de trabalho e também os valores que cada profissional recebe por hora desenvolvida. Existem acordos pessoais que fazem com que não seja possível padronizar esses valores. Nesse caso, o sistema deve separar os tipos de hora por grupos, atendendo os critérios da empresa. Os valores por essas horas são diferentes e a classificação consiste em:

a) Horas A: Horas utilizadas para treinamento ou implantação;

b) Horas B: Horas utilizadas para desenvolvimento de customizações de *software* para o cliente;

c) Horas C: São horas A ou B que são chamadas de acordo comercial. Essas horas não são cobradas do cliente, e consequentemente, não são pagas aos profissionais;

d) Horas D: São horas utilizadas no deslocamento do profissional até o cliente, São utilizadas para o cálculo do valor da agenda que será cobrado do cliente;

e) Horas E: Horas gastas com a alimentação do profissional. Devem ser lançadas na agenda, pois o profissional deve ter no mínimo 1 (uma) hora de descanso entre os períodos de trabalho.

O sistema também contém um cadastro de filiais, pois a região de trabalho citada anteriormente refere-se a qual filial o profissional está vinculado. Assim, o sistema poderá auxiliar na escolha do profissional mais próximo ao cliente, diminuindo os custos da empresa e também do cliente com o deslocamento do profissional. Nesse cadastro será necessário informar a região da filial, nome, Cadastro Nacional de Pessoa Jurídica (CNPJ), inscrição estadual (se existir), endereço, telefone e vincular a filial a um gerente. O gerente será o responsável por essa unidade e seu cadastro virá do cadastro de funcionários.

O sistema contém, ainda, um cadastro de clientes, com o nome fantasia, razão social, CNPJ, inscrição estadual, região, um campo *flag* para bloquear o atendimento ao cliente e que deverá habilitar um campo de texto com o motivo do bloqueio.

No cadastro de uma nova agenda será possível informar um título, descrição detalhada da agenda, o número da tarefa gerada para essa agenda, o solicitante da agenda em que o cliente e profissional estão vinculados, a data em a visita deverá ser realizada, a estimativa de tempo e o *status* de atendimento. O sistema fornecerá suporte às movimentações nessa agenda, ou seja, será possível encaminha a agenda apara outro profissional ou alterar a data de realização, e o *status*.

Todas as movimentações deverão ficar vinculadas ao funcionário que as realizou e devem conter um campo com a descrição do motivo da movimentação. No momento da conclusão da agenda o profissional informará o tempo foi gasto, as ações realizadas no cliente e deverá alterar o *status* com uma mensagem informando para aguardar o envio da ficha de atendimento. A ficha de atendimento é o meio pelo qual a empresa pode cobrar o cliente pelos atendimentos realizados, mas isso ocorrerá somente após a assinatura da ficha e o funcionário mudar o *status* do pedido para concluído.

O sistema permitirá a emissão de relatórios que informarão sobre quantas horas cada funcionário utilizou para realizar o serviço descrito na agenda. Também informará a meta mensal, o esforço necessário para alcançar essa meta, consulta de atendimentos realizados, agendas futuras, comissão dos profissionais e horas por cliente, dentre outros.

4.2 MODELAGEM DO SISTEMA

A Figura 2 apresenta o diagrama de casos de uso definidos para o sistema. Os casos de uso estão em granularidade de requisitos. Na sequência, (Quadro 2), os casos de uso são agrupados.. O sistema possuirá três atores:

a) Administrativo – ator com acesso a funcionalidades gerenciais do sistema. Tem acesso a todas as funcionalidades do sistema, exceto as específicas do colaborador (lançar horas, finalizar orçamento) e do cliente (autorização de orçamento e solicitação de cancelamento de agenda);

b) Colaborador – funcionário da empresa que realiza os serviços agendados. O colaborador tem acesso a lançar horas, finalizar orçamento, movimentar agenda, estimar horas, além de realizar consultas como, por exemplo, o tipo de horas;

c) Cliente – pessoas para as quais são realizados os serviços objeto de agendamentos. O cliente tem acesso à autorização de orçamento, solicitação de agenda, solicitação de consulta de horas e solicitação de cancelamento de agenda.

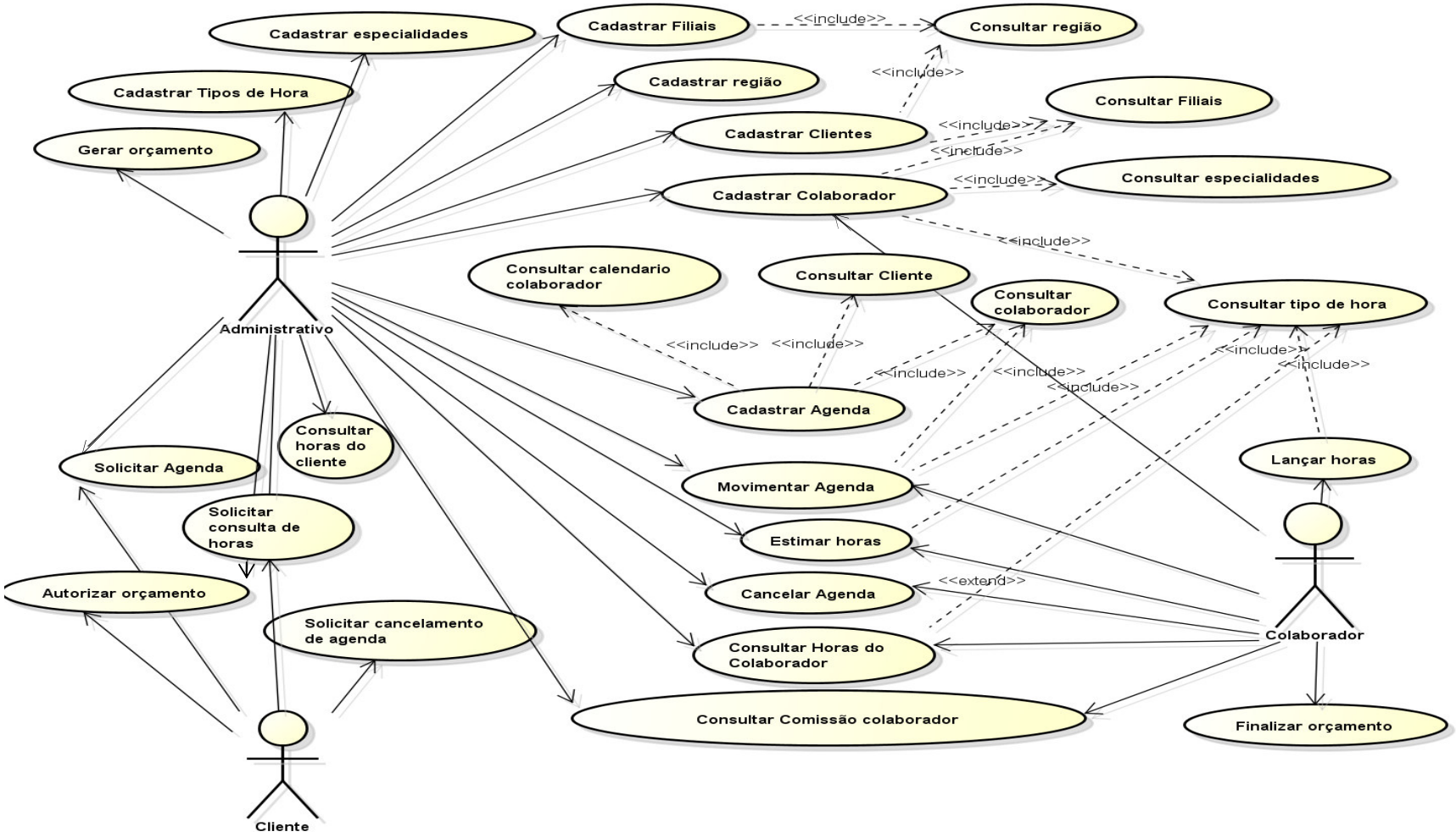

Figura 2 – Diagrama de casos de uso

O Quadro 2 apresenta agrupamentos de casos de uso e também casos de uso relacionados e estão representados na Figura 2.

Identificação do agrupamento	Objetivo	Casos de uso que o compõem
Manter tipo de horas	Permitir que atores adicionem, consultem, alterem ou removam tipos de horas.	Cadastrar tipos de horas Alterar tipos de horas Consultar tipos de horas Excluir tipos de horas
Manter filiais	Permitir que atores adicionem, consultem, alterem ou removam filiais.	Cadastrar filiais Alterar filiais Consultar filiais Excluir filiais Vincular Região
Manter regiões	Permitir que atores adicionem, consultem, alterem ou removam regiões.	Cadastrar regiões Alterar regiões Consultar regiões Excluir regiões
Manter clientes	Permitir que atores adicionem, consultem, alterem ou removam clientes.	Cadastrar clientes Alterar clientes Consultar clientes Excluir clientes Vincular região
Manter especialidades	Permitir que atores adicionem, consultem, alterem ou removam especialidades.	Cadastrar especialidades Alterar especialidades Consultar especialidades Excluir especialidades
Manter colaboradores	Permitir que atores adicionem, consultem, alterem ou removam colaboradores.	Cadastrar colaboradores Alterar colaboradores Consultar colaboradores Excluir colaboradores Vincular região Vincular especialidade Vincular filial
Manter agendas	Permitir que atores adicionem, consultem, alterem ou removam agendas.	Cadastrar agendas Alterar agendas Consultar agendas Excluir agendas Vincular orçamento Vincular cliente Vincular colaborador Movimentar agenda Finalizar agenda Cancelar agenda
Manter orçamentos	Permitir que atores adicionem, e consultem orçamentos.	Cadastrar orçamentos Consultar orçamentos Vincular horas Vincular colaborador

Quadro 2 – Listagem de casos de uso e requisitos

Os Quadros 3 a 6 apresentam a expansão de funcionalidades de inclusão, alteração, consulta e exclusão aplicáveis a todos os casos de uso de cadastro.

No Quadro 3 é apresentada a expansão da funcionalidade inclusão (cadastrar) dos casos de uso que a possuem.

1. Identificador do requisito: Cadastrar dados.
Descrição: Caso de uso que permite ao ator efetuar cadastros.
Evento Iniciador: Qualquer tela de cadastro disponibilizada pelo software.
Atores: Qualquer usuário do sistema com permissão para cadastro
Pré-condição: O usuário deve ter permissão para efetuar o cadastro solicitado.
Sequência de Eventos:
1 – O usuário informa os dados de entrada.
2 – O sistema valida os dados inseridos pelo usuário e caso estejam de acordo com a estrutura esperada pelo banco de dados insere os mesmos no banco.
3 – O sistema retorna uma mensagem informando se o cadastro foi efetuado ou alguma mensagem de erro que auxilie o usuário no cadastro.
Pós-Condição: Os dados inseridos devem ser validados.
Extensões: Caso ocorra erro nos dados informados o sistema deve solicitar a correção ou inserção de alguma informação faltante.

