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

PATRIKY ERNANY SILVA

UTILIZAÇÃO DO *FRAMEWORK* CAKEPHP PARA DESENVOLVIMENTO DE *WEBSITES* EM PHP

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

PATRIKY ERNANY SILVA

UTILIZAÇÃO DO *FRAMEWORK* CAKEPHP PARA DESENVOLVIMENTO DE WEBSITES EM PHP

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. M. Eng. Juliano Rodrigo Lamb.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Utilização do *framework* CakePHP para desenvolvimento de websites em PHP

Por

Patriky Ernany Silva

Este Trabalho de Diplomação (TD) foi apresentado às 10:20h do dia 01 de Dezembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Juliano Rodrigo Lamb, M.Eng UTFPR – Câmpus Medianeira (Orientador)	Prof. Fernando Schütz, Me UTFPR – Câmpus Medianeira (Convidado)
Prof. Márcio Angelo Matté UTFPR – Câmpus Medianeira (Convidado)	Prof. Juliano Rodrigo Lamb, M.Eng UTFPR – Câmpus Medianeira (Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado fôlego de vida e força para permanecer sempre firme nas minhas decisões, a toda minha família que sempre me apoiou e incentivou durante os momentos mais difíceis e deram conselhos que me fizeram superar obstáculos e seguir em frente com o trabalho e a minha futura esposa Marina Horn com a qual aprendi que a vida não é feita somente de felicidade, mais também de tristezas, mais que com sabedoria e uma companheira igual a ela, pode-se transformar os dias mais nublados em lindos dias de sol.

Agradeço também aos professores que compartilharão seus conhecimentos e fizeram de mim parte do que sou hoje. Em especial ao professor Juliano Rodrigo Lamb, o qual me orientou durante todo este trabalho de diplomação, pois sem a ajuda e o incentivo dele hoje não estaria concluindo mais esta etapa na minha vida.

“Ninguém ignora tudo. Ninguém sabe tudo. Todos nós sabemos alguma coisa. Todos nós ignoramos alguma coisa. Por isso aprendemos sempre.”

(Paulo Freire)

RESUMO

SILVA, E. Patriky. Utilização do *framework CakePHP* para desenvolvimento de *Web*sites em *PHP*. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

O desenvolvimento de *software* sem qualquer tipo de auxilio, como um *framework*, pode ser cansativo e tedioso. Este trabalho tem como foco demonstrar, por meio de um estudo experimental como desenvolver uma aplicação *Web* utilizando-se do *framework CakePHP*. *Frameworks* promovem agilidade e padronização ao desenvolvimento. O estudo experimental consistirá no desenvolvimento de um *blog*, onde são cadastradas as informações relativas a usuário e os respectivos *posts*. Verificou-se que o *CakePHP* proporciona agilidade no desenvolvimento, diminuindo o tempo gasto no desenvolvimento de sistemas.

Palavras-chave: *Internet*. *PHP*. MVC. *Web*.

ABSTRACT

SILVA, E. Patriky. Using the *CakePHP framework* for developing *Web*sites in *PHP*. 2011. Completion of course work (Technology Analysis and Systems Development), Federal Technological University of Parana. Mediatrix 2011.

The development of software without any help, as a framework, can be tiring and tedious. This work focuses on demonstrating, through an experimental study to develop a web application using the CakePHP framework. Frameworks promote agility and standardization development. The experimental study was to develop a blog, where information concerning registered user and their posts. It was found that CakePHP provides flexibility in the development, reducing the time spent on system development.

Keywords: Internet. PHP. MVC. Web.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

CMS	-	*Content Management System*
HTML	-	*HyperText Markup Language*
MER	-	Modelo Entidade Relacionamento
MVC	-	*Model – View – Controller*
PHP	-	*Php Hypertext Preprocessor*
POO	-	Programação Orientada Objetos
REST	-	*Representational State Transfer*
SGBD	-	Sistema Gerenciador de Banco de Dados
SQL	-	*Structured Query Language*

SUMÁRIO

1 INTRODUÇÃO

PHP é uma linguagem de *scripting* com HTML (BRITO, 2007). Surgiu por volta de 1994 criada por Rasmus Lerdorf, e grande parte da sintaxe é "emprestada" do *C*, *Java* e *Perl* com um pequeno conjunto de funcionalidades específicas. O objetivo da linguagem é permitir aos programadores *Web* escrever rapidamente páginas geradas dinamicamente.

> "Originalmente *PHP* significava *Personal Home Page*, porém, à medida que foi adquirindo maior funcionalidade e representando um papel mais sério na área do desenvolvimento *Web*, o nome precisou ser melhorado para indicar corretamente sua aplicação, logo, passou a ser chamado de *Php Hypertext Preprocessor* (Preprocessador de Hipertexto *Php*)". (BROUWER, 2002)

Esta é uma definição adequada de *PHP*. No entanto, contêm alguns termos a que talvez não se esteja habituado. Outra forma de pensar em *PHP* é que é uma eficaz linguagem de *scripting* que atua apenas no servidor e que o cliente não vê.

"Framework é uma estrutura, uma fundação para você criar a sua aplicação." (BELEM, 2009). Em outras palavras, um *framework* permite o desenvolvimento rápido de aplicações, o que economiza tempo e ajuda a desenvolver aplicações mais sólidas e seguras, além de reduzir a quantidade de código repetido.

A ideia de trabalho por trás de um *framework* no *PHP* está ligada ao modelo MVC (*Model View Controller*). O padrão de projeto MVC isola a lógica de negócio (como a aplicação funciona) da camada de exibição (a parte visual). O *Model* é o modelo da aplicação, onde são definidos propriedades e atributos dos personagens, o *View* é a camada de visualização, onde a aplicação apresenta o que foi obtido através do controle, em outras palavras, é a visualização do usuário final. A parte visual não deve ter nenhuma lógica de código, apenas a exibição dos dados, e no *Controller* é onde serão processadas todas as requisições feitas através da interface (*View*). O *Controller* acessa o *Model* a fim de obter determinadas informações, toda a lógica da aplicação é controlada pelos *Controllers* e nos *Models* são feitas validações e associações.

Existem vários *frameworks* para o *PHP* como o *CakePHP*, *Zend Framework* e o *Symfony*, mas aquele que vem se destacando, pela simplicidade, facilidade de uso e por ser *free open-source* (código aberto gratuito), no momento, é o *CakePHP* que,

por ter sido desenvolvido usando as bases e modelos do *Ruby on Rails*[1], é bastante focado no desenvolvimento ágil.

O *CakePHP* utiliza os padrões de projetos *ActiveRecord*, *Association Data Mapping*, *Front Controller* e claro, o MVC. Este, por conseguinte, possui classes e objetos adicionais que tem como objetivo proporcionar extensibilidade e reuso, para que possam adicionar funcionalidades à base MVC das aplicações. Para a extensão de controladores existe a classe *Component*, para a de visão existe a classe *Helper* e para a extensão de modelo existe a classe *Behavior*.

Este trabalho tem como objetivo a realização de um estudo detalhado sobre o *framework CakePHP* e alguns componentes que auxiliam no desenvolvimento utilizando o mesmo. Espera-se ainda identificar as vantagens do *framework* que encaixe melhor no desenvolvimento de uma aplicação *Web*, aplicando em um estudo experimental voltado ao desenvolvimento de um *blog*.

1.1 OBJETIVO GERAL

Desenvolver uma aplicação como estudo experimental com a funcionalidade de um *blog*, utilizando o *framework CakePHP*.

1.2 OBJETIVOS ESPECÍFICOS

Os principais objetivos específicos são:

- Desenvolver um referencial teórico sobre *frameworks*.
- Demonstrar alguns componentes que auxiliam no desenvolvimento das aplicações utilizando o *CakePHP*.
- Apresentar as vantagens e desvantagens do uso de *framework*.
- Criar a análise e o projeto do estudo experimental a ser desenvolvido.
- Desenvolver a aplicação utilizando o *CakePHP*, como estudo experimental.
- Testar a aplicação e as funcionalidades.

[1] *Ruby on rails* é um *framework* de desenvolvimento *Web* (gratuito e de código aberto) otimizado para a produtividade sustentável e a diversão do programador. Ele permite que o desenvolvedor escreva código de forma elegante, favorecendo a convenção ao invés da configuração. (MONTEIRO, 2011)

1.3 JUSTIFICATIVA

A utilização dos *frameworks PHP* deve-se a motivos, tais como:

- Agilizar o processo de desenvolvimento,
- Favorece o reuso de códigos,
- Estabilidade de funcionamento da aplicação.

Favorecendo o reuso dos códigos implementados em vários projetos, economiza tempo e diminui trabalho, pois o *framework* traz uma série de módulos pré-configurados para realizar tarefas que o desenvolvedor encontra tais como, conexão com o banco de dados e proteção contra ataques. (BELEM, 2009)

“A simplicidade é o que mais possibilita erros e falhas pelos principiantes, pois, nem todo código que funciona, necessariamente está correto e bem desenvolvido.” (BELEM, 2009)

O desenvolvimento de aplicações *Web* com o auxílio de um *framework* tornará mais simples a produção de projetos. Assim sendo, todo o código fica separado, pois este possui o padrão de projetos MVC. Caso opte-se pela não utilização do *framework* todo o código ficará junto e misturado, pois não oferecerá a funcionalidade que ofereceria com a utilização deste.

A escolha do *CakePHP* foi face a robustez do *framework*, por ser *free* (código aberto livre), ter grande documentação em português e por ser menos complicado e com custo reduzido de tempo para aprender. Deve-se lembrar que (como será visto mais adiante) não existe “o melhor” *framework*, existe o *framework* que mais se adapta ao que o desenvolvedor necessita.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho possui seis capítulos, sendo que o primeiro trata sobre a contextualização do tema abordado, define os objetivos e a justificativa do projeto.

