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

BRUNO HENRIQUE GASPARIM CORAZZA

APLICATIVO PARA CONTROLE DE ATUALIZAÇÕES EM BASES DE DADOS DE SISTEMAS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2014

BRUNO HENRIQUE GASPARIM CORAZZA

APLICATIVO PARA CONTROLE DE ATUALIZAÇÕES EM BASES DE DADOS DE SISTEMAS

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: **241**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **BRUNO HENRIQUE GASPARIM CORAZZA**.

Às 17:23 hrs do dia 6 de agosto de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Rúbia E. O. Schultz Ascari (Convidada) e Lucilia Yoshie Araki (Convidada), para avaliar o Trabalho de Diplomação do aluno Bruno Henrique Gasparim Corazza, matrícula 1167480, sob o título **Aplicativo para Controle de Atualizações em Bases de DAdos de Sistemas**; 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 17:53 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Rúbia E. O. Schultz Ascari, M.Sc.
Convidada

Profa. Lucilia Yoshie Araki, M.Sc.
Convidada

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

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

CORAZZA, Bruno Henrique Gasparim. Aplicativo para controle de atualizações em bases de dados de sistemas. 2014. 39 f. Trabalho de conclusão de curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2014.

A empresa que utilizará o sistema resultado deste trabalho possui uma base de dados comum que é replicado para cada um de seus clientes, que compartilham da mesma versão de um sistema *online*. Verificou-se, assim, que o desenvolvimento de um sistema para controle de atualizações dessas bases de dados poderia ser bastante útil, inclusive porque minimizaria o trabalho manual e permitiria um gerenciamento mais efetivo do que foi realizado na base de dados de cada um dos clientes do sistema. Assim, a proposta deste trabalho é a implementação de um aplicativo de controle de atualizações e versões das bases de dados. O aplicativo desenvolvido é para ambiente *web* e o desenvolvimento tem como base a linguagem PHP e o Zend Framework.

Palavras-chave: Linguagem PHP. Zend Framework. Aplicativos web.

ABSTRACT

CORAZZA, Bruno Henrique Gasparim. Application to control updates in databases of systems. 2014. 39 f. Trabalho de conclusão de curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2014.

The company that will use the system resultant from this work has a common database that is replicated to each of its clients, who share the same version of an online system. It was found, therefore, that the development of a system to control updates of these databases could be quite useful, especially because it would minimize manual work and allow a more effective management of what was done in the database of each client on the system. The proposal of this work is the implementation of an application that controls the updates and versions of databases. The application is designed for web environment and the development is based on PHP and Zend Framework language.

Keywords: PHP language. Zend Framework. Web system.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

IDE	*Integrated Development Environment*
GUI	*Graphical User Interface*
HTML	*Hipertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
MVC	*Model-View-Controller*
SQL	*Srtructured Query Language*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo são apresentadas as considerações iniciais, com o contexto no qual se insere a proposta deste trabalho, os seus objetivos e a justificativa. Por fim está a organização do texto, por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Para aplicativos de software desenvolvidos como um conjunto de módulos ou funcionalidades, em que clientes distintos utilizam funcionalidades iguais e também específicas, um aplicativo que auxilie no gerenciamento e nas atualizações dessas funcionalidades seria bastante útil.

Nesse tipo de aplicativo, quando é realizada uma alteração no projeto da base de dados que é compartilhada por todos os clientes, há a dificuldade em atualizar as demais bases de dados replicadas ("clonadas"), que são relativas a cada cliente. Na empresa para a qual está sendo desenvolvido o sistema proposto como resultado deste trabalho é necessário conectar manualmente em cada base de dados e executar linhas de comando *Srtructured Query Language* (SQL) para as atualizações necessárias.

Quando a quantidade de clientes é grande é considerável o tempo demandado para realizar essas operações. O tempo para realizar a atualização de uma base de dados é multiplicado pela quantidade de clientes. Assim, verificou-se a necessidade de desenvolver um sistema que permitisse o cadastro das bases de dados, por meio do seu endereço, usuário, senha e nome da base de dados. Com as bases cadastradas é possível selecionar bases específicas e criar rotinas para execução de código SQL realizadas em todas as bases selecionadas.

Para a implementação do sistema para realizar essas atualizações foi escolhida a linguagem PHP e o Zend Framework. A modelagem do aplicativo e o estudo das tecnologias foi realizado como estágio supervisionado pelo autor deste trabalho.

1.2 OBJETIVOS

O objetivo geral está relacionado ao resultado principal da realização deste trabalho e os objetivos específicos o complementam.

1.2.1 Objetivo Geral

Implementar um aplicativo para realização de atualizações comuns em bases de dados de clientes distintos.

1.2.2 Objetivos Específicos

- Implementar as funcionalidades do aplicativo para controle de atualizações utilizando a linguagem PHP e o Zend Framework;
- Agilizar e facilitar a realização de atualizações de bases de dados comuns entre clientes que compartilham o mesmo sistema.

1.3 JUSTIFICATIVA

Um sistema para controle de atualizações em bases de dados automatiza atividades que são realizadas de forma manual, facilitando e agilizando o processo de realização das mesmas. Além disso, proporciona uma maneira de gerenciar essas atualizações.

A escolha da linguagem PHP se justifica por ela permitir o desenvolvimento para *web* de maneira simples do ponto de vista do programador. A sintaxe da linguagem é de fácil entendimento e a interação com *Hipertext Markup Language* (HTML) também é facilitada. E os incrementos da própria linguagem (a versão 5 é orientada a objetos), o uso de *frameworks*, como o Zend que implementa o padrão de projetos *Model-View-Controller* (MVC), e de Ajax

facilitam e agilizam o trabalho de implementação.

As características da linguagem PHP como intuitividade, execução rápida, multiplataforma e de código fonte aberto a tornaram como uma das mais importantes para desenvolvimento para *web* (CUI et al., 2009). O uso do padrão MVC com essa linguagem provê uma maneira mais efetiva de geração de aplicações modulares e organizadas. Isso porque esse padrão permite dividir uma aplicação em camadas que podem ser analisadas e implementadas separadamente, por desacoplar modelo de dados e a forma de apresentação desses dados ao usuário. MVC auxilia a reduzir a complexidade da aplicação e a incrementar flexibilidade e reuso de código (CUI et al., 2009).

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta as considerações iniciais, o objetivo e a justificativa.

O Capítulo 2 apresenta o referencial teórico centrado no desenvolvimento de aplicações *web* utilizando o padrão de projetos MVC e o *framework* Zend.

O Capitulo 3 apresenta os materiais utilizados na modelagem e no desenvolvimento do sistema e o método. Os materiais de referem às ferramentas e às tecnologias utilizadas. O método apresenta as principais atividades realizadas para desenvolver o trabalho.

No Capítulo 4 está o resultado obtido da realização deste trabalho que é a modelagem e a implementação do sistema. A modelagem é apresentada por meio de diagramas e explicações textuais e a implementação por meio de telas e partes do código desenvolvido. Uma primeira versão da modelagem foi realizada como estágio curricular pelo autor deste trabalho, mas é apresentada visando facilitar o entendimento do sistema implementado.

Por fim, está o Capítulo 5 com a conclusão.

2 MVC E ZEND NO DESENVOLVIMENTO DE APLICAÇÕES WEB

Este capítulo apresenta o referencial teórico do trabalho que se refere ao desenvolvimento de aplicações *web* utilizando o padrão de projetos MVC e a linguagem PHP, juntamente com Zend Framework. O Capítulo inicia com a apresentação sobre padrões de projeto, centrando-se no MVC, em seguida é apresentado sobre *frameworks* para desenvolvimento *web*, focando o Zend, que é um *framework* para desenvolvimento em PHP que tem como base o padrão de projetos MVC.

2.1 PADRÕES DE PROJETO

Padrões provêem um meio de capturar o conhecimento sobre uma solução bem-sucedida no desenvolvimento de *software* (SELFA, CARRILLO, BOONE, 2006). Um padrão de projeto, ou *design pattern,* pode ser considerado como o reuso de experiências e de conhecimento adquiridos com o desenvolvimento de projetos na solução de problemas recorrentes. O problema e a respectiva solução são devidamente documentados e assim denominados um padrão de projeto porque se referem às situações específicas de projeto. Padrões de projeto são formas já testadas e documentadas de resolver certos tipos de problemas (MINETTO, 2007).

De acordo com Germoglio (2010) a aplicação de um padrão de projeto tem efeito mais restrito na solução. É comum que os padrões de projeto estejam associados à implementação do sistema, ao uso de tecnologias. Embora o que é reusado são conceitos que caracterizam problemas e as respectivas soluções.