Nome do fluxo alternativo (extensão)	Descrição
Informações incompatíveis	Se a informação for incompatível com o que o banco de dados solicita o sistema deve retornar um erro.

Inclusões: Validar dados inseridos.
Requisitos não funcionais:

Identificador	**Nome**	**Descrição**
RNF1.1	Informações válidas	Os dados só devem ser inseridos no banco se forem validados.

Quadro 3 – Casos de uso Cadastrar

A expansão da funcionalidade para alterar dados dos casos de uso que a possuem é apresentada no Quadro 4.

2. Identificador do requisito: Alterar dados.
Descrição: O caso de uso permite ao ator efetuar alterações em dados já salvos.
Evento Iniciador: Qualquer tela que permita alteração nos dados.
Atores: Usuário com permissões de alteração.
Pré-condição: O usuário deve ter permissão para efetuar a alteração.
Sequência de Eventos:
1 – O usuário pesquisa o registro que deseja alterar.
2 – O usuário altera os dados em tela.
3 – O sistema valida se as informações que o usuário deseja alterar permitem esse tipo de mudança.
4 – O sistema retorna uma mensagem ao usuário dizendo se foi possível ou não efetuar a alteração solicitada.
Pós-Condição: As informações alteradas devem estar disponíveis para consulta e alteração.

Extensões: Se for impossível realizar as alterações o sistema deve retornar uma mensagem ao cliente informando a situação.	
Nome do fluxo alternativo (extensão)	Descrição
Impossível realizar alterações	O sistema deverá informar ao usuário que as alterações nas informações não são possíveis caso não estejam disponíveis.
Inclusões: Validar alterações. **Requisitos não funcionais:**	
Identificador	**Nome** / **Descrição**
RNF 2.1	Alteração — Não poderão ser feitas alterações em alguns campos, pois existem outros locais do sistema que dependem dessa informação, o usuário não deve conseguir alteras esse tipo de informação.

Quadro 4 – Casos de uso alterar

A funcionalidade consultar dados dos casos de uso que a possuem é apresentada no Quadro 5.

3. Identificador do requisito: Consultar dados. **Descrição**: Este caso de uso permite consultar dados ou informações disponíveis no sistema. **Evento Iniciador**: Telas de consulta e relatórios disponíveis. **Atores**: Qualquer usuário com permissão de consulta **Pré-condição**: Se o usuário que solicitou a consulta for um colaborador algumas consultas trarão dados referentes a esse usuário somente. **Sequência de Eventos**: 1 – O usuário solicita consulta de dados. 2 – O sistema verifica se o usuário tem permissões para efetuar a consulta. 3 – O sistema retorna a consulta ou retorna um erro dizendo que não foi possível efetuar a consulta dos dados. **Pós-Condição**: Os filtros utilizados pelo usuário devem ser válidos. **Extensões**: Se o sistema não conseguir efetuar a consulta o usuário deve ser informado sobre o motivo do erro.	
Nome do fluxo alternativo (extensão)	Descrição
Indisponibilidade na consulta	Se não for possível efetuar a consulta o sistema deverá retornar ao usuário um aviso com o motivo da indisponibilidade.
Inclusões: Validar consultas **Requisitos não funcionais:**	
Identificador	**Nome** / **Descrição**
RNF 3.1	Filtro de colaborador — Um colaborador não poderá visualizar alguns dados de outro, portanto a pesquisa deve obedecer o filtro de usuário trazendo os dados somente do usuário logado.

Quadro 5 – Casos de uso consultar dados

No Quadro 6 é apresentada a expansão da funcionalidade exclusão dos casos de uso de cadastro.

4. Identificador do requisito: Excluir dados
Descrição: Caso de uso que possibilita a exclusão de dados.
Evento Iniciador: Qualquer tela que possibilite exclusão de dados.
Atores: Qualquer usuário com permissão de exclusão de dados.
Pré-condição: O usuário deve ter permissão para excluir os dados na tela em questão.
Seqüência de Eventos:
1 – O usuário solicita a exclusão dos dados.
2 – O sistema valida o usuário tem permissão para excluir os dados.
3 – O sistema valida se os dados que serão excluídos não são necessários em outras partes do sistema.
4 – O sistema retorna a mensagem de sucesso na exclusão ou de erro na exclusão
Pós-Condição: O sistema deve verificar se o usuário tem permissão para excluir os dados e também se os dados não são necessários em outras partes do sistema.
Extensões: Se a exclusão estiver bloqueada por algum motivo, o sistema deverá apresentar uma mensagem com o motivo desse bloqueio ao usuário.

Nome do fluxo alternativo (extensão)	Descrição
Exclusão bloqueada	O sistema deverá apresentar uma mensagem ao usuário com o motivo do bloqueio da exclusão caso ela não seja possível realizá-la.

Inclusões: Validar dados.
Requisitos não funcionais:

Identificador	**Nome**	**Descrição**
RNF 4.1	Excluir dados	Os dados só poderão ser excluídos se o usuário possuir permissão para a exclusão e os dados não forem necessários em outras partes do sistema.

Quadro 6 – Casos de uso excluir

4.3 APRESENTAÇÃO DO SISTEMA

Apenas usuários cadastrados podem logar-se no sistema, informando seu *login* e senha, conforme Figura 3.

Figura 3 – Tela de login do sistema com foco no botão entrar

A Figura 4 apresenta a tela inicial do sistema que permite a seleção de funcionalidades do sistema e apresenta, ainda, no topo o nome do colaborador que está logado no momento. Na imagem da Figura 4, o usuário está com o cursor sobre o item do menu que acessa a tela da tabela de cadastro de países.

Figura 4 – Tela inicial do sistema com foco no link para tabela de países

A Figura 5 apresenta a tela com a listagem dos países cadastrados. Essa tela permite alterar o cadastro de um país por meio de um duplo clique na coluna que apresenta os dados do referido país. A adição de um novo país é realizada por meio do botão "Adicionar". A exclusão também é realizada por meio da opção "Excluir" (última coluna da listagem).

Figura 5 – Tela responsável pelo cadastro de países

Um duplo clique na coluna que contém o registro que se deseja alterar habilita a funcionalidade apresentada na Figura 6. Esse procedimento cria dinamicamente um *input box* utilizando as tecnologias JavaScript e jQuery. O campo criado recebe o valor existente na coluna. Para salvar a alteração basta pressionar a tecla *enter*. A edição possui uma validação utilizando jQuery que evita que o usuário insira valores nulos para a coluna sendo editada. A

edição também pode ser feita pelo o botão “Editar” que faz com que a edição seja feita através de um formulário.

Figura 6 – Cadastro de países em modo de edição

Quando o usuário clicar no botão “Adicionar” é apresentado o formulário para adição de um novo registro (Figura 7). O formulário, nesse caso, possui os campos País e Discagem Direta Internacional (DDI). O campo “idPais” é numérico, chave primária e incrementado automaticamente à medida que os registros são inseridos no banco de dados. Essa informação é necessária apenas para identificar os registros armazenados na tabela de forma exclusiva e permitir o controle e relacionamentos das tabelas do banco de dados.

Após inserir a informação nas colunas, o usuário pode clicar no botão “Adicionar” e confirmar a operação ou cancelá-la, que, nesse caso, retornará ao estado de listagem. Quando o usuário confirmar a inserção, a tela retorna ao estado de listagem e o valor inserido estará na lista de países. A adição possui uma validação utilizando jQuery que evita que o usuário insira valores nulos. Os campos marcados com um asterisco permitem que o usuário identifique quais campos são de preenchimento obrigatório.

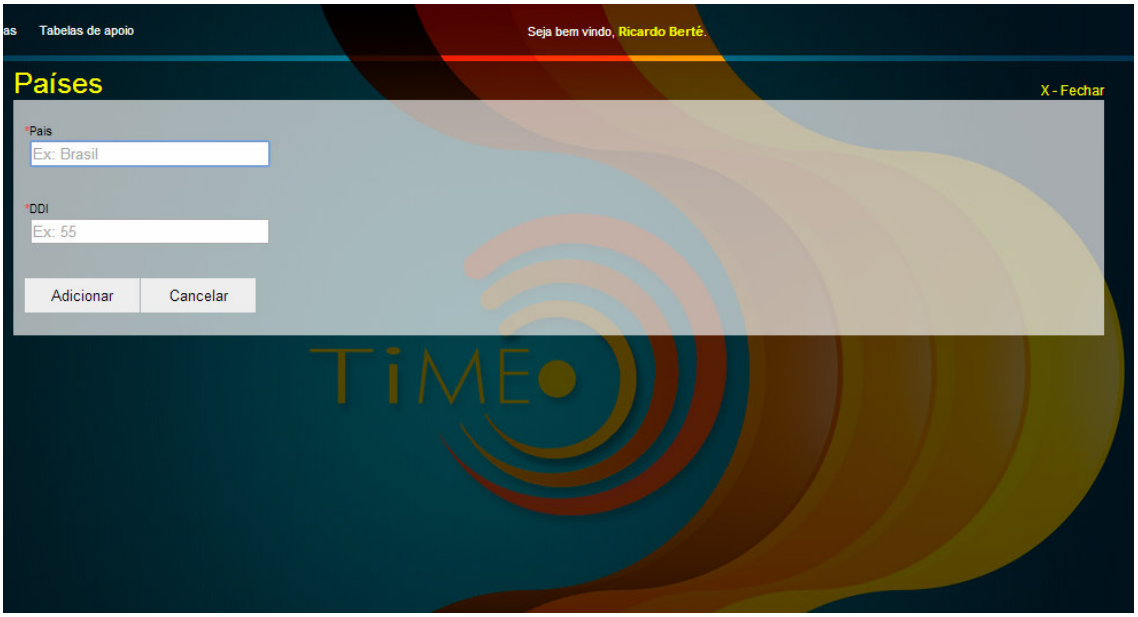

Figura 7 – Cadastro de países em modo de adição

A Figura 8 mostra o que ocorre quando o usuário clica no botão “Excluir”. Nesse caso é apresentada uma janela de confirmação informando qual o país que está sendo excluído e é solicitado que o usuário confirme a exclusão, podemos verificar que a janela não possui título definido, para a criação dessa confirmação foi utilizada a função *confirm()* nativa do *javascript* que não permite a adição de títulos. Se clicado no botão ”OK”, o país será automaticamente excluído da listagem. Se o usuário clicar no botão ”Cancelar” é direcionado ao estado anterior da tela.