O segundo busca fornecer um conceito teórico sobre as tecnologias e ferramentas que foram utilizadas do desenvolvimento do mesmo.

O capítulo três contempla o estudo experimental, onde foi mostrado na prática a utilização do *framework CakePHP* para o desenvolvimento de *Websites* em *PHP*.

O capítulo quatro demonstra os resultados obtidos com o desenvolvimento do *site* utilizando o *framework CakePHP*.

As conclusões finais sobre o projeto e as sugestões para trabalhos futuros são abordadas no capítulo cinco.

Por fim, estão as referências bibliográficas que fizeram parte do embasamento teórico do projeto.

2 REVISÃO DE LITERATURA

Este capítulo tem como objetivo fornecer um referencial teórico sobre as linguagens para a *Internet*, explanar o que são os *frameworks* e pontuar algumas considerações sobre os *frameworks Zend*, *Symfony* e *CakePHP*.

2.1 LINGUAGENS PARA A INTERNET

Para o desenvolvimento de sistemas *Web* existem várias linguagens de programação tais como *Java*, *PHP*, *C#, Java Script*, *Ruby*, entre outras, como é possível visualizar na FIGURA 1.

FIGURA 1 - Linguagens para *Internet*

O *TIOBE Programming Community Index* é um indicador da popularidade de linguagens de programação e o índice é atualizado uma vez por mês. As classificações são baseadas no número de engenheiros qualificados em todo o mundo, cursos e fornecedores terceirizados. Os motores de busca populares como *Google*, *Bing*, *Yahoo*, *Wikipédia*, *YouTube* e *Baidu* são usados para calcular as classificações. (Tiobe *Software,* 2011)

Segundo *TIOBE Software* (2011), a linguagem *PHP* encontra-se na quarta colocação das linguagens mais utilizadas para desenvolvimento de aplicações *Web*. A FIGURA 2 apresenta os resultados da pesquisa referentes ao mês de outubro de 2011, em qual colocação cada linguagem se encontra e quantas posições a mesma subiu ou desceu em relação a 2010.

Position Oct 2011	Position Oct 2010	Delta in Position	Programming Language	Ratings Oct 2011	Delta Oct 2010	Status
1	1	=	Java	17.913%	-0.25%	A
2	2	=	C	17.707%	+0.53%	A
3	3	=	C++	9.072%	-0.73%	A
4	4	=	PHP	6.818%	-1.51%	A
5	6	↑	C#	6.723%	+1.76%	A
6	8	↑↑	Objective-C	6.245%	+2.54%	A
7	5	↓↓	(Visual) Basic	4.549%	-1.10%	A
8	7	↓	Python	3.944%	-0.92%	A
9	9	=	Perl	2.432%	+0.12%	A
10	11	↑	JavaScript	2.191%	+0.53%	A
11	10	↓	Ruby	1.526%	-0.41%	A
12	12	=	Delphi/Object Pascal	1.104%	-0.45%	A
13	13	=	Lisp	1.031%	-0.05%	A
14	14	=	Transact-SQL	0.909%	+0.09%	A
15	23	↑↑↑↑↑↑↑↑	PL/SQL	0.903%	+0.30%	A-
16	24	↑↑↑↑↑↑↑↑	Lua	0.802%	+0.25%	A
17	16	↓	RPG (OS/400)	0.757%	+0.05%	A--
18	15	↓↓↓	Pascal	0.721%	-0.05%	A
19	-	=	Assembly*	0.622%	-	B
20	17	↓↓↓	Ada	0.609%	-0.09%	B

FIGURA 2 - Posições das Linguagens de Programação
Fonte: *TIOBE Software* (2011)

Na FIGURA 3 é possível verificar as tendências em longo prazo das 10 primeiras linguagens, desde o ano 2002 até o ano de 2011.

Com o diagrama da FIGURA 3, pode-se notar como a linguagem *PHP* vem crescendo, pois passou de aproximadamente 2.5% de utilização em 2002 para aproximadamente 7.5% em 2011.

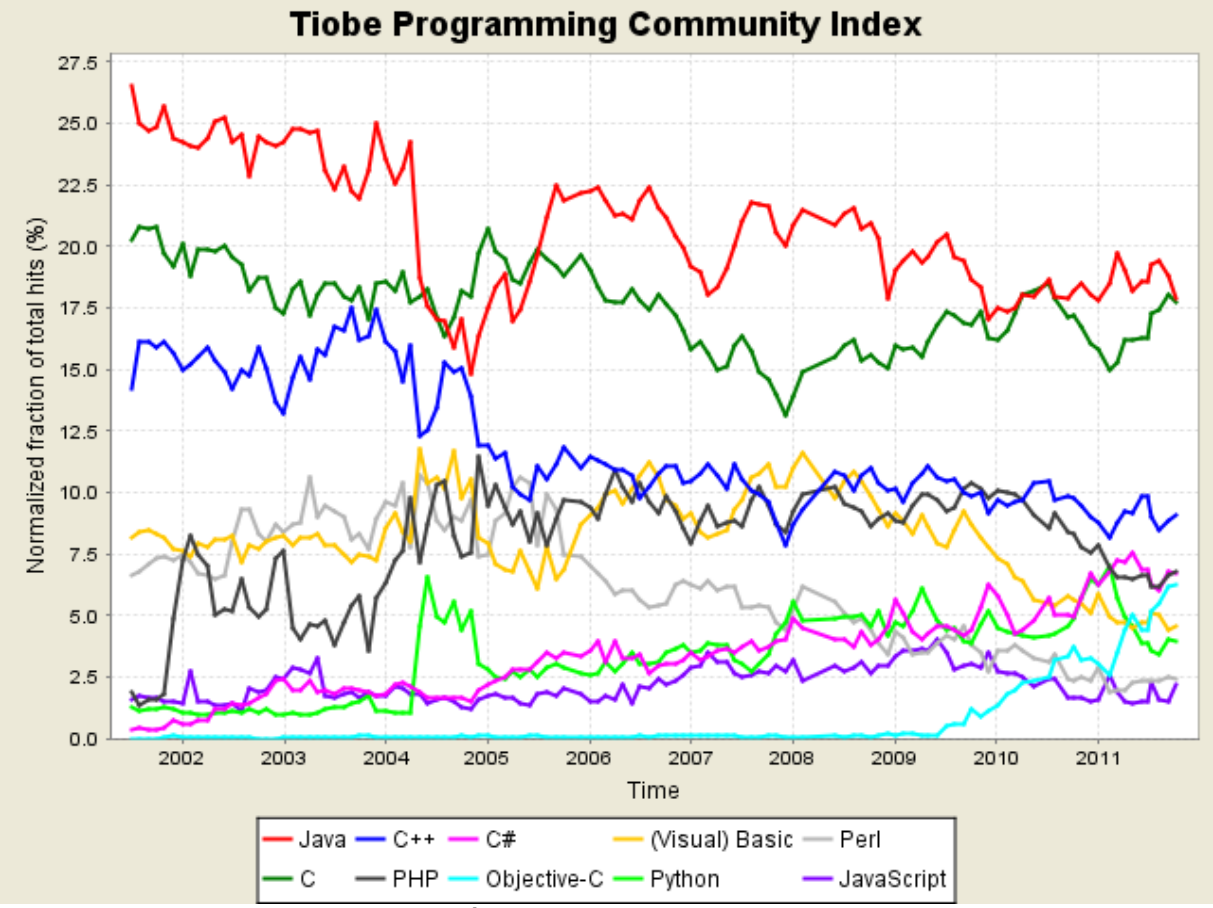

FIGURA 3 - Tendência a longo prazo das 10 linguagens
Fonte: *TIOBE Software* (2011)

2.2 PHP

A utilização de aplicativos *Web* tem se tornado indispensável no dia-a-dia das pessoas. Prefere-se acessar uma aplicação *on-line* que permite realizar as mesmas ou até mais tarefas que um aplicativo *desktop* fornece, sem ter o transtorno de precisar instalar o programa no próprio computador (LINGHAM, 2007).

Tanto as aplicações *desktop* como as aplicações *Web* precisam cuidar de algo que é essencial: a segurança. Uma das principais vantagens desta regra, que é poder ser acessada de qualquer lugar por meio da *Internet*, torna-se um dos pontos fracos.

> Aplicações *Web* são formadas por inúmeros recursos, como páginas dinâmicas, estáticas, imagens, *downloads*, *uploads*, relatórios etc. Grande parte desses recursos não pode estar disponível para qualquer pessoa que tentar acessá-lo e para isso as aplicações *Web* devem controlar o acesso dos usuários a estes recursos. (FRANZINI, 2009)

Para que essas aplicações estejam seguras existem alguns princípios que devem ser considerados, segundo BROWN et al (2005):

- **Autenticação**: O processo de provar a identidade para uma aplicação;
- **O controle de acesso de recursos**: Os meios pelos quais as interações com os recursos são limitados para os usuários, papéis (*roles*), ou programas com o objetivo de impor a integridade, confidencialidade, ou restrições de disponibilidade;
- **A integridade dos dados**: Os meios de garantir que um terceiro não alterou informações enquanto a mesma estava em trânsito;
- **Confidencialidade e privacidade dos dados**: Os meios utilizados para assegurar que a informação feita é disponível somente para usuários que estão autorizados a acessá-lo.

Desde o surgimento da Internet, ela se mostrou um mecanismo de anonimato, onde as pessoas poderiam se passar por outras ou até mesmo não se passar por nenhuma. Autenticação (FIGURA 4) é o processo onde uma pessoa garante quem ela realmente é com o intuito de obter alguma informação ou executar alguma ação.