Os benefícios do uso de padrões em um projeto são (MCCONNELL, 2004):

a) Redução da complexidade da solução ao prover abstrações reusáveis - um padrão de projeto define elementos e relações entre esses elementos, diminuindo a quantidade de conceitos novos a serem utilizados na solução. Uma solução já empregada pode ser utilizada em um contexto semelhante para resolver problemas semelhantes ou mesmo ser adaptada para ajustar-se às especificidades do novo problema.

b) Reuso - como padrões de projeto são soluções de projeto para problemas recorrentes, é possível que a implementação (parcial ou total) do padrão possa ser reusada,

facilitando o desenvolvimento.

c) Geração de alternativas - padrões de projeto distintos podem resolver o mesmo problema. Assim, é possível avaliar e escolher entre soluções possíveis, considerando os benefícios e as desvantagens de cada uma delas.

d) Facilidade de comunicação - os padrões de projeto descrevem conceitos e elementos que estarão presentes no projeto.

Buschmann et al. (1996) definiram 23 padrões de projetos, dentre os quais estão: *adapter*, *decorator, façade, observer* e *factory*. Existem outros padrões de projetos como, por exemplo, o *Model-View-Controller*.

O padrão MVC é apresentado a seguir por ser o utilizado para implementar o sistema obtido como resultado da realização deste trabalho.

2.1.2 Padrão *Model-View-Controller*

O *Model-View-Controller* (denominado como paradigma, modelo, arquitetura, padrão) foi inicialmente projetado para interfaces em aplicações implementadas com Smaltalk e tem se tornado um padrão de projeto para interfaces com usuários. Nesse padrão não há a necessidade de considerar a linguagem de implementação da aplicação e é bastante útil para aplicações nas quais os componentes de controle são alterados frequentemente (SELFA, CARRILLO, BOONE, 2006).

A separação entre o tratamento dos dados, a lógica de negócio e a apresentação dos dados é um aspecto muito relevante. É comum que seja necessário que informações fornecidas por sistemas sejam personalizadas para diferentes necessidades dos usuários. Para Selfa, Carrillo, Boone (2006) isso gera alguns problemas que precisam ser resolvidos, dentre os quais estão:

a) A mesma informação deve ser mostrada em formatos diferentes e em visões diferentes;

b) Mudanças em uma visão devem ser refletidas nas outras;

c) Mudanças na interface do usuário devem ser fáceis de implementar;

d) A funcionalidade central do sistema deve ser independente da interface para permitir que múltiplas interfaces coexistam e para que os mesmos dados sejam apresentados de maneiras distintas visando atender interesses dos diversos usuários que utilizam um mesmo

sistema.

O padrão MVC resolve esses problemas por meio da separação das funcionalidades do sistema da sua interface e incorpora um mecanismo para propagar as mudanças para as demais visões do sistema (SELFA, CARRILLO, BOONE, 2006).

O Model-View-Controller é um padrão de projeto (*design pattern*) frequentemente utilizado em aplicações que necessitam manter múltiplas visões de um mesmo conjunto de dados. O MVC provê uma separação clara de objetos em três partes: modelo, visão e controle. Por causa dessa separação, múltiplas visões e controles podem interagir com um mesmo modelo. E novos tipos de visões e controles podem interagir com o modelo sem necessidade de realizar alterações no modelo definido no projeto.

O MVC é um conceito de desenvolvimento e projeto que visa organizar a separação de uma aplicação em três partes ou componentes distintos e cada parte é especializada em uma tarefa (GAMMA et al., 2000; GONÇALVES et al., 2005; SWEAT, 2005; SELFA, CARRILLO, BOONE, 2006; CUI et al. 2009). Essas partes são:

a) Modelo (*model*) - é o domínio da representação dos dados com os quais a aplicação opera. O modelo está relacionado ao gerenciamento da visão. O modelo encapsula os dados do aplicativo e as regras de negócio. Muitos aplicativos utilizam um mecanismo de persistência (armazenamento) de dados, como, um banco de dados, por exemplo. MVC não menciona a camada de acesso aos dados porque é subentendido que a mesma está encapsulada pelo modelo.

b) Visão (*view*) - a visão exibe os dados ou informações da aplicação para o usuário. A visão extrai dados do modelo e os formata para apresentação ao usuário. A visão apresenta o modelo de forma a suportar interação tipicamente representada pelos elementos de interface com o usuário. Múltiplas visões com diferentes objetivos podem existir para um único modelo.

c) Controle (*control*) - o controle gerencia as atualizações da visão. O controle coordena o modelo e a visão exibe a interface correta ou executa ações que devem ser realizadas pela aplicação. O controle direciona o fluxo do aplicativo e recebe entradas, traduzindo-as para o modelo e para a visualização. É o controle quem interpreta as entradas do usuário (como de *mouse* e teclado) e orienta o modelo e a visão para as mudanças necessárias. O controle processa e responde a eventos (normalmente ações do usuário) e pode indiretamente invocar mudanças no modelo.

A Figura 1 é uma representação esquemática da interação entre modelo, visão e controle do padrão MVC e inclui o usuário, representando a interação com o sistema.

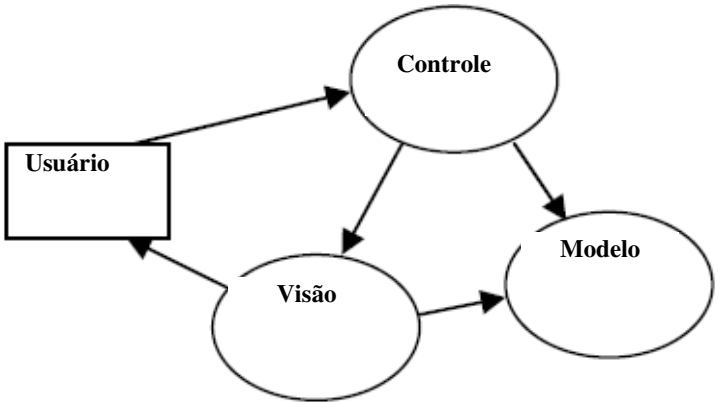

Figura 1 – MVC Architecture
Fonte: Selfa, Carrillo e Boone (2006, p. 49).

Pela representação esquemática da Figura 1, o padrão de projetos MVC pode ser considerado passivo, o que significa que o modelo não sabe da existência do controle e da visão. Por exemplo, o modelo é um texto que pode somente ser alterado pelo usuário. Contudo, na maioria dos casos o modelo deve ter uma ligação com a visão para informá-la de mudanças feitas em seu estado que são causadas por procedimentos internos. A visão e o controle estão sempre conectados. O controle se comunica com a visão para determinar quais objetos estão sendo manipulados pelo usuário e chamar os métodos do modelo para fazer mudanças nesses objetos. O modelo realiza as mudanças e notifica a visão para atualizá-la.

O MVC tem como principal objetivo separar dados e a lógica de negócios (*Model*) da interface do usuário (*View*) e do fluxo da aplicação (*Controller*). O objetivo é permitir que dados produzidos a partir das regras definidas pela lógica de negócio possam ser visualizados por meio de interfaces distintas. Na arquitetura *MVC*, a lógica de negócio (o modelo) não sabe quantas nem quais as interfaces com o usuário estão exibindo o seu estado. Para a visão não importa a origem dos dados, mas ela precisa garantir que sua aparência reflita o estado do modelo, ou seja, sempre que o estado do modelo muda, o modelo notifica as visões para que elas sejam atualizadas. Os estados refletem as alterações nos dados sendo apresentados.

O MVC apresenta benefícios para aplicações interativas permitindo múltiplas representações da mesma informação e promovendo reuso de código (SELFA, CARRILLO, BOONE, 2006). Além disso, as suas vantagens com relação a outros padrões de projeto são (LARMAN, 2007; MCHEICK, QI, 2011):

a) Baixo acoplamento – as funcionalidades de cada camada são implementadas de forma distinta.

b) Alta coesão – a separação em camadas auxilia a definir de forma clara as

funcionalidades de cada classe ou componente do sistema.

c) As visões provêem alta flexibilidade e agilidade – múltiplos modelos de visão podem ser criados, adicionados, modificados e eliminados dinamicamente. As visões são separadas do modelo.

d) Clareza de projeto – as funcionalidades relacionadas à apresentação, as operações com os dados e as regras de negócio estão bem separadas, facilitando, inclusive o trabalho por equipes distintas em um mesmo projeto.

e) Manutenção – as funcionalidades separadas por camadas facilitam a manutenção e a atualização do sistema. É possível, por exemplo, trocar componentes de acesso a dados sem alterar as demais camadas. O mesmo ocorre com a interface.

f) Alta escalabilidade – a implementação em camadas facilita a inclusão de funcionalidades.