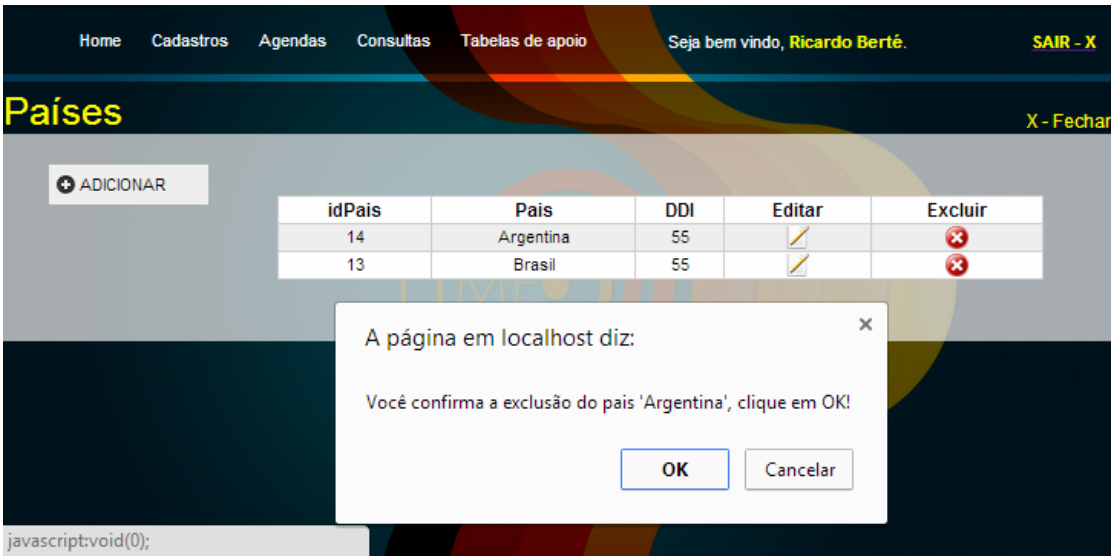

Figura 8 – Cadastro de países em modo de exclusão

A Figura 9 apresenta a tela de listagem dos colaboradores. Por se tratar de um cadastro com mais campos é apresentado um botão “Editar” em cada registro. No topo dessa tela é apresentado um mecanismo de busca que serve como padrão para outras telas e terá seu funcionamento apresentado posteriormente.

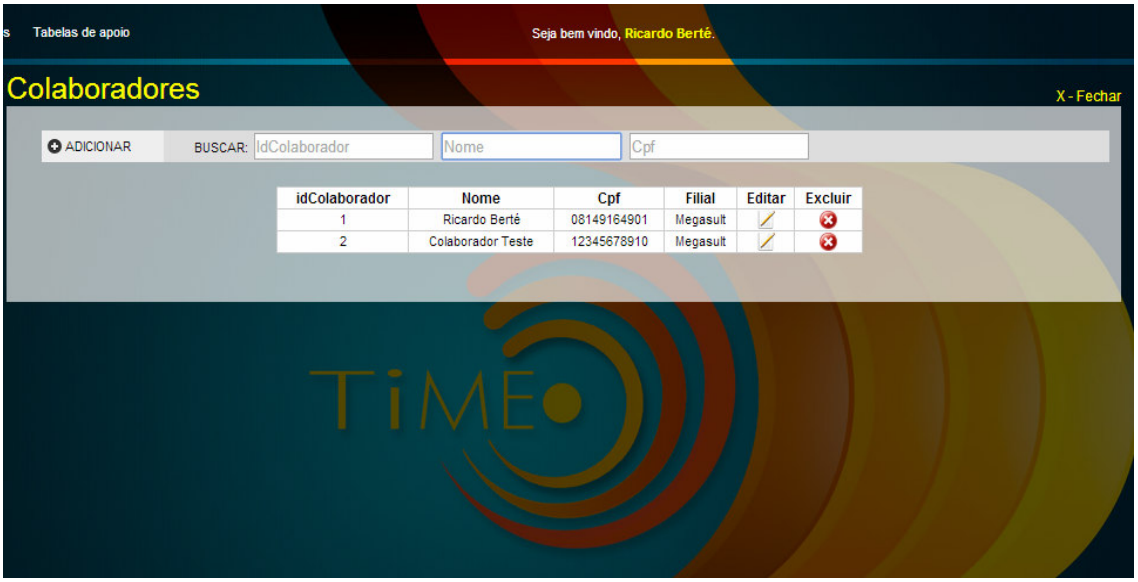

Figura 9 – Tela de listagem de colaboradores

A Figura 10 apresenta o resultado de uma busca na tela de listagem de colaboradores. Quando a busca é realizada o resultado é carregado de dinamicamente na listagem sem necessidade de recarregar a tela.

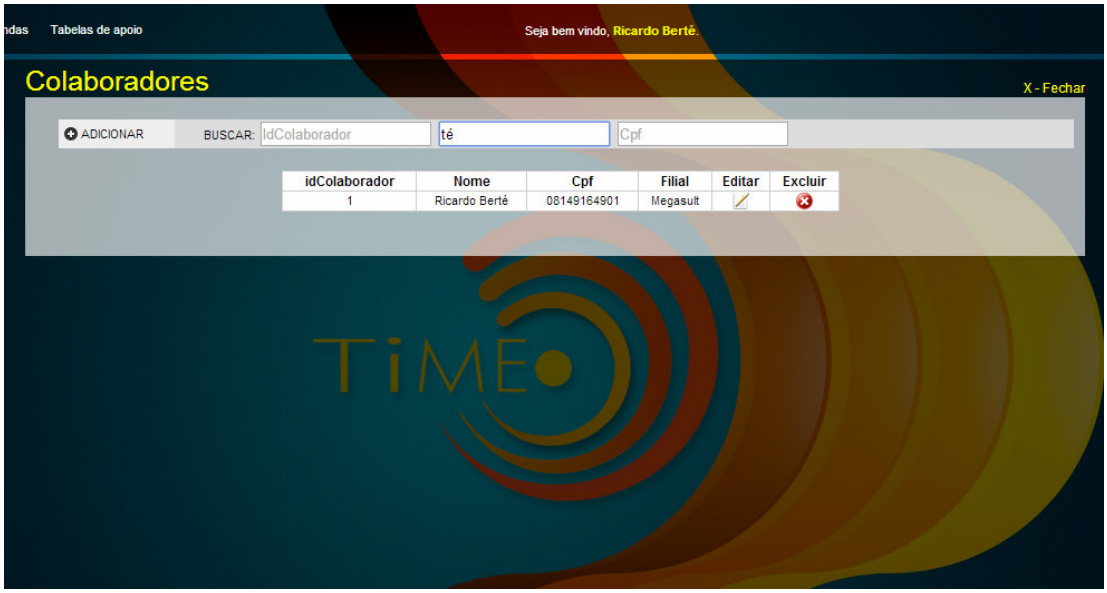

Figura 10 – Resultado da busca na tela de colaboradores

A Figura 11 mostra a tela de inserção de um novo colaborador com o campo "Especialidade" sendo editado. Na imagem é possível verificar que existem 3 (três) campos (especialidade, filial, cidade) que são carregados de dinamicamente. O funcionamento do mecanismo que carrega as informações será apresentado posteriormente.

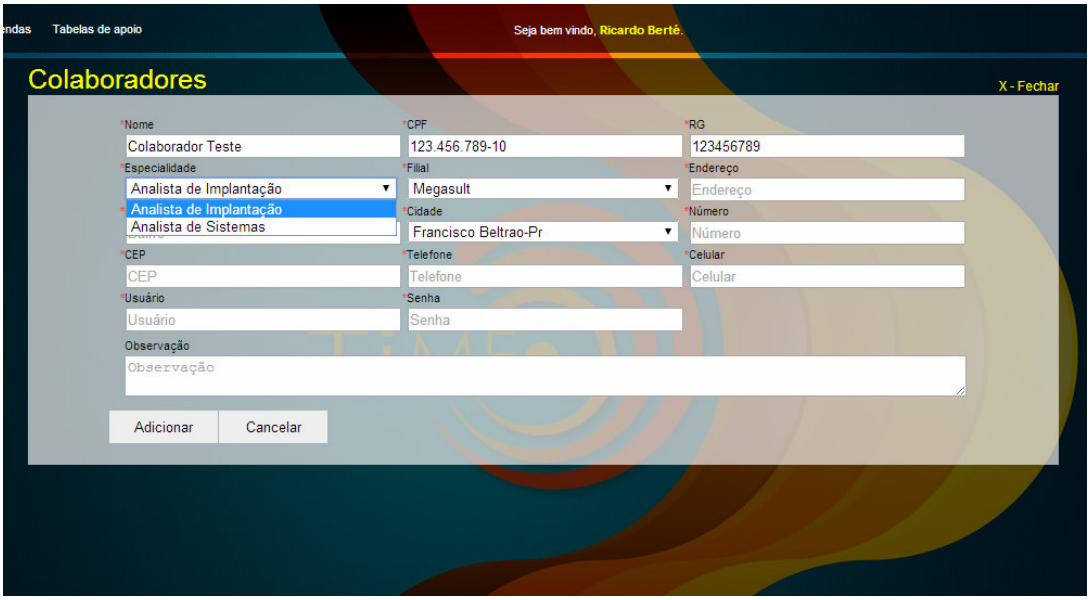

Figura 11 – Tela de inserção de um novo colaborador com o campo especialidade em edição

É possível observar também por meio da Figura 11 que o campo "CPF" contém uma máscara para auxiliar o usuário na interação com o sistema e evitar o preenchimento incorreto de informações básicas. Nesse exemplo são utilizadas máscaras também nos campos Código de Endereçamento Postal (CEP), telefone e celular. As máscaras do aplicativo são criadas utilizando-se do *plugin* jQuery chamado *Maskedinput*.

Após o usuário clicar no botão "Adicionar", o sistema verifica se os campos estão preenchidos e o registro do colaborador é carregando de dinamicamente. A tela de listagem apresentará o último registro inserido.

A Figura 12 mostra a listagem de agendas que faz parte da principal funcionalidade do sistema que é o cadastro e gerenciamento de agendas.

