UNIVERSIDADE FEDERAL DE SANTA CATARINA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SGD – SISTEMA DE GESTÃO DE DOCENTES

Florianópolis
2010

BRUNO HENRIQUE FERRONATO

SGD – SISTEMA DE GESTÃO DE DOCENTES

Monografia apresentada ao Departamento de Informática e de Estatística (INE), no curso de Sistemas de Informação da Universidade Federal de Santa Catarina, como requisito para a obtenção do diploma de Bacharel em Sistemas de Informção.

Orientador: Fernando Augusto da Silva Cruz

FLORIANÓPOLIS

2010

Universidade Federal de Santa Catarina
Bacharelado em Sistemas de Informação
Centro de Tecnologia

Trabalho de Conclusão de Curso de Bacharelado em Sistemas de Informação intitulado SGD – Sistema de Gestão de Docentes, de autoria de Bruno Henrique Ferronato, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Nereu Estanislau Burin
UFSC

Prof. Dr. João Bosco Mangueira Sobral
UFSC

Prof. Dr. Fernando Augusto da Silva Cruz
UFSC

Prof. Dr. Maria Marta Leite
Coordenador(a) do Curso de Sistemas de Informação

Data de aprovação: Florianópolis, Outubro de 2010

Endereço - Florianópolis, SC - CEP - Brasil - Tel.: (48) 3721-9424 - Fax: (48) 3721-9566

RESUMO

Neste trabalho é descrito o projeto e a implementação de um sistema para a gestão de docentes da Universidade Federal de Santa Catarina. O sistema foi desenvolvido para atender as necessidades da Comissão Permanente de Pessoal Docente, com a intenção de automatizar os seus processos rotineiros. Com isso foi desenvolvido um sistema que possibilita a centralização das informações dos professores para que facilmente seja possível gerar os relatórios de qualificação do professor, fazer consultas a dados cadastrais entre outros. O sistema foi desenvolvido para ser utilizado através da Internet utilizando tecnologias consolidadas e gratuítas, não tendo nenhum gasto com licenças proprietárias. Com o sistema implementado obteve-se um grande ganho de tempo e esforço nos trabalhos que eram executados manualmente e agora são realizados através do sistema.

Palavras-chave: Informatização, implementação, gestão.

ABSTRACT

This work describes the design and implementation of a system for the management of professors at the Federal University of Santa Catarina. The system was developed to meet the needs of the CPPD (Comissão Permanente de Pessoal Docente da UFSC), with the intention to automate their routine processes. Thus, we developed a system that centralizes the information for professors so that is possible easily generate the reports concerning the qualifications of the professor, to consult the registration information and other attributes. The system uses the Internet through technologies consolidated and gratuitous, and none spent on proprietary licenses. With the implemented system we obtained a large gain in time and effort in the works that were performed manually and are now performed through this system.

Palavras-chave: Computerization, implementation, management.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

CSS	Cascading Style Sheets (folha de estilos em cascata)
HTML	HyperText Markup Language (Linguagem de Marcação de Hipertexto)
JSON	JavaScript Object Notation (Notação de Objetos JavaScript)
MVC	Model view controller (Modelo Visão e controle)
PHP	Hypertext Preprocessor (Software como Serviço)
AJAX	Asynchronous Javascript And XML (Javascript e XML Assíncrono)
DOM.	Document Object Model (Modelo de Objeto de Documentos)
SGBD	Sistema de gerenciamento de banco de dados

SUMÁRIO

INTRODUÇÃO

A CPPD foi Constituída através do Decreto nº 94664/87 e regulamentada pela Portaria nº 475/87 do Ministério da Educação, para assessorar aos Orgãos Deliberativos Centrais na formulação, aperfeiçoamento e modificação da política de pessoal docente da UFSC. Está vinculada a Pró-Reitoria de Ensino de Graduação da UFSC.

Os assuntos tratados manualmente pelos funcionários da CPPD, inerentes aos processos de professores, tem se tornado bastante difícil devido ao crescimento da quantidade de processos a cada ano, e ao grande número de documentos necessários para o controle de uma progressão funcional.

Como solução, este trabalho propõe a criação de um sistema cliente-servidor, denominado SGD – Sistema de Gestão de Docentes, para disponibilizar o acesso rápido, *online,* de informações relativas aos processos tramitando na CPPD.

1.1 Tema

Desenvolvimento de um sistema que possibilite a informatização do processo de gestão das progressões funcionais dos docentes da UFSC.

1.2 Problema

Dentre as muitas e variadas razões que justificam a opção por um sistema informatizado de gerenciamento, Rowley (1994), aponta:

- Os computadores possibilitam a redução do número de tarefas repetitivas.
- Em geral, os dados serão inseridos uma única vez e, daí em diante, poderão ser acessados e modificados.

- Os sistemas informatizados podem ser mais baratos e mais eficientes;
- Podem propiciar a introdução de serviços que não existiam antes;
- Controle adicional de todas as funções que se consegue com a ajuda de informações gerenciais mais abrangentes que justificam um processo decisório mais eficaz.

De uma forma ampla, a finalidade da informatização é agilizar e aumentar a eficiência e a precisão na recuperação da informação.

Para tornar isso possível e modernizar a forma de trabalho, que hoje é toda feita manualmente, é que foi pensado no desenvolvimento do sistema SGD.

1.2.1 Objetivo geral

Organizar e agilizar os assuntos inerentes tratados pela CPPD, em relação aos processos de progressões funcionais dos professores da UFSC.

1.2.2 Objetivos específicos

- Desenvolver um sistema capaz de realizar a gerência dos docentes e suas progressões funcionais de maneira automatizada agilizando o processo de mudança de progressões funcionais dos professores.
- Permitir o cadastro de professores.
- Permitir a geração de relatórios de qualificação do professor.
- Permitr a pesquisa de professores por id, nome, número do SIAPE, matrícula UFSC e departamento.

1.3 Justificativa

A justificativa do trabalho se deve ao fato que atualmente todo o controle de uma progressão funcional de um professor ser feito manualmente, tornando o processo lento e burocrático.

A criação de um sistema para gerência desse processo tornará o processo mais ágil e facilitará o acompanhamento tanto do gestor quanto do professor que solicita sua progressão funcional.

Desenvolver estudos que permitam fornecer subsídios para fixação, aperfeiçoamento e modificação da política de pessoal e de seus instrumentos.

A progressão funcional é o crescimento funcional do servidor estável no exercício do cargo de provimento efetivo, nos níveis e referências do cargo, na classe da carreira, ou na carreira, conforme o plano de cargos ou carreira e vencimentos estabelecido para o órgão ou entidade, estruturado de forma vertical e horizontal, fundamentado na qualificação e no desempenho profissional.(BURIN et al., s.d.).

Através dos processos bienais de Progressão, os servidores são objetivamente avaliados pelos critérios de merecimento e antiguidade, para o preenchimento de um número de vagas pré-definidas para cada referência da classe de cargo.

Existem vários tipos de progressões funcionais. No que segue, listamos algumas delas.

- Estágio probatório
- Progressão funcional em estágio probatório
- Progressão funcional horizontal
- Progressão funcional vertical

- Reposicionamento
- Alteração de regime de trabalho 3º grau
- Alteração de regime de trabalho 1º e 2º graus
- Afastamento para pós-graduação
- Titulação
- Licença capacitação

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta as principais tecnologias necessárias ao desenvolvimento deste trabalho, sendo elas as liguagens de programação PHP, *javascript*, através do *framework jquery*, a utilização de um sistema de *templates*, o *Smarty*, a linguagem de estilos CSS e as ferramentas de análise e modelagem como o DBDesigner para o banco de dados e o JUDE para diagramas UML.

O capítulo também apresenta a arquitetura cliente-servidor, que segundo a wikipedia é um modelo computacional que separa clientes e servidores, sendo interligados entre si geralmente utilizando-se uma rede de computadores.

Por fim o capítulo aborda a questão dos *Design Patterns* que são soluções para problemas frequentes no desenvolvimento de sistemas orientados a objetos.

2.1 PHP

PHP (um acrónimo recursivo para "PHP: Hypertext Preprocessor") é uma linguagem de programação de computadores interpretada, livre e muito utilizada para gerar conteúdo dinâmico na World Wide Web.

A linguagem de programação PHP foi criada em 1994 por Rasmus Lerdorf. No início era formada por um conjunto de scripts voltados à criação de páginas dinâmicas que Rasmus utilizava para monitorar o acesso ao seu currículo na Internet. À medida que essa ferramenta foi crescendo em funcionalidades, Rasmus teve de escrever uma implementação em C, a qual permitia às pessoas desenvolverem de forma muito simples suas aplicações para web. Rasmus nomeou essa versão de PHP/FI (Personal Home Pages/Forms Interpreter) e decidiu disponibilizar seu código na web, em 1995, para compartilhar com outras pessoas, bem como receber ajuda e correção de bugs (DALL'OGLIO, 2007, p.14).

É uma linguagem de distribuição livre de código aberto, assim, qualquer pessoa pode usar, alterar e redistribuir para a comunidade.

2.2 Javascript

JavaScript é uma linguagem de programação criada por Brendan Eich para a Netscape em 1995, que a princípio se chamava LiveScript. A Netscape, após o sucesso inicial desta linguagem, recebeu uma colaboração considerável da Sun, a qual permitiu o uso do nome JavaScript para alavancar o Livescript, porém as semelhanças entre a linguagem da Sun o Java e o Javascript são quase nulas.

A linguagem foi criada para atender, principalmente, às seguintes necessidades:

- Validação de formulários no lado cliente (programa navegador);
- Interação com a página.

2.3 jQuery

jQuery é um *framework* para ajudar os desenvolvedores a se concentrarem na lógica dos sistemas da web e não nos problemas de incompatibilidade dos navegadores atuais.

Seu lema é: "Escrever menos e fazer mais".

Possui inúmeras vantagens em relação a escrita pura de javascript, entre elas estão o suporte a múltiplos navegadores, a reutilização de código através dos plugins, e a reduçao no número de linhas para a escrita de código. jQuery trabalha com AJAX (WIKIPEDIA) e DOM (WIKIPEDIA), implementa recursos de CSS (SILVA, Maurício).

2.4 Smarty

O Smarty é uma classe de *templates*. Funciona de uma forma que separe interface da lógica de programação e tem por objetivo, facilitar e melhorar o desenvolvimento de qualquer aplicação em PHP.

Por ser muito difundido no mundo inteiro, e estar ligado ao site oficial do PHP, o Smarty tem uma comunidade grande de desenvolvedores. Isso ajuda no suporte e discussão de melhorias.

Entre suas principais funcionalidades estão:

- Melhor desempenho de execução.
- Acaba com overhead de template, pois compila apenas uma vez.
- É extensível, ou seja, pode-se criar suas próprias funções,.
- Pode–se configura a sintaxe da *tag* de delimitação do template.
- Uso de funções condicionais de forma simplificada dentro dos templates.
- Sem limitação para *loops*, *If*, e outras estrutras lógicas.
- Modo de *cache* embutido.
- Arquitetura de *plugins*.

2.5 CSS

Cascading Style Sheets (ou simplesmente CSS) é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Ao invés de colocar a formatação dentro do documento, o desenvolvedor cria um *link* (ligação) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal, basta modificar apenas um arquivo.

2.6 DBDesigner

É uma ferramenta *CASE* para a modelagem de dados que trabalha com o modelo lógico, desenvolvida pela fabFORCE (www.fabforce.net) sob a licença GNU GPL (*General Public License*). É um software multiplataforma (Windows 2k/XP e Linux KDE/GNOME) implementado em Delphi/Kylix. Além de permitir a modelagem, criação e manutenção de bancos de dados, esta ferramenta possibilita também a engenharia reversa, gerando o modelo de dados a partir de um banco existente, e ainda possibilita o sincronismo entre o modelo e o banco.

Foi construída originalmente para oferecer suporte ao MySQL, porém oferece também suporte à engenharia reversa e sincronização a outros SGBDs como Oracle, SQL Server, SQLite e outros que permitam acesso via ODBC (*Open Database Connectivity*).

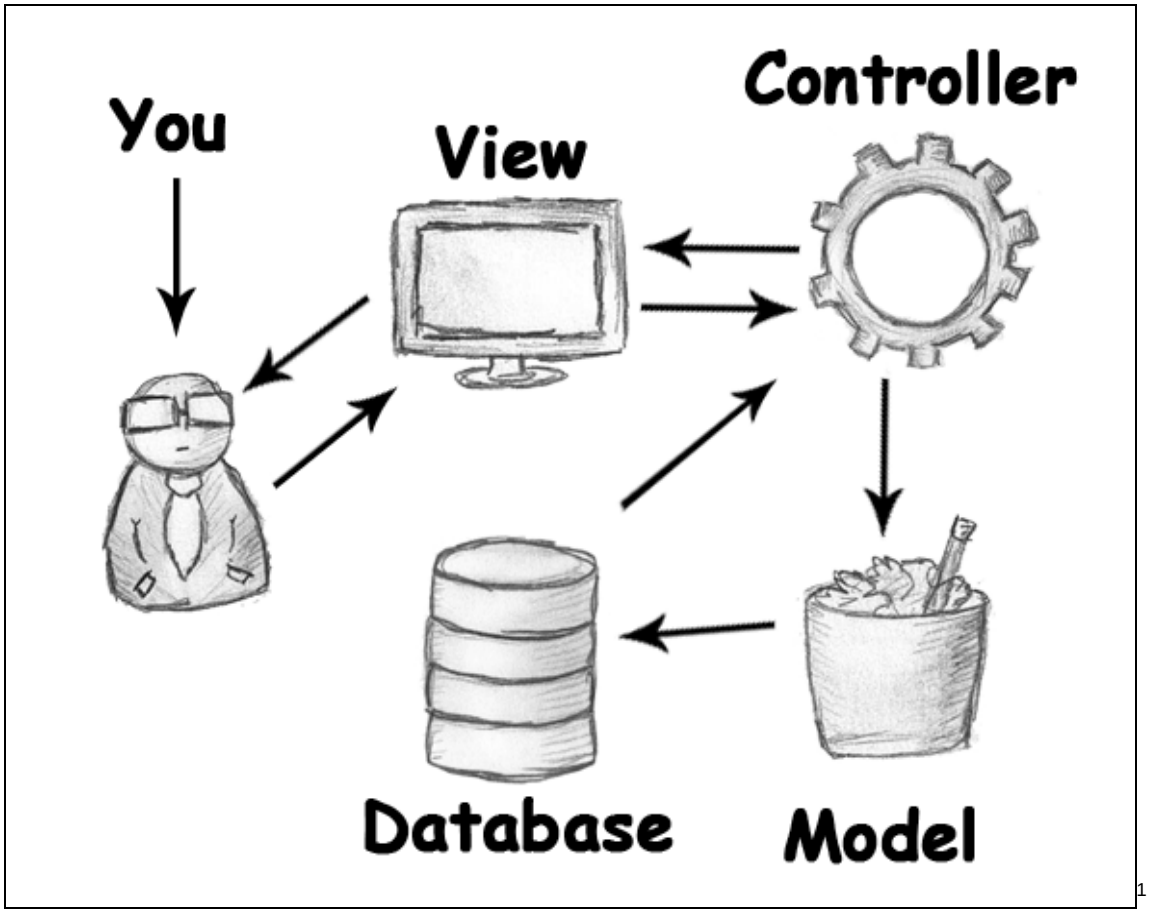
2.7 JUDE

JUDE é uma ferramenta para projetar sistemas (JUDE). É apropriado para uso em negócios, grandes modelos e criação de documentos. Provê diagramas UML2.0 e diagramas adicionais, funcionalidade de impressão melhorada, habilidade de fazer *merge* com outros projetos JUDE, especificações de caso-de-uso, *input-output* de modelos de/para arquivos XML, funções de copiar e colar em formato de vetor(EMF), e exportação de informações do projeto em formato CSV.