2.1.3 Padrão *Model-View-Controller* e PHP

PHP é uma linguagem de *script* e, assim, é comum que o código escrito em PHP seja incorporado em documentos HTML. Com essa forma de programação todas as funcionalidades do sistema são colocadas no mesmo arquivo que está o HTML que apresenta a interface do aplicativo. Essa forma de desenvolvimento gera dificuldades para, por exemplo, realizar alterações na página. Além disso, reusar fica mais difícil porque o código PHP responsável pelas regras de negócio e persistência dos dados e o HTML que define a interface com o usuário estão mesclados. E agregados a esses podem estar rotinas em linguagens de *scritpt*, como, *JavaScript*, por exemplo.

O código PHP e HTML escritos em um mesmo arquivo é definido como uma forma tradicional de utilizar linguagens de *script*, como é considerada PHP, para aplicativos *web*. Essa forma de implementação possui as seguintes vantagens, de acordo com Wang (2011):

a) A estrutura do código é simples, tornando-o fácil de entender e usar;

b) O código pode ser facilmente inserido em arquivo que compõe a página;

c) Programas PHP, páginas HTML e outros arquivos podem ser combinados em uma única página, reduzindo as requisições e melhorando a eficiência.

Apesar das vantagens, a estrutura tradicional da linguagem PHP apresenta desvantagens que são (WANG, 2011):

a) A grande flexibilidade de produzir o código pode resultar em dificuldades de leitura

e gerenciamento;

b) O modelo em cascata e de desenvolvimento rápido utilizado em estrutura PHP tradicional propicia o desenvolvimento de código de difícil manutenção e reuso.

Embora o desenvolvimento de código PHP de forma tradicional tenha vantagens, suas desvantagens podem gerar problemas que não podem ser negligenciados. Contudo, utilizando o padrão MVC é possível resolver esses problemas (WANG, 2011) porque o código PHP e o código HTML são separados, facilitando o entendimento e a depuração. Além disso, o uso de padrões e classes facilita a manutenção de código. Com MVC, o controle chama as classes do modelo para realizar as funcionalidades necessárias. E o controle recebe esse retorno e chama a visão. O controle utiliza as mensagens de retorno do modelo para apresentar na visão a página ao usuário.

2.2 FRAMEWORKS

Uma das grandes vantagens de uso de *frameworks* está na automatização de tarefas repetitivas (MINETTO, 2007). Por exemplo, as operações de manipulação de dados em uma base de dados são praticamente iguais para todas as tabelas de um sistema. Assim, a geração das funções para as operações de inclusão, exclusão, alteração e consulta pode ser automatizada por meio de *frameworks*. Minetto (2007) cita, ainda, como vantagem dos *frameworks* a separação da apresentação e da lógica de negócio, a facilidade de geração de testes automatizados e de documentação.

2.2.1 Zend Framework

O Zend Framework é um *framework* de desenvolvimento em PHP, que contém vários componentes que visam prover organização no desenvolvimento e também a reutilização de código (ZEND, 2014). O Zend provê o padrão MVC possibilitando separar a GUI (*Graphical User Interface*) e *templates* (visão) da lógica de negócio (controle) e dos dados (modelo).

A flexibilidade quanto à utilização de bibliotecas e até mesmo de outros *frameworks* é uma característica do Zend. Ele permite que, por exemplo, seja utilizado um *framework* de mapeamento objeto relacional na camada *model*. O Zend é composto por vários componentes,

sendo que cada um deles é responsável por uma tarefa específica. Na Figura 2 estão listados os principais componentes desse *framework* organizados por categorias.

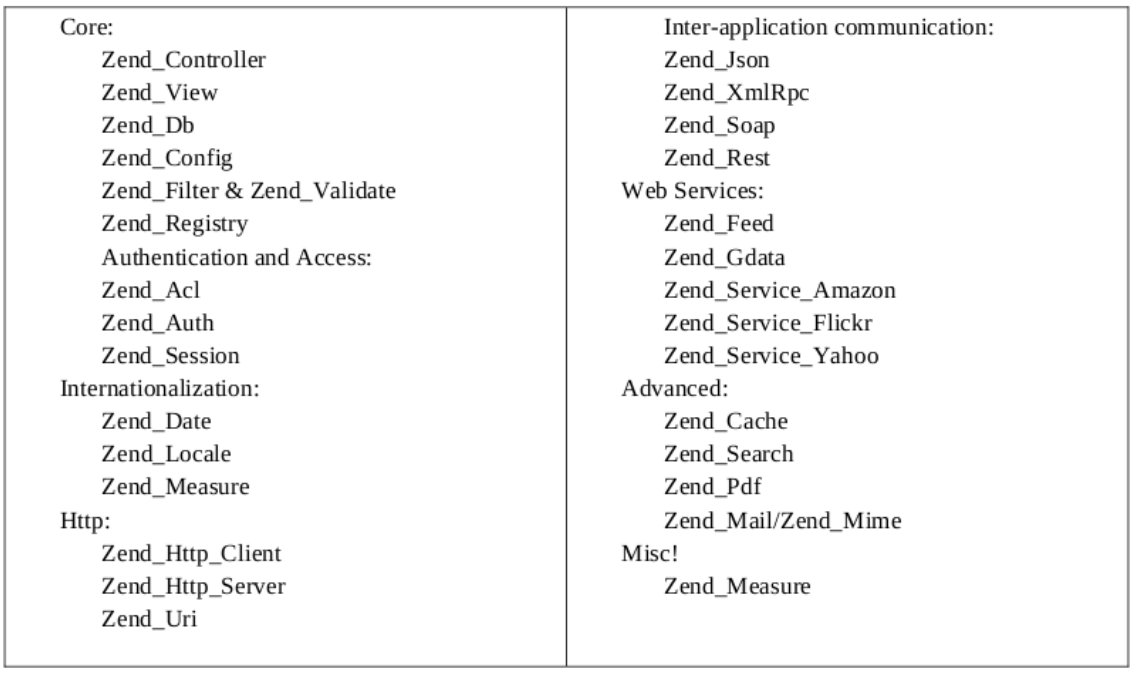

Figura 2 – Componentes principais do Zend
Fonte: Allen e Lo (2007, p. 10).

Os agrupamentos listados na Figura 2 são:

a) *Core* – componentes responsáveis pelas funcionalidades para trabalhar com o modelo MVC. O componente Zend_Controller provê os métodos e as classes para a camada controle. O componente Zend_View provê todos os mecanismos para a camada de visão.

b) *Authentication and Access* – os componentes desta categoria visam prover uma forma de autenticação e de autorização de acesso às funcionalidades do sistema.

c) *Internationalization* – componentes para prover internacionalização do sistema.

d) *Http* – componentes que permitem a realização de requisições com o protocolo HTTP (*Hypertext Transfer Protocol).*

e) *Inter-application communication* – os componentes Zend_Json, Zend_Xml_Rpc, Zend_Soap e Zend_Rest provem a comunicação com outras aplicações por meio do protocolo HTTP.

f) *Web Services* – interação com vários serviços *web*, tais como, o Yahoo pelo componente Zend_Service_Yahoo e o Flickr pelo Zend_Service_Flickr.

g) Advanced – componentes para tarefas como um gerador de arquivos pdf (Zend_Pdf) e um componente para envio de email (Zend_Mail).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados. Os materiais se referem às tecnologias e as ferramentas usadas para modelar e implementar o sistema. O método reporta a sequência das principais atividades realizadas para desenvolver este trabalho.

3.1 MATERIAIS

As tecnologias e ferramentas utilizadas para a modelagem e a implementação do sistema são apresentadas no Quadro 1.

Nome	Versão	Disponibilização/referência	Aplicação no projeto
Astah Community	6.2.1	http://astah.net/editions/community Astah (2014).	Ferramenta de modelagem dos casos de uso
MySQL Workbench	6.0 CE	http://www.mysql.com/products/workbench/ MySQL Workbench (2014).	Para a modelagem do diagrama de entidades e relacionamento do banco de dados
PHP	5.3.4	http://br.php.net/ PHP (2014).	Linguagem de programação
Zend Studio	10.0.0	http://www.zend.com/products/studio/ Allen (2007).	IDE (*Integrated Development Environment*) para codificação do sistema
Ajax		Asynchronous Javascript and XML GARRET (2005).	Para implementar o assincronismo de mensagens entre cliente e servidor
HTML	5.0	http://www.w3.org/TR/html5/ HTML 5 (2014).	Para o desenvolvimento da interface *web*
jQuery	1.8.2	http:// http://api.jquery.com Jquery (2014).	Na implementação da interface
Zend Framework	2.0	http://framework.zend.com/ Zend (2014).	Para implementação do padrão MVC
MySQLFront	5.0	http://www.mysqlfront.de/ MySQL Front (2014).	Gerenciamento do banco de dados
MySQL	5.1.53	http://www.mysql.com/ Milani (2007). MySQL (2014).	Banco de dados
Apache	2.2.17	http://www.apache.org/ Apache (2014).	Contêiner *web*

Quadro 1 – Ferramentas e tecnologias

3.2 MÉTODO

As etapas para a modelagem e a implementação do sistema para controle de atualizações de bases de dados seguiram o modelo sequencial linear proposto por Pressman (2008). O uso desse modelo é justificado porque o autor deste trabalho tem conhecimento dos aspectos de negócio e técnicos envolvidos no desenvolvimento do aplicativo, incluindo a definição dos requisitos. O autor trabalha na empresa para a qual o sistema foi implementado. As etapas definidas foram:

a) **Levantamento dos requisitos**

Esse sistema surge de uma necessidade de uma empresa. Assim, a definição dos requisitos foi realizada com o auxílio dos funcionários da empresa que realizam manualmente os procedimentos que serão realizados automaticamente pelo sistema desenvolvido. A definição dos requisitos foi realizada de forma completa como trabalho de estágio. Apenas ajustes, no sentido de melhoria, foram realizados.

b) **Análise e projeto**

Os requisitos definidos foram modelados sob a forma de casos de uso e diagrama de entidades e relacionamentos do banco de dados no trabalho de estágio.

c) Implementação

A implementação foi realizada utilizando o ambiente Zend Studio.

d) Testes

Os testes realizados tiveram o objetivo de verificar erros de código e a interação com o sistema.