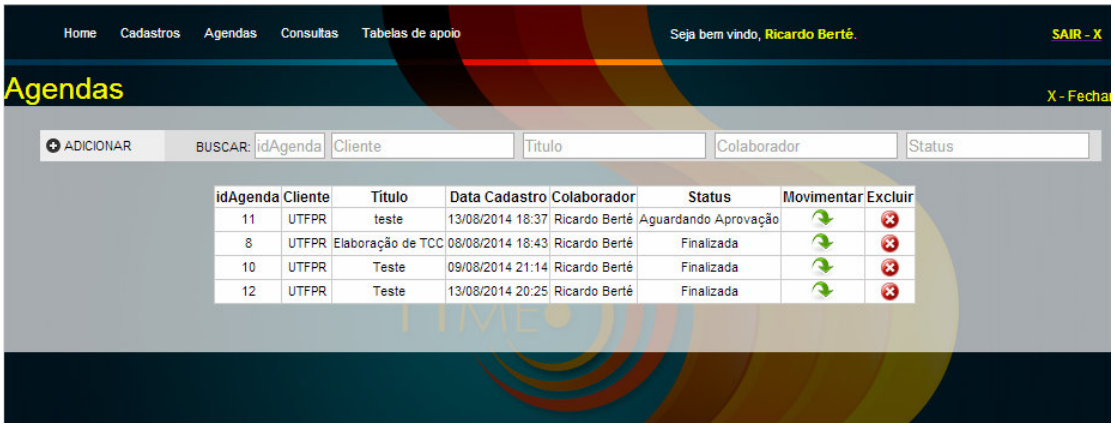

Figura 12 – Listagem de agendas

A tela apresentada na Figura 12 possui alguns pontos peculiares se comparada às outras telas do sistema: ao invés do botão "Editar", existe o botão "Movimentar". Como requisito da direção da empresa que fará uso do sistema, as agendas não podem ser alteradas depois de sua criação, o que pode ser alterado está disponível nas movimentações. Na Figura 13 está exposta a tela de movimentação em seu estado inicial, porém com o campo "Selecionar movimentação" em estado de edição, mostrando as possíveis movimentações dessa agenda.

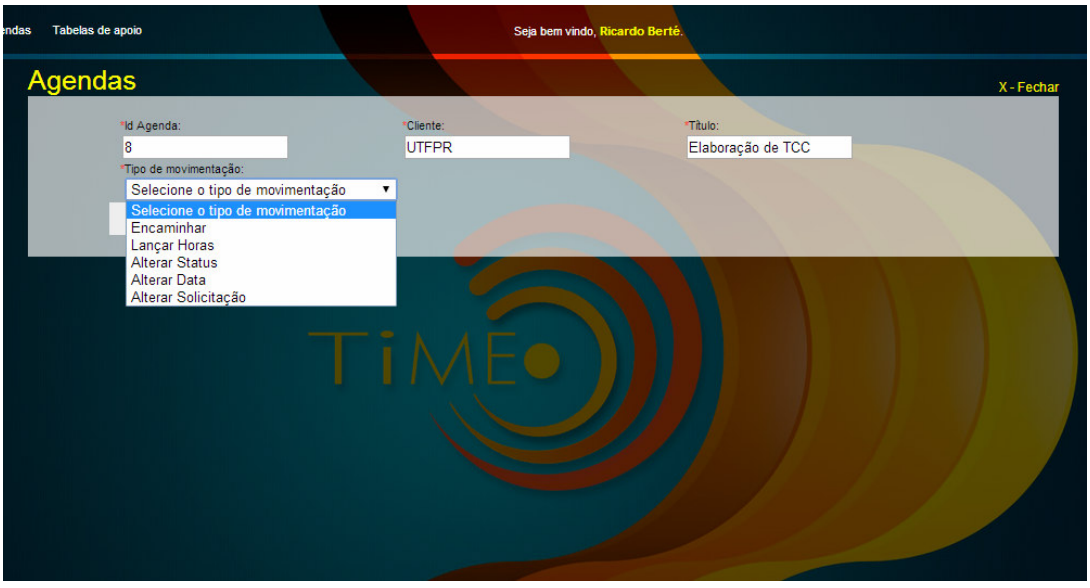

Figura 13 – Tela de agendas em modo de movimentação de agenda

A Figura 14 apresenta o formulário caso o usuário selecione a opção para encaminhar a agenda.

Figura 14 – Formulário para encaminhamento da agenda

Por meio da Figura 14 é possível observar que foram apresentados os campos "colaborador origem" e "colaborador destino", que armazenam o colaborador responsável pela agenda e para qual o usuário deseja encaminhar respectivamente. É, ainda, apresentado o botão "Salvar" em uma posição que mantém a linha de alteração que o usuário está seguindo. Quando o usuário selecionar a alteração e clicar no botão "Salvar" é apresentada uma mensagem informando que a movimentação foi salva com sucesso e o sistema é direcionado para a tela de listagem das agendas.

A Figura 15 apresenta a segunda opção de movimentação de agendas que é o lançamento de horas.

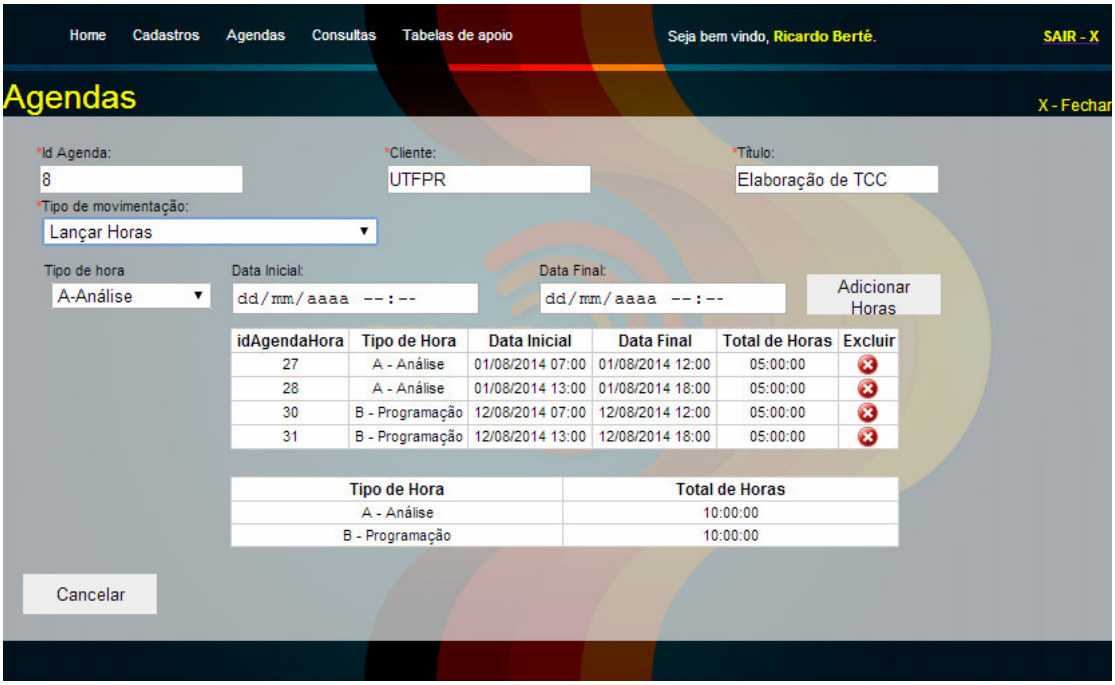

Figura 15 – Formulário para lançar horas

A opção de lançamento de horas possibilita que o usuário lance horas na agenda. Os tipos de horas são cadastrados nas tabelas de apoio. O processo consiste na seleção do tipo de hora, inserção da data e hora inicial e da data e hora final. Conforme informado pelo cliente, as horas devem ser lançadas em períodos, ou seja, o analista deve efetuar um lançamento de horas do tempo gasto durante a manhã e outro do tempo gasto durante a tarde.

Na Figura 15 também é possível observar um totalizador de horas que o usuário utilizará para monitorar o quanto já lançou de horas nessa agenda. Na listagem das horas lançadas por período, o botão "Excluir" permite apagar o lançamento de horas. Não existe possibilidade de alteração nos lançamentos, eles devem ser excluídos e inseridos novos. Toda a atualização das tabelas de listagem é feita de modo dinâmico evitando o recarregamento da página.

A Figura 16 mostra a terceira opção de movimentação que permite a alteração do *status*.

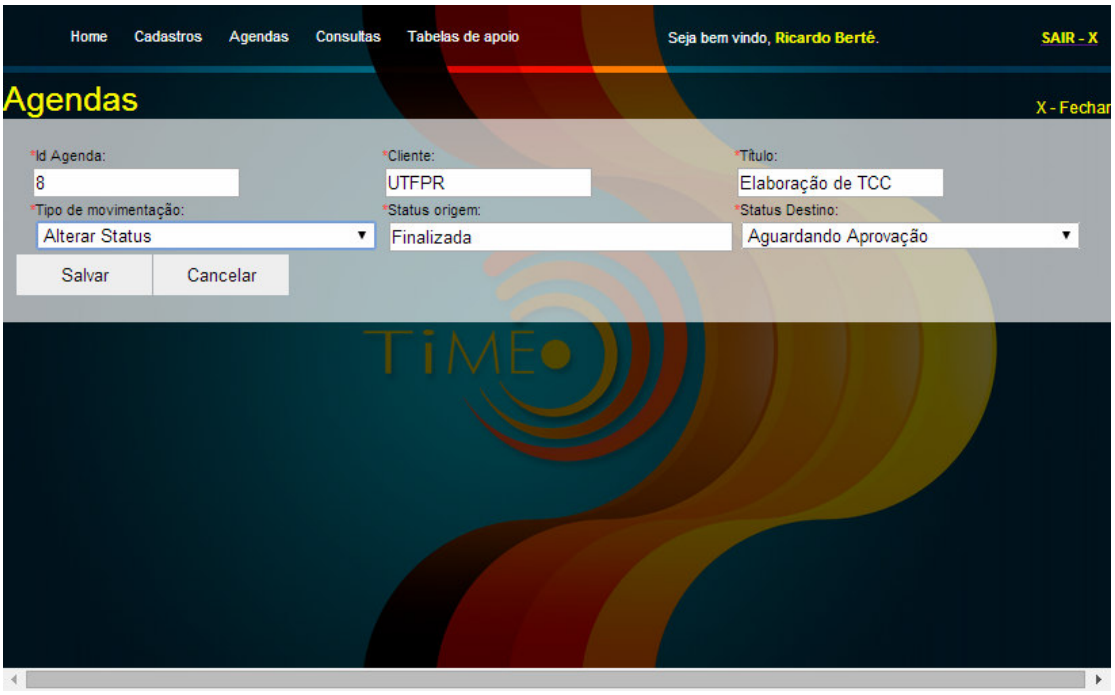

Figura 16 – Formulário para alteração de *status*

A opção de alteração de *status* segue o mesmo conceito apresentado no encaminhamento que foi citado anteriormente.

A Figura 17 apresenta a quarta possibilidade de movimentação da agenda que permite a alteração da data.

