Alfamídia
Programação: Design Pattern - MVC com PHP

Visite www.alfamidiaonline.com.br para outros conteúdos e cursos gratuitos e pagos

AVISO DE RESPONSABILIDADE

As informações contidas neste material de treinamento são distribuídas “NO ESTADO EM QUE SE ENCONTRAM”, sem qualquer garantia, expressa ou implícita. Embora todas as precauções tenham sido tomadas na preparação deste material, a Alfamídia Prow não têm qualquer responsabilidade sobre qualquer pessoa ou entidade com respeito à responsabilidade, perda ou danos causados, ou alegadamente causados, direta ou indiretamente, pelas instruções contidas neste material ou pelo software de computador e produtos de hardware aqui descritos.

08/2011 – Versão 1.0

Alfamídia Prow
http://www.alfamidia.com.br

Unidade 1
PHP com MVC

1.1 Sobre o Curso

O Curso Sistemas de Comércio Eletrônico com MVC com PHP apresenta aos alunos os conceitos e técnicas envolvidos no desenvolvimento de uma aplicação de comércio eletrônico.

Além de compreender noções importantes, como validação de usuários e gerenciamento de sessões.

1.2 Estrutura do Curso

Primeiramente veremos como preparar o ambiente onde a aplicação será executada.

As unidades seguintes abordam tópicos como: autenticação de usuários e segurança, como manter o usuário logado (sessões) e acesso à base de dados, entre outros.

Unidade 2
Padrões de Projeto

2.1 Padrões de Projeto

"Alguém já resolveu os seus problemas!"

Os padrões de projeto de software ou padrões de desenho de software, também muito conhecido pelo termo original em inglês, Design Patterns, descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de software orientados a objetos. Um padrão de projeto estabelece um nome e define o problema, a solução, quando aplicar esta solução e suas consequências.

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

Os padrões de projetos são na realidade um conjunto de procedimentos e metodologias para resolver questões recorrentes. Estes padrões foram documentados e permitem que a equipe de desenvolvimento (analistas, programadores, etc) possam adaptar integralmente ou parcialmente este modelo de solução de acordo com sua necessidade.

2.2 Orientação a Objetos e organização em camadas

Um dos maiores desafios das equipes de desenvolvimento de aplicações é o de criar aplicativos cada vez mais seguros, eficientes, de fácil manutenção, reutilizáveis e em prazos cada vez menores. Neste sentido a utilização do paradigma de orientação a objetos se transformou na opção natural para se atingir este grau de maturidade de produção dos sistemas.

Ao criarmos uma aplicação utilizando o conceito de orientação a objetos o sucesso no seu desenvolvimento este diretamente atrelado á arquitetura que vamos usar para construir a aplicação.

Podemos observar claramente que a tendência indica que esta arquitetura estará baseada na organização da aplicação em camadas e na observação dos padrões utilizados pelo mercado.

A organização em camadas é o principal ponto para a independência entre os componentes e esta independência é que vai atingir os objetivos de eficiência, escalabilidade, reutilização e facilidade de manutenção.

Inicialmente para gerar aplicativos multicamadas temos a sensação de que o processo é mais complexo, mas se entrarmos a fundo na compreensão da evolução tecnológica que levou a chegar neste desenho, vamos perceber que é bem clara a necessidade de utilização desta configuração.

O termo camada pode significar uma separação física ou uma camada lógica, no caso dos Padrões de projeto, consideramos a camada como uma referência a separação de responsabilidades.

Unidade 3
As diferentes arquiteturas de Aplicações

3.1 Aplicações de Uma Camada

A estrutura física dos computadores sempre exerceu uma forte influencia no padrão de programação, seja limitando a quantidade de memória disponível para rodar a aplicação ou restringindo os recursos visuais que poderiam ser apresentados.

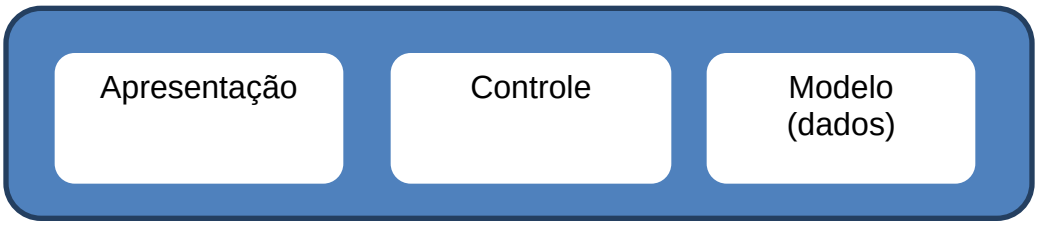
Neste sentido a programação, inicialmente era conhecida como monolítica, ou seja os papeis de Apresentação dos dados, armazenamento e controle dos mesmos eram armazenados em um mesmo computador.