4 RESULTADOS

Este capítulo apresenta a modelagem e a implementação de um aplicativo para controle de atualizações de bases de dados.

4.1 ESCOPO DO SISTEMA

Ao efetuar alterações no projeto geral da base de dados há a dificuldade em atualizar as demais bases de dados clonadas, relativas a cada cliente. Atualmente para realizar essa tarefa, na empresa para o qual o sistema objeto deste trabalho foi implementado, é necessário conectar manualmente cada base de dados e executar as instruções SQL necessárias. Essas instruções realizadas em cada base podem ser geradas por qualquer sistema de projeto de banco de dados ou escritas pelo usuário do sistema. Com uma grande quantidade de clientes é muito grande o tempo necessário para que uma atualização de base de dados de um mesmo sistema seja efetuada na base individual de cada cliente.

A solução encontrada foi desenvolver um aplicativo que permitisse cadastrar bases de dados, por meio de seu endereço, usuário, senha e nome do banco. Posteriormente seria possível selecionar dentre as bases de dados cadastradas e criar uma rotina que executaria um código SQL nessas bases.

Esse aplicativo surge da necessidade de executar o mesmo comando em várias bases de dados replicadas. Por meio do aplicativo, o usuário pode cadastrar bases de dados para posteriormente executar comandos SQL nas bases que necessitar.

O aplicativo manterá um histórico que é gerado a partir de qualquer execução de código no sistema. Nesse histórico ficará armazenado a base de dados que foi alterada, a data e a hora da alteração, o usuário que executou a alteração, o código que foi executado e a resposta (*feedback*) retornada pelo sistema.

4.2 MODELAGEM DO SISTEMA

O diagrama de casos de uso (Figura 3) apresenta as principais funcionalidades do sistema e os atores definidos.

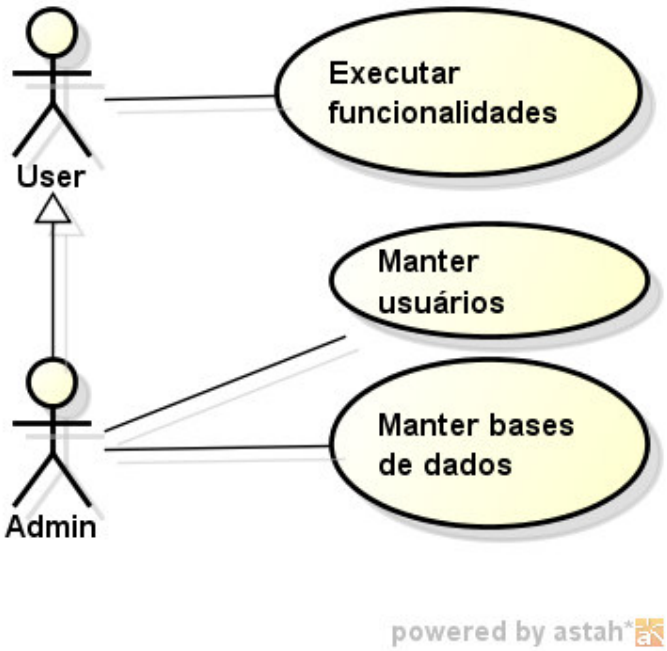

Figura 3 – Diagrama de casos de uso

De acordo com a representação da Figura 3 existem dois atores para o sistema. O ator denominado “User” que executa as funcionalidades, ou seja, as atualizações nas bases de dados cadastradas. Essas atualizações se referem a executar instruções SQL. E o ator “Admin”, o administrador, que realiza manutenção de usuários do sistema e das bases de dados.

A Figura 4 apresenta o diagrama de entidades e relacionamentos elaborado.

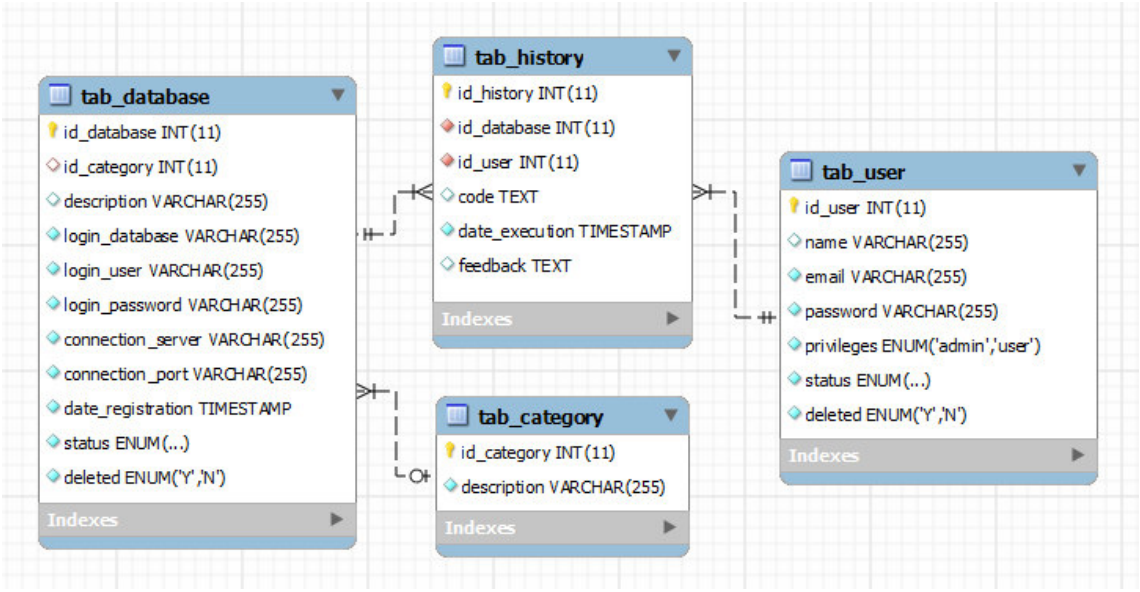

Figura 4 – Diagrama de entidades e relacionamentos

A seguir são apresentadas as tabelas constantes no diagrama apresentado na Figura 4.

A tabela tab_database apresentada no Quadro 2 armazena as bases de dados que poderão ser modificados através do sistema.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_database	Integer	Não	Sim	Não	Campo chave de identificação da base de dados.
id_category	Integer	Sim	Não	Sim	Chave estrangeira, oriunda da tabela de category.
description	Varchar	Sim	Não	Não	Campo utilizado para que o usuário grave uma descrição ou identificação para a base.
login_database	Varchar	Não	Não	Não	Nome da base de dados.
login_user	Varchar	Não	Não	Não	Usuário utilizado para conectar à base de dados.
login_password	Varchar	Não	Não	Não	Senha utilizada para conectar à base de dados.
connection_server	Varchar	Não	Não	Não	Endereço para conectar à base de dados.
connection_port	Varchar	Não	Não	Não	Porta utilizada para a conexão à base de dados.
date_registration	Timestamp	Não	Não	Não	Data de registro da base de dados.
status	Enum	Não	Não	Não	Informa se o registro está habilitado ou desabilitado.
deleted	Enum	Não	Não	Não	Utilizando o método de exclusão lógica, determina se o registro é excluído ou não.

Quadro 2 – Tabela tab_database

A tabela tab_user apresentada no Quadro 3 armazena os usuários que terão acesso ao sistema.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_user	Integer	Não	Sim	Não	Campo chave de identificação do usuário.
name	Varchar	Sim	Não	Não	Nome do usuário.
email	Varchar	Não	Não	Não	Email do usuário.
password	Varchar	Não	Não	Não	Senha de acesso ao sistema.
privileges	Enum	Não	Não	Não	Privilégio do usuário, administrador ou usuário.
status	Enum	Não	Não	Não	Informa se o registro está habilitado ou desabilitado.
deleted	Enum	Não	Não	Não	Utilizando o método de exclusão lógica, determina se o registro é excluído ou não.