2.8 MVC (Model-view-controller)

Segundo a Wikipedia, MVC é um padrão de arquitetura de software que visa a separação da lógica de negócio da lógica de apresentação, permitindo o desenvolvimento, teste e manutenção em componentes isolados.

Para exemplificar, a figura 1 mostra o fluxo de funcionamento das requisições realizadas e como elas interagem dentro do sistema.



[1]

Figura 1 - Fluxo de funcionamento do modelo MVC

[1] Imagem retirada de: ROCKET, Ror. Rails MVC architecture – Explained the easy way. apr. 2010. Disponível em: <http://rorrocket.com/2010/04/17/the-rails-mvc-architecture-explained-the-easy-way/>. Acesso em: 11 nov. 2010

2.9 UML - LINGUAGEM DE MODELAGEM UNIFICADA

Basicamente, a UML permite que desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significados, isto é, semântica. É uma notação independente de processos, embora o RUP (*Rational Unified Process*) tenha sido especificamente desenvolvido utilizando a UML

Abaixo são listados os pincipais diagramas UML1.4 e UML2.0

- **Diagrama de Classes:**

  Descreve a estrutura de um sistema, mostrando o sistema de classes, seus atributos e os relacionamentos entre as classes.

- **Diagrama de Casos de Uso:**

  Descreve a funcionalidade fornecida por um sistema em termos de atores, seus objetivos representados como casos de uso, e as dependências entre os casos de uso.

- **Diagrama de Estados:**

  Diagramas de Estados mostram os diferentes estados de um objeto durante sua vida, e o estímulo que faz com que o objeto mude seu estado.

- **Diagrama de Atividades:**

  Descreve o negócio passo-a-passo, fluxos operacionais de componentes em um sistema. Um diagrama de atividades mostra o fluxo global de controle do sistema.

- **Diagrama de Seqüência:**

Mostra como objetos se comunicam uns com os outros em termos de uma seqüência de mensagens. Também indica a expectativa de vida dos objetos em relação a essas mensagens.

- **Diagrama de Comunicação(Colaboração):**

  Normalmente é utilizado como complemento do diagrama de seqüência, porém possui um enfoque diferente, concentrando-se em como os objetos estão vinculados através de mensagens.

- **Diagrama de Componentes:**

  Ilustra como as classes deverão se encontrar organizadas através da noção de componentes de trabalho.

2.10 Cliente-Servidor

O padrão de arquitetura Cliente/Servidor é um padrão muito utilizado nas mais diversas aplicações. O princípio básico de uma arquitetura Cliente-Servidor é a separação da arquitetura em dois lados: o cliente e o servidor. O cliente é responsável pela *interface* com o usuário, sendo que o usuário poderá através desta *interface* solicitar a execução de serviços no servidor. O servidor trabalha em função destas solicitações do cliente, ou seja, ele é um simples executor de serviços solicitados. Pode-se dizer, de modo geral, que o cliente é a parte ativa, enquanto o servidor é a parte passiva.

Para um cliente solicitar um serviço ao servidor, é necessário haver um meio de comunicação. Este meio de comunicação é normalmente estabelecido pela rede de computadores. É possível, também, ter um cliente em execução no próprio servidor. O modelo de solicitação de um serviço deve ser conhecido pelo servidor e pelo cliente. Isto permite uma troca de mensagens entre o servidor e seus clientes.

Logo, havendo um cliente, um servidor com serviços e um meio de comunicação entre o cliente e o servidor, está estabelecida uma arquitetura Cliente-

Servidor. O padrão de arquitetura Cliente-Servidor é utilizado em serviços de bancos de dados, aplicações de *groupware*, Internet e outros (SILVA, 2009).

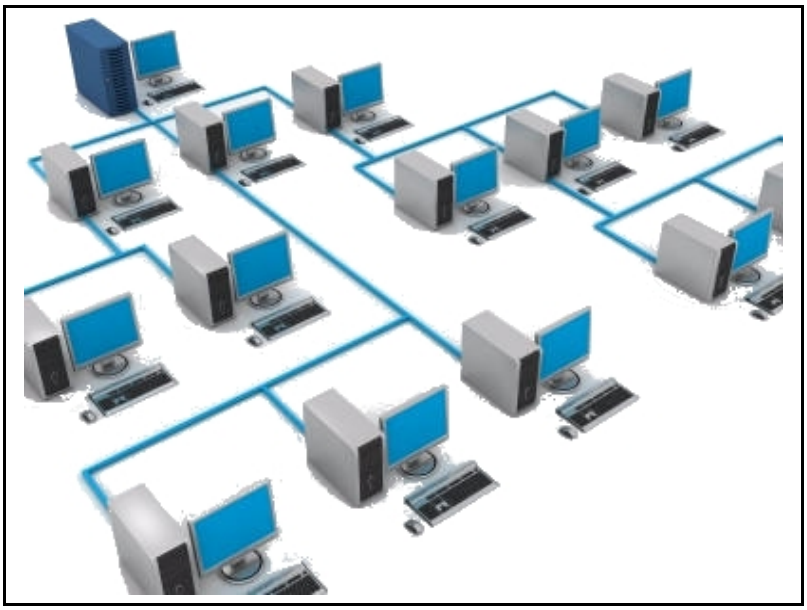


Figura 2 - Representação da arquitetura cliente-servidor[2]

[2] Imagem retirada de: STROPARO, Elder. Cliente-servidor. mai. 2010. Disponível em: <http://elderstroparo.blogspot.com/2010/05/cliente-servidor.html>. Acesso em: 07 nov. 2010

2.11 Design patterns

Design Pattern, que foi traduzido como padrões de projeto para português, são soluções padrão para determinados problemas recorrentes no desenvolvimento de sistemas de software orientados a objetos. Um padrão de projeto estabelece um nome e define o problema, a solução, quando aplicar esta solução e suas conseqüências.

Os padrões de projeto visam facilitar a reutilização de soluções de desenvolvimento, isto é, soluções na fase de projeto do software, sem considerar reutilização de código. Também acarretam um vocabulário comum de desenvolvimento, facilitando a comunicação, documentação e aprendizado dos sistemas de software (WIKIPEDIA).

A seguir é são apresentados os principais padrões de projetos divididos por famílias.

- **Padrões de criação**
    - *Abstract Factory*
    - *Builder*
    - *Factory Method*
    - *Prototype*
    - *Singleton*
- **Padrões estruturais**
    - *Adapter*
    - *Bridge*

- *Composite*
- *Decorator*
- *Façade*
- *Flyweight*
- *Proxy*

- **Padrões comportamentais**
  - *Chain of Responsability*
  - *Command*
  - *Interpreter*
  - *Iterator*
  - *Mediator*
  - *Memento*
  - *Observer*
  - *State*
  - *Strategy*
  - *Template Method*
  - *Visitor*

- **Padrões GRASP**
  - *Controller*
  - *Creator*

- *Expert*
- *Law of Demeter*
- *Low Coupling / High Coehsion*
- *Polymorphism*
- *Pure Fabrication*

3 METODOLOGIA DE DESENVOLVIMENTO

O sistema SGD surgiu com a necessidade de se informatizar os processos rotineiros realizados na manipulação de processos de pedidos de progressões funcionais realizados pelos professores da UFSC à CPPD.

Foi criado inicialmente como trabalho de disciplina de integração curricular dos alunos Leandro Gobbo & Aloysio Nandi Tiscoski, utilizando *webservice* para o cadastro e visualização das informações pertinentes ao sistema e aos professores.

Após um estudo preliminar do sistema existente e depois de minunciosa análise, verificou-se que o sistema foi concebido segundo o modelo cliente/servidor utilizando PHP como linguagem e Mysql como SGBD.

Após isso, verificou-se que seria necessário uma mudança de paradigma, de um sistema estruturado para um sistema orientado a objetos. Tal mudança seria bastante significativa, porém tornaria o sistema mais modular e mais fácil de manutenção, além de todas as vantagens da orientação a objetos.

Para o banco de dados, fez-se um trabalho de engenharia reversa para criar o modelo entidade-relacional. Todavia, os interrelacionamentos não foram extraídos através da ferramenta DBDesigner. De posse das tabelas, fez-se a análise dos relacionamentos entre as mesmas e recuperou-se os interrelacionamentos entre essas.

Uma vez feito isso, foram sendo acrescentadas as tabelas ao modelo entidade-relacional, para dar suporte ao novo modelo de banco de dados. Isto acarretou, praticamente, no desenvolvimento de um novo modelo de banco de dados. A figura 3 mostra uma visão parcial do modelo de tabelas vigentes.

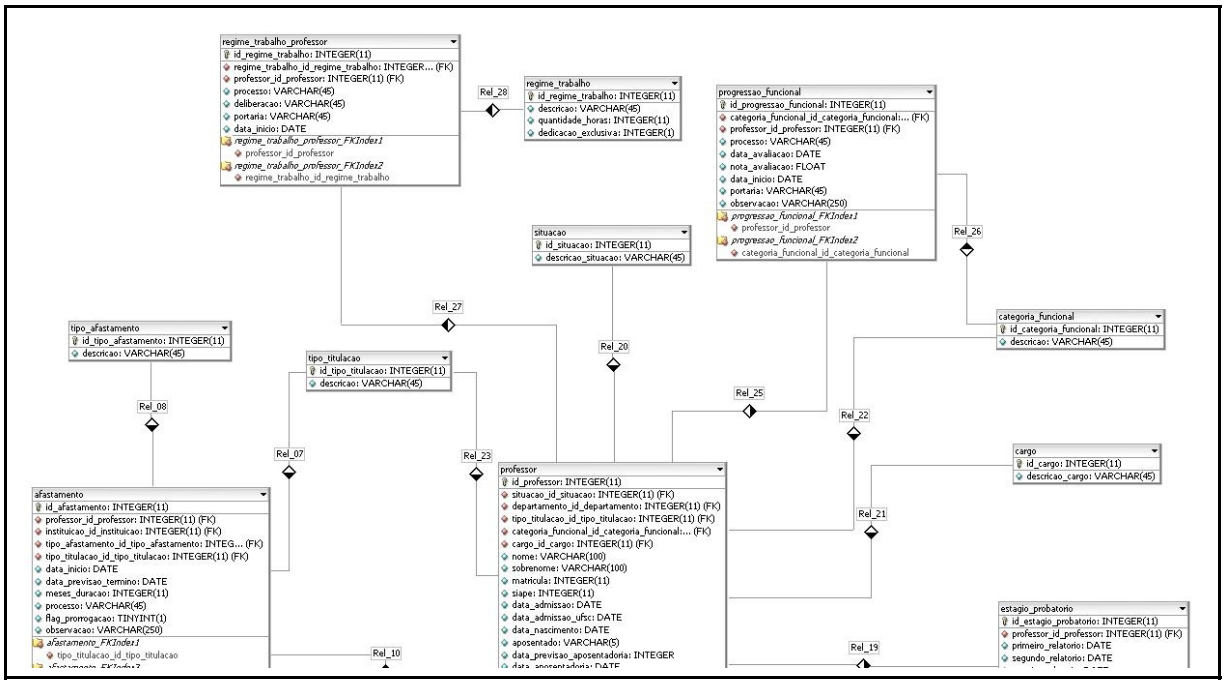


Figura 3 - Visão parcial do modelo entidade-relacionamento.

Uma vez pronta a modelagem do banco de dados, partiu-se para a modelagem do sistema orientados a objetos, usando o paradigma norteado pelo padrão MVC. Este padrão proporcionou uma separação mais clara de todas as funcionalidades existentes no sistema. A figura 4 mostra a estrutura dos diretórios do sistema.

Figura 4 - Estrutura de diretórios do sistema

Desse modo, sobre a ótica desse padrão, todas as funcionalidades por hora existentes foram reescritas com base no padrão supracitado.

Como linguagem de suporte do sistema, optou-se pela linguagem Javascript com auxílio do *framework* Jquery.

O *framework* Jquery foi escolhido devido a uma série de vantagens, entre elas tamanho, velocidade, acesso direto a qualquer componente DOM, grande variedade de efeitos e animações, uso simplificado de AJAX, cross-browser, suporte a *plugins* entre outras.

Um gerenciador de *templates* também foi adicionado. No caso o Smarty para a organização do sistema e a separação do código, dessa forma mantendo a parte visual separada da parte lógica da aplicação.

3.1 Fluxo de funcionamento do sistema

A seguir iremos explicar como é o funcionamento das requisições dentro do sistema.

Quando o sistema acaba de carregar todo o conteúdo HTML do arquivo index.php, são carregados os arquivos Javascript que serão responsáveis pela “escuta” dos eventos efetuados no sistema. Um evento pode ser um clique em um botão, acesso a um *link*, alteração de valor de um campo de seleção entre outros.

Quando esse evento é disparado, geralmente, uma requisição é enviada ao *FrontController*, o qual irá decidir qual o *Controller* que é responsável por atender àquela requisição.

O *Controller* por sua vez, se necessário, fará uma requisição ao seu *Model* que executará uma consulta ao banco de dados, e retornando o conteúdo solicitado ao *Controller* que a chamou. De posse dos dados solicitados, o *Controller* irá chamar a sua *View* para fazer a apresentação desses dados ao usuário.

A figura 5 mostra como se dá o fluxo das requisições entre os diversos arquivos do sistema SGD.