O fato de inicialmente os computadores, de forma geral, serem de grande porte e via de regra não possuírem comunicação entre eles, também foi responsável por esta necessidade de agrupamento das camadas, mesmo que estas possuam papéis diferenciados.

Este desenho de desenvolvimento trazia uma série de complicadores para o desempenho de construção, bem como para o processo de utilização das mesmas. Dentre estes problemas podemos citar: uma grande quantidade de linhas de código e de manutenção extremamente complicada.

As camadas lógicas de entrada do usuário, verificação, lógica de negócio e acesso a banco de dados estava presente em um mesmo lugar.

Visualmente podemos reproduzir a estrutura monolítica da seguinte forma:



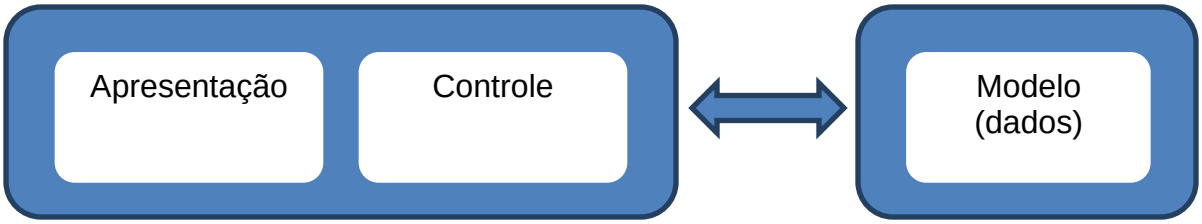
3.2 Aplicações de Duas Camadas

Com a evolução tecnologia e a descentralização da informação, sugiram as redes de computadores, o que possibilitou inicialmente a separação da base de dados das demais camadas.

Este novo recurso, fez com que surgisse a arquitetura em duas camadas, onde aparece a figura do servidor de banco de dados e dos clientes, que ainda possuíam a Apresentação dos dados bem como todas as regras de Controle.

Esta configuração trouxe consigo a vantagem de permitir a distribuição das informações em espaços físicos distintos, mas compartilhados. No entanto a manutenção do versionamento das aplicações contidas nos clientes se transformou em um verdadeiro exercício de guerra.

Visualmente podemos observar esta arquitetura da seguinte forma:



3.3 Aplicações de Três Camadas

Graças a evolução das redes de comunicação, dentre elas a internet, ficou evidente a necessidade de separar a lógica de negócio da interface com o usuário. O objetivo é o de que os usuários das redes possam acessar as mesmas aplicações sem ter que instalar estas aplicações em suas máquinas locais.

Como a lógica do aplicativo, inicialmente contida no cliente(cliente rico) não reside mais na máquina do usuário este tipo de cliente passou a ser chamado de cliente pobre ou magro.(thin).

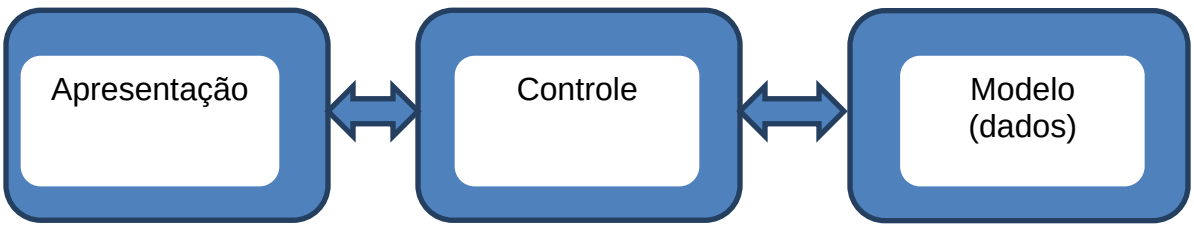
Neste modelo o aplicativo é movido para o Servidor e um navegador Web é usado como um cliente magro. O aplicativo é executado em servidores Web com os quais o navegador Web se comunica e gera o código HTML para ser exibido no cliente.

Neste modelo a lógica de apresentação esta separada em sua própria camada lógica e física. A separação em camadas lógicas torna os sistemas mais flexíveis permitindo que as partes possam ser alteradas de forma independente.

As funcionalidades da camada de negócio podem ser divididas em classes e essas classes podem ser agrupadas em pacotes ou componentes reduzindo as dependências entre as classes e pacotes estes por sua vez podem ser reutilizadas por diferentes partes do aplicativo e até por aplicativos diferentes. O modelo de 3 camadas tornou-se a arquitetura padrão para sistemas corporativos com base na Web.