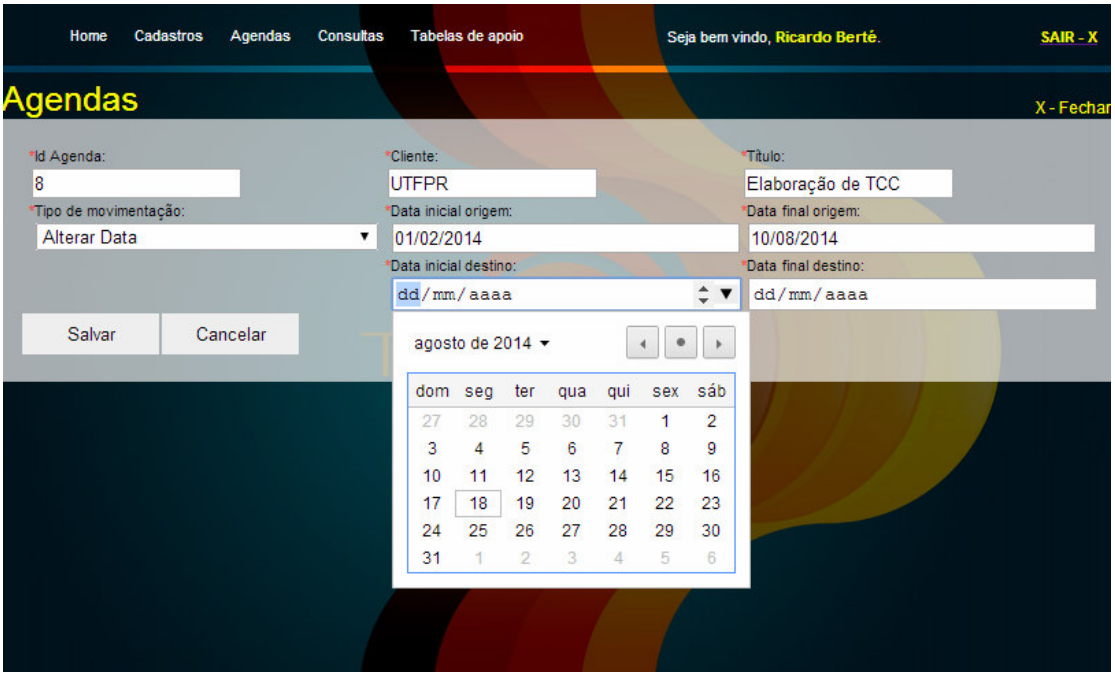

Figura 17 – Formulário para alteração de data

A alteração de datas possui um leiaute um pouco diferente dos demais. Para alterar a data, o usuário deve selecionar as duas datas. A data inicial origem apresenta a data em que seria iniciada a agenda e a data final origem, a data em que seria o final da agenda. Os campos de destino são os campos que recebem as novas datas de início e fim da agenda. Esses campos são utilizados para que o usuário possa informar uma nova data para realização da agenda, caso o cliente queira alterar a data de realização da agenda. Todos os campos de data dos formulários de edição ou adição são do tipo *"date"* que foi implantado no HTML5 e cria um *datepicker* para que o usuário tenha maior facilidade no preenchimento de datas.

Na Figura 18 é apresentada a movimentação de agenda que permite a alteração da solicitação do cliente.

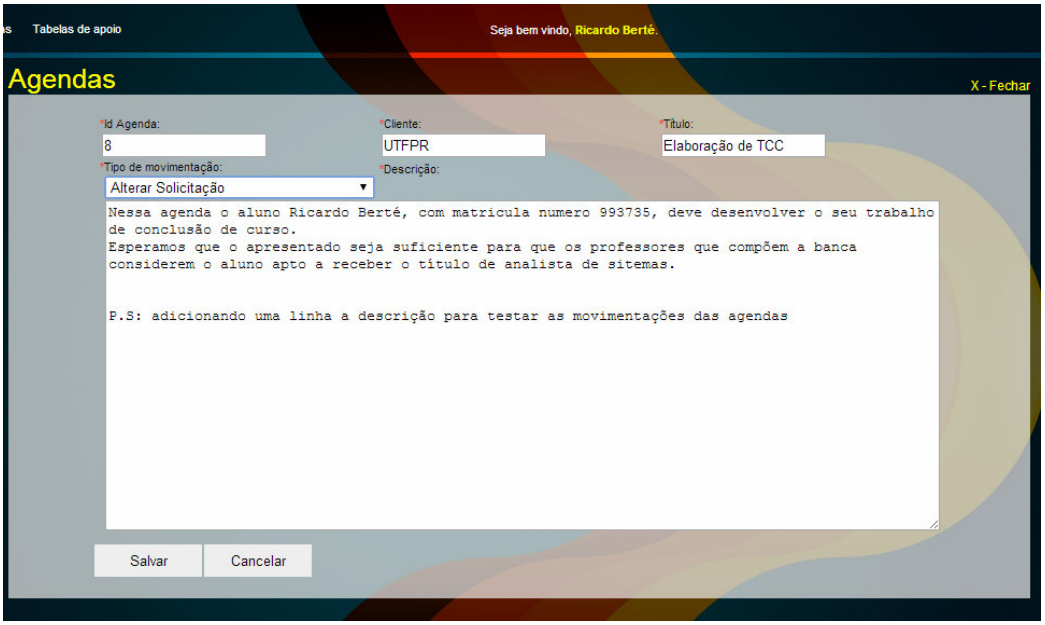

Figura 18 – Formulário para alteração de solicitação

A alteração de solicitação carrega um campo tipo *textarea* com a atual solicitação do cliente podendo ser editada. Todas as movimentações feitas são armazenadas em uma tabela do banco de dados que serve como *log*. Em futura versão do sistema será implementada uma tela para a consulta das movimentações das agendas.

Essas são as possíveis movimentações levantadas no processo de análise. Caso seja necessária alguma outra movimentação como, por exemplo, a troca de cliente, a agenda deve ser excluída e deverá ser criada uma nova agenda.

A Figura 19 mostra a tela de detalhes da agenda.

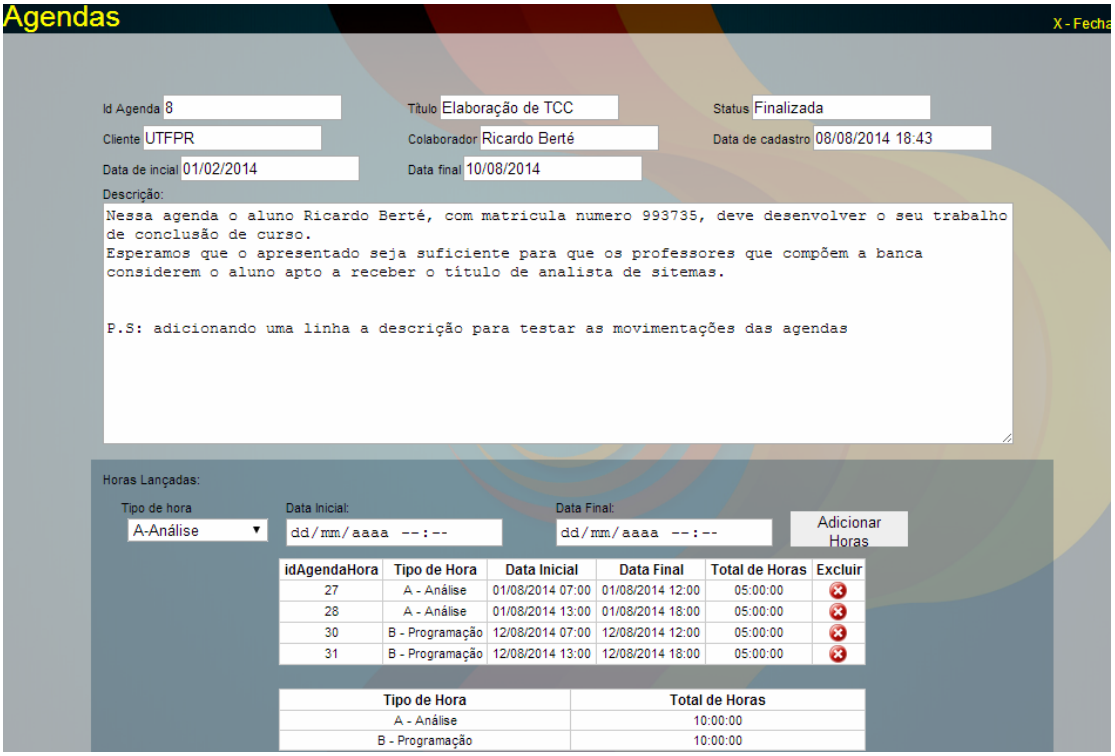

Figura 19 – Detalhe da agenda

A Figura 19 apresenta a tela da listagem dos detalhes da agenda, as informações básicas como: id, título, *status*, cliente, colaborador, data de cadastro, data prevista para início, data prevista para fim, solicitação, além das horas lançadas, os totalizadores de horas para essa agenda. Foi mantida nessa tela a opção que permite inserir horas por solicitação do cliente. Nessa tela todas as informações são carregadas de dinamicamente evitando o recarregamento da página.

4.4 IMPLEMENTAÇÃO DO SISTEMA

Esta seção mostra a estrutura do sistema e também o que é executado entre as camadas *model*, *view* e *controller* que são as camadas propostas por esse padrão de projetos.

A metodologia MVC fornece um padrão de organização para aplicativos que faz com que a manutenção posterior seja mais simples. Isso ocorre porque cada pasta tem um conteúdo pré-definido em termos de tipo de arquivo e cada tipo de arquivo tem uma função pré-definida. A Figura 20 apresenta a estrutura de pastas de acordo com o padrão MVC.

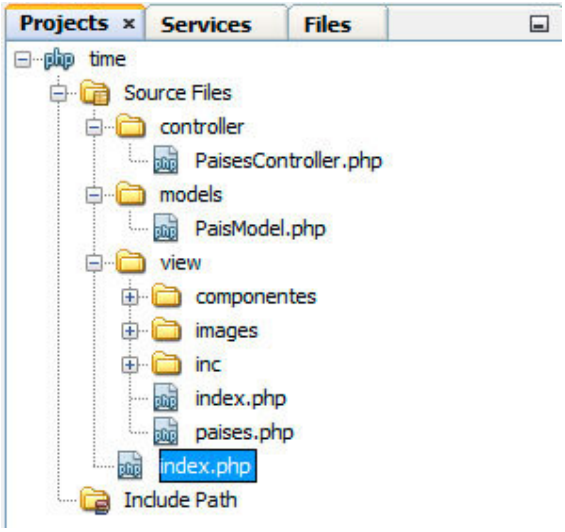

Figura 20 – Estrutura de pastas MVC

A pasta “Source Files” é padrão criada pela IDE NetBeans no momento que o projeto é criado. Essa pasta tem a função de armazenar todos os arquivos da aplicação.