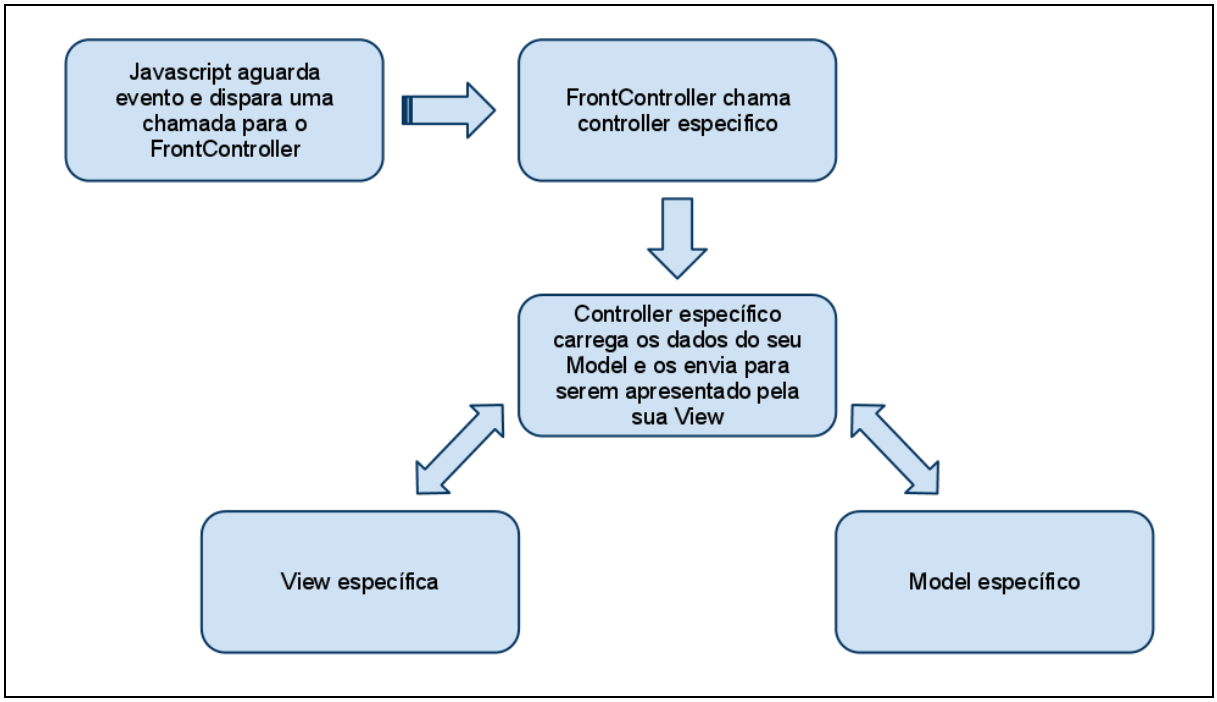


Figura 5 - Fluxo de requisições no sistema SGD

3.2 Requisitos

De um modo geral, os requisitos de um sistema podem ser divididos em dois grandes grupos: **requisitos funcionais** e **requisitos não-funcionais**.

Os requisitos funcionais descrevem como o sistema funciona, é aquilo que deve ser feito pelo sistema. São a descrição das diversas funções que os usuários querem ou precisam que o software faça.

Os requisitos não-funcionais dizem como determinada tarefa deve ser feita. Estão relacionados com padrões de qualidade como confiabilidade, performance, e robustez.

3.2.1 Funcionais

Permite o cadastro de novos países, UF's, municípios, instituições, centros, departamentos, tipos de afastamentos de professores, tipos de titulações, categorias funcionais e regimes de trabalhos.

Permite o cadastro de um novo professor, do seu regime de trabalho, do seu afastamento e de sua progressão funcional.

Emite relatório de centros, departamentos, professores, diretores dos centros, professores por departamento e departamento por centro.

Permite a alteração dos dados cadastrais, assim como a senha de acesso do usuário logado no sistema.

3.2.2 Não funcionais

O sistema deve funcionar nos navegadores Firefox e Google Chrome.

O tempo de resposta do sistema não deve ultrapassar 10 segundos.

O sistema deverá disponibilizar para o usuário recurso de visualização do processamento que está sendo executado.

Somente usuários com credenciais de acesso poderão ter acesso ao sistema.

O sistema deve ser fácil de usar. As operações devem ser ágeis e consumir pouco tempo de processamento.

3.3 Padroes de codificação

No sistema SGD, alguns padrões de codificação foram adotados para uma melhor legibilidade e organização do código.

Para a escrita de variáveis, foi utilizado o padrão *camelCase*, a identação do código é feita através de *tabs,* cada qual conténdo quatro espaços. Todas as funções escritas em PHP possuem comentários no formato utilizado pelo *phpDoc*, o qual poderá ser utilizado para documentar o sistema.

Para a escrita das funcoes em Javascript, procurou-se escrever as funções como sendo métodos de objetos dessa maneira, temos um maior controle sobre o fluxo da execução dos métodos.

A nomenclatura das classes de controle terminam com a letra ‘C’ maiúscula, as de modelo terminam com a letra ‘M’ e as de visão com a letra ‘V’.

3.4 Ferramentas

Para o desenvolvimento do sistema foram utilizadas diversas ferramentas, entre as principais, citamos abaixo:

PhpMyAdmin - É um programa desenvolvido em PHP para administração do MySQL pela Internet. A partir deste sistema é possível criar e remover bases de dados, criar, remover e alterar tabelas, inserir, remover e editar campos, executar códigos SQL e manipular campos-chaves. O phpMyAdmin é muito utilizado por programadores web que muitas vezes necessitam manipular bases de dados.

Mysql Query Browser - O MySQL Query Browser é uma ferramenta gráfica fornecida pela MySQL AB para criar, executar e otimizar solicitações SQL em um ambiente gráfico. Assim como o MySQL Administrator foi criado para administrar um

servidor MySQL, o MySQL Query Browser foi criado para auxiliar você a selecionar e analisar dados armazenados dentro de um banco de dados MySQL.

Enquanto todas as solicitações executadas no MySQL Query Browser também podem ser executadas pela linha de comando, utilizando-se o utilitário mysql, o MySQL Query Browser permite a execução e edição dos dados de maneira gráfica, que é mais intuitiva para o usuário. Foi projetado para trabalhar com versões 4.0 ou superiores do servidor MySQL.

Zend Studio - O Zend Studio é um IDE (*Integrated Development Environment*) comercial e proprietário para PHP desenvolvido pela Zend Tecnologia, baseado no PDT, *plugin* de desenvolvimento PHP do Eclipse em PDT (o projeto PDT é liderado pela Zend).

É integrado com o Zend Server e Zend Framework, permitindo aos desenvolvedores usufruir de um ambiente PHP completo.

FileZilla – É um cliente FTP, SFTP e FTPS de código livre para Microsoft Windows e GNU/Linux. É distribuído em licença GNU General Public License.

Começou como um projeto de estudo em ciência da computação por Tim Kosse e dois colegas na segunda semana de janeiro de 2001.

FireBug - É uma extensão para o Mozilla Firefox que adiciona ao navegador inúmeras ferramentas para facilitar a tarefa de desenvolvimento de páginas web. Possibilita a identificação e eliminação de erros de programação, edição e também o monitoramento de CSS, HTML e JavaScript presentes em qualquer página da internet.

Para quem cria *sites* e utiliza o Firefox, pode ter a facilidade de realizar todas as inspeções em qualquer página *web*, sendo ainda um recurso interessante para quem está iniciando com a marcação HTML, já que é possível entrar em um *site* específico e verificar a marcação utilizada nele.

WAMP – É o acrônimo para a combinação: Windows, Apache, MySQL e PHP. É o termo usado para denominar os *softwares* que efetuam a instalação

automática de vários *softwares* de forma que facilitem e agilizem a instalação dos mesmos.

Em geral é usado WAMP para dizer que é um instalador de Apache, Mysql e PHP para Windows, sendo denominados como LAMP os *softwares* que tem a mesma destinação para sistemas operacionais LINUX e MAMP para Macintosh.

Google Code - É um sistema da companhia Google para interesse de programadores em desenvolvimento de softwares. O *site* contém código-fonte aberto e uma lista de serviços que suportam a API pública do Google.

Vale ressaltar que o Google Code é de suma importância para o desenvolvimento do sistema SGD, pois oferece todo o suporte necessário para o gerenciamento do sistema.

A seguir apresentaremos as principais vantagens disponibilizadas pelo sistema de gerenciamento de projetos mencionado acima.

- Disponibiliza um repositório para o armazenamento do código-fonte do sistema incluindo controle de versão. Essa funcionalidade é bastante útil, pois dessa forma podemos manter o histórico da evolução do sistema, como também serve como um *backup* do código. Também facilita o compartilhamento do codigo, ao oferecer um endereço de onde é possível fazer o seu *download*.

- Permite o desenvolvimento colaborativo, dessa forma vários desenvolvedores podem trabalhar no mesmo projeto, de forma descentralizada, mantendo o projeto sempre atualizado, com rápida correções de *bugs* e aumento na qualidade do código desenvolvido.

- Sistema de *Issues*, os quais são os problemas encontrados no sistema ou novas funcionalidades. Pode ser alimentado pelos desenvolvedores ou pelos próprios usuários do sistema em desenvolvimento.

- Sistema de *Wiki*, o qual pode ser usado para criar a documentação do sistema, tutoriais ou qualquer outra informação relevante na utilização do sistema que se está desenvolvendo.

- Oferece espaço para que se possam disponibilizar arquivos de *downloads*, atualmente, disponibiliza 2048Mb de espaço para esses arquivos.

- Permite a pesquisa no código-fonte, assim, de maneira rápida e *online*, podemos fazer uma busca em todo o código-fonte do sistema para encontrar determinado termo.

- Gerenciamento de papéis de usuários no sistema, dessa forma podemos definir o que cada participante de desenvolvimento pode alterar.

- Navegação *online* na estrutura de arquivos e diretórios do sistema, não sendo necessário dessa maneira, fazer o *download* do código-fonte para realizar essa tarefa.

- Integração do sistema de gerenciamento do projeto com o Google Analitics, dessa maneira, podemos ter um relatório geral e atualizado da utilização do sistema.

- Integração com o Google Groups, oferecendo um ambiente para discussões relativas ao sistema.

Subversion - Também conhecido por SVN é um sistema de controle de versão desenhado especificamente para ser um substituto moderno do CVS, o qual se considera ter algumas limitações.

A seguir, algumas telas das principais ferramentas utilizadas. A figura 6 mostra a tela principal de codificação da ferramenta Zend Studio.

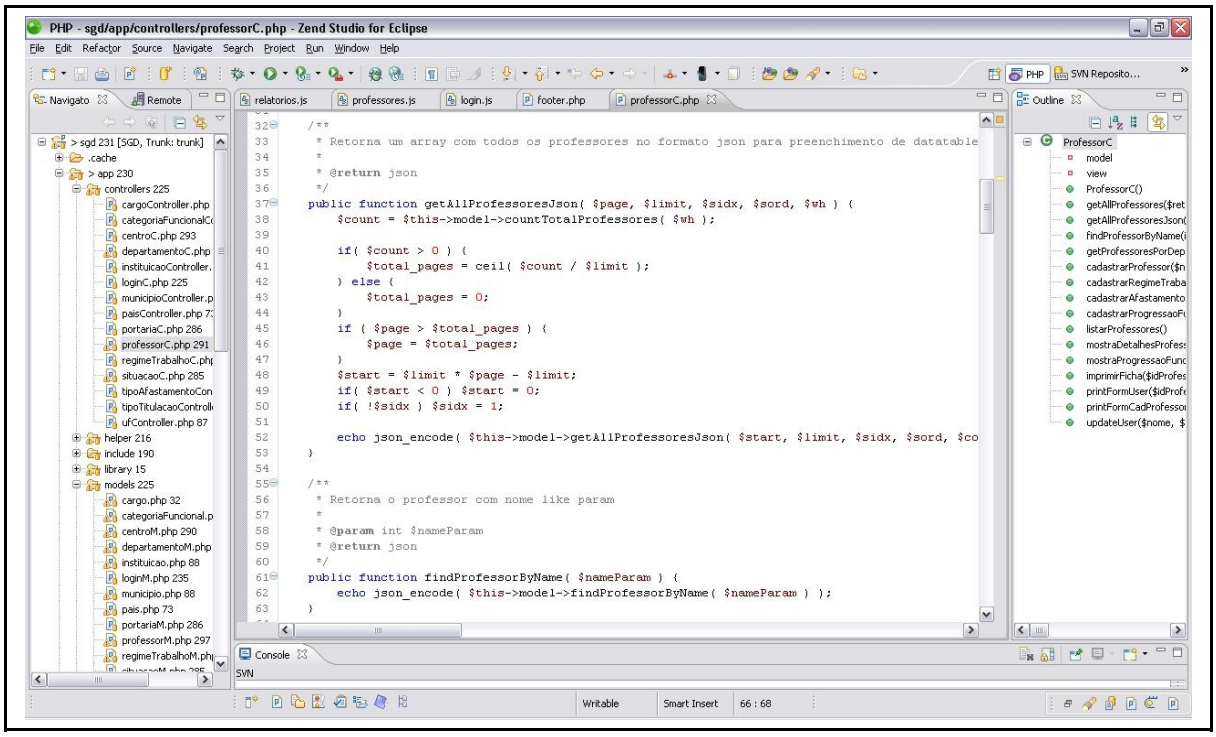


Figura 6 - Zend Studio

A figura 7 mostra a tela principal da ferramenta DBDesigner com uma visão ampla do modelo do banco de dados do sistema SGD.

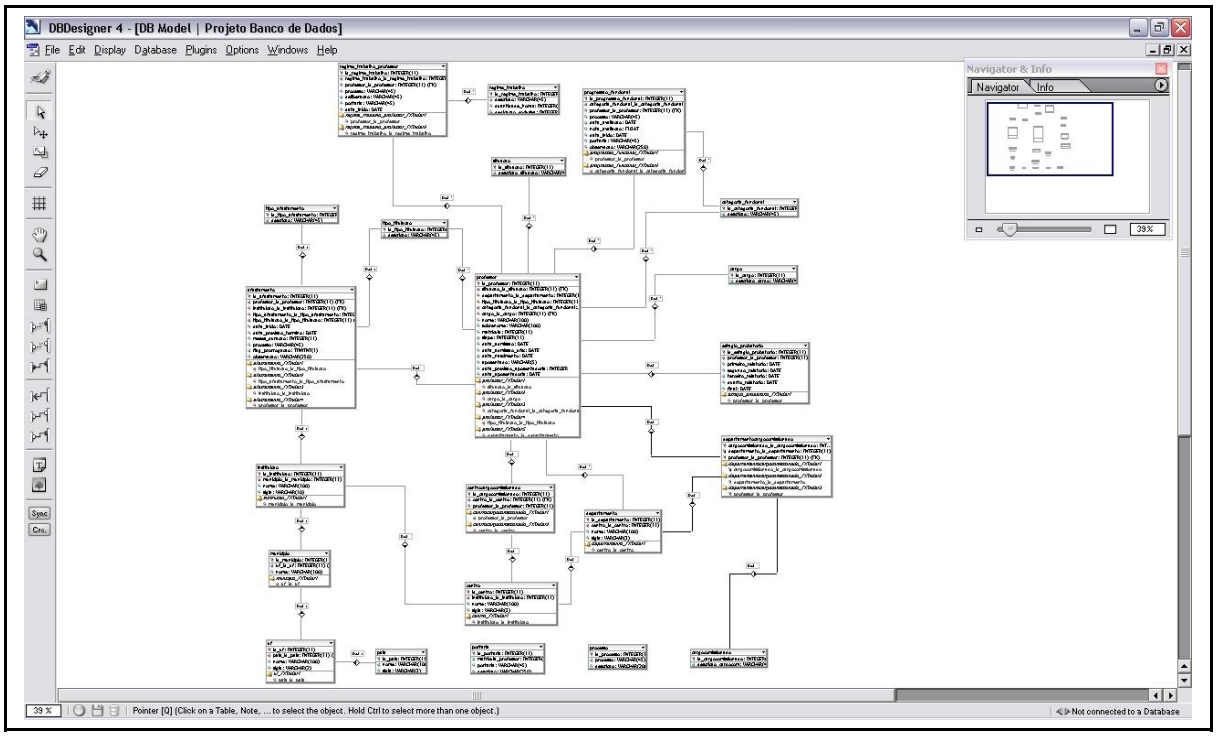


Figura 7 – DBDesigner

A figura 8 contém a tela do Firebug na parte inferior da imagem com uma requisição ao servidor para a exibição dos professores e outra de retorno, com os dados retornados pelo banco de dados no format JSON.