A modelagem orientada a objetos ajuda a promover a modularidade, pois os objetos encapsulam seus dados (propriedades, estados) e oferecem funcionalidades através de seus métodos. Projetando-se de forma adequada os objetos podem ter reduzidas as dependências entre si ficando assim fracamente acoplados e serão mais fáceis de manter e evoluir.

Visualmente podemos definir assim:



Unidade 4
A arquitetura do padrão MVC

4.1 Introdução

O Model-View-Controller (MVC) é um padrão de arquitetura de aplicação gerado a partir da comunidade SmallTalk na década de 1970 por Reenskaug Trygve. E foi popularizada para uso na web com o advento do Ruby on Rails em 2004.

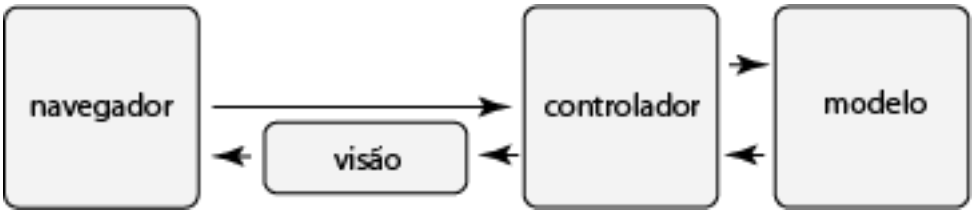
A abordagem MVC é composta por três tipos de objetos:

• Model (Modelo). Objetos que implementam a lógica para o domínio de dados do aplicativo. Muitas vezes, os objetos do Modelo recuperam e armazenam seus estados em um banco de dados. Por exemplo, um objeto Produto pode recuperar informações de um banco de dados, operar sobre elas, e em seguida, escrever informações atualizadas de volta a uma tabela no banco de dados.

• View (Visão). Objeto que exibe a interface do aplicativo com o usuário (UI), criado a partir dos dados do Modelo. Um exemplo de uma Visão seria a representação dos dados de um objeto de modelo Produto, que apresenta ao usuário caixas de textos com informações do Nome, Descrição, Preço e Categoria do Produto.

• Controller (Controle). São componentes que lidam com a interação do usuário, trabalhar com o Modelo e, finalmente, toma a decisão de qual Visão mostrar. Em um aplicativo MVC, a Visão exibe informações de Modelo e o Controle manipula e responde a entrada do usuário em uma interação.

O padrão MVC ajuda a criar aplicações que separam os diferentes aspectos da aplicação (lógica de requisição/entrada, a lógica do negócio e lógica UI), proporcionando um acoplamento fraco entre esses elementos. O padrão especifica onde cada tipo de lógica deve ser estar na aplicação. A lógica da interface do usuário pertence à Visão. Lógica de requisição/entrada pertence ao Controller. A lógica do negócio pertence ao Modelo. Esta separação ajuda você a gerenciar a complexidade quando você cria um aplicativo, porque ele permite que você concentre-se em um aspecto da aplicação de cada vez. Por exemplo, você pode se concentrar na visão sem depender da lógica de negócios.



4.2 Arquitetura MVC

Quando falamos de uma arquitetura que respeite o modelo de três camadas físicas ocorre uma divisão no aplicativo de modo que a lógica de negócio esteja localizada na parte central das três camadas físicas. Este comportamento recebe a nomenclatura de "camada física intermediária" ou "camada física de negócios". A maior parte do código escrito reside na camada de apresentação e na camada de negócio.

Este layout de distribuição lógica e física dos sistemas é reconhecido como a arquitetura MVC - (Modelo, Visualização, Controle) que nos proporciona uma forma de facilitar o processo de manutenção e apresentação dos dados de uma aplicação.

Quando falamos de MVC, não estamos nos referindo a algo novo na área do Desenvolvimento, e sim de uma arquitetura que foi originalmente desenvolvida para mapear as tarefas tradicionais de entrada, processamento e saída para o modelo de interação com o usuário. A utilização do padrão MVC transforma o processo de mapear esses conceitos no domínio de aplicações Web multicamadas, mais simples eficaz e rápido.