Quadro 3 – Tabela tab_user

A tabela tab_category apresentada no Quadro 4 armazena as categorias em que o usuário deseja organizar suas bases de dados cadastrados.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_category	Integer	Não	Sim	Não	Campo chave de identificação da categoria.
description	Varchar	Não	Não	Não	Descrição ou identificação definida pelo usuário para a categoria.

Quadro 4 – Tabela tab_category

A tabela tab_history apresentada no Quadro 5 armazena o histórico de operações efetuadas pelo sistema.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id_history	Integer	Não	Sim	Não	Campo chave de identificação do histórico registrado.
id_database	Integer	Não	Não	Sim	Identifica à qual base de dados este histórico pertence.
id_user	Integer	Não	Não	Sim	Identifica qual usuário executou a ação.
code	Text	Não	Não	Não	Registra o código executado.
date_execution	Timestamp	Não	Não	Não	Registra a data de execução.
feedback	Text	Não	Não	Não	Registra o retorno fornecido pela linguagem utilizada.

Quadro 5 – Tabela tab_history.

4.3 APRESENTAÇÃO DO SISTEMA

Figura 5 apresenta a tela de *login* do sistema. Para acessar o sistema o usuário deve informar um *email* e a respectiva senha previamente cadastrados para que possa ser identificado pelo sistema e tenha seus privilégios concedidos.

Restringe-se ao usuário que não seja administrador apenas executar os códigos SQL nos bancos selecionados. Já ao administrador são permitidas todas as outras funcionalidades, que são: cadastrar usuários, categorias, bases de dados bem como ter acesso aos históricos de execução.

Figura 5 – Tela de login

A Figura 6 apresenta a listagem de bases de dados (*databases)* cadastradas, bem como o botão para adicionar uma nova *database* e uma nova categoria. Além disso, na listagem de cada item cadastrado é exibido o ícone de edição para que por meio do mesmo seja possível alterar ou excluir o registro.

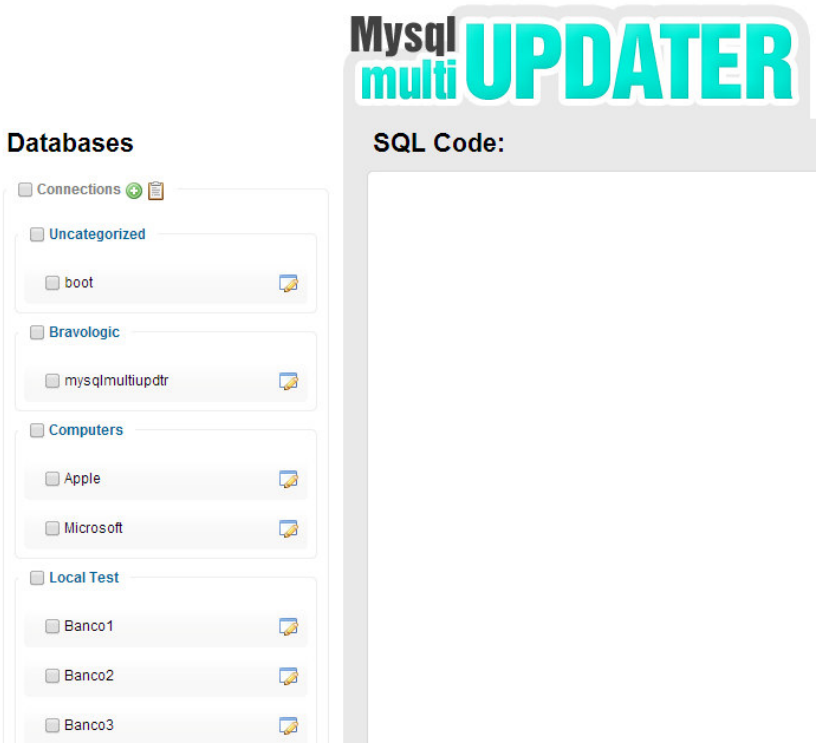

Figura 6 – Listagem de databases cadastradas

Ao clicar no ícone para adicionar uma *database* é exibida a tela apresentada na Figura 7. Para representar os campos obrigatórios é exibido um asterisco ao lado do nome do campo. A validação é realizada dinamicamente por meio de jQuery.

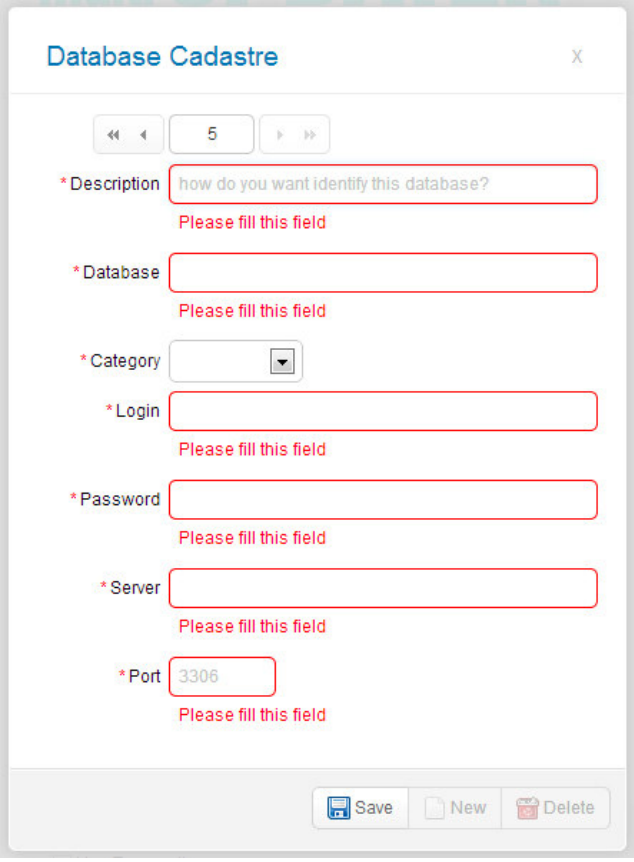

Figura 7 – Cadastro de nova database

Ao clicar no ícone de edição exibido na listagem na Figura 6, é aberta a mesma tela utilizada para o cadastro de uma nova *database*. Porém, com os *inputs* já preenchidos para facilitar a edição. Isto pode ser observado na Figura 8.

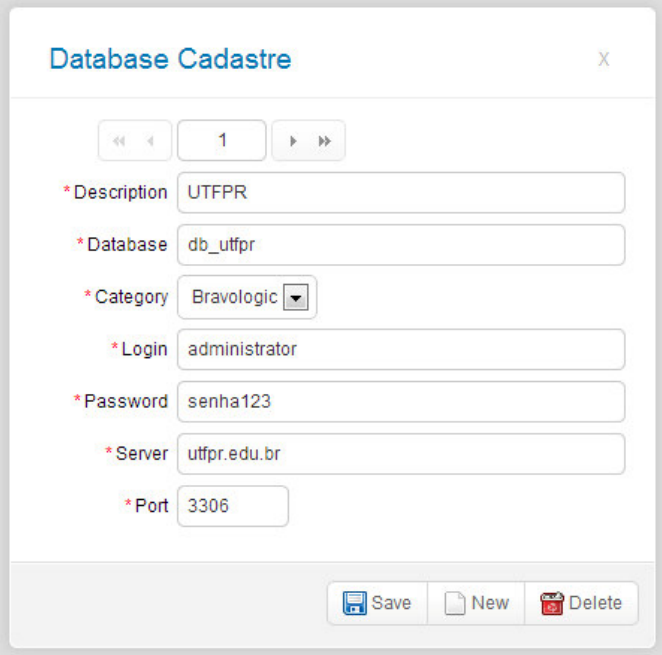

Figura 8 – Edição de cadastro de database

A Figura 9 complementa a Figura 6, mostrando por completo a área destinada para que o código SQL que será executado seja inserido. Logo acima está o botão ‘Users’ que ao ser clicado abre o cadastro de usuários e o botão ‘History’ que ao ser clicado exibe um relatório com o histórico de execuções de código feitas pelo sistema, relatório este que será exibido na Figura 10. Mais ao topo e a direita está a saudação ao usuário e o botão para sair do sistema.