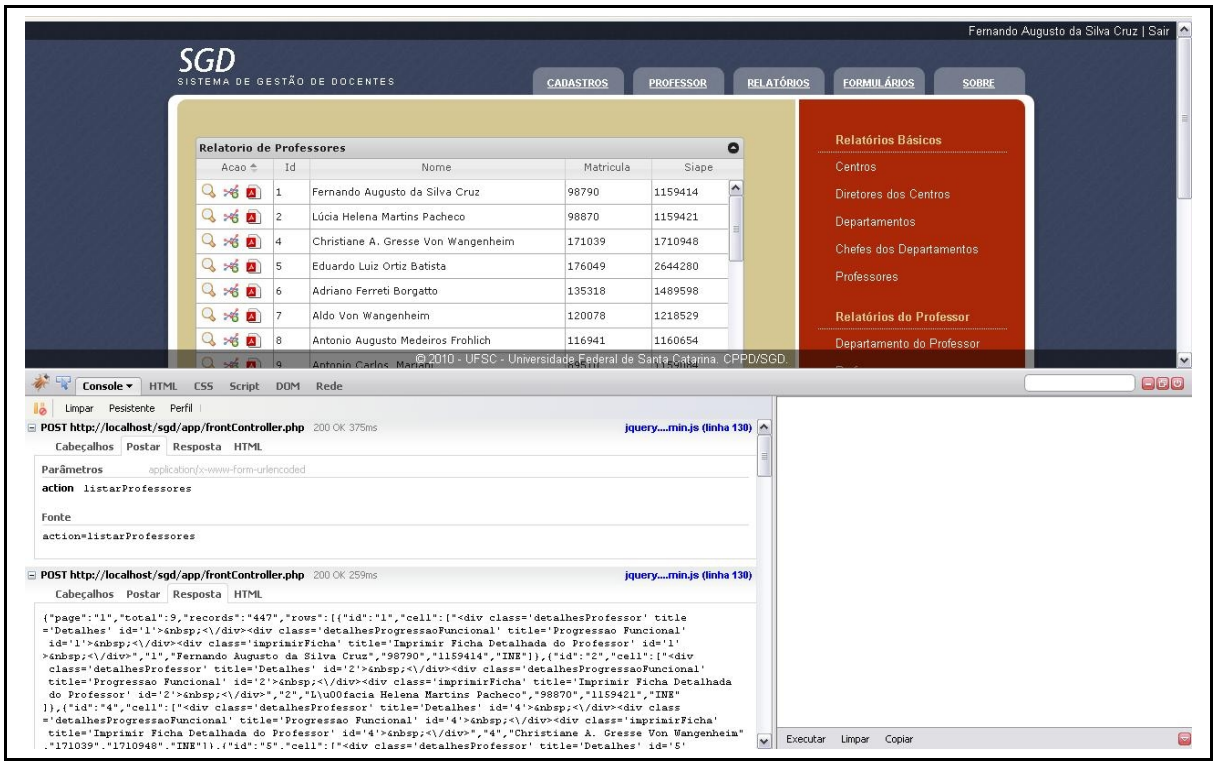


Figura 8 – Firebug

A figura 9 exibe o sistema para gerência de projetos do Google, exibindo a tela dos *"issues"* (problemas) que foram cadastrados para o sistema. Um *issue* pode ser uma correção ou nova funcionalidade necessária ao sistema. Através desse sistema, usuário e programador tem um contato mais direto para modificações no sistema, além de permitir manter o histórico das alterações realizadas. Ainda, podemos realizar pesquisas no código-fonte escrito, em todas as versões do sistema, de maneira rápida e eficiente.

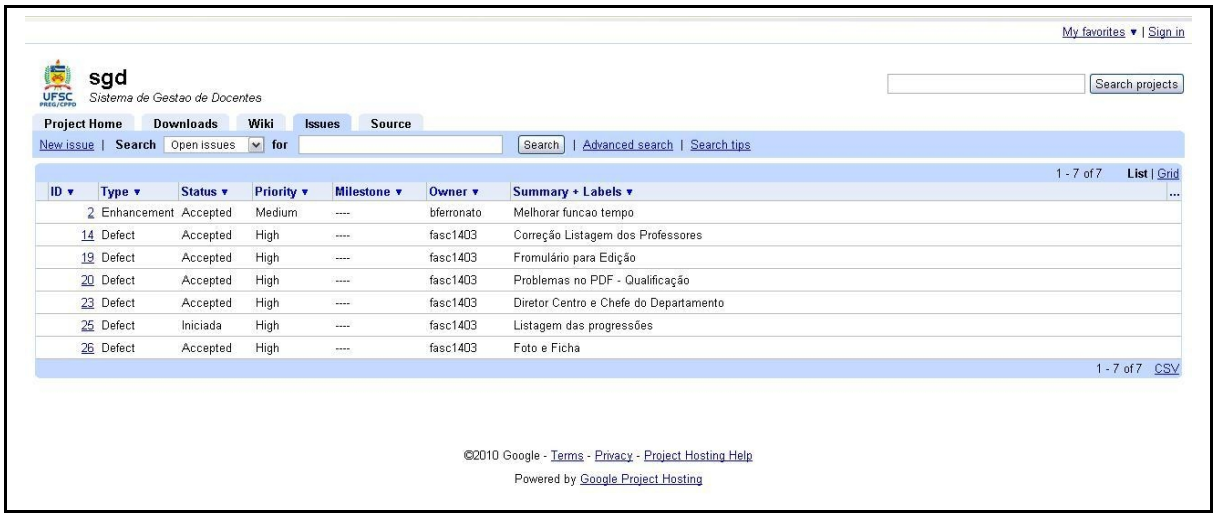


Figura 9 - Google Code – SGD

A figura 10 mostra a tela da Ferramenta *JUDE* com uma visão parcial dos casos-de-uso do sistema.

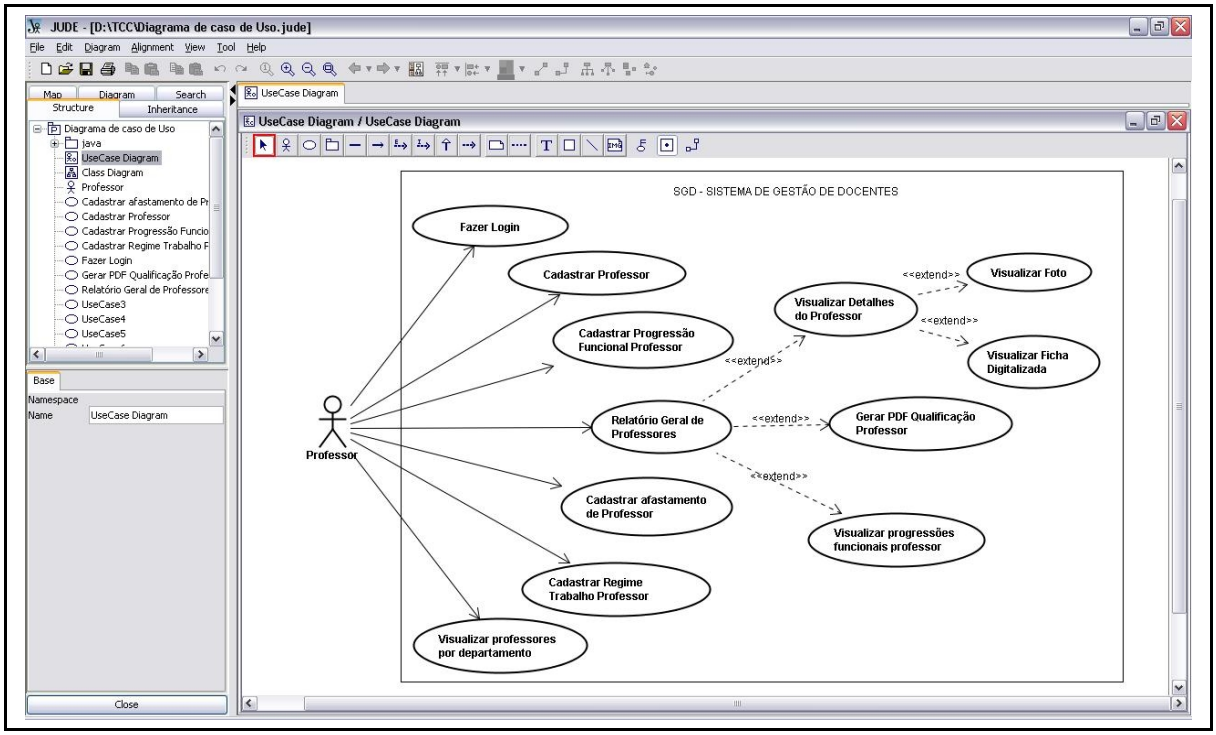


Figura 10 – Casos-de-uso em Jude

4 IMPLEMENTAÇÃO

O sistema iniciou a partir de uma estrutura pré-concebida. Essa estrutura contava com o *layout* do sistema, algumas classes e formulários com assinaturas de métodos SOAP não implementados e o banco de dados.

A partir disso, começou-se o desenvolvimento desse sistema. A primeira ação que foi desenvolvida, foi a criação do modelo do banco de dados a partir do banco já existente. Esse desenvolvimento foi realizado fazendo-se a exportação do SQL de construção do banco de dados. Com isso em mãos, um novo modelo de banco de dados foi criado utilizando a ferramenta DBDesigner e os campos e relacionamentos entre as tabelas foram sendo desenhadas no modelo.

O modelo do banco de dados foi muito importante para o desenvolvimento do sistema, pois foi através dele que foram observados os inter-relacionamentos entre as tabelas e como era possível acessar determinado dado.

Após o desenho do modelo do banco de dados, começou-se a implementação do código-fonte. Uma das primeira mudanças realizadas, foi a centralização da conexão com o banco de dados, a qual era criada a cada funcionalidade executada no sistema. Esse modo foi alterado pelo padrão de projeto *singleton*, o qual garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.

Com o padrão implementado, começou-se a alterar os arquivos para separar o que era apresentação de dados do que era processamento de dados. Essa alteração consistiu também, com o inicio da criação das classes de objetos. Cada funcionalidade já existente, foi reescrita utilizando-se o paradigma da orientação a objetos. Na finalização dessa funcionalidade, a mesma funcionalidade escrita nos padrões estruturados era excluída.

Com isso obteve-se a separação do código no padrão MVC. O próximo passo, foi a inclusão do gerenciador de *templates* Smarty*,* o qual nos oferece uma série de vantagens, entre as principais, podemos citar um melhor desempenho de

execução, *cache* de *templates*, suporte a *plugins*, melhor organização do código, restrição de escrita de código php na camada de visualização.

Organizada a *View do Modelo* MVC*,* partimos agora para a inclusão do Javascript no sistema. O Javascript é utilizado através do *framework* Jquery o qual faz todo o roteamento das chamadas do sistema, as requisições AJAX, a validação dos formulários no lado *cliente, o* bloqueio de requisições durante a execução de uma funcionalidade, a apresentação de mensagens ao usuário entre outras.

Como decisão de projeto, procurou-se criar uma estrutura de sistema no qual todo o Javascript necessário ficasse centralizado, não somente na organização em diretórios como também em seu carregamento, e que todas as funcionalidade do sistema funcionassem via AJAX.

Isso foi feito realizando a inclusão de todo o Javascript necessário no sistema no final da página index.html. A inclusão dos arquivos obdece determinada ordem para evitar a chamada de funções que ainda não tenham sido declaradas. A ordem é, primeiro o *framework* Jquery*,* depois todos os *plugins* Jquery e por fim os arquivos escritos para o sistema.

A partir desse momento já podemos começar a escrever as funções de controle de fluxo (requisições AJAX), que são associadas a *links* ou botões no carregamento do Javascript, ou estabelecidas após o carregamento de uma página. Essas funções tem normalmente o seguinte comportamento: um *POST* com uma variável informando qual o *template* deve ser carregado, e em seguida, a inclusão do seu conteúdo em determinada *div* do sistema. Após o carregamento do *template*, o *callback* da função faz as associações necessárias aos novos *links* e botões.

Para o cadastro de dados, o comportamento é um pouco diferente, ao clicar no botão cadastrar, todos os campos obrigatórios são verificados antes de enviar a requisição ao servidor. Caso os dados não estejam em conformidade, a requisição não é enviada. Essa é uma verificação bastante útil e rápida, quando não são necessários dados do banco de dados. Essa verificação é a validação no lado do cliente e apesar de suas vantagens, ela não deve ser usada como o único método

de validação de entrada de dados, pois pode ser facilmente burlada com ferramentas de edição de Javascript.

Também buscou-se a centralização na inclusão de arquivos no lado do servidor, para isso foi criada uma classe central de controle chamada *frontController,* que centraliza as inclusões de outras páginas e é o primeiro a receber qualquer requisição ao sistema.

Ao receber uma requisição, o *frontController* chama o *controller* da função passada como parâmetro e o método correspondente.

Existem vários *controllers* na aplicação e eles foram organizados de maneira que cada um representasse uma entidade no sistema. Assim temos *controllers* responsáveis por professores, centros, departamentos, entre outras entidades.

Cada um desses *controllers* é responsável por fazer a comunicação com suas respectivas *views* e *models.*

Por fim, foi incluído o controle de acesso ao sistema, sendo a partir de então, necessário fazer *login* utilizando o número SIAPE e uma senha de acesso. Com a funcionalidade de controle de acesso desenvolvida, foi possível criar uma nova funcionalidade para o usuário, o qual agora, pode alterar suas informações cadastrais.

4.1 Caracterização arquitetura cliente-servidor

A seguir será caracterizada a arquitetura cliente-servidor desenvolvida no sistema SGD, apresentando a funcionalidade de cadastro de professor como exemplo.

Visto que o sistema foi desenvolvido para funcionar via *web,* temos que seu uso se dará através de um navegador, neste caso, o navegador é a parte cliente, da arquitetura cliente-servidor. A figura 11 mostra uma visão parcial do formulário para preenchimento dos dados do novo professor.

Figura 11 - Formulário de cadastro de professor

Após preenchermos o formulário e clicarmos no botão cadastrar, uma validação dos dados é feita no lado do cliente, antes de enviar os dados ao servidor. Para exemplo, deixaremos alguns campos de dados sem preenchimento, para exibir esta validação, a qual é exibida na figura 12.