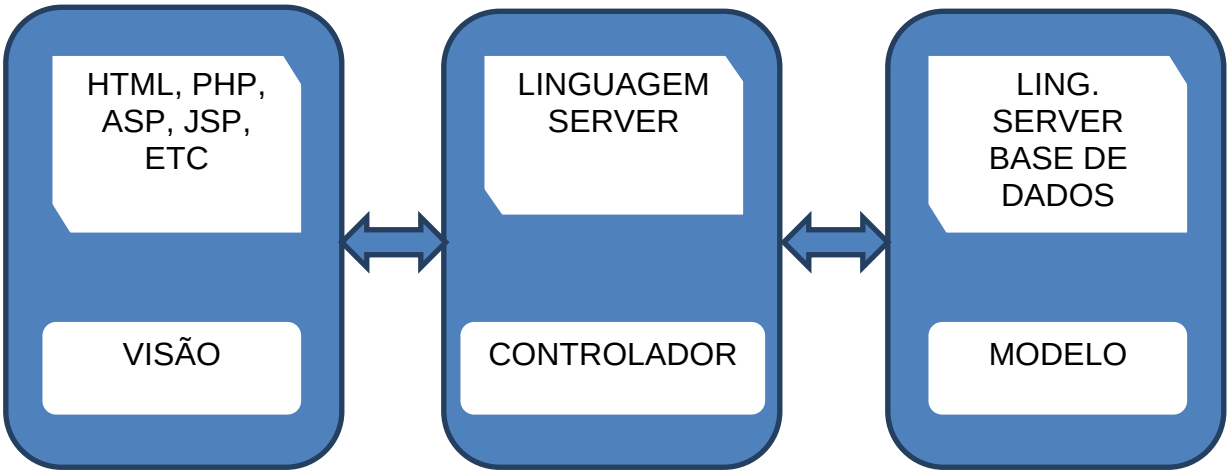
Na arquitetura MVC o modelo representa os dados da aplicação e as regras do negócio que determinam como será o acesso e a modificação dos dados. O modelo também é responsável por acessar o repositório de dados e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo.

Um componente de visualização apresenta o conteúdo de uma parte específica do modelo e encaminha para o controlador as ações do usuário. Além disso, a visualização também acessa também os dados do modelo via controlador e define como esses dados devem ser apresentados.

Um controlador define como irá de comportar a aplicação, é ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Em um cliente de aplicações Web essas ações do usuário poderiam ser cliques de botões ou seleções de menus. As ações realizadas pelo modelo incluem ativar processos de negócio ou alterar o estado do modelo.

Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta a solicitação do usuário. Há normalmente um controlador para cada conjunto de funcionalidades relacionadas.

A arquitetura de 3 camadas que esta representada abaixo é uma implementação do modelo MVC . O modelo MVC esta preocupado em separar a informação de sua apresentação.