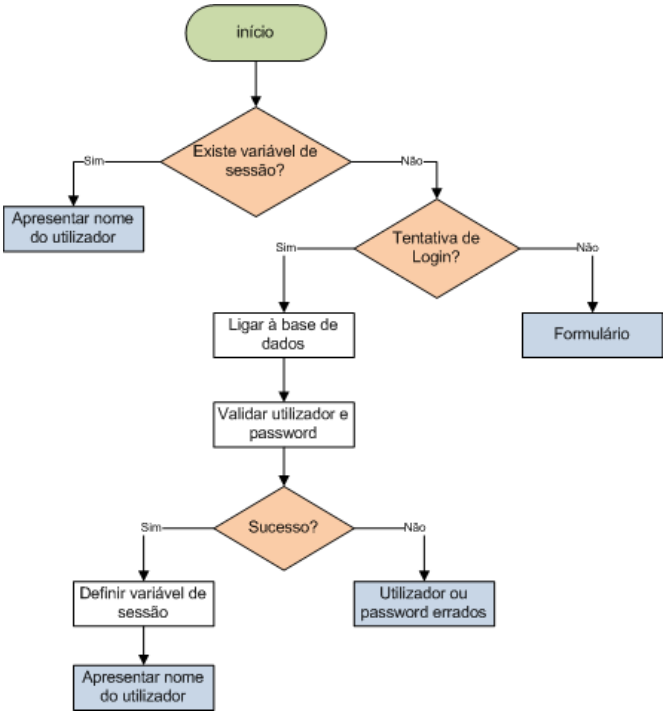

FIGURA 4 - Rotina para autenticação em PHP
Fonte: VELOSO (2010)

A autenticação geralmente é feita em uma tela de *acesso (login)* onde a pessoa informa as credencias (geralmente nome de usuário e senha) e caso forem corretas, ela tem a autenticação confirmada pela aplicação.

Autorização consiste na etapa de dar permissão à pessoa autenticada para acessar algum recurso ou realizar alguma ação. Em sistemas onde existem vários tipos de usuários, um administrador de sistemas definirá quais usuários terão acesso e quais privilégios de execução esses terão.

Segundo FRANZINI (2009), o mecanismo de autorização é composto de dois momentos:

- Ato de atribuir permissões ao usuário do sistema no momento do cadastro, ou posterior.
- Ato de checar as permissões que foram atribuídas ao usuário no momento do acesso.

A FIGURA 4 apresenta o esquema de autenticação, onde é possível verificar-se que para obter sucesso, deve-se informar *login* e *password* e os mesmos serem idênticos aos armazenados na base de dados, caso contrário à autenticação falha.

A FIGURA 5 demonstra o esquema do funcionamento de uma página *Web* em *PHP*.

FIGURA 5 - Esquema do funcionamento de uma página *Web* em *PHP*
Fonte: *PHP*BRASIL (2006).

2.3 FRAMEWORKS

Juntamente com a busca por soluções ágeis e produtivas é que começaram a surgir às primeiras ferramentas com estas características, também conhecidas como *frameworks*. Segundo ELTON (2007), estes além de facilitarem a vida dos programadores, também trazem um ótimo ganho de rendimento no desenvolvimento de aplicações.

2.3.1 DEFINIÇÃO

Em sua essência, *framework* é o conjunto de organização de código, junto com bibliotecas de desenvolvimento, apresentadas de um jeito que facilita o desenvolver de um projeto. (BROWN, 2008)

De acordo com FAYAD *et al.* (2006), *"*um *framework* consiste em um conjunto de classes que se relacionam e representam uma solução incompleta. Um *framework* é o esqueleto de uma aplicação que pode ser customizado por um desenvolvedor da aplicação*"*.

Ainda de acordo com FAYAD e*t al.* (2006), "um *framework* orientado a objetos é um projeto reutilizável de *software* definido por um conjunto de classes abstratas e pela maneira pela qual as instâncias dessas classes colaboram entre si"*.*

Segundo ZEMEL (2009), existem muitas vantagens quanto a utilização de *frameworks* que são:

- **Utilidade**: Um dos objetivos principais dos *frameworks* é auxiliar no desenvolvimento de aplicações e *softwares*.
- **Segurança**: Os *frameworks* mais eficazes são projetados de modo a garantir a segurança de quem programa .
- **Extensibilidade**: Os *frameworks* permitem extender funcionalidades nativas.
- **Economia de tempo**: As funções que manipulam as imagens são de um caminho bem mais exigível, precisando de mais tempo na realização em suma, com o uso de um *framework* pode-se realizar-se em pouquissimo tempo.
- **Ajuda fácil**: Os desenvolvedores de *frameworks* geralmente disponibilizam material de qualidade nos *Websites*, com uma vasta documentação a

respeito. Além disso, a maioria dos *frameworks* têm uma comunidade de desenvolvedores dispostos a se ajudarem entre si.

As desvantagens de se utilizar um *framework* residem na questão de terem uma metodologia própria de desenvolvimento. Os *frameworks* tem uma curva de aprendizagem que pode ser bem elevadas e raramente tem *interfaces* prontas, ou seja, essencialmente se começa do zero qualquer projeto. (BROWN, 2008)

Não existe um *framework* melhor ou pior no mercado. Existem sim, diferentes abordagens que podem se enquadrar melhor em um contexto especifico. (BRAGANÇA, 2010)

Segundo BROWN (2008) *"*você tem um projeto específico, que tem necessidades específicas, o projeto tende a ser um projeto pequeno ou grande, que vai receber manutenção diária e são esperadas melhorias constantes, ou terá um projeto que será entregue em fases, com funcionalidades adicionadas ao longo do tempo, essa é sem dúvida a melhor opção*".*

No QUADRO 1 pode-se observar quais são os 10 *frameworks* mais utilizados, observa-se que o *CakePHP* vem na terceira posição.

QUADRO 1 - Top 10 *Frameworks* para *PHP*

Top 10 Hot PHP Frameworks	
1 Yii	(126 votes)
2 CodeIgniter	(98 votes)
3 CakePHP	(76 votes)
4 Zend	(69 votes)
5 Symfony	(51 votes)
6 PHPDevShell	(41 votes)
7 Prado	(28 votes)
8 Akelos	(24 votes)
9 ZooP	(5 votes)
10 QPHP	(5 votes)

Fonte: Adaptado de *PHPFramework* (2011)

Pode-se constatar e discernir o que é *framework*, assim sendo, verifica-se o que este não é, de acordo com FARIA (2009) "ele não é um CMS[2] (*Content Management System*), não é incompatível com *AJAX*, [...] não é fácil de aprender e ele não faz mágica sozinho".

[2] CMS (Content Management System), ou Sistema de Gerenciamento de Conteúdos. É um sistema que permite gerenciar conteúdos. CMS é uma ferramenta que permite a um editor criar, classificar e publicar qualquer tipo de informação em uma página *Web* (ALVAREZ, 2008).

O *framework* não é um CMS pois ele não gerencia o conteúdo do *site*, porém, nada impede de criar fazendo o uso dele.

Existem *frameworks*, como o *CakePHP* por exemplo, que fornece alguns *helpers* para trabalhar de forma mais simples com *AJAX*, mais se o *framework* não oferecer esses *helpers*, é possivel escrever os arquivos *JavaScripts* e utilizá-los.

Deve-se ter o cuidado para não utilizar um *framework Web* para desenvolver sistema *desktop* pois não haverá sucesso no mesmo. É necessário que se utilize uma arquitetura bem definida, sendo assim será necessário organiza-se e manter o projeto organizado.

Para conseguir utilizar o *framework*, antes deve-se ter um pouco de conhecimento de POO (Programação Orientada Objetos) além da arquitetura MVC (*Model-View-Controller*) que o mesmo utiliza. No caso da utilização do *framework CakePHP* sem conhecimento da programação OO, o programador não verificará a vantagem, por exemplo, dos *helpers* e components, seja na criação ou utilização. A arquitetura oferecida pelo *framework* auxiliará a entender onde colocar o que, ou seja, quem é o *model*, *view* e *controller* e o que colocar dentro de cada um.

O *framework* ajudará a manter o código organizado e fornecendo algumas facilidades genéricas, permitindo que o desenvolvedor projete as aplicações cada vez mais rápido.

2.3.2 *FRAMEWORK* SYMFONY

O Symfony (FIGURA 6) é um dos *frameworks* mais eficientes. Todas as tarefas são modulares e o *framework* utiliza diferentes camadas para manejar os dados. É bastante utilizado para projetos com grandes funcionalidades. (JAQUES, 2009)

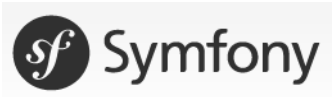

FIGURA 6 - Logo *Symfony*
Fonte: *Symfony* (2011)

Symfony é um *framework* que fornece uma estrutura para desenvolvimento de aplicações *Web* complexas. Ele usa a maioria das melhores práticas de

desenvolvimento *Web* e integra grandes bibliotecas de terceiros. *Symfony* tem uma complexidade maior de se utilizar que o *CakePHP*, exigindo a utilização de linha de comando para executar comandos de configuração e para criar aplicativos. É repleto de características eficientes, tem uma boa documentação, e está em constante atualização, graças à ativa e útil comunidade. (BRAGANÇA, 2010).

2.3.3 *FRAMEWORK* ZEND

Zend Framework é um *framework* extremamente eficaz, porém com uma curva de aprendizado um pouco maior. Desenvolvido pela *Zend Technologies*, *Zend Framework* está licenciado sob a licença *New BSD license*. Estendendo as funcionalidades do *PHP*, *Zend Framework* é baseado na simplicidade, orientado a objeto, boas práticas de desenvolvimento, licenciamento corporativo amigável e uma ágil base de códigos testados rigorosamente. *Zend Framework* é focado na construção segura, confiável e de modernas aplicações *Web* 2.0. (BRAGANÇA, 2010)

*Zend Framework (*FIGURA 7) é um *framework free open-source* que provê funcionalidades para o desenvolvimento de aplicações e serviços *Web* utilizando-se a linguagem de programação *PHP*. Propõe-se a oferecer uma biblioteca de recursos de grande poder, fornecendo soluções modernas, robustas e seguras para o desenvolvedor. Inclui, também, diversos componentes de integração com *APIs* de serviços, como *Flickr*, *Amazon*, *Delicious*, entre outros. (NET, 2008).