Dentro dessa pasta existe a pasta “controller”, que armazena todos os arquivos responsáveis por tratar as requisições enviadas pela camada *view*, carregar as classes envolvidas no processo e executar os métodos solicitados pela *view*.

A pasta “models” contém as classes responsáveis pela interação com o banco de dados, retornando os resultados solicitados pela *view* ao *controller*.

A pasta “view” contém todos os arquivos responsáveis por receber as requisições e criar a visualização do conteúdo para o usuário.

As Listagens 1 a 4 a seguir contêm um exemplo de como a *view* se comporta para efetuar requisições ao *controller* que, por sua vez, irá requisitar ao *model* que retornará os valores para a *view* que é a responsável pela formatação das informações para o usuário.

Na Listagem 1 estão as funções contidas dentro do arquivo “paises.php”, que é a *view* responsável pelo gerenciamento dos países. Nessa Listagem é apresentado somente o

procedimento para carregar os países de maneira dinâmica e não todas as funções contidas no arquivo.

```
<script>

$(document).ready(
   function(){
      getPaises();
   }
);

function getPaises() {
      $.post('../controller/PaisesController.php',
            {
                  action: 'getPaises'
            },
            function(data) {
                  $('.botao').show();
                  renderPaises(data);
            },
            "json"
      );
}

</script>
```

Listagem 1 – Exemplo de funções presentes na view, paises.php

De acordo com o trecho de código apresentado na Listagem 1, quando o documento está carregado é chamada a função “getPaises()”. Dentro dessa função há um “POST” que é direcionado para a *Uniform Resource Locator* (URL) do *controller* referente a essa *view*. Na Listagem 2 está o código do *controller* que foi acionado pela função da *view*. Esse *controller* inicia a classe “pais” que será utilizada e executa os métodos contidos dentro dela. No caso da função “getPaises()”, que passa o parâmetro “action” com o valor “getPaises”, será executado o método “getPaises()” da classe que está no *model* e está exposto na Listagem 3.

```
<?php
function __autoload($className){
      include_once("../models/".$className."Model.php");
}

$paises=new Pais("127.0.0.1","root","","dbTime");

if(!isset($_POST['action'])) {
      print json_encode(0);
      return;
}

switch($_POST['action']){
      case 'getPaises':
            print $paises->getPaises();
      break;
      case 'deletePais':
            $pais = new stdClass;
            $pais = json_decode($_POST['pais']);
```

```
            print $paises->deletePais($pais);
        break;
        case 'updatePais':
            $pais = new stdClass;
            $pais = json_decode($_POST['pais']);
            print $paises->updatePais($pais);
        break;
        case 'addPais':
            $pais = new stdClass;
            $pais = json_decode($_POST['pais']);
            print $paises->addPais($pais);
        break;

}

exit();

?>
```

Listagem 2 – Exemplo de controller, paisesController.php

A Listagem 3 apresenta o método "getPaises()" da classe que está no *model*.

```
<?php

class Pais {

    private $dbh;

    public function __construct($host,$user,$pass,$db)    {
        $this->dbh = new PDO("mysql:host=".$host.";dbname=".$db,$user,$pass);
    }

    public function getPaises(){
        $sbh = $this->dbh->prepare("SELECT * FROM paises order by pais");
        $sbh->execute();
        return json_encode($sbh->fetchAll());
    }

    public function deletePais($pais){
        $sth = $this->dbh->prepare("DELETE FROM paises WHERE IdPais=?");
        $sth->execute(array($pais->IdPais));
        return json_encode(1);
    }
    public function updatePais($pais){
        $sth = $this->dbh->prepare("UPDATE paises SET ". $pais->campo ."=? WHERE
idPais=?");
        $sth->execute(array($pais->novoValor, $pais->idPais));
        return json_encode(1);
    }
    public function addPais($pais){
        $sth = $this->dbh->prepare("INSERT INTO paises (pais, ddi) VALUES(?,?)");
        $sth->execute(array($pais->pais, $pais->ddi));
        return json_encode($this->dbh->lastInsertId());
    }
}
?>
```

Listagem 3 – Exemplo de model, paisModel.php

O método “getPaises()” do *model* retorna uma consulta feita no banco de dados em formato *JavaScript Object Notation* (JSON). A *view* por sua vez recebe esses dados e executa a função “renderPaises()”, exposta na Listagem 4, passando os dados em formato JSON retornados.

```
function renderPaises(jsonData) {
	//alert(JSON.stringify(jsonData));
	var table = '<table id="tablePaises"
class="tabelaApoio"><thead><tr><th>idPais</th><th>Pais</th><th>DDI</th><th>Excluir</th
></tr></thead><tbody>';

	$.each( jsonData, function(i,pais){
		table += '<tr class="trOver">';
		table += '<td field="idPais" idPais="'+pais.IdPais+'">'+pais.IdPais+'</td>';
		table += '<td campo="pais" ondblclick="tornaEditavel(this);"
onkeypress="verificaEnter()"  idPais="'+pais.IdPais+'" onclick=()>'+pais.Pais+'</td>';
		table += '<td campo="ddi" ondblclick="tornaEditavel(this);"
onkeypress="verificaEnter()" idPais="'+pais.IdPais+'">'+pais.DDI+'</td>';
		table += '<td><a href="javascript:void(0);"
onclick=(deletePais($(this).attr("idPais"),$(this).attr("pais"))) idPais="'+pais.IdPais+'"
pais="'+pais.Pais+'">excluir</a></td>';
		table += '</tr>';
	});

	table += '</tbody></table>';

	$('div#conteudo').html(table);
}
```

Listagem 4 – Exemplo de função que recebe o retorno da *view* pais.php

A função renderPaises, apresentada na Listagem 4, tem a responsabilidade de organizar a visualização para o usuário, carregando os países retornados na *div* com o *id* “conteúdo”. Tudo isso de maneira dinâmica, isto é, sem recarregar o sistema em momento algum. A Listagem 5 mostra como é o HTML da busca de colaboradores.

```
<div class="busca">
	BUSCAR:
	<input	type="text"	placeholder="IdColaborador"	onfocus="this.value('')"
onkeypress="getColaboradores('IdColaborador',this.value)"/>
	<input	type="text"	placeholder="Nome"
onkeypress="getColaboradores('Nome',this.value)"/>
	<input	type="text"	placeholder="Cpf"
onkeypress="getColaboradores('CPF',this.value)"/>
	</div>
```

Listagem 5 – Exemplo de HTML de uma busca

No código da Listagem 5 existem três campos e o método “onkeypress” deles executa a função “getColaboradores” passando como parâmetros o nome do campo e o valor que será utilizado no filtro. Na Listagem 6 está o código da função “getColaboradores”.

```
function getColaboradores(campo,filtro) {
	$('.botao').show();
	$('.busca').show();
	var Filtro = new Object();
	Filtro.campo = campo;
	Filtro.valor = filtro;

	var FiltroJson = JSON.stringify(Filtro);


   $.post('../controller/ColaboradoresController.php',
		{
			action: 'getColaboradores',
		 filtro: FiltroJson
		},
		function(data) {
			$('.botao').show();
			renderColaboradores(data);
		},
		"json"
   );
}
```

Listagem 5 – Função getColaboradores

A função getColaboradores recebe os dois parâmetros passados pelo campo de busca, então é criado um objeto com o nome Filtro que possui dois atributos, o campo e o valor, que são o campo que será comparado na busca e o valor que será utilizado para comparação, respectivamente. Após criado o objeto é acionado o *controller* "ColaboradoresController" que irá direcionar ao método correto do "*model* . Na Listagem 6 está o código do *controller*.

```
 <?php
function __autoload($className){
   include_once("../models/".$className."Model.php");
}

$colaboradores=new Colaborador("127.0.0.1","dbTime","root","");

if(!isset($_POST['action'])) {
   print json_encode(0);
   return;
}

switch($_POST['action']){
   case 'getColaboradores':
		$filtro = new stdClass;

		$filtro = json_decode($_POST['filtro']);
		print $colaboradores->getColaboradores($filtro);

   break;
      case 'deleteColaborador':
		$colaborador = new stdClass;
		$colaborador = json_decode($_POST['colaborador']);
		print $colaboradores->deleteColaborador($colaborador);
   break;
      case 'editaColaborador':
```

```
                $colaborador = new stdClass;
                $colaborador = json_decode($_POST['colaborador']);
                print $colaboradores->editaColaborador($colaborador);
    break;
      case 'addColaborador':
                $colaborador = new stdClass;
                $colaborador = json_decode($_POST['colaborador']);
            print $colaboradores->addColaborador($colaborador);
    break;
}

exit();

?>
```

Listagem 6 – ColaboradoresController

O *controller* instancia um objeto da classe "Colaborador", sendo assim possível o acesso aos métodos da classe. No caso do sistema desenvolvido foi passado via *post* uma variável com o nome "action" que possui o valor "getColaboradores". Na estrutura do *switch* caso o valor seja "getColaboradores" uma variável com nome "filtro" recebe o objeto com nome "filtro" enviado via *post* e aciona o método "getColaboradores" do *model* passando esse objeto para o mesmo como parâmetro.

Na Listagem 7 é apresentado o código do método "getColaboradores" do *model* "ColaboradorModel" utilizado no *controller* da Listagem 6, não será inserido o código inteiro para não prejudicar a leitura do documento.

```
public function getColaboradores($filtro){
        if($filtro->valor != ''){
            $sbh = $this->dbh->prepare("SELECT colaboradores.*,
filiais.IdFilial,filiais.NomeFantasia FROM colaboradores INNER JOIN filiais on filiais.IdFilial
= colaboradores.IdFilial WHERE colaboradores.".$filtro->campo." like '%".$filtro->valor."%'");
            $sbh->execute(array($filtro->campo, $filtro->valor));
            return json_encode($sbh->fetchAll());

        }else{
            $sbh = $this->dbh->prepare("SELECT colaboradores.*,
filiais.IdFilial,filiais.NomeFantasia FROM colaboradores INNER JOIN filiais on filiais.IdFilial
= colaboradores.IdFilial");
            $sbh->execute();
            return json_encode($sbh->fetchAll());
        }
    }
```

Listagem 7 – ColaboradoresController

O método "getColaboradores" recebe o objeto "filtro" e verifica se o atributo "valor" desse objeto está valorizado. Caso esteja, executada uma consulta filtrando os resultados e caso não esteja valorizado, a consulta é executada retornando todos os resultados.