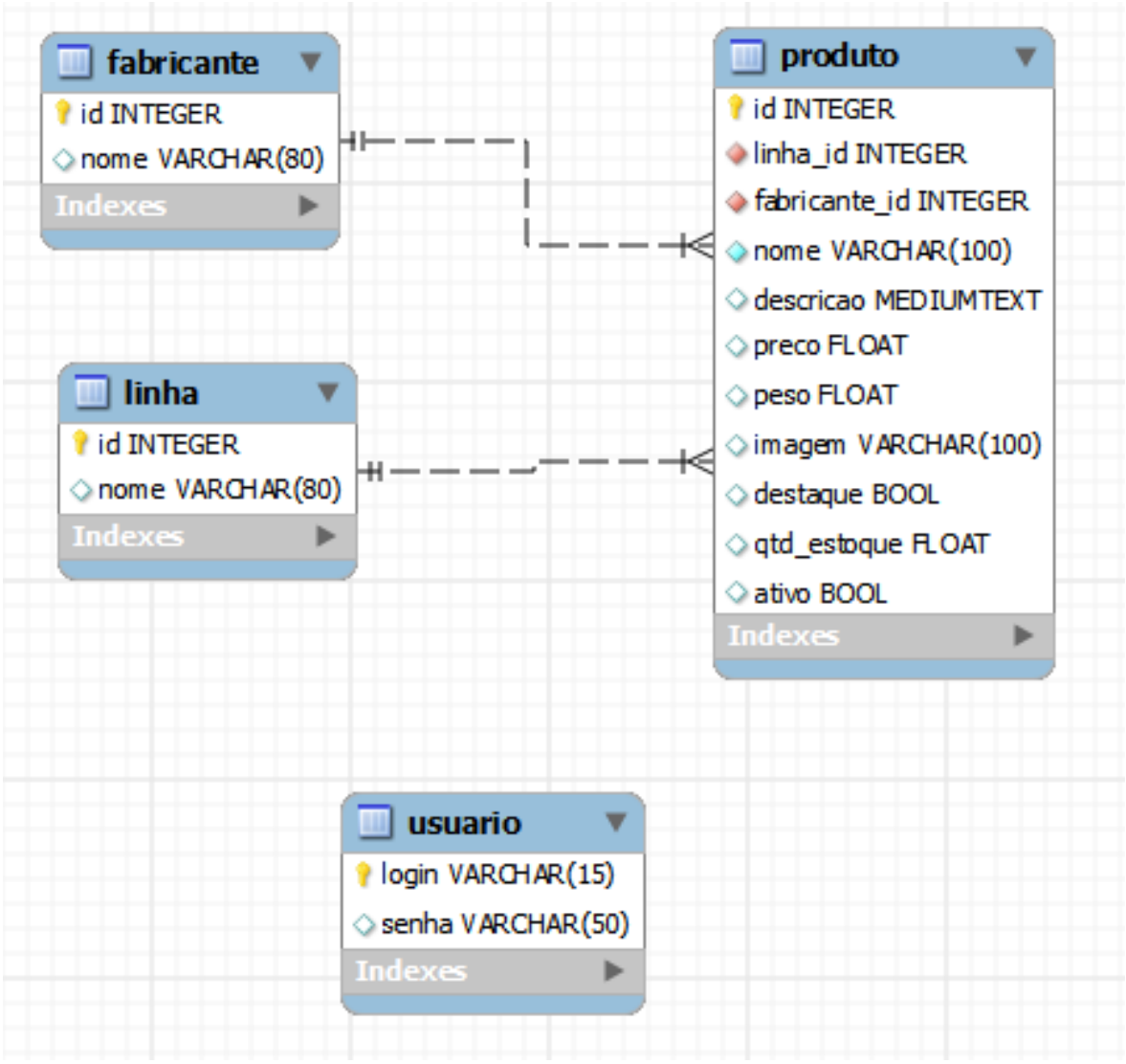


Unidade 5
Criando uma aplicação MVC

Vamos construir uma aplicação usando MVC com PHP que consistirá em um sistema de gerenciamento de produtos para loja virtual. Essa aplicação terá um controle de acesso que irá incluir, ler, atualizar e excluir (CRUD = create, read, update, delete) produtos do catálogo de uma loja virtual.

5.1 Definindo o modelo de dados

Nossa aplicação irá alimentar a tabela produto que tem a seguinte estrutura:



Script SQL para geração:

```
-- -----------------------------------------------------
-- TABELA fabricante
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `fabricante` (
      id INT(11) NOT NULL AUTO_INCREMENT,
      nome VARCHAR(80) NULL, PRIMARY KEY (id)
) ENGINE=InnoDB;

-- -----------------------------------------------------
-- TABELA linha
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS linha (
      id INT(11) NOT NULL AUTO_INCREMENT,
      nome VARCHAR(80) NULL, PRIMARY KEY (id)
) ENGINE=InnoDB;


-- -----------------------------------------------------
-- TABELA produto
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS produto (
      id INT(11) NOT NULL AUTO_INCREMENT,
      linha_id INT(11) NOT NULL,
      fabricante_id INT(11) NOT NULL,
      nome VARCHAR(100) NOT NULL,
      descricao MEDIUMTEXT NULL,
      preco FLOAT NULL,
      peso FLOAT NULL,
      imagem VARCHAR(100) NULL,
      destaque TINYINT(1) NULL,
      qtd_estoque FLOAT NULL,
```

```
      ativo TINYINT(1) NULL,
      PRIMARY KEY (id),
      INDEX produto_FKIndex1 (fabricante_id ASC),
      INDEX produto_FKIndex2 (linha_id ASC),
      CONSTRAINT fk_01 FOREIGN KEY (fabricante_id) REFERENCES
fabricante (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,
      CONSTRAINT fk_02 FOREIGN KEY (linha_id) REFERENCES linha (id)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
) ENGINE=InnoDB;
```

E vamos alimentar as tabelas linha e fabricante com alguns valores:

```
INSERT INTO linha (nome) VALUES ('Camiseta Oficial');
INSERT INTO linha (nome) VALUES ('Camiseta Treino');
INSERT INTO linha (nome) VALUES ('Camiseta Comemorativa');

INSERT INTO fabricante (nome) VALUES ('Adidas');
INSERT INTO fabricante (nome) VALUES ('Dal Ponte');
INSERT INTO fabricante (nome) VALUES ('Olympicus');
```

Também vamos criar e alimentar uma tabela para o login do usuário no sistema:

```
-- -----------------------------------------------------
-- TABELA usuario
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS usuario (
      login VARCHAR(15) NOT NULL ,
      senha VARCHAR(50) NULL ,
      PRIMARY KEY (login)
)ENGINE = InnoDB;

-- USUÁRIO INICIAL DO SISTEMA
INSERT INTO usuario (login, senha) VALUES ('admin', MD5('123'));
```

Repare que não vamos armazenar a senha da maneira usual, simplesmente guardando o seu valor no banco de dados. Isso deixaria o usuário utilizador do sistema vulnerável, pois se o mesmo utilizar a sua senha em mais de um sistema, ele ficaria exposto em caso de invasão ao nosso site.