FIGURA 7 - Logo *Zend*
Fonte: *Zend Framework* (2011)

2.3.4 *FRAMEWORK* CAKEPHP

O *CakePHP* (FIGURA 8) é reconhecido pela simplicidade e pela alta produtividade que ele proporciona. Baseado no *framework Rails* para *Ruby* ele

funciona através de um gerador de código que produz todo o *CRUD* deixando para o programador desenvolver a regra de negócio de cada sistema (SCHIRM, 2010).

Assim como no *Rails*, o CakePHP utiliza como roteamento padrão o *RESTful* (*Representational State Transfer*). O termo originou-se no ano de 2000, em uma tese de doutorado sobre a *Web*, realizado pelo norte-americano Roy Fielding. REST é uma arquitetura cliente/servidor sem estado, no qual os recursos são identificados por suas URL's e são manipulados através de suas representações. (GUNDERLOY, 2008)

Dentro da arquitetura REST, são utilizados os métodos *GET, POST, PUT* e *DELETE* do HTML, que servem para manipular os recursos. As URL's seguem um padrão que às divide em 4 partes: www.dominio.com / controlador / ação / parâmetro. (INDIA, 2010).

FIGURA 8 - Logo *CakePHP*
Fonte: FOUNDATION (2011)

2.3.4.1 Definição

A melhor definição para o *CakePHP* diz que ele permite que os usuários *PHP* de todos os níveis possam desenvolver rapidamente aplicações *Web* robustas.

> *CakePHP* é um *framework* escrito em *PHP* que tem como principais objetivos oferecer uma estrutura que possibilite aos programadores de *PHP* de todos os níveis, desenvolverem aplicações robustas rapidamente, sem perder flexibilidade. O *CakePHP* é baseado no *framework Ruby on Rails* e utiliza *design patterns* conhecidos, como *ActiveRecord*, *Association Data Mapping*, *Front Controller* e *MVC* (*Model-View-Controller*) e recursos exclusivos para uso de *AJAX*. (NET, 2008)

2.3.4.2 Vantagens

Vantagens de utilizar o *CakePHP*, segundo CARLOS SILVA (2007):

- Comunidade ativa, amigável;

- Flexibilidade de Licenciar;
- Compatibilidade com *PHP*4 e *PHP*5;
- *Scaffolding* da aplicação;
- *CRUD* integrado para a interação da base de dados e perguntas simplificadas;
- *Template* rápido e flexível (sintaxe de *PHP*);
- Listas flexíveis do controle de acesso;
- Utiliza *RESTful*;
- Existem outras vantagens como Segurança, Ajax, *Session* e outras.

Segundo CALAÇA (2010), os motivos de utilizar o *CakePHP* são o aprendizado fácil e rápido, por ele ser um *software* livre, tem grande comunidade brasileira e muita documentação em português e possui vários componentes já prontos.

2.3.4.3 Desvantagens

Segundo CALAÇA (2010), as desvantagens de utilizar-se o *CakePHP* são:

- A compatibilidade com *PHP*4 pode ser um problema.
- Várias convenções pré-definidas.
- Acoplamento relativamente alto.

Segundo LOPES (2010) utilizar-se o *CakePHP* requer uma quantidade maior de tempo para analisar e modelar o sistema, requer pessoal especializado e não é aconselhável para pequenas aplicações.

2.3.4.4 Comparação entre Zend e CakePHP

Existe fatores em que o CakePHP é mais produtivo de utilizar-se em relação ao Zend, porém em outro fator pode-se existir o inverso e o Zend mostrar-se mais produtivo.

Conforme CHAD (2008), pode-se observar na TABELA 1, onde é apresentado características de ambos os *frameworks* e apontado uma faixa de escala utilizada

pelo mesmo. A ordem da escala é: *poor* (ruim), *fair* (regular), *good* (bom), *excellent* (excelente), sendo o *poor* o nível mais baixo e *excellent* o mais alto.

TABELA 1 - Comparação de recursos do CakePHP e Zend *Framework*

Feature	CakePHP	Zend *Framework*
License	MIT	BSD
Compatibility	4 and 5	5.1.4 or later
Documentation	Good	Excelente
Community	Google, Group, IRC, Articles	Wiki, Lists, Chat
Tutorial/Sample Availability	Excelente	Fair
MVC	Strict	Flexible
Conventions	Strict	Optional
Configuration	PHP file	PHP array, XML, or INI files
Database Abstraction	PHP, PEAR, ADODB	PHP, PDO
Security	ACL – based	ACL – based
Data Handling	Good	Excellent
Caching	Good	Excellent
Sessions	Excellent	Excellent
Logging/Debugging	Good	Excellent
Templating	PHP – based	PHP – based
Helpers	Good	Excellent
JavaScript / Ajax	Good	Fair
Web Services	Good	Excellent
Localization	Good	Excellent
Unit Testing	Yes	Yes

Fonte: CHAD (2008)

3 MATERIAS E MÉTODOS

A aplicação *Web* desenvolvida para este trabalho baseia-se em um micro *blog*, onde determinada pessoa pode se cadastrar como autor, realiza novos *Posts* incluindo o respectivo nome como autor e comentar em *Posts* publicados por outros autores ou em *Posts* publicados por si mesmo.

3.1 ANÁLISE E PROJETO DO SISTEMA

As funcionalidades do sistema podem ser visualizadas na FIGURA 9, através do diagrama de casos de uso.

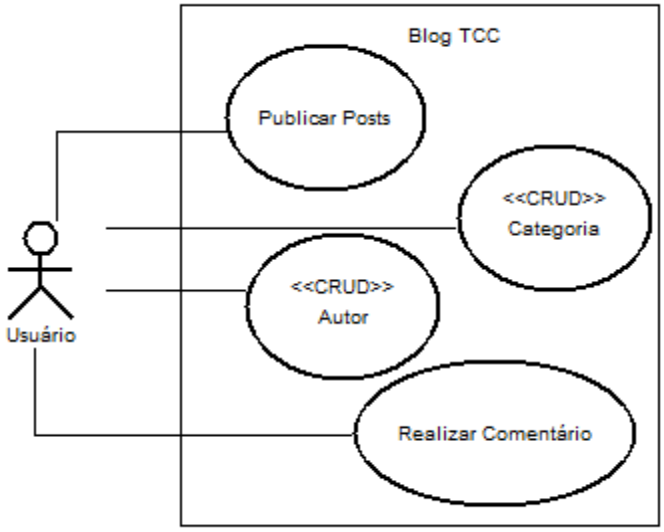

FIGURA 9 - Casos de Uso

3.1.1 Organização dos Requisitos

Conforme WAZLAWICK (2004), requisitos podem ser classificados em casos de uso, conceitos e consultas. Casos de uso são as funcionalidades do sistema, conceitos são passíveis de operações CRUD e consultas são os relatórios do sistema. Essa divisão permite ao analista/projetista identificar os pontos críticos do sistema e consequentemente projetar ou dividir tarefas adequadamente.

3.1.1.1 Casos de Uso

A listagem dos casos de uso do sistema é exibida através da TABELA 2, onde é possível verificar as funcionalidades do sistema, bem como os atores envolvidos.

TABELA 2 - Listagem de casos de uso

Nome	Atores	Descrição
Publicar *Posts*	Usuário	O sistema deve permitir ao usuário cadastrar novo *Post*. O usuário informa o Autor e a Categoria, bem como o Título e o Conteúdo do *post* e manda gravar.
Realizar Comentário	Usuário	O sistema deve permitir ao usuário visualizar o *post* completo. Informa seu nome, *e-mail* e o conteúdo do comentário e manda gravar.

3.1.1.2 Conceitos

Os conceitos remetem aos dados armazenados pelo sistema, e suas restrições. As operações do sistema, que possibilitam (I) Inclusão, (A) Alteração, (E) Exclusão e (C) Consulta, podem ser visualizadas através da TABELA 3. Um caso de uso pode ser um conceito, mas a recíproca não é verdadeira.

TABELA 3 - Listagem de conceitos e operações de manutenção

Conceito	I	A	E	C	Observação
Posts	x			x	Para inserção é necessário selecionar um autor e uma categoria.
Categoria	x	x	x	x	
Autor	x	x	x	x	
Comentário	x			x	Para inserção ou consulta dos comentários deve-se visualizar o *post* completo.

3.1.1.3 Consultas

As consultas que serão efetuadas pelo sistema podem ser visualizadas através da TABELA 4. Basicamente, o sistema deve de emitir relatórios para todos os conceitos que foram armazenados no sistema.

TABELA 4 - Listagem de Consultas

Nome	Observação
Posts Cadastrados.	Exibe a listagem completa de *Posts* cadastrados
Autores Cadastrados.	Exibe a listagem completa de autores cadastrados
Categorias Cadastradas.	Exibe a listagem completa de categorias cadastradas
Comentários Efetuados.	Exibe a listagem completa de *comentários* efetuados

3.1.2 Diagrama de Atividades

O funcionamento do sistema para a publicação de um *post* pode ser visualizado na FIGURA 10, através do diagrama de atividades.