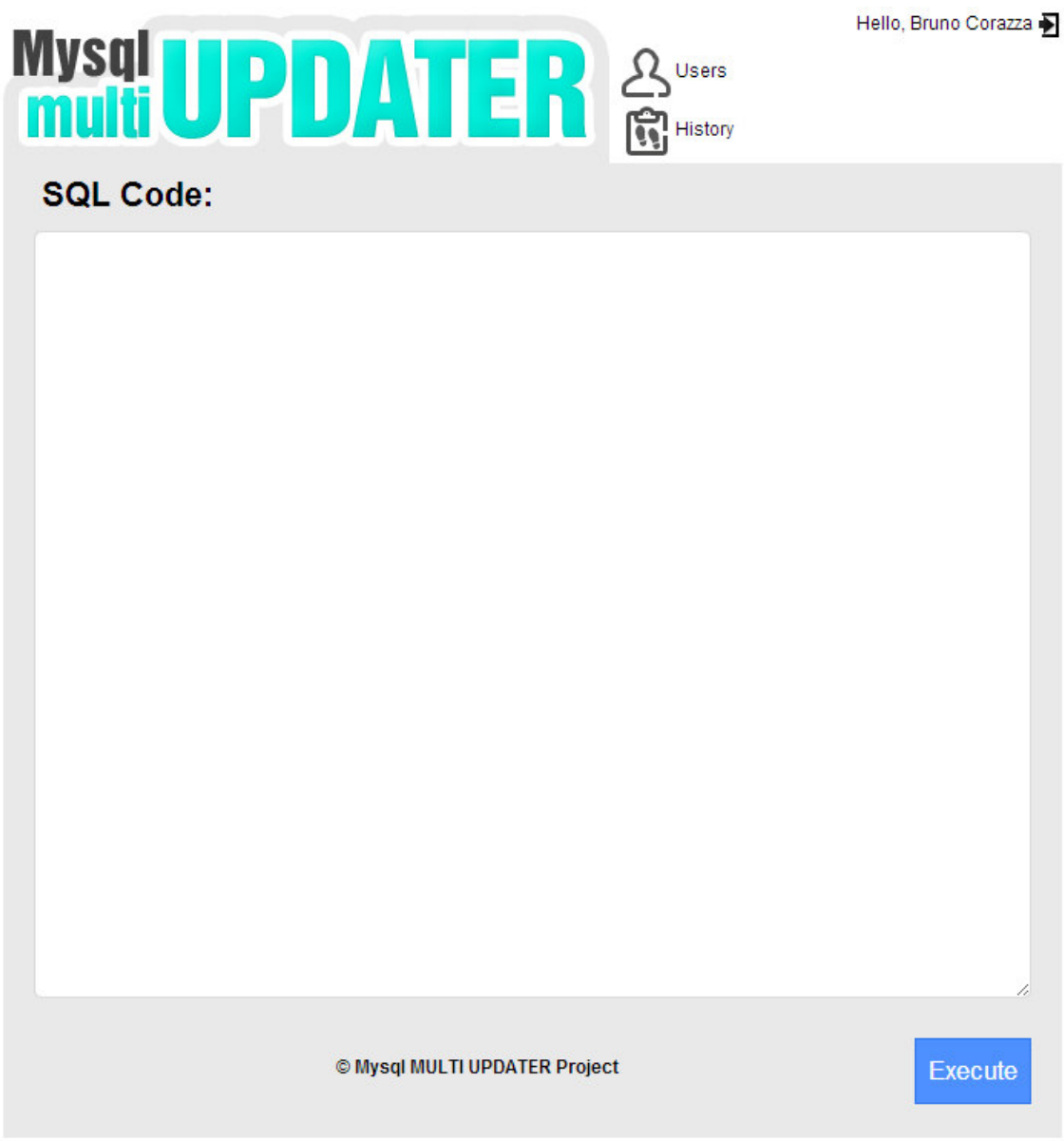

Figura 9 – Complementação da tela principal do sistema

Na Figura 10 está um exemplo de relatório do sistema. Esse relatório exibe todas as execuções de código feitas através do sistema, bem como qual usuário as realizou, quais *databases* foram afetadas, o código que foi executado, a data e hora da execução e o retorno obtido, chamado de *feedback* do sistema.

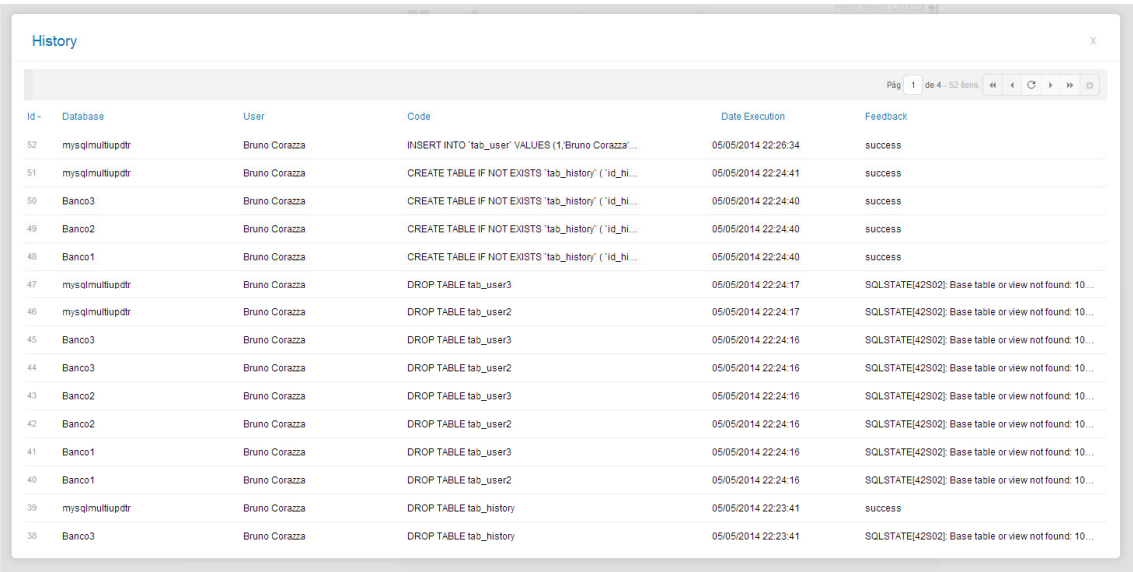

Figura 10 – Relatório de histórico de execução

Na Figura 11 está a imagem do sistema executando sua principal funcionalidade, a execução de códigos MySql em múltiplas bases de dados. No canto esquerdo estão as *databases* selecionadas que terão o código executado. Já no centro da Figura estão os códigos inseridos no campo de execução. Na parte inferior da imagem é observada uma mensagem de sucesso apresentada em cor verde que pode ser substituída quando necessário por uma mensagem vermelha de erro, discriminando e relatando os erros ocorridos para cada *database*.

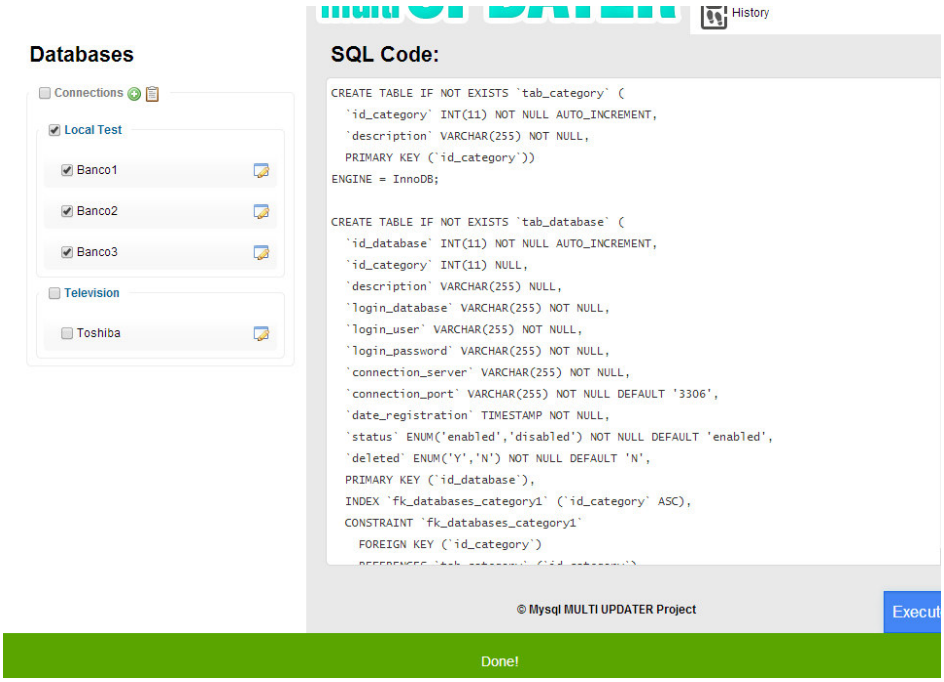

Figura 11 – Execução da funcionalidade principal do sistema

4.4 IMPLEMENTAÇÃO DO SISTEMA

Na Figura 8 é exibida a estrutura de pastas gerada pelo Zend Framework com base nos fundamentos do MVC, sendo assim, fica possível visualizar de uma forma mais didática a interação entre as camadas *model, view* e *controller.* Na Figura 8 é possível perceber a divisão por módulos do sistema dentro da primeira pasta, chamada *module.* A pasta denominada *App,* se refere ao módulo sendo trabalhado. Dentro desta pasta estão as configurações personalizadas deste módulo.