Por isso vamos usar a função MD5 para gerar um HASH da mesma. Hash, em termos mais técnicos, é uma sequência de caracteres (letras ou números) gerada por um algoritmo de dispersão que transforma uma (possível) grande quantidade de dados em uma pequena quantidade de forma quase exclusiva.

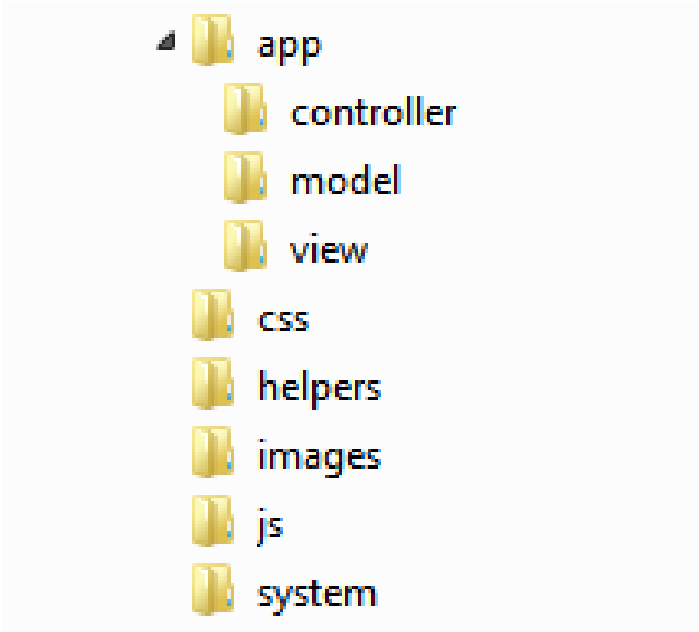
A seqüencia de Hash busca identificar uma informação de maneira única, sendo um método para transformar dados visando um resultado exclusivo.Por exemplo, criando identificações para uma mensagem de correio eletrônico, uma senha, uma chave criptográfica ou mesmo um arquivo.

No nosso caso, o HASH gerado pela senha do usuário será armazenado no banco. Não há como descobrir qual senha gerou aquele HASH. A única maneira possível são os métodos de força bruta, onde o atacante tenta adivinhar qual a senha que gera um determinado HASH.

Para verificarmos se um usuário é autêntico no sistema, vamos aplicar a função MD5 sobre a senha fornecida e verificar se é igual ao HASH armazenado.

5.2 Estrutura de Pastas

Nosso sistema vai utilizar a seguinte estrutura de pastas para o site:



As pastas css, images e js irão conter os arquivos de css, imagens e Java script respectivamente. A pasta helpers conterá as classes de terceiros que vamos utilizar para desenvolver o sistema e a pasta system, todos os objetos e arquivos de configuração. A pasta system irá conter os arquivos de configuração do sistema e também as classes que irão compor

A pasta app irá conter os três elementos básicos do MVC: controladores, models e views.

Como sugere o próprio nome, helpers são para ajudar com as tarefas. Cada helper é um conjunto de funções relacionadas a uma determinada "categoria de tarefas" ou uma classe ou conjunto de classes que são utilizadas para alguma tarefa específica.

No nosso caso, vamos utilizar os helpers listados a seguir.

5.3 Classe Template Power

Quando acessamos sistemas via WEB, temos contato direta com a camada de Visão através de arquivos HTML , XML , ASP , PHP, etc. Graças a crescente ascensão deste padrão sugiram vários modelos que se propuseram a oferecer um facilitador na criação desta camada.

Dentre estes modelos, que chamamos de Templates, podemos destacar o Template Power e o Smarty. Vamos utilizar o Template Power para nos ajudar a implementar esse conceito do MVC. Caso deseje conhecer mais sobre o Template Power acesse: http://templatepower.codocad.com/ e para obter mais informações sobre o Smarty acesse: http://www.smarty.net/.

5.4 Classe para Upload de arquivos

Nosso sistema fará o upload de imagens para serem vinculadas aos produtos. Para isso, vamos precisar de uma classe que recebe esses arquivos, valide e certifique de que os mesmos são seguros para estarem em nosso site e que armazene os mesmos dentro de uma pasta específica.