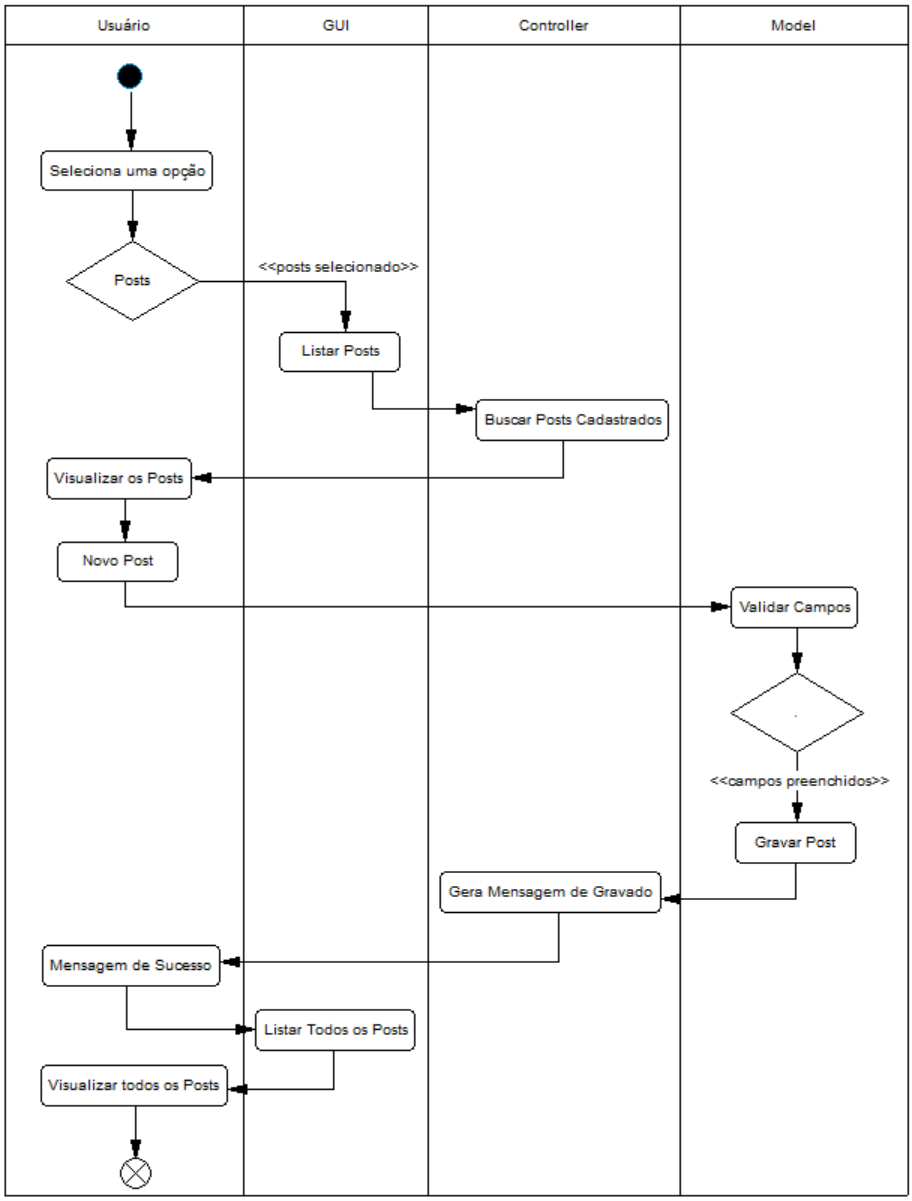

FIGURA 10 - Diagrama de Atividades

3.1.3 Diagrama de Classes

A arquitetura entre as classes pode ser visualizada na FIGURA 11, através do diagrama de classes.

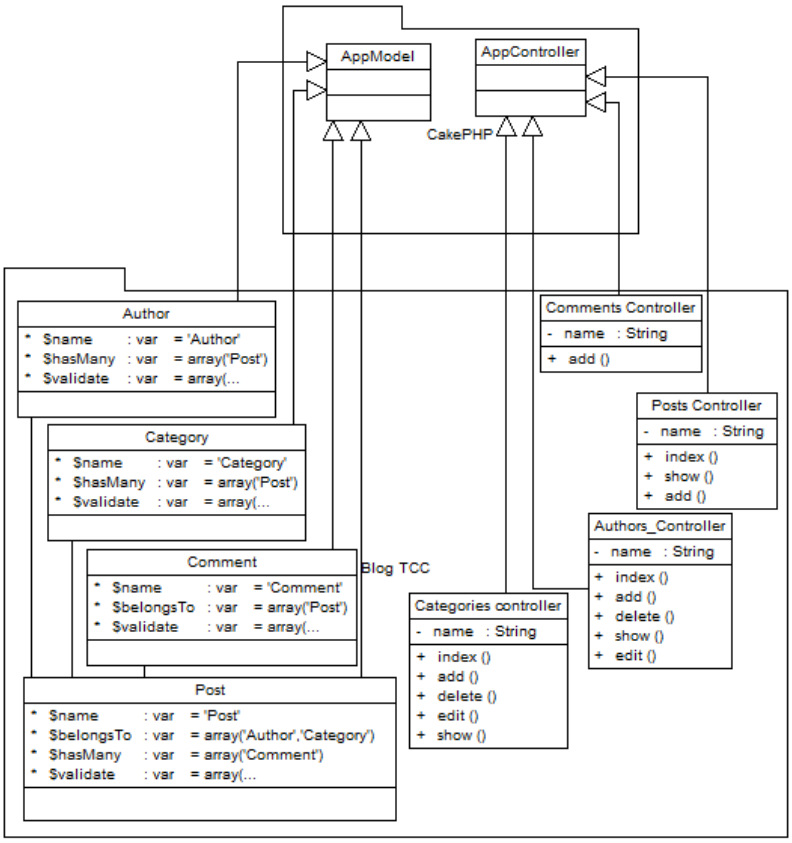

FIGURA 11 - Diagrama de Classes

Na FIGURA 11 pode-se observar ainda 2 pacotes: **CakePHP** e ***Blog* TCC**. O objetivo desta notação é reforçar que no CakePHP tudo *extende* das classes `AppModel` e `AppControler`. A arquitetura do estudo experimental concentra-se no segundo pacote (*Blog* TCC).

3.1.4 Máquina de Testes

Para demonstrar o funcionamento foi desenvolvida uma aplicação *Web* utilizando-se:

- *Notepad*++, como editor de texto;
- O *framework* para desenvolvimento *Web CakePHP* na versão 1.3.10;
- SGBD *MySQL*, no gerenciamento do banco de dados;
- O Xampp na versão 1.7.7 como servidor *Web;*
- *Windows* 7.

3.2 ESQUEMA DO BANCO DE DADOS

Para o desenvolvimento desta aplicação foi necessário criar 4 tabelas: `autor`, *`post`*, `comentario` e `categoria`.

Foram utilizados os nomes em inglês, tanto nos nomes das tabelas como nos nomes dos atributos das mesmas, pois é convenção do *CakePHP*. Poderiam ser usados termos em português sem nenhum impedimento, caso utilize-se dessa maneira nos *models* deverá ser especificado qual nome pertence a qual tabela, como pode-se observar no QUADRO 2.

QUADRO 2 - Exemplo utilizando nome da tabela em português

```php
<?php
class Author extends AppModel{
    var $name     = 'Author'; //nome do model
    var $useTable = 'autores';//nome da tabela está em português
```

A FIGURA 12 apresenta o MER (Modelo Entidade Relacionamento) do sistema, com as respectivas tabelas, bem como os atributos.

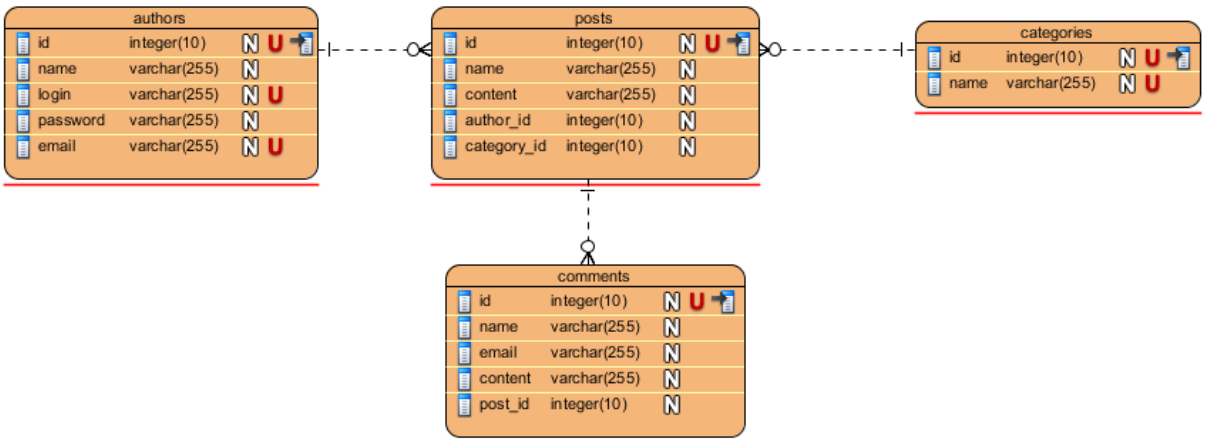

FIGURA 12 - Esquema do Banco de Dados

3.3 CONFIGURANDO O BANCO NA APLICAÇÃO

Com o banco de dados criado, é necessário dizer a aplicação qual banco utilizar, para isso deve-se navegar até a pasta *config* que se localiza dentro de *app* e

editar o arquivo *datasabe.php*. O QUADRO 3 mostrará como deverá ficar o arquivo *database.php* depois de editado.

QUADRO 3 - Arquivo *database.php* editado

```
class DATABASE_CONFIG {
var $default = array(
    'driver' => 'mysql',
    'persistent' => false,
    'host' => 'localhost',
    'login' => 'root',
    'password' => 'password',
    'database' => 'tccpatriky',
    'prefix' => '',
);

var $test = array(
    'driver' => 'mysql',
    'persistent' => false,
    'host' => 'localhost',
    'login' => 'user',
    'password' => 'password',
    'database' => 'test_database_name',
    'prefix' => '',
);
```

No arquivo `database.php` deve ser colocado o nome do banco de dados em *database*, neste caso foi usado o nome `tccpatriky`, o *password* (caso tenha) em *password*, o *acesso (login)* , o *host* de acesso e o *driver* utilizado.