Esse comportamento faz com que o método utilizado para a busca e para listagem geral seja o mesmo, evitando, assim, a construção de novos métodos, simplificando o código e agilizando o desenvolvimento.

Ao final da consulta o método retorna os resultados em formato *JavaScript Object Notation* (JSON) e a função "getColaboradores" da *view* que aguardava o retorno dispara a função "renderColaboradores" passando o retorno como parâmetro.

Na Listagem 8 está exposta a função "renderColaboradores".

```
function renderColaboradores(jsonData) {
    var table = '<table id="tableColaboradores"
class="tabelaApoio"><thead><tr><th>idColaborador</th><th>Nome</th><th>Cpf</th><th>Fil
ial</th><th>Editar</th><th>Excluir</th></tr></thead><tbody>';

  $.each( jsonData, function(i,colaborador){
        table += '<tr class="trOver">';
        table += '<td field="IdColaborador"
IdColaborador="'+colaborador.IdColaborador+'">'+colaborador.IdColaborador+'</td>';
            table += '<td campo="Nome" IdColaborador="'+colaborador.IdColaborador+'"
onclick=()>'+colaborador.Nome+'</td>';
        table += '<td campo="CPF" IdColaborador="'+colaborador.IdColaborador+'"
onclick=()>'+colaborador.CPF+'</td>';
        table += '<td campo="Bloqueado" IdColaborador="'+colaborador.IdColaborador+'"
>'+colaborador.NomeFantasia+'</td>';
        table += '<td><a href="javascript:void(0);"
onclick=(editaColaborador("'+colaborador.IdColaborador+'"))><img class="btExcluir"
src="images/editar.png" /></a></td>';
            table += '<td><a href="javascript:void(0);" colaborador="'+colaborador.Nome+'"
onclick=(deleteColaborador("'+colaborador.IdColaborador+'",$(this).attr("colaborador")))><img
class="btExcluir" src="images/excluir.png" /></a></td>';
            table += '</tr>';
    });

  table += '</tbody></table>';

  $('div#conteudo').html(table);
}
```

Listagem 8 – Função "renderColaboradores"

A função "renderColaboradores" recebe o retorno do *model* como parâmetro e monta uma tabela com os valores.

A edição do colaborador por sua vez utiliza o mesmo método "getColaboradores" para buscar o colaborador que o usuário deseja editar e assim montar o formulário de edição. A diferença está no método de renderização executado. A função executa a função "createFormEdicao" quando recebe o retorno do *model*. A Listagem 9 apresenta o código da função "editaColaborador".

```
function editaColaborador(filtro) {
	var Filtro = new Object();
	Filtro.campo = 'idColaborador';
	Filtro.valor = filtro;

	var FiltroJson = JSON.stringify(Filtro);

   $.post('../controller/ColaboradoresController.php',
		{
			action: 'getColaboradores',
		 filtro: FiltroJson
		},
		function(data) {
			createFormEdicao(data);
		},
		"json"
   );
}
```

Listagem 9 – Função "editaColaborador"

Na Listagem 10 está o código da função "createFormEdição".

```
function createFormEdicao(jsonData) {
	$('.botao').hide();
	$('.busca').hide();


	$.each( jsonData, function(i,colaborador){
	  var form = '<div style="width:810px;margin:auto">';

	  form += '<input type="hidden" id="IdColaborador" IdColaborador="'+colaborador.IdColaborador+'" value="'+colaborador.IdColaborador+'"/>';


	  form += '<div class="divFormIncluir">';
	  form += '<span><span class="fonteVermelha">*</span>Nome</span>';
	  form += '<input class="inputFormIncluir" type="text" id="Nome" name="Nome" value="'+colaborador.Nome+'" placeholder="Nome"/>';
	  form += '</div>';
	  form += '<div class="divFormIncluir">';
	  form += '<span><span class="fonteVermelha">*</span>CPF</span>';
	  form += '<input class="inputFormIncluir" type="text" id="CPF" name="CPF" value="'+colaborador.CPF+'" placeholder="CPF"/>';
	  form += '</div>';

	  form += '<div class="divFormIncluir">';
	  form += '<span><span class="fonteVermelha">*</span>RG</span>';
	  form += '<input class="inputFormIncluir" type="text" id="RG" name="RG" value="'+colaborador.RG+'" placeholder="RG"/>';
	  form += '</div>';


	  form += '<div class="divFormIncluir">';
	  form += '<span><span class="fonteVermelha">*</span>Especialidade</span></br>';
	  form += '<select class="inputFormIncluir" id="especialidade" name="especialidade">';

	  $.post('../controller/EspecialidadesController.php',
```

```
        {
            action: 'getEspecialidades'
        },
        function(data) {
            $.each( data, function(i,especialidade){
                if(especialidade.IdEspecialidade == colaborador.IdEspecialidade) {
                    form += '<option value="'+especialidade.IdEspecialidade+'" selected>'+especialidade.Especialidade+'</option>';
                }else{
                    form += '<option value="'+especialidade.IdEspecialidade+'">'+especialidade.Especialidade+'</option>';
                }
            });

            form += '</select>';
            form += '</div>';


            form += '<div class="divFormIncluir">';
            form += '<span><span class="fonteVermelha">*</span>Filial</span>';
            form += '<select class="inputFormIncluir" id="filial" name="filial">';
            $.post('../controller/FiliaisController.php',
            {
                action: 'getFiliais'
            },
            function(data) {
                $.each( data, function(i,filial){
                    if(filial.IdFilial == colaborador.IdFilial){
                        form += '<option value="'+filial.IdFilial+'" selected="true">'+filial.NomeFantasia+'</option>';
                    }else{
                        form += '<option value="'+filial.IdFilial+'">'+filial.NomeFantasia+'</option>';
                    }

                });

                form += '</select>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Endereço</span>';
                form += '<input class="inputFormIncluir" type="text" id="Endereco" name="Endereco" value="'+colaborador.Endereco+'" placeholder="Endereço"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Bairro</span>';
                form += '<input class="inputFormIncluir" type="text" id="Bairro" name="Bairro" value="'+colaborador.Bairro+'" placeholder="Bairro"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Cidade</span>';
                form += '<select class="inputFormIncluir" id="cidade" name="cidade">';
                $.post('../controller/CidadesController.php',
                {
                    action: 'getCidades'
                },
                function(data) {
                    $.each( data, function(i,cidade){
```

```
                    if(colaborador.IdCidade == cidade.IdCidade){
                        form += '<option value="'+cidade.IdCidade+'" selected="true">'+cidade.Cidade+'-'+cidade.Sigla+'</option>';
                    }else{
                        form += '<option value="'+cidade.IdCidade+'">'+cidade.Cidade+'-'+cidade.Sigla+'</option>';
                    }

                });

                form += '</select>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Número</span>';
                form += '<input class="inputFormIncluir" type="text" id="Numero" name="Numero" value="'+colaborador.Numero+'" placeholder="Número"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>CEP</span>';
                form += '<input class="inputFormIncluir" type="text" id="CEP" name="CEP" value="'+colaborador.CEP+'" placeholder="CEP"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Telefone</span>';
                form += '<input class="inputFormIncluir" type="text" id="Telefone" name="Telefone" value="'+colaborador.Telefone+'" placeholder="Telefone"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Celular</span>';
                form += '<input class="inputFormIncluir" type="text" id="Celular" name="Celular" value="'+colaborador.Celular+'" placeholder="Celular"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Usuário</span>';
                form += '<input class="inputFormIncluir" type="text" id="Usuario" name="Usuario" value="'+colaborador.usuario+'" placeholder="Usuário"/>';
                form += '</div>';

                form += '<div class="divFormIncluir">';
                form += '<span><span class="fonteVermelha">*</span>Nova Senha</span>';
                form += '<input class="inputFormIncluir" type="password" id="Senha" name="Senha" value="" placeholder="Nova senha"/>';
                form += '</div>';

                form += '<div style="margin-left:14px;padding-top:190px; width:600px;">';
                form += '<br/>Observação';
                form += '<textarea style="margin-bottom:10px;width:770px" class="inputFormIncluir" id="Observacao" name="Observacao">';
                form += colaborador.Observacao;
                form += '</textarea>';
                form += '</div>';

                form += '<div>';
                form += '<button type="button" id="edtCliente" onclick="salvaEdicaoColaborador()">Salvar</button>';
```

```
                form += '<button type="button" id="cancelar"
onclick="getColaboradores()">Cancelar</button>';
                form += '</div>';
                form += '</div>';

                $('#conteudo').html(form);

                $("#Celular").mask("(99) 9999-9999");
                $("#Telefone").mask("(99) 9999-9999");
                $("#CEP").mask("99999-999");
                $("#CPF").mask("999.999.999-99");
            },"json");
        }, "json");
    }, "json");
});
}
```

Listagem 10 – Função "createFormEdicao"

A função apresentada na Listagem 10 tem como objetivo básico criar o formulário de edição do colaborador. Na Listagem 11 está um trecho de código da função "createFormEdicao" de relativa importância que serve como exemplo para a busca de campos dinâmicos do sistema, ou seja, a criação de campos a partir de resultados retornados de consulta a tabelas do banco.

```
            form += '<div class="divFormIncluir">';
            form += '<span><span class="fonteVermelha">*</span>Cidade</span>';
            form += '<select class="inputFormIncluir" id="cidade" name="cidade">';
            $.post('../controller/CidadesController.php',
            {
               action: 'getCidades'
            },
            function(data) {
               $.each( data, function(i,cidade){
                  if(colaborador.IdCidade == cidade.IdCidade){
                     form += '<option value="'+cidade.IdCidade+'"
selected="true">'+cidade.Cidade+'-'+cidade.Sigla+'</option>';
                  }else{
                     form += '<option value="'+cidade.IdCidade+'">'+cidade.Cidade+'-
'+cidade.Sigla+'</option>';
                  }

               });

               form += '</select>';
               form += '</div>';
```

Listagem 11 – Criação de *select* com campos da base de dados

Na Listagem 11 pode-se verificar que no meio da função "createFormEdicao" é executada uma requisição que busca as cidades cadastradas e as compara com a cidade do cadastro do colaborador valorizando o campo e inserindo as cidades para que possa ser selecionada uma delas para a edição. Esse método é utilizado em todos os campos dinâmicos do sistema.