Usaremos a classe a seguir para essa função:

Arquivo: helpers/class.Upload.inc.php

```php
<?php
/**
 * Classe para tratamento de Upload de arquivos
 *
 */
class Upload
{
	// propridades da classe
	public $Campo;
	public $Arquivo;
	public $PastaDestino;
	public $Sobrescrever;
	public $ehUpload;
	public $Erro = 0;
	// extensões aceitas por default para upload
	public $ExtValidas = array('jpeg', 'jpg', 'jpe', 'gif', 'png');

	/**
	 * Construtor da classe
	 * @param string $campo : nome do campo file que está enviando o arquivo
```

```
        * @param string $pastaDestino : pasta onde será guardado o arquivo
        * @param bool $sobrescrever : sobrescrever o arquivo caso ele já exista
        * @param array $validos : array com exensões aceitas para upload
        */
      public function Upload($campo, $pastaDestino, $sobrescrever, $validos="")
      {
            // recebe os parâmetros
            $this->Campo = $campo;
            $this->PastaDestino = $pastaDestino;
            $this->Sobrescrever = $sobrescrever;
            $this->ehUpload = false;
            if(is_array($validos) && count($validos)>0)
            {
                  $this->ExtValidas = $validos;
            }

            // trata o upload, verificando se o arquivo foi enviado
            if(isset($_FILES[$campo]))
            {
                  if($_FILES[$campo]['error']==UPLOAD_ERR_OK)
                  {
                        // não houve erro de envio, verifica arquivo
                        $this->Arquivo = $_FILES[$campo]['name'];
                        if($this->ehArquivoPermitido($this->Arquivo))
                        {
                              // a extensão do arquivo é valida
                              $arqDestino = $pastaDestino .'/'. $this-
>Arquivo;
                              if(!$sobrescrever)
                              {
                                    if(is_file($arqDestino))
                                    {
                                          // renomeia arquivo de destino
                                          $this->Arquivo = uniqid() .'_'.
$this->Arquivo;
                                          $arqDestino = $pastaDestino .'/'.
$this->Arquivo;
                                    }
                              }

      move_uploaded_file($_FILES[$campo]['tmp_name'],$arqDestino);
                        }
                  }else{
                        // o PHP nos informa que algum erro ocorreu
                        $this->Erro = $_FILES[$campo]['error'];
                  }
            }
      }

      // verifica se o arquivo tem extensã válida
      function ehArquivoPermitido($arquivo)
      {
            // separa as partes do nome do arquivo pelo ponto
            $partes = explode(".", $arquivo);
            // retorna o valor do último elemento do array gerado
```

```
            $extensao = end($partes);

            if( array_search($extensao, $this->ExtValidas))
            {
                  // arquivo ok
                  return true;
            }
            // arquivo não permitido
            return false;
      }
}
```

5.5 Classe BD

Essa classe será responsável por abstrair o acesso ao banco de dados. Em linhas gerais, ela centraliza todos os comandos de acesso ao banco de dados. Caso seja necessário mudar de SGBD a qualquer momento, só precisamos reescrever essa classe, sem alterar qualquer outro componente da nossa aplicação.

Arquivo: helpers/class.BD.inc.php

```
<?php


class BD

{

      public $Servidor;

      public $Usuario;

      public $Link;

      public $Banco;

      public $Charset;


      // para iniciar automaticamente meu objeto com a conex?o
aberta

      public function __construct($servidor, $usuario, $senha,
$banco, $charset )

      {

            $this->Abrir($servidor, $usuario, $senha, $banco,
$charset );

      }

      // para encerrar a conex?o ao banco

      public function __destruct()

      {
```

```
            $this->Fechar();
      }


      public function Abrir($servidor, $usuario, $senha, $banco,
$charset )
      {
            // conecta ao servidor
            $this->Link = mysql_connect($servidor, $usuario, $senha)
                  or die("Não foi possível conectar ao servidor
$servidor "
                        . " com o usuario $usuario e senha *****.<br
/>Erro: " . mysql_error());
            // seleciona o banco de dados
            mysql_select_db($banco, $this->Link)
                  or die("Não foi possível abrir o banco $banco no
servidor $servidor.<br />"
                        . mysql_error());
            // acerta o cjto de caracteres da conex?o: utf8 (UTF-
8),
            // latin1 (ISO-5589-1/europeu ocidental)
            mysql_set_charset($charset);

            $this->Servidor = $servidor;
            $this->Usuario = $usuario;
            $this->Banco = $banco;
            $this->Charset = $charset;


      }


      public function Fechar()
      {
            if($this->Link!=null)
            {
                  @mysql_close($this->Link);
                  $this->Link = null;
            }
```

```
	}
}

class Consulta
{
	public $resultado;
	public $BD;

	public function __construct( $sql, $bd)
	{

		if(!$bd->Link)
		{
			$bd->Abrir();
			die("entrou");
		}
		$this->resultado = mysql_query($sql, $bd->Link);
		if($this->resultado)
		{
			// comando executado com sucesso
		}else{
			die( "Erro ao executar comando $sql<br />"
				.mysql_error());
		}
		$this->BD = $bd;
	}
	// retorna a quantidade de registros resultante do
	// comando SQL
	public function Linhas()
	{
		return mysql_num_rows($this->resultado);
	}
	// retorna um valor de um campo espec?fico
	// ex: $obj->Campo(0, 'titulo')
	public function Campo($linha, $campo)
```

```
	{
		return mysql_result($this->resultado, $linha, $campo);

	}

	public function InsertId()
	{
		return mysql_insert_id($this->BD->Link);
	}

}
```