3.4 CRIANDO OS MODELS DA APLICAÇÃO

Para cada tabela do banco, um *model* existirá com o mesmo nome da tabela, porém no singular. Nos *models* serão implementados as validações e serão informados os tipos de relacionamentos entre as tabelas. Caso os nomes das tabelas estejam em português, deverá ser especificado qual é a tabela que o *model* fará referência. Para os nomes das tabelas e os atributos utilizados nesta aplicação foi-se utilizado à convenção do *CakePHP*, sendo assim não foi necessário informar qual o nome da tabela, pois o *model* identifica como sendo a tabela pelo próprio nome do *model*.

No *model author, foi feito* o relacionamento *hasMany* (um pra muitos) *Post*, e no *model Post* foi feito o relacionamento *belongsTo* (pertence à) *author*, para estabelecer o relacionamento *One-to-Many* (um-para-muitos) entre as duas tabelas.

No QUADRO 4 pode-se observar o relacionamento e as validações no *model author*.

QUADRO 4 - *Model author*

```php
<?php
class Author extends AppModel{
    var $name     = 'Author'; //nome do model
    var $hasMany = array('Post');

    var $validate = array(
        'name' => array(
            'rule' => 'notEmpty',
            'message' => 'Campo Obrigatorio'
        ),
        'login' => array(
            'rule' => 'notEmpty',
            'message' => 'Campo Obrigatorio'
        ),
        'password' => array(
            'rule' => 'notEmpty',
            'message' => 'Campo Obrigatorio'
        ),
        'email' => array(
            'rule' => 'notEmpty',
            'message' => 'Campo Obrigatorio'
        )
    );
}
?>
```

No QUADRO 5 pode-se visualizar como ficaram os relacionamentos e as validações no *model Post*.

QUADRO 5 - *Model Post*

```php
<?php
class Post extends AppModel{
    var $name = 'Post';

    var $belongsTo = array('Author','Category');
    var $hasMany = array('Comment');

    var $validate = array(
        'name' => array(
            'rule' => 'notEmpty',
            'message' => 'Campo Obrigatorio'
        ),
        'content' => array(
            'rule' => 'notEmpty',
            'message' => 'Campo Obrigatorio'
        )
    );
}
?>
```

3.5 CRIANDO OS CONTROLLERS DA APLICAÇÃO

Os componentes do tipo *controller* são responsáveis por atender as requisições, tratar e gerar uma resposta. Para cada *model* existirá um *controller*, já os *controllers*, diferentemente dos *model*, devem estar no plural e no final dos respectivos nomes devem conter "*_controller*". Os *controllers* estendem a classe *AppController*.

Controllers podem incluir qualquer número de métodos que são referidos como *actions* (ações). As *actions* são métodos do controlador utilizados para mostrar as *views*. Uma *action* vai ser sempre um método do controlador.

O QUADRO 6 exibe a *action index* do controlador *authors_controller*, que será responsável por mostrar todos os autores cadastrados.

QUADRO 6 - *Action index* do *controller authors_controller*

```php
<?php
class AuthorsController extends AppController{
    var $name = 'Authors';

    //metodo listar todos
    function index(){
        $authors = $this->Author->find('all',array(
            'order' => array('Author.name' => 'asc')
            )
        );
        $this->set('authors',$authors);
    }
}
```

Esta *action* gerará uma *view* chamada *index*, onde serão exibidos os dados dos autores conforme desejado pelo desenvolvedor. No QUADRO 7 pode-se observar a *view index* gerada por esta *action*.

QUADRO 7 - *View index* gerada pela *action index*

```
<h1>Listagem de Autores</h1>
<?php echo $html->link('Novos Autores',array('controller' => 'authors','action' => 'add'))?>
<table>
        <tr>
            <th>Nome</th>
            <th>Email</th>
            <th>Ações</th>
        </tr>

<?php foreach($authors as $autor): ?>
        <tr>
            <td><?php echo $autor['Author']['name'] ?></td>
            <td><?php echo $autor['Author']['email'] ?></td>
            <td>
                <?php echo $html->link("Exibir",array(
                    'action' => 'show',
                    $autor['Author']['id']));?> /
                <?php echo $html->link("Deletar Autor",array(
                    'action'=>'delete',
                    $autor['Author']['id']),null,'Deseja Excluir?')?> /
                <?php echo $html->link("Editar Autor",array(
                    'action' => 'edit',
                    $autor['Author']['id']))?>
            </td>
        </tr>
<?php endforeach; ?>
</table>
```

Para cada *action* de cada controlador, será gerada uma *view* que exibirá os dados para o cliente.

O QUADRO 8 mostra a *action add*, responsável por passar os dados para armazenar, exibir uma mensagem informando que o autor foi salvo e em seguida redirecionar a página.

QUADRO 8 - *Action add*

```
//metodo cadastrar
    function add(){
    //se não estiver vazio dados do form
            if(!empty($this->data)){
                if($this->Author->save($this->data)){//no metodo save passar
                //os valores vindo do form por $this->data.
                    //se salvou o Post
                    $this->Session->setFlash('Autor Salvo');
                    $this->Redirect(array("action" => "index"));
                }else{
                    $this->Session->setFlash('Erro');
                }
            }
    }
```

O QUADRO 9 apresenta a *action delete*, responsável por excluir o autor e gerar uma mensagem de sucesso ou erro e redirecionar a página.

QUADRO 9 - *Action delete*

```
//metodo excluir
    function delete($id = null){
        if(!empty($id)){
            if($this->Author->delete($id)){
                $this->Session->setFlash('Autor deletado com sucesso');
                $this->Redirect(array("action" => "index"));
            }else{
                $this->Session->setFlash('Erro');
            }
        }
    }
```

O QUADRO 10 mostra a action *show*, que irá passar o *id* do autor e chamará a *view* que exibirá os dados do mesmo.

QUADRO 10 - *Action show*

```
//metodo show - exibir detalhes
    function show($id = null){
        $this->Author->id = $id;
        $this->set('author',$this->Author->read());
    }
```

No QUADRO 11 pode-se verificar a *action edit*, que passará o *id* do autor para a *view* de edição, em seguida verificará se houve alguma alteração ou não nos dados, caso positivo gerará uma mensagem informando que o mesmo foi editado.

QUADRO 11 - *Action edit*

```
function edit($id = null){
        $this->Author->id = $id;
        $this->set('author',$this->Author->find($id));
        if(empty($this->data)){
        //o Posto vai ser recuprado por ID,
        //caso não venha nenhuma alteração no form os dados vão ser os proprios atuais do Post
            $this->data = $this->Author->read();
        }else{
            if($this->Author->save($this->data)){
                $this->Session->setFlash("Author {$id} foi editado com sucesso.");
                $this->redirect(array("action" => "index"));
            }
        }
    }
```

Todas estas *actions* pertencem ao controlador *authors_controller*, e cada uma destas *actions* poderão gerar uma *view*.

As *action* não precisam, necessariamente, gerar uma *view*, como é o caso da *action delete*, onde o próprio controlador gera uma resposta informando que o autor foi deletado com sucesso.

É possível criar os *controllers* utilizando o *scaffolding* do CakePHP, o QUADRO 12 mostra como utilizar este *helper*.

QUADRO 12 - *Controller* criado utilizando *scaffolding* no CakePHP

```php
<?php
class AuthorsController extends AppController{
    public $scaffold;
}
?>
```

Já utilizando o *framework* Zend, é um pouco mais complexo como pode-se visualizar no QUADRO 13.

QUADRO 13 - *Controller* criado utilizando scaffolding no Zend

```php
<?php
require_once 'Zend.php';

function __autoload($className) {
        // fall back to zend framework
        Zend::loadClass($className);
    }

$controller = Zend_Controller_Front_Scaffold::getInstance();
$controller->setControllerDirectory('controllers');
$controller->scaffold('Blog', new BlogTable() );
$controller->scaffold('Mood', new MoodTable() );
$controller->scaffoldRelate('Blog', 'mood-id', 'Mood', 'id');

$controller->dispatch();
?>
```

O mesmo *controller* criado utilizando o Symfony tem a complexidade de utilizar-se linha de comando, como pode ser visualizado no QUADRO 14.

QUADRO 14 - *Controller* criado utilizando scaffolding no Symfony

```
Symfony> propel-init-crud myapp artigo artigo
```

3.6 CRIANDO AS VIEWS DA APLICAÇÃO

As *views* devem estar dentro do diretório referente ao controlador que as gera. Assim, no caso das *views* do controlador *authors_controller*, as *views* estarão dentro do diretório *authors*, por conseguinte dentro deste diretório estarão todas as

views geradas pelas *actions* do controlador *authors_controller* que serão a *index*, *edit*, *add* e a *show*, como mostra a FIGURA 13.

FIGURA 13 - *Views* geradas pelo *authors_controller*

O QUADRO 15 apresenta o código da *view add*, responsável pela criação do formulário de cadastro de novos autores.

QUADRO 15 - *View add*

```
<h3>Novo Author</h3>

<?php
    echo $form->create('Author'); //form para Post
    echo $form->input('name');
    echo $form->input('login');
    echo $form->input('password');
    echo $form->input('email');
    echo $form->end('Salvar AUthor');//fecha o form e cria o submit com o texto
?>
```

O QUADRO 16 mostra o código da *view edit*, responsável por mostrar os dados já cadastrados para efetuar-se a edição dos mesmos.

QUADRO 16 - *View edit*

```
<h3>Editar Author <?php echo $author['Author']['name']; ?></h3>

<?php
    echo $form->create('Author',array('action' => 'edit')); //form para Post
    echo $form->input('name');
    echo $form->input('login');
    echo $form->input('password');
    echo $form->input('email');
    //echo $form->input('id',array('type' => 'hidden'));
    echo $form->end('Editar Autor');//fecha o form e cria o submit com o texto
?>
```

No QUADRO 17 pode-se visualizar o código da *view index*, que exibirá os autores cadastrados e os respectivos e-mails para contato, bem como as ações de exibir, excluir e editar autor.