Logo abaixo está a pasta *Controller* na qual encontram-se as classes de controle utilizadas pelo sistema. Na pasta *Model* estão presentes os arquivos com as classes de interação com a base de dados.

No mesmo nível da pasta src está a pasta *view* que possui dentro dela sua divisão por módulos, por sua vez, possui dentro dela a pasta *app* que traz dentro dela uma pasta para cada *Controller* criado. Há ainda a pasta *database* que faz referência ao *DatabaseController*, e em seu interior existe os arquivos no formato *phtml* que fazem parte da camada *view*.

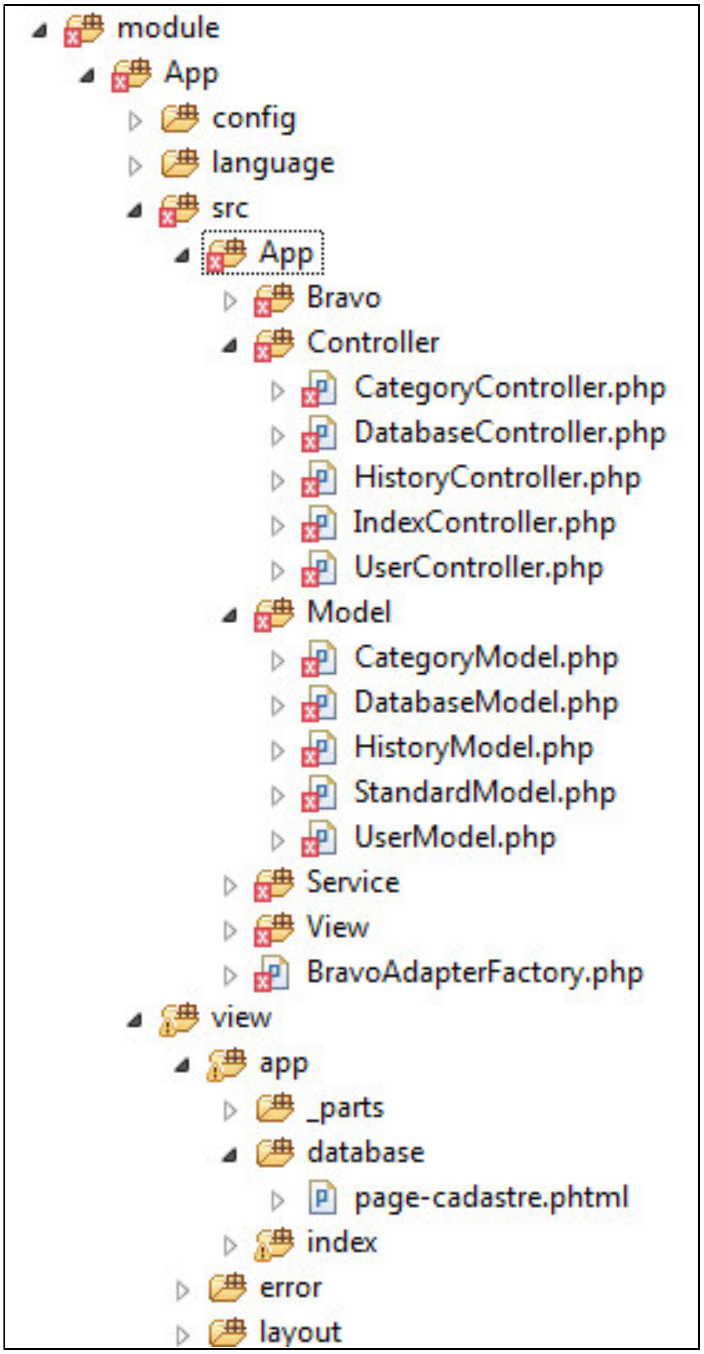

Figura 8 – Estrutura Zend Framework de pastas

Na Listagem 1 é apresentado o código do componente de gerenciamento de formulários criado em jQuery.

```
$('#databasePageCad__form').bravoForm({
        table                   : 'tab_database',
        primaryKey                      : 'id_database',
        populateUrl                     : '/app/database/search',
        registerUrl                     : '/app/database/register',
        idBravoAutocompleteProp : {
                populateUrl  : '/app/database/search',
                populateCols : {
                        id : ['id_database','id_database'],
                        title     : ['description','description'],
                        where   : ['id_database', 'description']
                        }
        },
        customValidation: function(){
                return true;
        },
        continuarCadDisplay: false,
        onAdd: function(){
                database__categorySearch();
        },
        afterPopulate: function(populate){
                database__categorySearch(populate.data[0].id_category);
        },
        onRegisterSuccess: function(){
                index__searchDatabase();
        }
});
```

Listagem 1 – Componente de formulário

Para utilizar esse componente alguns parâmetros devem ser definidos, como a tabela utilizada na base de dados para a gravação deste formulário, a chave primária desta tabela, a *url* que será utilizada para registrar as informações coletadas por ele e também a *url* que será utilizada para popular os campos em caso de edição.

Ainda neste componente observa-se a existência de alguns eventos, como o *onAdd, afterPopulate e onRegisterSuccess.* No primeiro é definido o que será executado ao criar um formulário para adicionar alguma informação à base de dados. Já no segundo é definido o que será executado após popular um formulário, em caso de estar editando alguma informação já gravada no banco de dados. No terceiro é definido o que deve ser executado em caso de sucesso ao registrar alguma informação no banco de dados.

Na Listagem 2 observa-se a estrutura de código que recebe as informações enviadas pelo formulário apresentado na Figura 7. Essa classe trata transações de banco de dados para evitar que registros inconsistentes sejam gravados. Utiliza-se também um controle de múltiplas requisições para que não seja possível a execução de dois códigos simultaneamente na mesma base de dados.

Após tratadas, as informações são enviadas ao *model* correspondente para que sejam

gravadas de acordo com a sua tabela.

Depois de registradas, as informações retornam ao componente de formulário no formato Json via Ajax, sendo possível assim o tratamento das mesmas de forma dinâmica sem a necessidade de recarregar a página.

```
public function registerAction(){
	$databaseModel = new DatabaseModel($this);
	$response = new BravoResponse();
	$adapter = $this->getServiceLocator()->get('db_cliente');
	$adapter->getDriver()->getConnection()->execute("SET SESSION TRANSACTION ISOLATION
LEVEL SERIALIZABLE");
	$adapter->getDriver()->getConnection()->beginTransaction();
	$transBlock = true;
	while($transBlock == true){
		try {
			foreach($_REQUEST['r'] as $requestKey => $request){
				$response->setData($databaseModel->register($request, array('insideTransaction' =>
true)));
				$adapter->getDriver()->getConnection()->commit();
			}
			$transBlock = false;
		} catch ( \Exception $e ) {
			$adapter->getDriver()->getConnection()->rollBack();
			if(strstr($e->getMessage(), '1213 Deadlock'))
				$transBlock = true;
			else{
				$transBlock = false;
				$response->setError(array('message'  => $e->getMessage()));
			}
		}
	}
	echo $response->render($_REQUEST['_responseFormat']);
	exit();
}
```

Listagem 2 – Tratamento de dados do formulário

Na Listagem 3 é exemplificado como é efetuada uma consulta à base de dados de forma dinâmica e diretamente na camada de visualização.

Para que isso seja possível foi utilizado o componente Ajax presente no jQuery. Por meio desse componente é possível definir qual *url* acessará dinamicamente, bem como o tipo de dados que serão enviados e os parâmetros. É possível também definir o que será feito com o retorno oferecido pela página carregada dinamicamente, tanto no caso de sucesso como no caso de erro.

Como exibido na imagem, em caso de sucesso será percorrido o conteúdo retornado pela página e será criada uma *string* concatenando as opções do que virá a ser um campo do tipo *select* da linguagem *html* e será exibido na camada de visualização.

```
function database__categorySearch(id_category){
	$.ajax({
		url: '/app/category/search',
		dataType : 'json',
		data: { r : [{
			_debug: false
			}],
			_responseFormat : 'json'
		},
		error : function(xhr, status, error){
			 flashMessage({
				template : 'error',
				msg : 'Erro interno',
				dadosTec : 'Population of category <br/>URL: /app/category/search <br>' +
xhr.responseText
			 });
		},
		success	: function(data){
			if(data.data[0]){
				var html = '<option value=""></option>';
				$.each(data.data[0], function(){
				html += '<option value="' + this.id_category + '" ' + ((id_category == this.id_category) ?
'selected' : '' ) + '>';
				html += '	' + this.description;
				html += '</option>';
				});
	$('#databasePageCad__id_category').html(html);
			}
		}
	});
}
```

Listagem 3 – Consulta dinâmica em Ajax

Na Listagem 4 é apresentado o código que implementa a funcionalidade principal do sistema. Primeiramente o comando SQL inserido no campo de execução é repartido pelo caractere ‘;’ que por padrão da linguagem define a limitação entre uma *query* e outra. Posteriormente é executada uma consulta para obter as informações das *databases* selecionadas e informadas por parâmetro. Esta consulta é necessária para que posteriormente seja possível conectar-se a cada *database* e executar os códigos repartidos anteriormente.