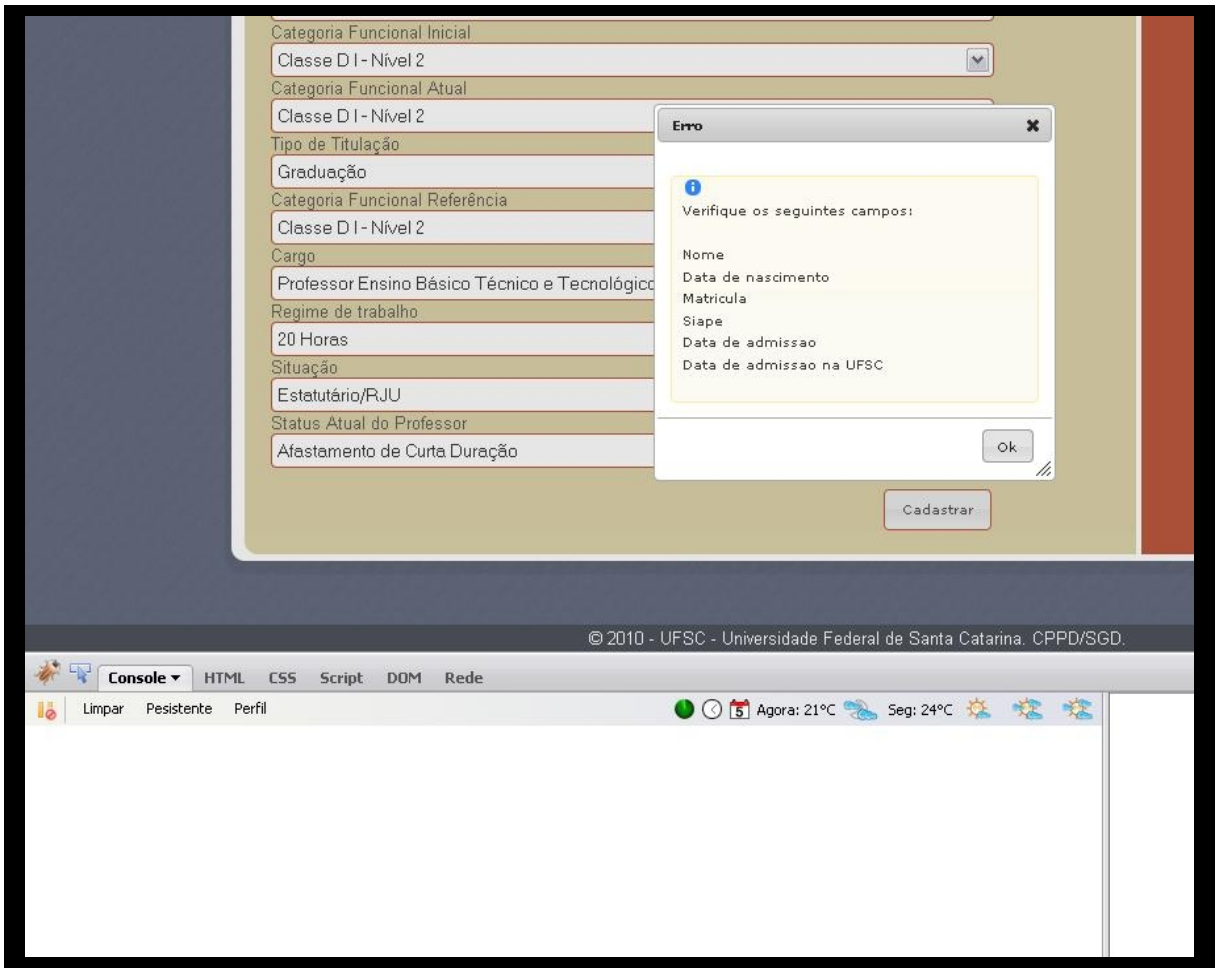


Figura 12 - Validação dados cadastro lado cliente

Como podemos ver na figura 12, foi realizada uma validação do dados de entrada no sistema, informando o usuário que determinados campos deveriam ser verificados, também podemos perceber através da ferramenta Firebug que nenhuma requisição foi enviada ao servidor, assim, todo esse processo ocorreu somente no lado do cliente.

A seguir, iremos alterar o código Javascript para que não seja feita a validação do campo nome no lado cliente, mas manteremos a validação no lado servidor, e iremos realizar um novo cadastro de professor sem o campo nome preenchido.

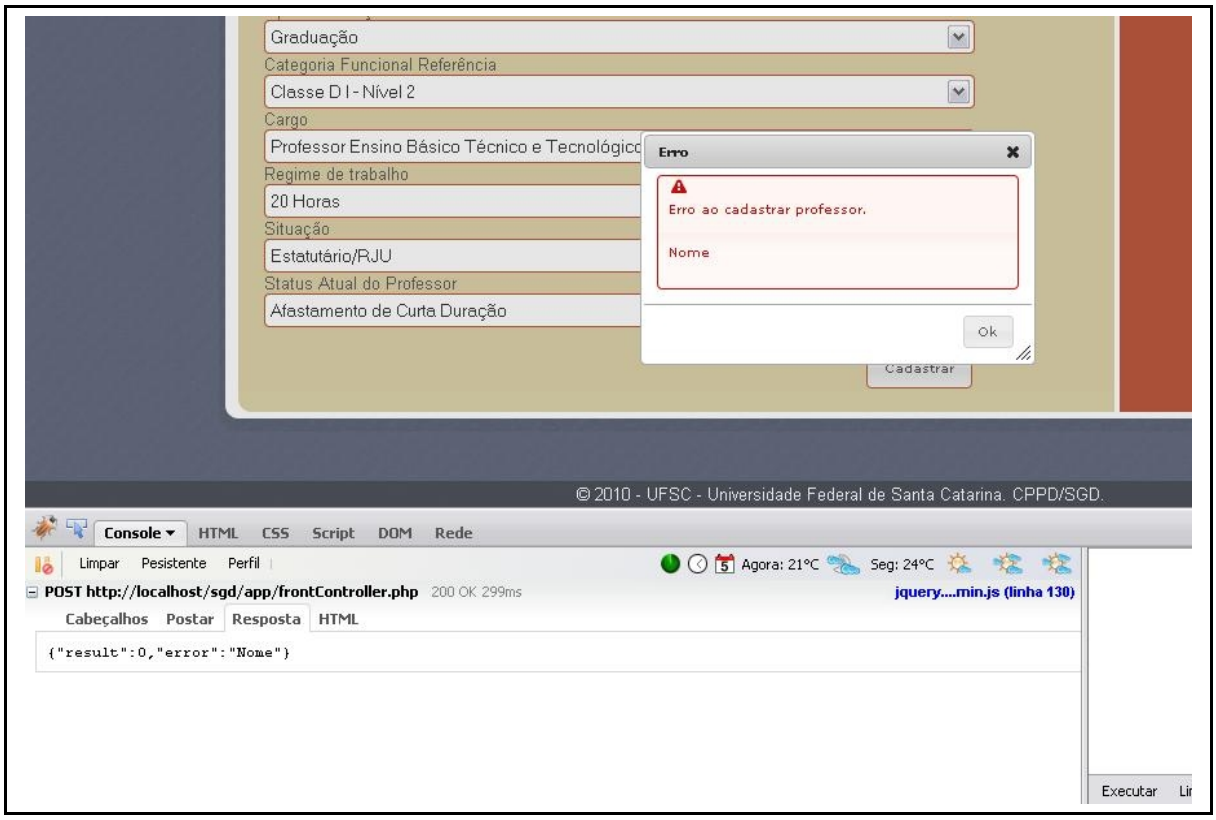


Figura 13 - Validação dados cadastro lado servidor

Como podemos ver na imagem 13, uma mensagem de erro foi exibida ao usuário, informando sobre um erro ao realizar o cadastro do professor, esse erro foi gerado propositalmente, para que pudessemos ver a validação dos dados no lado do servidor. Analisando o console da ferramenta Firebug, podemos perceber que uma requisição foi enviada ao servidor, e como resposta, obtivemos um JSON, o qual nos informa que o resultado foi *0*(zero) e que o erro foi **Nome**.

Por último, iremos fazer um cadastro de professor corretamente para analisarmos a requisição enviada pelo cliente e a resposta do servidor.

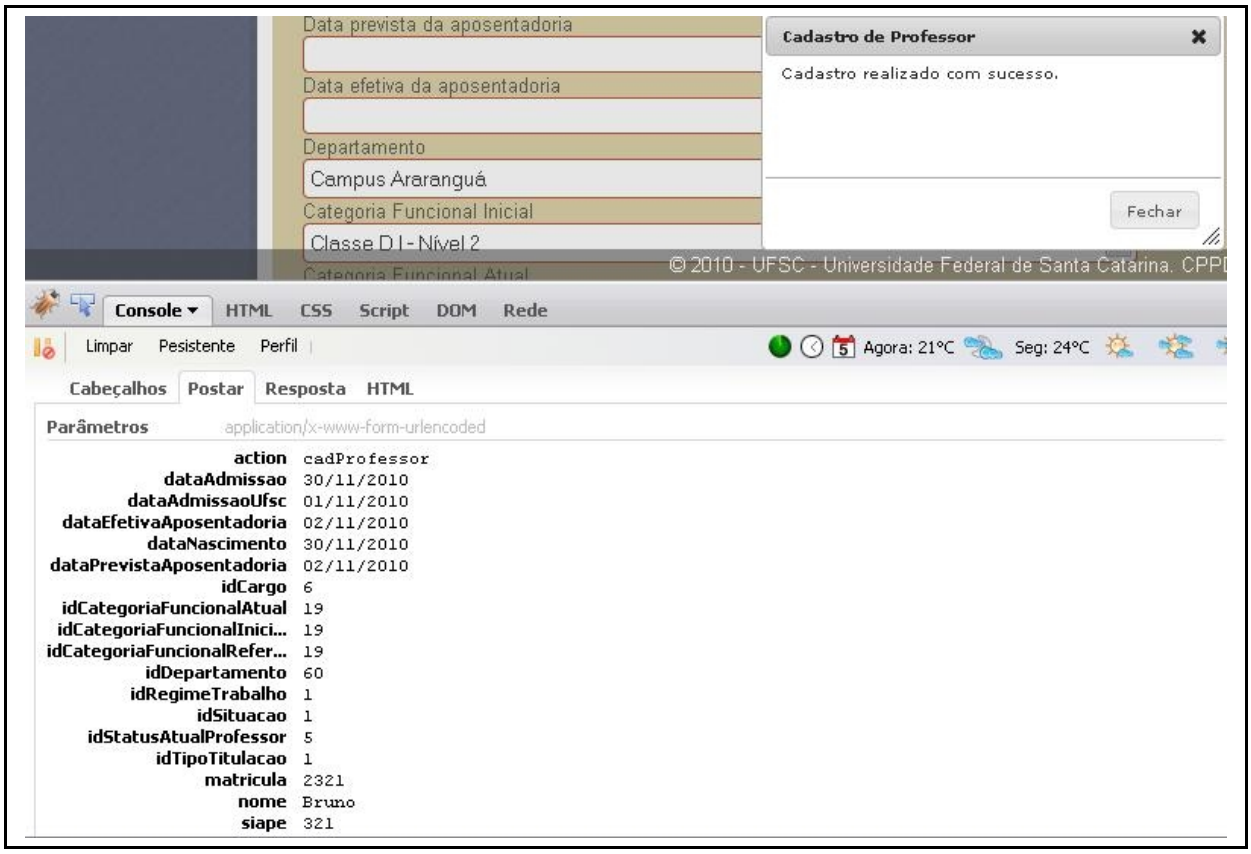


Figura 14 - Requisicão enviada pelo cliente

Na figura 14 podemos ver todos os parâmetros enviados pelo cliente ao servidor, também podemos notar que existe um parâmetro de nome *action*, que é o parâmetro identificador da requisição feita pelo cliente e que será analisado pelo *frontController* para fazer o seu correto direcionamento dentro do sistema.

Como resposta a requisição enviada, podemos ver na figura 15 o JSON nos informando que o *result* foi igual a 1, neste caso, interpretado pelo Javascript como sucesso no cadastro de um novo professor.

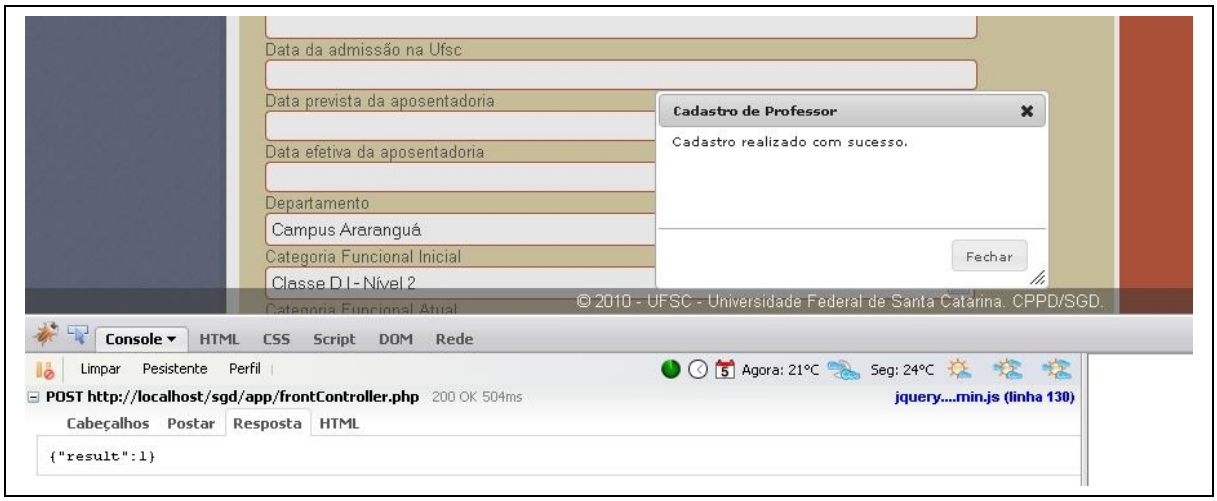


Figura 15 - Resposta retornada pelo servidor

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 Conclusão

Uma vez colocado em uso o sistema trouxe significativos benefícios para os usuários entre os quais a geração automatizada das qualificações dos professores.

As informações que eram mantidas e preservadas em meios físicos através de fichas de papel, agora serão todas digitalizadas e incluídas no sistema. Tal procedimento é necessário, pois caso haja alguma dúvida em relação ao dado cadastrado, bastará uma consulta à ficha digitalizada para seu esclarecimento, além.de manter, historicamente, os dados.

Além disso, a criação do sistema trouxe um enorme ganho na busca de informações relativas aos professores, na geração de relatórios e na agilidade com que a qualificação de um professor pode ser obtida, ganhando também na qualidade dos dados. Evita-se assim, erros de transcrição da informação da ficha de papel para cada novo documento criado.

5.2 Trabalhos futuros

Como trabalhos futuros a serem implementados na aplicação, podemos listar alguns abaixo:

- Criação de tabelas para a avaliação dos trabalhos dos professores durante o período de uma progressão.

- Análise semântica das informações para que o preenchimento dessas informações seja todo automatico.
- Integração do banco de dados do sistema ao banco de dados do servidor central da UFSC para que as informações referentes ao ensino e pesquisa do professor possam ser utilizadas de forma que alguns dados sejam preenchidos automaticamente, diminuindo significativamente o trabalho do no preenchimento de formulários de cálculo de processos, sendo esse cálculo realizado todo automaticamente.

6 REFERÊNCIAS

DALL'OGLIO, Pablo. **PHP Programando com orientação a Objetos.** São Paulo:Novatec, 2007. 574 p.