QUADRO 17 - *View index*

```
<h1>Listagem de Autores</h1>
<?php echo $html->link('Novos Autores',array(
                        'controller' => 'authors',
                        'action' => 'add'))?>
<table>
        <tr>
            <th>Nome</th>
            <th>Email</th>
            <th>Ações</th>
        </tr>

<?php foreach($authors as $autor): ?>
        <tr>
            <td><?php echo $autor['Author']['name'] ?></td>
            <td><?php echo $autor['Author']['email'] ?></td>
            <td>
                <?php echo $html->link("Exibir",array(
                    'action' => 'show',
                    $autor['Author']['id']));?> /
                <?php echo $html->link("Excluir Autor",array(
                    'action'=>'delete',
                    $autor['Author']['id']),null,'Deseja Excluir?')?> /
                <?php echo $html->link("Editar Autor",array(
                    'action' => 'edit',
                    $autor['Author']['id']))?>
            </td>
        </tr>
<?php endforeach; ?>
</table>
```

O QUADRO 18 apresenta a *view show*, que exibirá todos os dados do respectivo autor.

QUADRO 18 - *View show*

```
<h1>dados do autor</h1>

<ul>
    <li>ID: <?php echo $author['Author']['id'] ?></li>
    <li>Nome: <?php echo $author['Author']['name']?></li>
    <li>Login : <?php echo $author['Author']['login']?></li>
    <li>Password: <?php echo $author['Author']['password']?></li>
    <li>Email: <?php echo $author['Author']['email']?></li>
</ul>

<?php echo $html->link('Voltar',array('action' => 'index')) ?>
```

4 RESULTADOS E DISCUSSÕES

Ao acessar o sistema será exibida a página inicial que é a página de boas vindas com um texto introdutório, a FIGURA 14 exibe esta página.

FIGURA 14 - Página inicial do *blog*

O sistema possui cadastros de autores e categorias (FIGURA 15), sendo que estes fazem parte de um *Post*, ou seja, cada um que for gravado terá uma categoria e um autor. Desta forma, antes de cadastrar o *Post* deve-se ser cadastrado o autor e a categoria referente ao mesmo.

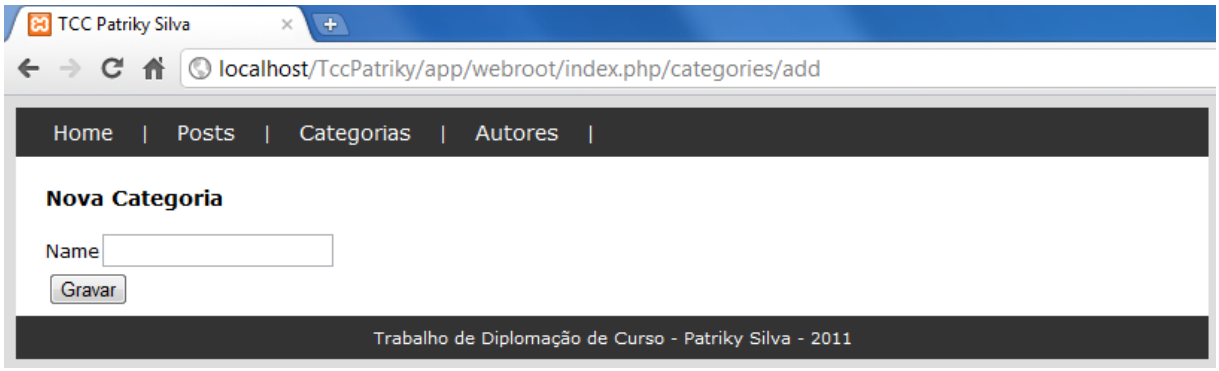

FIGURA 15 - Página de cadastro de Categoria

O CakePHP utiliza o roteamento *RESTful*, e como pode ser visualizado na *URL* da FIGURA 15, a mesma está dividida em: www.dominio.com / controlador / ação

- www.dominio.com é localhost/TccPatriky/app/webroot/index.php, pois está sendo utilizado um servidor local;

- Controlador é *categories*;
- Ação é *add.*

Sendo assim, o CakePHP irá chamar a *action* (ação) *add* do *controller categories*, que adicionará uma nova categoria.

A FIGURA 16 mostra a página de cadastro de novos autores, onde deve ser informado o nome, *acesso (login)* , *password* e *email* do mesmo.

FIGURA 16 - Página de cadastro de Autor

Depois de cadastrado o autor e a categoria do *Post*, é possível realizar o cadastro do mesmo. A FIGURA 17 apresenta a página de cadastro de *Posts*.

FIGURA 17 - Página de cadastro de *Post*

Depois de efetuado o cadastro, pode-se visualizar o nome e *e-mail* do autor bem como as ações possíveis de serem efetuadas. Na FIGURA 18 pode-se visualizar os autores cadastrados.

FIGURA 18 - Autores cadastrados

A FIGURA 19 exibe as categorias cadastradas e as ações que são possíveis efetuar.

FIGURA 19 - Categorias cadastradas

A FIGURA 20 apresenta os *Posts* cadastrados, onde é mostrado um pequeno resumo do *Post*. A visualização do *Post* completo é possível acessando o *link* `Post Completo`.

FIGURA 20 - *Posts* cadastrados

Depois de efetuado o cadastro dos *Posts*, é exibido um pequeno resumo sobre o mesmo e a quantidade de comentários. Se o usuário desejar visualizar o *Post* completo será mostrado um *link* para o *Post* completo. Quando acessado para visualização o *Post* completo, pode-se cadastrar comentários referentes aquele *Post*. Na FIGURA 21 pode-se observar a página que mostra o *Post* completo com o formulário para cadastro de comentário.

FIGURA 21 - Página do *Post* completo e cadastro de comentário

Visualizando a URL da FIGURA 21, pode-se verificar que além do controlador e da ação, agora a mesma possui o parâmetro. Sendo assim a URL dividiu-se em www.dominio.com / controlador / ação / parâmetro, sendo que o parâmetro é a última informação da URL (1).

Depois de realizado o comentário, é possível visualizá-lo conforme mostra a FIGURA 22.

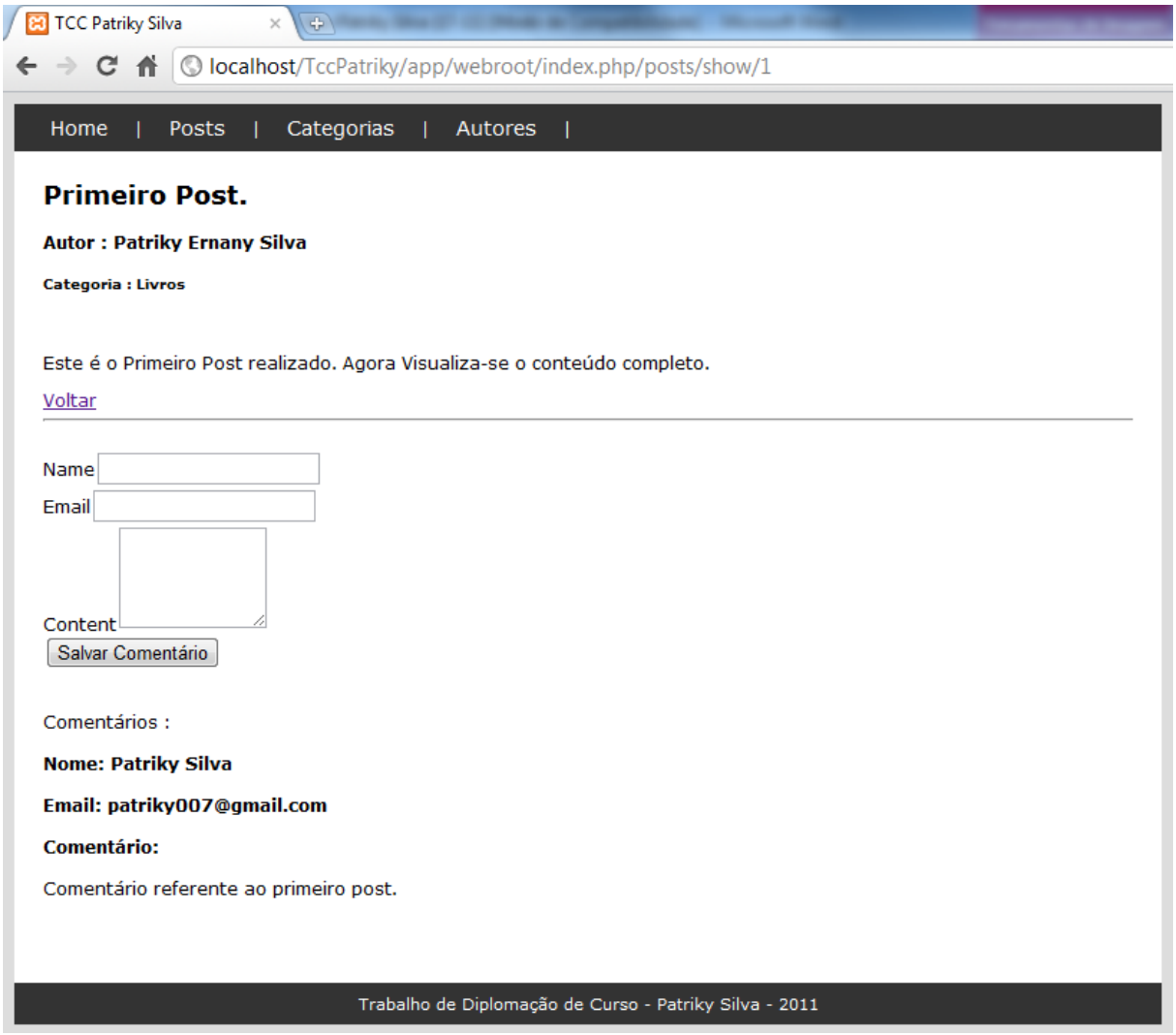

FIGURA 22 - Visualização dos comentários

É possível visualizar a quantidade de comentários de cada *Post*, como pode-se observar através da FIGURA 23.