Após encontrar as bases selecionadas na tela principal, elas são percorridas uma a uma. Nesse processo o algoritmo se conecta à base atual e logo após percorre a cadeia de códigos gerados no primeiro processo descrito nesta listagem. Ao executar cada código é registrado também o histórico relativo, com seus dados e com o retorno de sucesso de execução ou erro quando ocorrido.

```
public function execAction(){
	$session		= new Container('user');
	$response		= new BravoResponse();
	$databaseModel		= new DatabaseModel($this);
	$historyModel	= new HistoryModel($this);

/**
	 * Reparte o comando sql
	 */
	$arrExplodeComandoSql = explode(';', $_POST['sqlCommand']);

	/**
	 * Busca bases de dados que serão utilizadas
	 */
	$arrExplodeIds = explode(',', $_REQUEST['ids']);

	foreach($arrExplodeIds as $id){
		if($condSql)
			$condSql .= ' OR ';
		$condSql .= 'id_database = ' . $id;
	}

	$databaseSearch = $databaseModel->search(array('where' => $condSql));
	if(count($databaseSearch) > 0) foreach($databaseSearch as $database){
		unset($adapter);
		unset($bancoInvalido);
		try{
			/**
			 * Connecta no banco
			 */
			$adapter = new Adapter(array(
					'driver'   => 'Pdo_Mysql',
					'hostname' => $database->connection_server,
					'port'    => $database->connection_port,
					'database' => $database->login_database,
					'username' => $database->login_user,
					'password' => $database->login_password
			));

			/**
			 * Testa conexão
			 */
			$adapter->getCurrentSchema();

		}catch (\Exception $e) {
			$response->setError(utf8_encode('<br />' . $database->description . ': ' . $e->getMessage()));
			/**
			 * Registra erro de conexão
			 */
			$historyModel->register(array(
					'_status'		=> 'add',
					'id_database'	=> $database->id_database,
					'id_user'		=> $session->offsetGet('data')->id_user,
					'feedback'		=> utf8_encode($e->getMessage())
			), array('insideTransaction' => true));
			$bancoInvalido = true;
		}

		if(!$bancoInvalido){
			/**
```

```
                    * Percorre os comandos repartidos pelo delimitador ';'
                    */
                    foreach($arrExplodeComandoSql as $comando){
                        unset($error);
                        if($comando != ''){
                            try{
                                /**
                                 * Executa comando
                                 */
                                $response->setData($adapter->query($comando,
Adapter::QUERY_MODE_EXECUTE));
                            }catch (\Exception $e) {
                                $response->setError(utf8_encode('<br />' . $database->description . ': ' . $e-
>getMessage()));
                                $error = $e->getMessage();
                            }
                            /**
                             * Registra histórico da operação
                             */
                            $historyModel->register(array(
                                    '_status'       => 'add',
                                    'id_database'   => $database->id_database,
                                    'id_user'       => $session->offsetGet('data')->id_user,
                                    'code'              => $comando,
                                    'feedback'          => ($error) ? utf8_encode($error) : 'success'
                            ), array('insideTransaction' => true));
                        }
                    }
                }
            }
            echo $response->render('json');
            exit();
}
```

Listagem 4 – Execução de códigos SQL em múltiplas bases de dados

5 CONCLUSÃO

Este trabalho de conclusão de curso teve como objetivo modelar e implementar um sistema para realizar instruções SQL em bases de dados específicas e previamente cadastradas. O sistema foi desenvolvido com o Zend Framework 2, que utiliza o modelo MVC como estrutura de desenvolvimento e tem como padrão de linguagem de desenvolvimento a PHP.

As dificuldades encontradas no que se referem aos recursos da linguagem e implementação das funcionalidades do sistema foram facilmente transpostas com a leitura de documentações e projetos disponibilizados pelo próprio site do desenvolvedor do *framework*.

Este trabalho foi de grande valia para fixar e esclarecer alguns pontos que não haviam sido explorados em relação às tecnologias utilizadas e para aprofundar ainda mais um conhecimento utilizado diariamente.

Vale destacar também a grande viabilidade da utilização do Zend Framework, pois ele proporciona de maneira fácil e bem documentada a organização fundamental para o andamento do trabalho. Além disso, existem muitos componentes e facilidades proporcionadas que em caso de não utilização deste *framework* teriam a necessidade de serem construídos, tornando assim o desenvolvimento mais longo e complexo.

O desenvolvimento do aplicativo foi concluído alcançando todos os requisitos solicitados. O mesmo vem sendo utilizado como proposto inicialmente para facilitar as tarefas diárias de atualizações e controle de versões de *databases* na empresa onde o acadêmico trabalha. Novas funcionalidades e atualizações poderão ser implantadas de acordo com necessidades que venham a ser detectadas. Uma dessas funcionalidades visando facilitar o uso do sistema é a possibilidade de carregar um arquivo com extensão.sql ou .txt diretamente para o sistema, sem a necessidade de colar o código na interface do aplicativo como é feito na versão atual.

REFERÊNCIAS

ALLEN, Rob; LO, Nick. **Zend framework in action**. MEAP Editon, 2007.

APACHE. **Apache project**. Disponível em: < http://httpd.apache.org/>. Acesso em: 21 jan. 2014.

ASTAH. **Astah community**. Disponível em: < http://astah.net/features/uml-features>. Acesso em: 20 jan. 2014.

BUSCHMANN, Frank; MEUNIER, Regine; ROHNERT, Hans; SOMMERLAD, Peter; STAL, Michael, **Pattern-oriented software architecture**, v. 1: A System of Patterns. John Wiley & Sons, 1996.

CUI, Wei; HUANG, Lin; LIANG, LiJing; LI, Jing. **The research of PHP development framework based on MVC pattern**. In: Fourth International Conference on Computer Sciences and Convergence Information Technology, p. 947-949, 2009.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GARRETT, Jesse James. **Ajax: a new approach to web applications**. 2005. Disponível em <http://www.adaptivepath.com/ideas/essays/archives/000385.php>. Acesso em: 2 jan. 2014.

GERMOGLIO, Guilherme. **Arquitetura de software**. Texas: Rice University, 2010.

GONÇALVES, Rodrigo F.; GAVA, Vagner L.; PESSÔA, Marcelo; S. P. SPINOLA, Mauro. M. **Uma proposta de processo de produção de aplicações web**. Revista Produção, v. 15, n. 3, p. 376-389, Set./Dez. 2005.

HTML 5. **W3C HTML 5.0**. Disponível em: < http://www.w3.org/TR/html5/>. Acesso em: 20 jan. 2014.

JQUERY, **Jquery**. Disponível em: <http://jquery.com>. Acesso em: 15 jan. 2014.

LARMAN, Craig. **Utilizando UML e padrões:** uma introdução a análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

MCCONNELL, Steve. **Code complete**. 2 ed., Microsoft Press, 2004.

MCHEICK, Hamid; QI, Yan. **Dependency of components in MVC distributed architecture**. IEEE Canadian Conference on Electrical and Computer Engineering (IEEE CCECE 2011), p. 91-694, 2011.

MILANI, André. **MySQL – guia do programador**. São Paulo: Novatec, 2007.

MINETTO, Elton Luís. **Frameworks para desenvolvimento em PHP**. São Paulo: Novatec, 2007.

MYSQL FRONT. **MySQL Front**. Disponível em: <http://www.mysqlfront.de/>. Acesso em: 21 jan. 2014.

MYSQL. **MySQL.** Disponível em: <http://www.mysql.com>. Acesso em: 20 jan. 2014.

MYSQL WORKBENCH. **MySQL Workbench 6.0.** Disponível em: <http://www.mysql.com/products/workbench/>. Acesso em: 20 jan 2014.

PHP. **Linguagem PHP**. Disponível em: <http://www.php.net>. Acesso em: 18 jan. 2014.

PRESSMAN. Roger. **Engenharia de software**. São Paulo: MacGraw-Hill, 2008.

SELFA, Diana M.; CARRILLO, Maya; BOONE, Maria del Rocío. **A database and web application based on MVC architecture**. 16th International Conference on Electronics, Communications and Computers (CONIELECOMP '06), p. 48-53, 2006.

SWEAT, Jason. E. **PHP architect's: guide to PHP design patterns**. Marco Tabini and Associates. Canada, 2005.

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

ZEND. **Zend framework.** Disponível em <http://framework.zend.com/manual/en/>. Acesso em: 13 jan. 2014.