Em relação à movimentação de agendas, na Listagem 12 é apresentada a função que cria os campos utilizados na movimentação.

```
function createCamposMovimentacao(idAgenda, idTipoMovimentacao) {
var Filtro = new Object();
Filtro.campo = 'IdAgenda';
Filtro.valor = idAgenda;

var FiltroJson = JSON.stringify(Filtro);
var campos = '';

$.post('../controller/AgendasController.php',
    {
        action: 'getAgendas',
        filtro: FiltroJson
    },
    function(data) {
      if(idTipoMovimentacao == '7'){
        $.each( data, function(i,agenda){
          campos +=  '<div class="divFormIncluir">';
          campos  +=  '<span><span  class="fonteVermelha">*</span>Colaborador origem:</span></br>';
          campos  +=  '<input  id="origem"  name="'+agenda.IdColaborador+'" class="inputFormIncluir" type="text" value="'+agenda.nome+'" readonly/>';
          campos +=  '</div>';

          campos += '<div class="divFormIncluir">';
          campos  +=  '<span  style="margin-top:2px;"><span class="fonteVermelha">*</span>Colaborador Destino:</span></br>';
          campos += '<select class="inputFormIncluir" id="destino" name="destino">';
          $.post('../controller/ColaboradoresController.php',
          {
              action: 'getColaboradores'
          },
          function(data) {
            $.each( data, function(i,colaborador){
              campos  +='<option value="'+colaborador.IdColaborador+'">'+colaborador.Nome+'</option>';
            });
            campos +=  '</select>';
            campos +=  '</div>';

            campos +=  '<div class="divFormIncluir">';
            campos +=  '<button style="float:right; margin-top:10px; margin-right:-3px;" type="button"  id="movimentaAgenda" onclick="salvaMovimentacao(\'7\',\"+agenda.IdAgenda+'\',\'IdColaborador\',origem.name,destino.value)">Salvar</button>';
            campos +=  '</div>';

            $('#conteudoMovimentacao').html(campos);
          },"json");
        });
      }else if(idTipoMovimentacao == '8'){
        $.each( data, function(i,agenda){

          $.post('agendaHoras.php',
          {
            idAgenda: agenda.IdAgenda
          },
          function(data) {
```

```
                    $('#conteudoMovimentacao').html(data);
                });
            });
        } else if(idTipoMovimentacao == '9'){
            $.each( data, function(i,agenda){
                campos += '<div class="divFormIncluir">';
                campos += '<span><span class="fonteVermelha">*</span>Status origem:</span></br>';
                campos += '<input id="origem" class="inputFormIncluir" type="text" name="'+agenda.IdAgendaStatus+'" value="'+agenda.agendastatus+'" readonly/>';
                campos += '</div>';

                campos += '<div class="divFormIncluir">';
                campos += '<span style="margin-top:2px;"><span class="fonteVermelha">*</span>Status Destino:</span></br>';
                campos += '<select class="inputFormIncluir" id="destino" name="destino">';
                $.post('../controller/AgendaStatusController.php',
                {
                    action: 'getAgendaStatus'
                },
                function(data) {
                    $.each( data, function(i,agendaStatus){
                      campos +='<option text="'+agendaStatus.AgendaStatus+'" value="'+agendaStatus.IdAgendaStatus+'">'+agendaStatus.AgendaStatus+'</option>';
                    });

                    campos += '</select>';
                    campos += '</div>';

                    campos += '<div class="divFormIncluir">';
                    campos += '<button style="float:right; margin-top:10px; margin-right:-3px;" type="button" id="movimentaAgenda" onclick="salvaMovimentacao(\'9\',\"+agenda.IdAgenda+'\',\'IdAgendaStatus\',origem.name,destino.value)">Salvar</button>';
                    campos += '</div>';

                    $('#conteudoMovimentacao').html(campos);
                },"json");
            });
        }else if(idTipoMovimentacao == '11'){
            $.each( data, function(i,agenda){
                campos += '<input type="hidden" id="origem" value="'+agenda.Descricao+'"/>';
                campos += '<div><br/><br/><br/>';
                campos += '<span style="margin-left:10px;"><span class="fonteVermelha">*</span>Descrição:</span></br>';
                campos += '<textarea id="destino" style="margin-bottom:10px;margin-left:15px;width:770px;height:300px;">';
                campos += agenda.Descricao;
                campos += '</textarea>';
                campos += '</div>';

                campos += '<div style="float:left;margin-top:-10px; margin-left:5px;margin-right:5px;">';
                campos += '<button style="float:right; margin-top:10px; margin-right:-3px;" type="button" id="movimentaAgenda" onclick="salvaMovimentacao(\'11\',\"+agenda.IdAgenda+'\',\'Descricao\',origem.value,destino.value)">Salvar</button>';
                campos += '</div>';

                $('#conteudoMovimentacao').html(campos);
```

```
                });
            }else if(idTipoMovimentacao == '10'){
                $.each( data, function(i,agenda){
                campos +=  '<div class="divFormIncluir">';
                campos  +=  '<span><span  class="fonteVermelha">*</span>Data  inicial
origem:</span></br>';
                campos  +=  '<input  id="origemDataInicial"  class="inputFormIncluir"
type="text" value="'+agenda.DataInicial+'" />';
                campos  +=  '<span><span  class="fonteVermelha">*</span>Data  inicial
destino:</span></br>';
                campos  +=  '<input  id="dataInicial"  class="inputFormIncluir"  type="date"
value="'+agenda.DataInicial+'" />';
                campos +=  '</div>';

                campos +=  '<div class="divFormIncluir">';
                campos  +=  '<span><span  class="fonteVermelha">*</span>Data  final
origem:</span></br>';
                campos +=  '<input id="origemDataFinal"  class="inputFormIncluir" type="text"
value="'+agenda.DataFinal+'" />';
                campos  +=  '<span><span  class="fonteVermelha">*</span>Data  final
destino:</span></br>';
                campos  +=  '<input  id="dataFinal"  class="inputFormIncluir"  type="date"
date="'+agenda.DataFinal+'" />';
                campos +=  '</div>';

                campos  +=  '<div  style="float:left;margin-top:-10px;  margin-left:5px;margin-
right:5px;">';
                campos  +=  '<button  style="float:right;  margin-top:10px;  margin-right:-3px;"
type="button"                                                          id="movimentaAgenda"
onclick="salvaMovimentacaoHoras('+agenda.IdAgenda+')">Salvar</button>';
                campos +=  '</div>';

                $('#conteudoMovimentacao').html(campos);
                });
            }

        },"json");
}
```

Listagem 12 – função "createCamposMovimentacao"

O formulário para a seleção do tipo de movimentação é criado da mesma maneira que os formulários de edição de outras telas do sistema. A diferença está na função "createCamposMovimentacao", dependendo do tipo de movimentação solicitado pelo usuário ela carrega os campos necessários para a movimentação da tarefa.

Esse comportamento faz com que a solicitação de novos tipos de movimentação pelo cliente exija a interação do programador na alteração da referida função.

5 CONCLUSÃO

Este trabalho teve como principal objetivo a implementação de um sistema para gerenciamento de agenda de visitas de profissionais de *software* para clientes. A linguagem utilizada foi o PHP e a implementação foi organizada seguindo o padrão de projetos MVC.

A linguagem PHP possui sintaxe simples e muitos recursos que facilitam e agilizam a implementação. Contudo, requer disciplina do programador porque a sua simplicidade permite que seja desenvolvido código desestruturado. Isso pode ocasionar problemas no entendimento do que foi implementado e de manutenção do código.

O recurso *JavaScript Object Notation* (JSON) utilizado em vários pontos do sistema faz com que a interação do usuário fique mais intuitiva e a navegação mais leve, pois o peso de processamento do servidor é transferido para o computador cliente, além de evitar várias requisições cliente-servidor simultâneas, que aumentam o tempo de carregamento.

Uma desvantagem notada durante o desenvolvimento foi a perca do recurso multi-janelas. Porém em uma versão futura isso poderá ser aplicado tendo em vista que uma solução já foi levantada.

A utilização do HTML5 faz com que a implementação seja muito mais simples do que a utilização somente do jQuery, realizado em implementações mais antigas. Como exemplo pode-se citar a manipulação de campos de data, transparências e cantos arredondados, que dependiam de um tempo hábil grande para serem implementados e agora são feitos por funções nativas do HTML 5 de maneira rápida e fácil.

Alguns requisitos do sistema foram alterados no decorrer do projeto e deverão ser adequados para a utilização do sistema pelo usuário final. Assim como a implementação de funcionalidades complementares deverão ser realizadas para atender aos interesses e necessidades dos futuros usuários do sistema.

Os objetivos propostos foram atendidos, pois o sistema permite controlar as horas realizadas pelos analistas e clientes. O que deve ser melhorado é o retorno dos dados em forma de informação útil para o setor financeiro, de faturamento e RH do cliente.

REFERÊNCIAS

BLAHA, Michael; JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2**. 2ª ed. Rio de Janeiro: Elsevier, 2006.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**, 2. ed. Addison-Wesley, 2006.

BURBECK, Steve. **Application programming in Smalltalk-80: how to use Model-View-Controller (MVC),** 1997, University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive. Disponível em: <http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>. Acesso em: 23 jun. 20134.

CHO, Euan S.; KIM, Soo D.; RHEW, Sung Y.; LEE, Sang D.; KIM, Chang G. **Object-oriented web application architectures and development strategies**. In: Asia Pacific Software Engineering Conference and and International Computer Science Conference 1997 (APSEC '97/ICSC '97), 1997, p. 322 – 331.

CHOLAKOV, Nikolaj. **On some drawbacks of the PHP platform**. In: International Conference on Computer Systems and Technologies (CompSysTech'08), 2008, p. II.7-1 - II.7-6.

MAGELA, Rogério. **Engenharia de software aplicada. Fundamentos**. Rio de Janeiro: Alta Books, 2006.

MCHEICK, Hamid; YAN Qi. **Dependency of components in MVC distributed architecture**. In: 24th Canadian Conference on Electrical and Computer Engineering (CCECE), p. 691-694, 2011.

MICROSOFT. **Microsoft, MSDN, Model-View-Controller**. Disponível em: <http://msdn.microsoft.com/en-us/library/ff649643.aspx>. Acesso em 25 jul. 2014.

PACIEVITCH, Thais. **Analista de sistemas**. Disponível em: <http://www.infoescola.com/profissoes/analista-de-sistemas/>. Acesso em: 15 out. 2013.

QIU, Xiaohong; PALLICKARA, Shrideep; UYAR, Ahmet. **Making SVG a web service in a message-based MVC architecture**, 2004. Disponível em: <http://surface.syr.edu/cgi/viewcontent.cgi?article=1121&context=eecs>. Acesso em: 24. Jul. 2014.

SHEGALOV, German; WEIKUM, Gerhard. **EOS2: unstoppable stateful PHP**. In: 32nd International Conference on Very Large Data Bases (VLDB 2006), 2006, p. 1223-1226.

WANG, Guanhua. **Application of lightweight MVC-like structure in PHP**. In: International Conference on Business Management and Electronic Information (BMEI), v. 2, 2011, p. 74-77.