Wikipedia. **JavaScript**. Disponível em: <http://pt.wikipedia.org/wiki/JavaScript>. Acesso em 20 set. 2010.

Wikipedia. **JQuery**. Disponível em: <http://pt.wikipedia.org/wiki/JQuery>. Acesso em 20 set. 2010.

Wikipedia. **Cascading Style Sheets**. Disponível em: <http://pt.wikipedia.org/wiki/Cascading_Style_Sheets>. Acesso em 21 set. 2010.

FEITOSA, Ciro. **Smarty e PHP, tudo a ver**. Ciro Feitosa. [s.l.] jan. 2006. Disponível em: <http://cirofeitosa.com.br/post/smarty-e-php-tudo-a-ver>. Acesso em 21 set. 2010

Jude Design e Comunication. **Jude.** Disponível em: < http://jude.change-vision.com/jude-web/product/jude_pl.html>. Acesso em 20/10/2010.

MySql Manual. **Introdução ao programa MySQL Query Browser.** Disponível em: <http://dev.mysql.com/doc/query-browser/pt/mysql-query-browser-introduction.html>. Acesso em 23 set. 2010.

Pergamim. **Informatizar Por Que**. Disponível em: <http://www.pergamum.pucpr.br/redepergamum/trabs/Camila_K_Burin-Informatizar_por_que.pdf>. Acesso em 17 out. 2010

Prefeitura Porto Alegre. **Progressão Funcional**. Disponível em: <http://www2.portoalegre.rs.gov.br/sma/default.php?p_secao=86>. Acesso em 17 out. 2010

Portal do Servidor. **Progressão Funcional**. Disponível em: <http://www.portaldoservidor.sc.gov.br/index.php?option=com_docman&task=doc_download&gid=463>. Acesso em 17 set. 2010

DevMedia. **DBDesigner: uma ferramenta gratuita para modelagem de dados**. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=6840>. Acesso em 23 set. 2010

Wikipedia. **PhpMyAdmin**. Disponível em: <http://pt.wikipedia.org/wiki/PhpMyAdmin>. Acesso em 23 set. 2010

Wikipedia. **FileZilla**. Disponível em: <http://pt.wikipedia.org/wiki/FileZilla>. Acesso em 22 set. 2010

Baixaki. **Firebug**. Disponível em: <http://www.baixaki.com.br/download/firebug.htm>. Acesso em 22 set. 2010

Wikipedia. **WAMP**. Disponível em: <http://pt.wikipedia.org/wiki/WAMP>. Acesso em 22 set. 2010

Wikipedia. **Subversion**. Disponível em: <http://pt.wikipedia.org/wiki/Subversion>. Acesso em 23 set. 2010.

Wikipedia. **Google Code**. Disponível em: <http://pt.wikipedia.org/wiki/Google_Code 17/10/2010>. Acesso em 25 set. 2010.

Wikipedia. **PHP**. Disponível em: <http://pt.wikipedia.org/wiki/PHP>. Acesso em 24 set. 2010.

Wikipedia. **UML**. Disponível em: <http://pt.wikipedia.org/wiki/UML>. Acesso em 27 set. 2010.

Silva, Danilo Rodrigues da. Evolução da Arquitetura Cliente/Servidor:Definição, caminhos e rumos. Disponível em: <http://knol.google.com/k/danilo-rodrigues-da-silva/evolução-da-arquitetura-cliente-servidor/anrd8mpq3bji/3>. Acesso em 07/11/2010.

Silva, Maurício Samy. O Módulo Transition das CSS3. Disponível em: < http://maujor.com/>. Acesso em 13/11/2010.

7 GLOSSÁRIO

Div Elemento criado a partir do HTML 4.0 com o objetivo de fornecer um mecanismo para agrupar e estrutura os documentos.

Subversion Sistema de controle de versão.

Mercurial Ferramenta multi-plataforma de controle de versão.

8 ANEXO A – SCRIPT DE BANCO DE DADOS

```
CREATE TABLE regime_trabalho (
  id_regime_trabalho INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao VARCHAR(45) NOT NULL,
  quantidade_horas INTEGER(11) UNSIGNED NOT NULL,
  dedicacao_exclusiva INTEGER(1) UNSIGNED NOT NULL,
  PRIMARY KEY(id_regime_trabalho)
)TYPE=InnoDB;

CREATE TABLE situacao (
  id_situacao INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao_situacao VARCHAR(45) NOT NULL,
  PRIMARY KEY(id_situacao)
)TYPE=InnoDB;

CREATE TABLE processo (
  id_processo INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  processo VARCHAR(45) NOT NULL,
  descricao VARCHAR(200) NOT NULL,
  PRIMARY KEY(id_processo)
)TYPE=InnoDB;

CREATE TABLE pais (
  id_pais INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  nome VARCHAR(100) NOT NULL,
  sigla VARCHAR(3) NOT NULL,
```

```
  PRIMARY KEY(id_pais)
)TYPE=InnoDB;

CREATE TABLE portaria (
  id_portaria INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  matricula_professor INTEGER(11) UNSIGNED NOT NULL,
  portaria VARCHAR(45) NOT NULL,
  descricao VARCHAR(250) NOT NULL,
  PRIMARY KEY(id_portaria)
)TYPE=InnoDB;

CREATE TABLE tipo_afastamento (
  id_tipo_afastamento INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao VARCHAR(45) NOT NULL,
  PRIMARY KEY(id_tipo_afastamento)
)TYPE=InnoDB;

CREATE TABLE cargocomissionado (
  id_cargocomissionado INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao_cargocom VARCHAR(45) NOT NULL,
  PRIMARY KEY(id_cargocomissionado)
)TYPE=InnoDB;

CREATE TABLE cargo (
  id_cargo INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao_cargo VARCHAR(45) NOT NULL,
  PRIMARY KEY(id_cargo)
```

```
)TYPE=InnoDB;


CREATE TABLE tipo_titulacao (
  id_tipo_titulacao INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao VARCHAR(45) NOT NULL,
  PRIMARY KEY(id_tipo_titulacao)
)TYPE=InnoDB;


CREATE TABLE categoria_funcional (
  id_categoria_funcional INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  descricao VARCHAR(45) NOT NULL,
  PRIMARY KEY(id_categoria_funcional)
)TYPE=InnoDB;


CREATE TABLE uf (
  id_uf INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  pais_id_pais INTEGER(11) UNSIGNED NOT NULL,
  nome VARCHAR(100) NOT NULL,
  sigla VARCHAR(2) NOT NULL,
  PRIMARY KEY(id_uf),
  INDEX uf_FKIndex1(pais_id_pais),
  FOREIGN KEY(pais_id_pais)
    REFERENCES pais(id_pais)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
)TYPE=InnoDB;
```

```
CREATE TABLE municipio (
  id_municipio INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  uf_id_uf INTEGER(11) UNSIGNED NOT NULL,
  nome VARCHAR(100) NOT NULL,
  PRIMARY KEY(id_municipio),
  INDEX municipio_FKIndex1(uf_id_uf),
  FOREIGN KEY(uf_id_uf)
    REFERENCES uf(id_uf)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE instituicao (
  id_instituicao INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  municipio_id_municipio INTEGER(11) UNSIGNED NOT NULL,
  nome VARCHAR(100) NOT NULL,
  sigla VARCHAR(10) NULL,
  PRIMARY KEY(id_instituicao),
  INDEX instituicao_FKIndex1(municipio_id_municipio),
  FOREIGN KEY(municipio_id_municipio)
    REFERENCES municipio(id_municipio)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE centro (
  id_centro INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
  instituicao_id_instituicao INTEGER(11) UNSIGNED NOT NULL,
  nome VARCHAR(100) NOT NULL,
  sigla VARCHAR(3) NOT NULL,
  PRIMARY KEY(id_centro),
  INDEX centro_FKIndex1(instituicao_id_instituicao),
  FOREIGN KEY(instituicao_id_instituicao)
    REFERENCES instituicao(id_instituicao)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE departamento (
  id_departamento INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  centro_id_centro INTEGER(11) UNSIGNED NOT NULL,
  nome VARCHAR(100) NOT NULL,
  sigla VARCHAR(3) NOT NULL,
  PRIMARY KEY(id_departamento),
  INDEX departamento_FKIndex1(centro_id_centro),
  FOREIGN KEY(centro_id_centro)
    REFERENCES centro(id_centro)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE professor (
  id_professor INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  situacao_id_situacao INTEGER(11) UNSIGNED NOT NULL,
```

```
departamento_id_departamento INTEGER(11) UNSIGNED NOT NULL,
tipo_titulacao_id_tipo_titulacao INTEGER(11) UNSIGNED NOT NULL,
categoria_funcional_id_categoria_funcional INTEGER(11) UNSIGNED NOT NULL,
cargo_id_cargo INTEGER(11) UNSIGNED NOT NULL,
nome VARCHAR(100) NOT NULL,
sobrenome VARCHAR(100) NOT NULL,
matricula INTEGER(11) UNSIGNED NOT NULL,
siape INTEGER(11) UNSIGNED NOT NULL,
data_admissao DATE NOT NULL,
data_admissao_ufsc DATE NOT NULL,
data_nascimento DATE NOT NULL,
aposentado VARCHAR(5) NULL,
data_previsao_aposentadoria INTEGER UNSIGNED NULL,
data_aposentadoria DATE NULL,
PRIMARY KEY(id_professor),
INDEX professor_FKIndex1(situacao_id_situacao),
INDEX professor_FKIndex2(cargo_id_cargo),
INDEX professor_FKIndex3(categoria_funcional_id_categoria_funcional),
INDEX professor_FKIndex4(tipo_titulacao_id_tipo_titulacao),
INDEX professor_FKIndex5(departamento_id_departamento),
FOREIGN KEY(situacao_id_situacao)
  REFERENCES situacao(id_situacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
FOREIGN KEY(cargo_id_cargo)
  REFERENCES cargo(id_cargo)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  FOREIGN KEY(categoria_funcional_id_categoria_funcional)
    REFERENCES categoria_funcional(id_categoria_funcional)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  FOREIGN KEY(tipo_titulacao_id_tipo_titulacao)
    REFERENCES tipo_titulacao(id_tipo_titulacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  FOREIGN KEY(departamento_id_departamento)
    REFERENCES departamento(id_departamento)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE estagio_probatorio (
  id_estagio_probatorio INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  professor_id_professor INTEGER(11) UNSIGNED NOT NULL,
  primeiro_relatorio DATE NOT NULL,
  segundo_relatorio DATE NOT NULL,
  terceiro_relatorio DATE NOT NULL,
  quarto_relatorio DATE NOT NULL,
  final DATE NOT NULL,
  PRIMARY KEY(id_estagio_probatorio),
  INDEX estagio_probatorio_FKIndex1(professor_id_professor),
  FOREIGN KEY(professor_id_professor)
    REFERENCES professor(id_professor)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE regime_trabalho_professor (
  id_regime_trabalho INTEGER(11) UNSIGNED NOT NULL,
  regime_trabalho_id_regime_trabalho INTEGER(11) UNSIGNED NOT NULL,
  professor_id_professor INTEGER(11) UNSIGNED NOT NULL,
  processo VARCHAR(45) NULL,
  deliberacao VARCHAR(45) NULL,
  portaria VARCHAR(45) NULL,
  data_inicio DATE NOT NULL,
  PRIMARY KEY(id_regime_trabalho),
  INDEX regime_trabalho_professor_FKIndex1(professor_id_professor),
  INDEX regime_trabalho_professor_FKIndex2(regime_trabalho_id_regime_trabalho),
  FOREIGN KEY(professor_id_professor)
    REFERENCES professor(id_professor)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  FOREIGN KEY(regime_trabalho_id_regime_trabalho)
    REFERENCES regime_trabalho(id_regime_trabalho)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE progressao_funcional (
  id_progressao_funcional INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
  categoria_funcional_id_categoria_funcional INTEGER(11) UNSIGNED NOT NULL,
  professor_id_professor INTEGER(11) UNSIGNED NOT NULL,
  processo VARCHAR(45) NULL,
  data_avaliacao DATE NULL,
  nota_avaliacao FLOAT NULL,
  data_inicio DATE NOT NULL,
  portaria VARCHAR(45) NULL,
  observacao VARCHAR(250) NULL,
  PRIMARY KEY(id_progressao_funcional),
  INDEX progressao_funcional_FKIndex1(professor_id_professor),
  INDEX progressao_funcional_FKIndex2(categoria_funcional_id_categoria_funcional),
  FOREIGN KEY(professor_id_professor)
    REFERENCES professor(id_professor)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(categoria_funcional_id_categoria_funcional)
    REFERENCES categoria_funcional(id_categoria_funcional)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
)TYPE=InnoDB;


CREATE TABLE centrocargocomissionado (
  id_cargocomissionado INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  centro_id_centro INTEGER(11) UNSIGNED NOT NULL,
  professor_id_professor INTEGER(11) UNSIGNED NOT NULL,
  PRIMARY KEY(id_cargocomissionado),
  INDEX centrocargocomissionado_FKIndex1(professor_id_professor),
```

```
    INDEX centrocargocomissionado_FKIndex2(centro_id_centro),
    FOREIGN KEY(professor_id_professor)
      REFERENCES professor(id_professor)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    FOREIGN KEY(centro_id_centro)
      REFERENCES centro(id_centro)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
  )TYPE=InnoDB;


  CREATE TABLE departamentocargocomissionado (
    cargocomissionado_id_cargocomissionado INTEGER(11) UNSIGNED NOT NULL,
    departamento_id_departamento INTEGER(11) UNSIGNED NOT NULL,
    professor_id_professor INTEGER(11) UNSIGNED NOT NULL,
          PRIMARY KEY(cargocomissionado_id_cargocomissionado, departamento_id_departamento,
professor_id_professor),
    INDEX departamentocargocomissionado_FKIndex1(cargocomissionado_id_cargocomissionado),
    INDEX departamentocargocomissionado_FKIndex2(departamento_id_departamento),
    INDEX departamentocargocomissionado_FKIndex3(professor_id_professor),
    FOREIGN KEY(cargocomissionado_id_cargocomissionado)
      REFERENCES cargocomissionado(id_cargocomissionado)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    FOREIGN KEY(departamento_id_departamento)
      REFERENCES departamento(id_departamento)
        ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  FOREIGN KEY(professor_id_professor)
    REFERENCES professor(id_professor)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)TYPE=InnoDB;

CREATE TABLE afastamento (
  id_afastamento INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  professor_id_professor INTEGER(11) UNSIGNED NOT NULL,
  instituicao_id_instituicao INTEGER(11) UNSIGNED NOT NULL,
  tipo_afastamento_id_tipo_afastamento INTEGER(11) UNSIGNED NOT NULL,
  tipo_titulacao_id_tipo_titulacao INTEGER(11) UNSIGNED NOT NULL,
  data_inicio DATE NOT NULL,
  data_previsao_termino DATE NOT NULL,
  meses_duracao INTEGER(11) UNSIGNED NOT NULL,
  processo VARCHAR(45) NOT NULL,
  flag_prorrogacao TINYINT(1) UNSIGNED NOT NULL,
  observacao VARCHAR(250) NULL,
  PRIMARY KEY(id_afastamento),
  INDEX afastamento_FKIndex1(tipo_titulacao_id_tipo_titulacao),
  INDEX afastamento_FKIndex2(tipo_afastamento_id_tipo_afastamento),
  INDEX afastamento_FKIndex3(instituicao_id_instituicao),
  INDEX afastamento_FKIndex4(professor_id_professor),
  FOREIGN KEY(tipo_titulacao_id_tipo_titulacao)
    REFERENCES tipo_titulacao(id_tipo_titulacao)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  FOREIGN KEY(tipo_afastamento_id_tipo_afastamento)
    REFERENCES tipo_afastamento(id_tipo_afastamento)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  FOREIGN KEY(instituicao_id_instituicao)
    REFERENCES instituicao(id_instituicao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  FOREIGN KEY(professor_id_professor)
    REFERENCES professor(id_professor)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)TYPE=InnoDB;
```