FIGURA 23 - Quantidade de comentários

Este trabalho é restrito a excluir ou alterar os *Posts* ou comentários, já os autores e as categorias podem ser alteradas e excluídas, pode-se visualizar os dados, tanto da categoria como do autor. A FIGURA 24 mostra a visualização dos dados da categoria.

FIGURA 24 - Visualização dos dados da categoria

A FIGURA 25 apresenta a visualização de todos os dados do respectivo autor, onde é exibido o *acesso (login)* e o *password*, caso o mesmo não recorde.

FIGURA 25 - Visualização dos dados do autor

O *acesso (login)* e o *password* do autor estão sendo armazenado, porém não são utilizados no estudo experimental, já que o mesmo não possui acesso restrito a páginas. Logo, quando o mesmo possuir, deve-se cuidar para esses dados não serem visualizados por outros autores.

5 CONSIDERAÇÕES FINAIS

Este capítulo trata das considerações finais que foram abstraídas durante o desenvolvimento do projeto. Também serão sugeridas algumas ideias para continuidade da pesquisa e aplicação.

5.1 CONCLUSÃO

Após a apresentação e realização do experimento, pode-se verificar que o *framework*:

- oferece vários *helpers* para o desenvolvimento de aplicações *PHP*;
- permite o desenvolvimento rápido, utilizando-se *scaffolding*;
- não necessita configuração para desenvolvimento;
- não utiliza linhas de comando para criar aplicações;
- utiliza o roteamento RESTful;

Tendo-se como base um dos *frameworks* mais utilizados para desenvolvimento de *sites*, o *CakePHP*, possibilitou a obtenção de bons resultados tratando-se de organização nos códigos, na agilidade do processo, na facilidade de manutenção e no aprendizado simples e rápido.

O CakePHP, por meio do padrão de projetos MVC, mostrou-se eficaz organizando todo o código da aplicação, separando a lógica de negócio da parte visual apresentada para o requisitante (*browser*).

5.2 TRABALHOS FUTUROS

Como sugestão para trabalhos futuros, fica a ideia de explorar melhor as funcionalidades do *CakePHP* como o acesso a páginas restritas e validação por e-mail no cadastro do autor, evitando que pessoas de má fé se passem por outra pessoa e, utilizar outro *framework* para o desenvolvimento de sistemas *Web*.

REFERÊNCIAS BIBLIOGRÁFICAS

A Linguagem PHP. Disponivel em: <http://www6.ufrgs.br/engcart/PDASR/linguagens.html>. Acesso em: 13 outubro 2011.

ALVAREZ,. **O que é um CMS**, 2008. Disponivel em: <http://www.criarweb.com/artigos/o-que-e-um-cms.html>. Acesso em: 13 outubro 2011.

BELEM, T. **Frameworks no PHP:** O que, quando, porque e qual?, 2009. Disponivel em: <http://*blog*.thiagobelem.net/frameworks-no-php-o-que-quando-porque-e-qual/>. Acesso em: 04 dez. 2011.

BRAGANÇA, W.. Wanderson C. Bragança. **Frameworks PHP**, 2010. Disponivel em: <http://www.wbraganca.com/2010/08/frameworks-php/>. Acesso em: 11 outubro 2011.

BRITO, L. M. P. PHP-NET. **O que é PHP e qual a sua Sintaxe**, 2007. Disponivel em: <http://www.php-pt.com/index.php?option=com_content&task=*view*&id=13&Itemid=28>. Acesso em: 04 dez. 2011.

BROUWER,. **The PHP Beginners Tutorial**, 2002. Disponivel em: <http://www.revistaphp.com.br/print.php?id=20>. Acesso em: 15 out. 2011.

BROWN, G. Revista PHP / Frameworks. **Uma Introdução aos frameworks**, 2008. Disponivel em: <http://www.revistaphp.com.br/artigo.php?id=210>. Acesso em: 11 outubro 2011.

CALAÇA, O. SlideShare. **Introdução ao *Framework* CakePHP**, 2010. Disponivel em: <http://www.slideshare.net/otaviocx/introduo-ao-*framework*-cakephp>. Acesso em: 11 outubro 2011.

CHAD. PHPFrameworks. **notes on choosing a php *framework* a comparison of cakephp and the zend *framework***, 2008. Disponivel em: <http://www.phpframeworks.com/news/p/476/notes-on-choosing-a-php-*framework*-a-comparison-of-cakephp-and-the-zend-*framework*>. Acesso em: 27 nov. 2011.

CHEUNG,. webappers. **Comparison Between Zend and CakePHP *Framework***, 2008. Disponivel em: <http://www.webappers.com/2008/12/12/comparison-between-zend-and-cakephp-*framework*/>. Acesso em: 27 nov. 2011.

DARLAN, D. **O que é um *framework*?**, 2008. Disponivel em: <http://www.oficinadanet.com.br/artigo/683/o_que_e_um_*framework*>. Acesso em: 12 outubro 2011.

FARIA, T. TulioFaria.net. **O QUE UM *FRAMEWORK* NÃO É**, 2009. Disponivel em: <http://www.tuliofaria.net/o-que-um-*framework*-nao-e/>. Acesso em: 11 outubro 2011.

FAYAD, W. E. et al. TCC Jhony. **UMA ANÁLISE ENTRE FRAMEWORKS DE PHP**, 2006. Disponivel em: <http://www2.unochapeco.edu.br/~jhony/tcc.pdf>. Acesso em: 11 outubro 2011.

FOUNDATION, CakePHP. **CakePHP**, 2011. Disponivel em: <http://cakephp.org/>. Acesso em: 26 nov. 2011.

GIBERSON,. Zend *Framework*. **Zend**_***Controller***_**Front**_**Scaffold**, 2007. Disponivel em: <http://*framework*.zend.com/wiki/display/ZFPROP/Zend_*Controller*_Front_Scaffold>. Acesso em: 27 nov. 2011.

GUNDERLOY,. **Roteamento Rails de fora para dentro**, 2008. Disponivel em: <http://guias.rubyonrails.com.br/routing.html#o-que--rest>. Acesso em: 05 dez. 2011.

INDIA, R. The Bakery. **Desenvolvimento de aplicações Web RESTful no CakePHP**, 2010. Disponivel em: <http://bakery.cakephp.org/por/articles/rightwayindia/2010/10/09/desenvolvimento_de_aplica%C3%A7%C3%B5es_web_restful_no_cakephp>. Acesso em: 05 dez. 2011.

JAQUES,. PHPIT. **Frameworks PHP – Qual é o melhor pra você?**, 2009. Disponivel em: <http://www.phpit.com.br/artigos/frameworks-php-qual-e-o-melhor-pra-voce.phpit>. Acesso em: 11 outubro 2011.

KROLOW,. Cobaia.net. **Começando com o CakePHP – Tutorial introdutório ao *Framework* CakePHP**, 2008. Disponivel em: <http://cobaia.net/2008/09/comecando-com-o-cakephp-tutorial-introdutorio-ao-*framework*-cakephp/>. Acesso em: 12 outubro 2011.

LISBOA, F. G. D. S. **Zend *Framework* - Desenvolvendo em PHP 5 orientado a objetos com MVC**. Disponivel em: <http://www.olivreiro.com.br/pdf/livros/cultura/2429865.pdf>. Acesso em: 12 outubro 2011.

LOPES, L. SlideShare. **Cake Php**, 2010. Disponivel em: <http://www.slideshare.net/lauralopes/cake-php-2356380?src=related_normal&rel=4790577>. Acesso em: 11 outubro 2011.

NET, R. O. D. Oficida da Net. **Frameworks:** CakePHP, 2008. Disponivel em: <http://www.oficinadanet.com.br/artigo/989/frameworks_cakephp>. Acesso em: 11 outubro 2011.

NET, R. O. D. Oficina da Net. **Frameworks:** Zend, 2008. Disponivel em: <http://www.oficinadanet.com.br/artigo/994/frameworks_zend>. Acesso em: 11 outubro 2011.

POTENCIER,. The Definitive Guide to symfony. **Chapter 14 - Generators**. Disponivel em: <http://www.symfony-project.org/book/1_0/14-Generators>. Acesso em: 27 nov. 2011.

SCHIRM,. O Genial Soluções Web. **O que é um *Framework*?**, 2010. Disponivel em: <http://www.ogenial.com.br/*blog*/o-que-e-*framework*/>. Acesso em: 11 outubro 2011.

SILVA, C. Revista PHP. ***Framework* CakePHP**, 2007. Disponivel em: <http://www.revistaphp.com.br/artigo.php?id=93>. Acesso em: 11 outubro 2011.

TIOBE *Software*. **TIOBE Programming Community Index for October 2011**. Disponivel em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Acesso em: 13 outubro 2011.

Top 10 Ranking PHP Frameworks. Disponivel em: <http://www.phpframeworks.com/top-10-php-frameworks/>. Acesso em: 13 outubro 2011.

VELOSO, N. NV.net. **Rotina para autenticação em PHP**, 2010. Disponivel em: <http://nveloso.net/index.php/rotina-para-autenticacao-em-php/>. Acesso em: 04 dez. 2011.

WAZLAWICK, R. S. **Análise e Projeto de Sistemas de Informação Orientados a Objetos**. Rio de Janeiro: Campus, 2004.

ZEMEL,. CodeIgniter Brasil. **O que é um *framework*? definição e benefícios de se usar frameworks**, 2009. Disponivel em: <http://codeigniterbrasil.com/passos-iniciais/o-que-e-um-*framework*-definicao-e-beneficios-de-se-usar-frameworks/>. Acesso em: 11 outubro 2011.