9 ANEXO B – TELAS

Figura 16 - Tela de login do sistema SGD

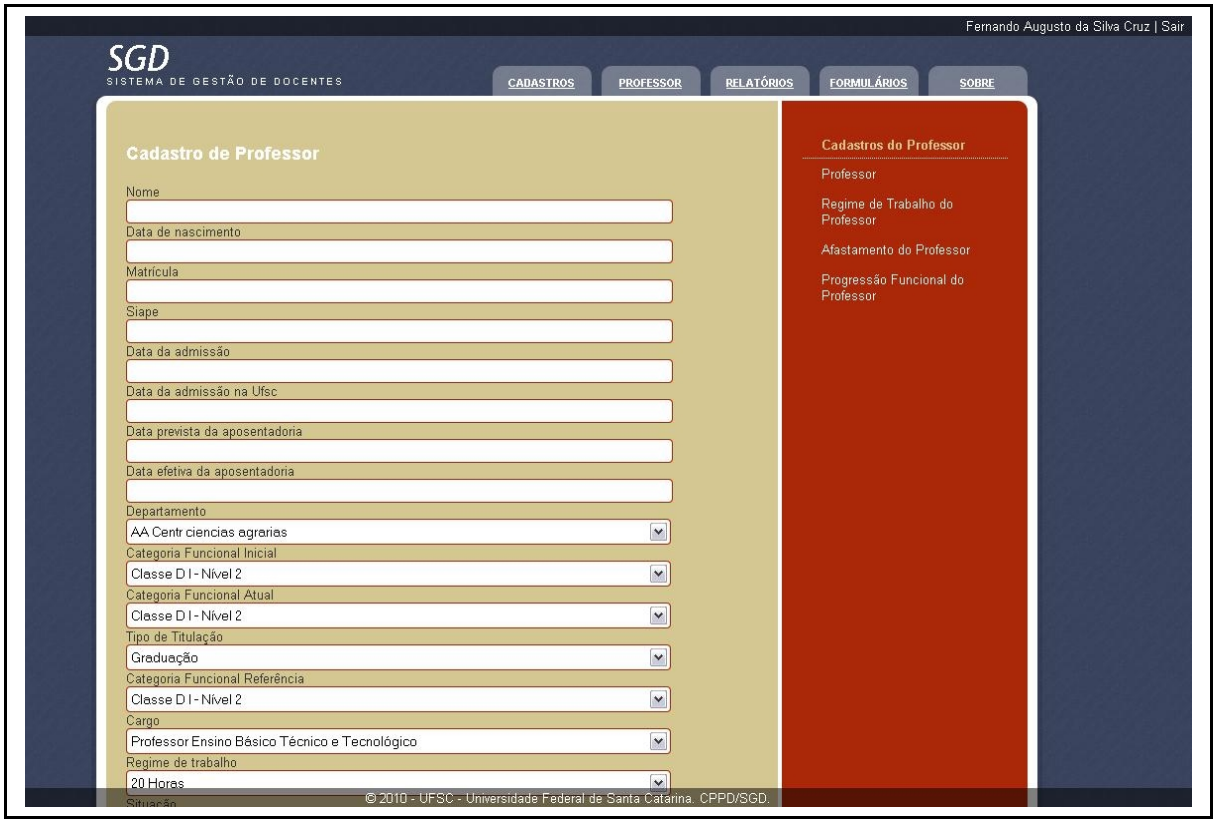


Figura 17 - Cadastro de professor

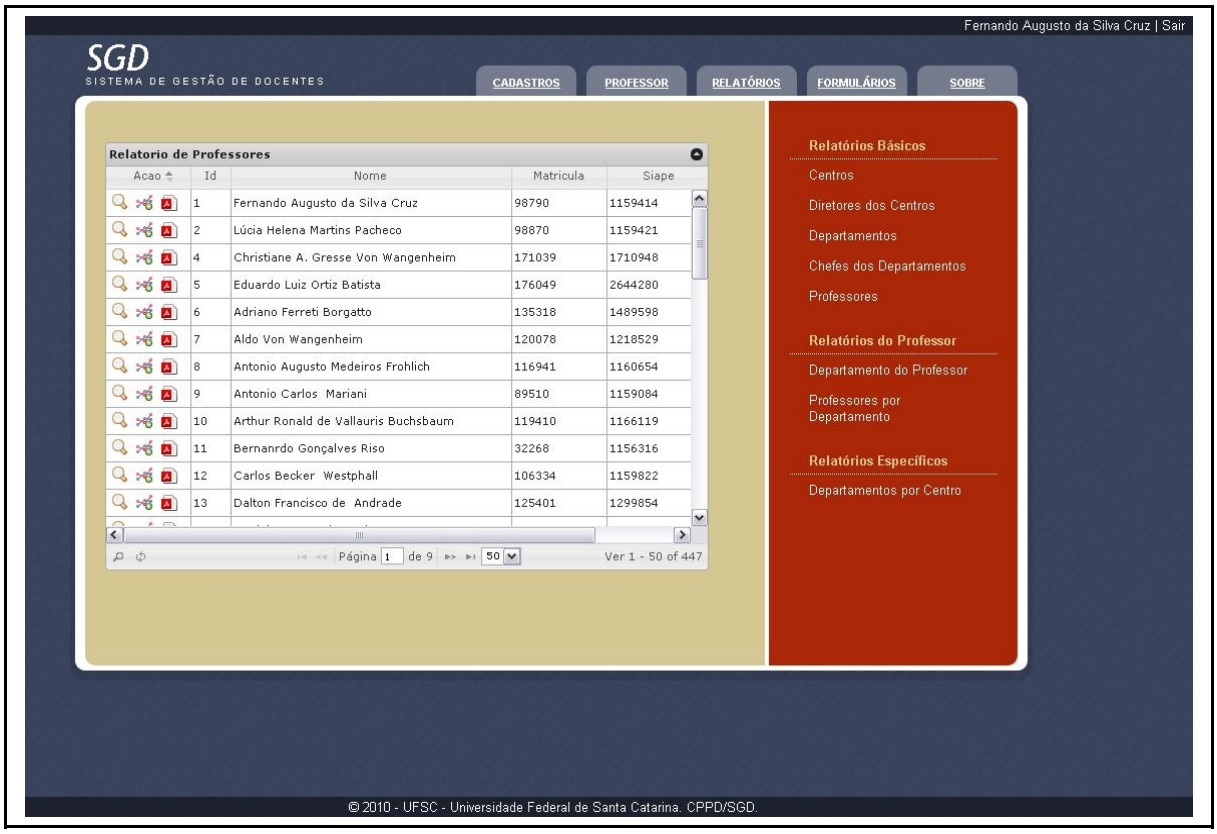


Figura 18 - Relatório de professores

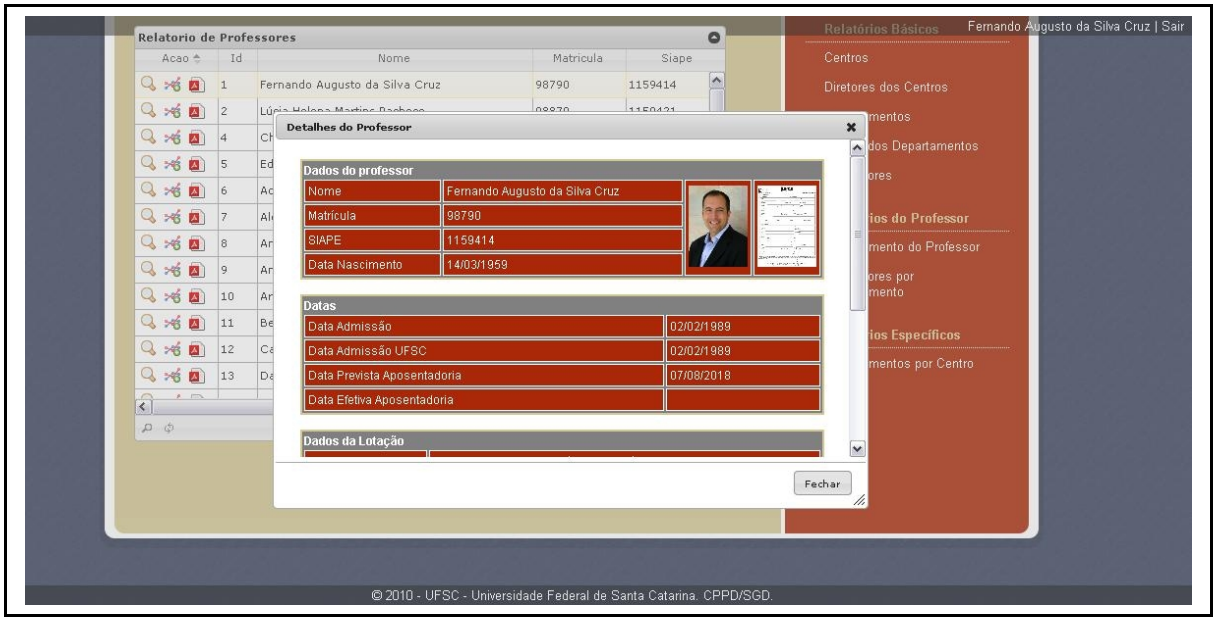
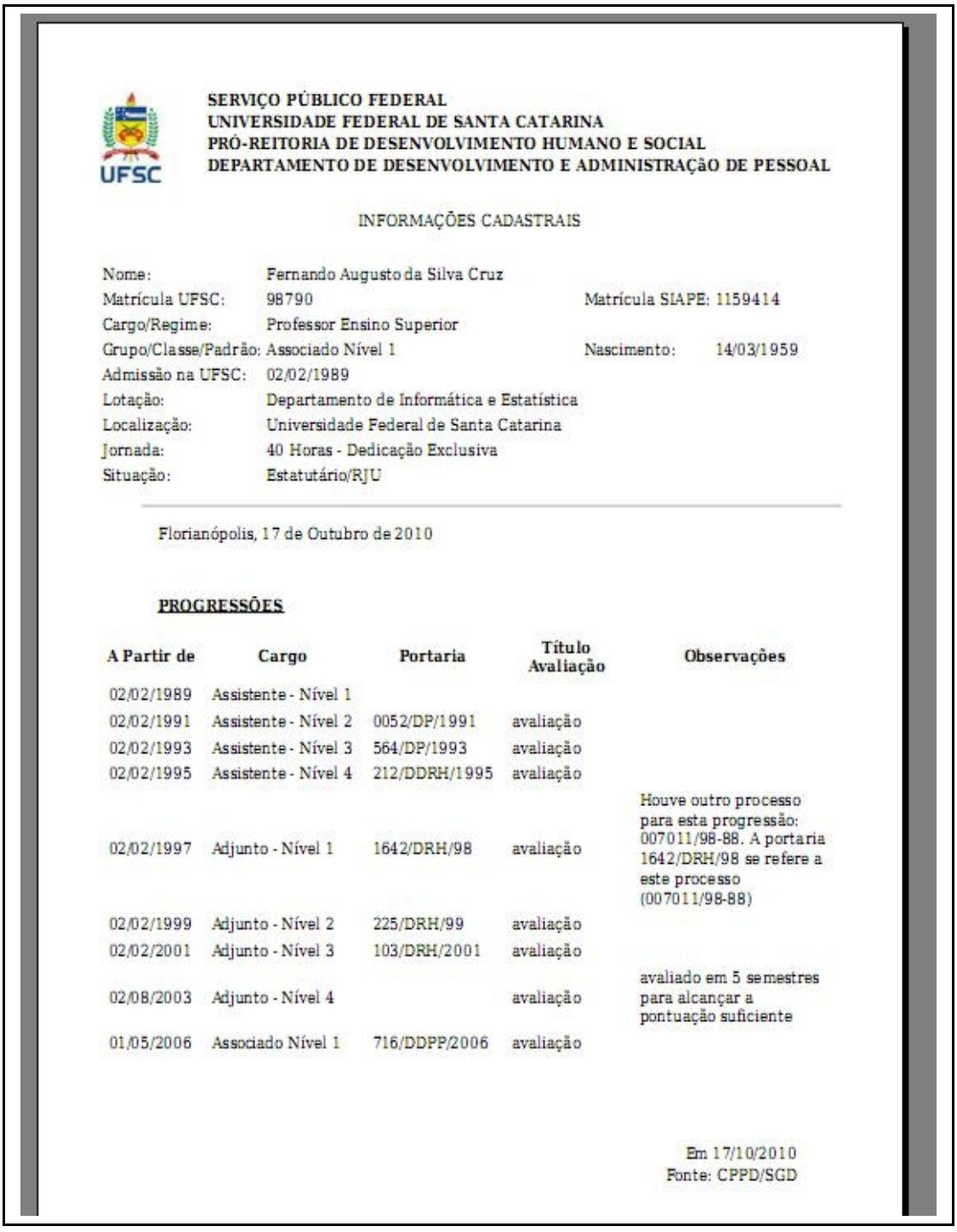


Figura 19 – Detalhes do professor



SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE SANTA CATARINA
PRÓ-REITORIA DE DESENVOLVIMENTO HUMANO E SOCIAL
DEPARTAMENTO DE DESENVOLVIMENTO E ADMINISTRAÇãO DE PESSOAL

INFORMAÇÕES CADASTRAIS

Nome:	Fernando Augusto da Silva Cruz		
Matrícula UFSC:	98790	Matrícula SIAPE:	1159414
Cargo/Regime:	Professor Ensino Superior		
Grupo/Classe/Padrão:	Associado Nível 1	Nascimento:	14/03/1959
Admissão na UFSC:	02/02/1989		
Lotação:	Departamento de Informática e Estatística		
Localização:	Universidade Federal de Santa Catarina		
Jornada:	40 Horas - Dedicação Exclusiva		
Situação:	Estatutário/RJU		

Florianópolis, 17 de Outubro de 2010

PROGRESSÕES

A Partir de	Cargo	Portaria	Título Avaliação	Observações
02/02/1989	Assistente - Nível 1			
02/02/1991	Assistente - Nível 2	0052/DP/1991	avaliação	
02/02/1993	Assistente - Nível 3	564/DP/1993	avaliação	
02/02/1995	Assistente - Nível 4	212/DDRH/1995	avaliação	
02/02/1997	Adjunto - Nível 1	1642/DRH/98	avaliação	Houve outro processo para esta progressão: 007011/98-88. A portaria 1642/DRH/98 se refere a este processo (007011/98-88)
02/02/1999	Adjunto - Nível 2	225/DRH/99	avaliação	
02/02/2001	Adjunto - Nível 3	103/DRH/2001	avaliação	
02/08/2003	Adjunto - Nível 4		avaliação	avaliado em 5 semestres para alcançar a pontuação suficiente
01/05/2006	Associado Nível 1	716/DDPP/2006	avaliação	

Em 17/10/2010
Fonte: CPPD/SGD

Figura 20 - Relatório de qualificação do professor gerado pelo sistema

```
24          $("#cadMunicipio").click(function() {
25              gb.processing();
26              var params = { "action":"printFormCadMunicipio" };
27              $('#content').load("app/frontController.php", params, function() {
28                  $("#cadastrarMunicipio").button().click(function() {
29                      cadastros.municipio.valida();
30                  });
31                  gb.processingClose();
32              });
33          });
```

Figura 21 – Javascript para carregamento do formulário de cadastro de município

```
263     case 'printFormCadMunicipio':
264         $municioC = new municipioController();
265         $municioC->printFormCadMunicipio();
266     break;
267     case 'cadMunicipio':
268         $municioC = new municipioController();
269         $municioC->cadastrar( $nome, $idUf  );
270     break;
```

Figura 22 - Roteamento pelo frontController

```
22      /**
23       * Imprime o formulario para o cadastro de um novo municipio
24       *
25       * @return void
26       */
27      public function printFormCadMunicipio() {
28          $ufC = new UfController();
29          $ufs = $ufC->getAll();
30          $municipioV = new MunicipioV();
31          $municipioV->printFormCadMunicipio( $ufs );
32      }
33
34      /**
35       * Realiza o cadastro de um novo municipio
36       *
37       * @return json
38       */
39      public function cadastrar( $nome, $idUf ) {
40          $municipioDAO = new Municipio();
41
42          $erro = array();
43          if ( empty( $nome) ) $erro[] = 'Nome';
44          if ( empty( $idUf) ) $erro[] = 'Municipio';
45
46          if ( count( $erro ) == 0 ) {
47              $return = $municipioDAO->cadastrarMunicipio( $nome, $idUf );
48          } else {
49              $return->result = 0;
50              $return->error = join( '<br />', $erro );
51          }
52          echo json_encode( $return );
53      }
```

Figura 23 - Controller do Municipio

```php
<?php

class MunicipioV {

    function MunicipioV() {}

    function printFormCadMunicipio( $ufs ) {
        $smarty = new Smarty();
        $smarty->template_dir = 'views/municipio/templates/';
        $smarty->compile_dir  = '../tmp/templates_c/';
        $smarty->cache_dir    = '../tmp/cache/';
        $smarty->config_dir   = 'views/configs/';
        $smarty->assign( "ufs", $ufs );
        $smarty->display('municipio.tpl');
    }
}

?>
```

Figura 24 – Utilização do Smarty para o envio dos dados ao template

```
<h1>Cadastro de Munic&iacute;pio</h1>
<div>Nome</div>
<input type="text" id="nome" name="nome" value="" maxlength="100" class="input ui-corner-all width100" />
<div>UF</div>
<select id="idUf" class="select ui-corner-all width100">
{foreach from=$ufs item=uf}
    <option value="{$uf->idUf}">{$uf->nome}</option>
{/foreach}
</select>
<p></p>
<p>
    <button id="cadastrarMunicipio" class="right button">Cadastrar</button>
</p>
```

Figura 25 - Template do formulário de cadastro de município

```
        cadastra: function( params ) {
            $.post("app/frontController.php", params, function( response ) {
                if ( response.result == 1 ) {
                    $("#cadMunicipio").click();
                    var msg = 'Cadastro realizado com sucesso.';
                    gb.message( msg, 'Cadastro de Municipio' );
                } else {
                    var msg = 'Erro ao cadastrar Municipio.<br /><br />' + response.error;
                    gb.errorMessage( msg, 'Erro' );
                }
            }, "json" );
```

Figura 26 – Javascript para o cadastro de um município

```php
    /**
     * Realiza o cadastro de um novo municipio
     *
     * @return stdClass
     */
    public function cadastrarMunicipio( $nome, $idUf ) {
        $conexao = Conexao::con();
        $return = new stdClass();

        $nome = utf8_decode( $nome );

        $sql[] = "INSERT INTO municipio ( id_uf, nome )";
        $sql[] = "VALUES (";
        $sql[] = "'{$idUf}', '{$nome}'";
        $sql[] = ")";

        if ( mysqli_query( $conexao, join( ' ', $sql ) ) ) {
            $return->result = 1;
        } else {
            $return->result = 0;
            $return->error = mysqli_error( $conexao );
        }
        return $return;
    }
```

Figura 27 – Classe do modelo o qual realiza a persistência dos dados no banco.

Apêndice 1 – Artigo

Fonte: O Autor

SGD – Sistema de Gestão de DocenteS

Bruno Henrique Ferronato

Departamento de Informática e Estatística(INE)

Universidade Federal de Santa Catarina(UFSC) – Florianópolis, SC - Brasil

bferronato@gmail.com

Abstract*. This work describes the design and implementation of a system for the management of professors at the Federal University of Santa Catarina. The system was developed to meet the needs of the CPPD (Comissão Permanente de Pessoal Docente da UFSC), with the intention to automate their routine processes. Thus, we developed a system that centralizes the information for professors so that is possible easily generate the reports concerning the qualifications of the professor, to consult the registration information and other attributes. The system uses the Internet through technologies consolidated and gratuitous, and none spent on proprietary licenses. With the implemented system we obtained a large gain in time and effort in the works that were performed manually and are now performed through this system.*

Resumo. *Neste trabalho é descrito o projeto e a implementação de um sistema para a gestão de docentes da Universidade Federal de Santa Catarina. O sistema foi desenvolvido para atender as necessidades da Comissão Permanente de Pessoal Docente, com a intenção de automatizar os seus processos rotineiros. Com isso foi desenvolvido um sistema que possibilita a centralização das informações dos professores para que facilmente seja possível gerar os relatórios de qualificação do professor, fazer consultas a dados cadastrais entre outros. O sistema foi desenvolvido para ser utilizado através da Internet utilizando tecnologias consolidadas e gratuítas, não tendo nenhum gasto com licenças de proprietárias. Com o sistema implementado obteve-se um grande ganho de tempo e esforço nos trabalhos que eram executados manualmente e agora são realizados através do sistema.*

Introdução

A CPPD foi Constituída através do Decreto nº 94664/87 e regulamentada pela Portaria nº 475/87 do Ministério da Educação, para assessorar aos Órgãos Deliberativos Centrais na formulação, aperfeiçoamento e modificação da política

de pessoal docente da UFSC. Está vinculada a Pró-Reitoria de Ensino de Graduação da UFSC.

Os assuntos tratados manualmente pelos funcionários da CPPD, inerentes aos processos de professores, tem se tornado bastante difícil devido ao crescimento da quantidade de processos a cada ano, e ao grande número de documentos necessários para o controle de uma progressão funcional.

Como solução, este trabalho propõe a criação de um sistema cliente-servidor, denominado SGD – Sistema de Gestão de Docentes, para disponibilizar o acesso rápido, *online,* de informações relativas aos processos tramitando na CPPD.

Metodologia de desenvolvimento

O sistema SGD surgiu com a necessidade de se informatizar os processos rotineiros realizados na manipulação de processos de pedidos de progressões funcionais realizados pelos professores da UFSC à CPPD.

As linguagens utilizadas no sistema foram o PHP, como linguagem de programação, o Mysql como banco de dados, o jquery para linguagem de script, o Smarty para o gerenciamento dos templates e css para o design do sistema.

Também foi utilizado o Google Code para realizar todo o gerenciamento do projeto (http://code.google.com/p/sgd/), como também o controle de versão do código fonte.

Para iniciar o sistema, fez-se a engenharia reversa de um banco de dados já existente para se obter o modelo entidade relacional. Após isso, desenvolveu-se o sistema seguindo o padrão MVC e o paradigma de orientação a objetos, para se ter uma melhor organização do código e também a separação das classes de controle, visão e persistência.

No sistema, toda a inclusão de scripts e arquivos é centralizada para facilitar a manutenção do sistema, além disso, todas as requisições ao sistema passam por um controlador central que faz o roteamento das requisições dentro do sistema.

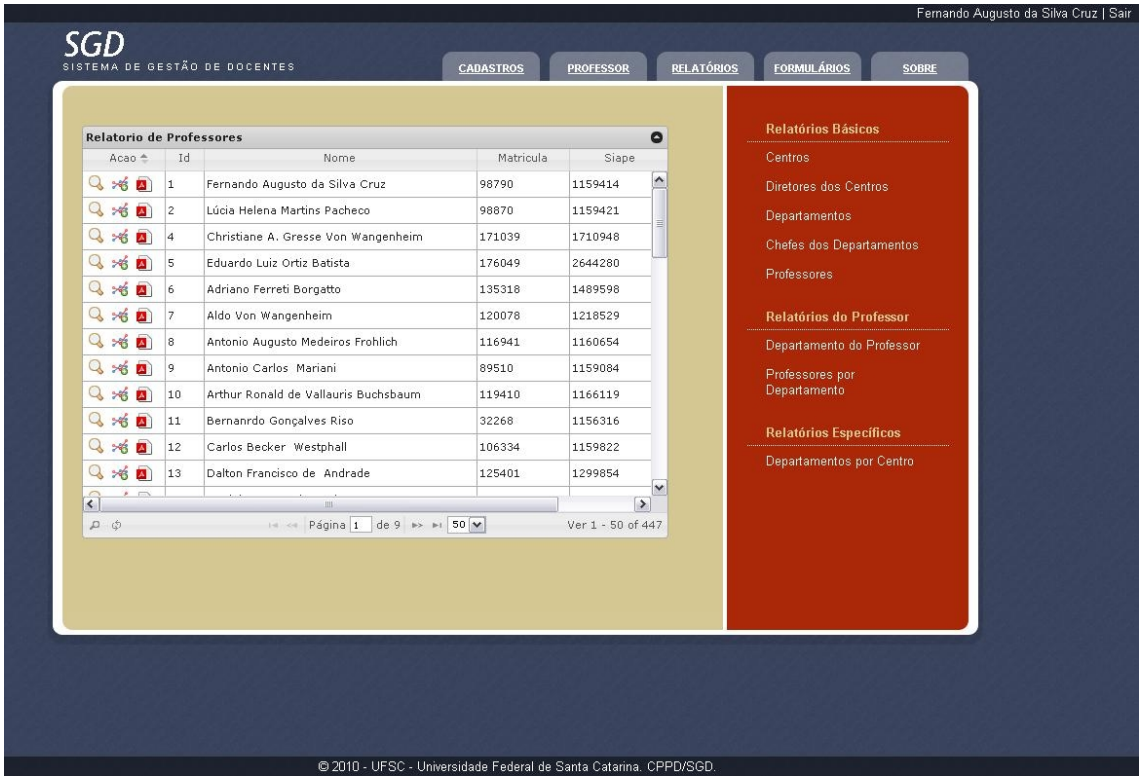


Figura 1 – Tela de relatório de professores

Ferramentas

Para o desenvolvimento do sistema, foram utilizadas diversas ferramentas, entre as principais podemos citar:

- PhpMyAdmin para a administração e manipulação do banco de dados.
- Zend Studio para a codificação do sistema.
- FileZilla como cliente de FTP para realizar o envio de arquivos para o servidor central
- Firebug, que é um plugin do Firefox para o debug do código.
- Wamp, que é um pacote com as instalações do Apache, Mysql e PHP.
- DBDesigner para o desenho do banco de dados.

Conclusão

Uma vez colocado em uso o sistema trouxe significativos benefícios para os usuários entre os quais a geração automatizada das qualificações dos professores.

As informações que eram mantidas e preservadas em meios físicos através de fichas de papel, agora serão todas digitalizadas e incluídas no sistema. Tal procedimento é necessário, pois caso haja alguma dúvida em relação ao dado cadastrado, bastará uma consulta à ficha digitalizada para seu esclarecimento, além de manter, historicamente, os dados.

Além disso, a criação do sistema trouxe um enorme ganho na busca de informações relativas aos professores, na geração de relatórios e na agilidade com que a qualificação de um professor pode ser obtida, ganhando também na qualidade dos dados. Evita-se assim, erros de transcrição da informação da ficha de papel para cada novo documento criado.

Trabalhos Futuros

Como trabalhos futuros a serem implementados na aplicação, podemos listar alguns abaixo:

- Criação de tabelas para a avaliação dos trabalhos dos professores durante o período de uma progressão.

- Análise semântica das informações para que o preenchimento dessas informações seja todo automático.

- Integração do banco de dados do sistema ao banco de dados do servidor central da UFSC para que as informações referentes ao ensino e pesquisa do professor possam ser utilizadas de forma que alguns dados sejam preenchidos automaticamente, diminuindo significativamente o trabalho do no preenchimento de formulários de cálculo de processos, sendo esse cálculo realizado todo automaticamente.

Referências

DALL'OGLIO, Pablo. **PHP Programando com orientação a Objetos.** São Paulo:Novatec, 2007. 574 p.

Wikipedia. **JavaScript**. Disponível em: <http://pt.wikipedia.org/wiki/JavaScript>. Acesso em 20 set. 2010.

Wikipedia. **JQuery**. Disponível em: <http://pt.wikipedia.org/wiki/JQuery>. Acesso em 20 set. 2010.

FEITOSA, Ciro. **Smarty e PHP, tudo a ver**. Ciro Feitosa. [s.l.] jan. 2006. Disponível em: <http://cirofeitosa.com.br/post/smarty-e-php-tudo-a-ver>. Acesso em 21 set. 2010

MySql Manual. **Introdução ao programa MySQL Query Browser.** Disponível em: <http://dev.mysql.com/doc/query-browser/pt/mysql-query-browser-introduction.html>. Acesso em 23 set. 2010.

Pergamim. **Informatizar Por Que**. Disponível em: <http://www.pergamum.pucpr.br/redepergamum/trabs/Camila_K_Burin-Informatizar_por_que.pdf>. Acesso em 17 out. 2010

Prefeitura Porto Alegre. **Progressão Funcional**. Disponível em: <http://www2.portoalegre.rs.gov.br/sma/default.php?p_secao=86>. Acesso em 17 out. 2010

Portal do Servidor. **Progressão Funcional**. Disponível em: <http://www.portaldoservidor.sc.gov.br/index.php?option=com_docman&task=doc_download&gid=463>. Acesso em 17 set. 2010

DevMedia. **DBDesigner: uma ferramenta gratuita para modelagem de dados**. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=6840>. Acesso em 23 set. 2010

Wikipedia. **FileZilla**. Disponível em: <http://pt.wikipedia.org/wiki/FileZilla>. Acesso em 22 set. 2010

Silva, Danilo Rodrigues da. **Evolução da Arquitetura Cliente/Servidor:Definição, caminhos e rumos**. Disponível em: <http://knol.google.com/k/danilo-rodrigues-da-

silva/evolução-da-arquitetura-cliente-servidor/anrd8mpq3bji/3>. Acesso em 07/11/2010.