Unidade 6
Front Controller

Vamos precisar de um controlador principal que irá inicializar a nossa aplicação. Ele é o Front Controller, que irá centralizar todos os pedidos e requisições feitos a aplicação. Todos os links deverão ser construídos apontando para ele.

Ele será responsável por identificar qual controlador deverá ser criado e qual o método que deverá ser acionado para cada situação. Para isso ele deverá receber, quando necessário dois parâmetros:

- controlador : indica o controlador a ser acionado, quando não for informado o controlador default será o controlador 'index' que exibirá a página inicial da aplicação. Ex: índex.php?controlador=produto
- método : o método do controlador a ser acionado, quando não for informado será utilizado o valor default 'index'. Ex.: índex.php?controlador=produto&método=listar.

Ele deverá estar localizado na raiz do site.

Arquivo: índex.php

```
<?php
// classes base
require_once 'system/class.Controller.inc.php';
require_once 'system/class.Model.inc.php';
require_once 'system/class.View.inc.php';

// models
require_once 'app/model/class.LoginModel.inc.php';
require_once 'app/model/class.ProdutoModel.inc.php';
require_once 'app/model/class.FabricanteModel.inc.php';
require_once 'app/model/class.LinhaModel.inc.php';

// controllers
require_once("app/controller/class.IndexController.inc.php");
require_once("app/controller/class.ProdutoController.inc.php");
require_once("app/controller/class.LoginController.inc.php");

// classes obrigatórias
```

```php
require_once 'helpers/class.BD.inc.php';
require_once 'helpers/class.TemplatePower.inc.php';
require_once 'helpers/class.Upload.inc.php';

// configuração sistema
require_once 'system/config.inc.php';

// controlador e método default
$classe = "IndexController";
$metodo = "Index";
// ou aqueles que forem informados
if(isset($_REQUEST['controlador']))
{
	$controlador = $_REQUEST['controlador'];
	$classe = $controlador . 'Controller';

	if(isset($_REQUEST['metodo']))
	{
		$metodo = $_REQUEST['metodo'];
	}

}

// Executa o controlador
$controlador = new $classe();
$controlador->$metodo();
```

Unidade 7
Camada de Apresentação

A camada de visualização (VIEW) ou apresentação é o elemento responsável pela interação direta com o usuário final. Neste elemento não há a necessidade do conhecimento de como a informação foi obtida ou onde ela foi obtida, mas apenas um foco na exibição da mesma.

Nesta camada as interações com o usuário são disparadas e a comunicação com o modelo ocorre através do monitoramento destes eventos pela camada de controle.

Quando acessamos sistemas via WEB, temos contato direta com esta camada através de arquivos HTML , XML , ASP , PHP, etc. Graças a crescente ascensão deste padrão sugiram vários modelos que se propuseram a oferecer um facilitador na criação destas camadas.

Dentre estes modelos, que chamamos de Templates, podemos destacar o Template Power e o Smarty. Vamos utilizar o Template Power por sua simplicidade que será um facilitador para compreendermos melhor estes conceitos do MVC. Caso deseje conhecer mais sobre o Template Power acesse: http://templatepower.codocad.com/ e para obter mais informações sobre o Smarty acesse: http://www.smarty.net/

Criaremos aqui uma camada que permita visualizar uma lista de produtos cadastrados em um banco de dados. Não vamos nos deter na sintaxe e semântica da linguagem utilizada em Template Power.

Vamos utilizar os seguintes arquivos:

Arquivo: app/view/master.htm

```
<html>
<head>
     <title>Gerenciamento de Conteúdo</title>
     <link href="css/estilo.css" type="text/css" rel="stylesheet"
/>
     <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
</head>
<body>
     <div id="cabecalho">
          <h1> Gerenciamento de Conteúdo </h1>
     </div>
     <div id="menu">
          <ul>
               <!-- START BLOCK : menu-logado -->
```

```
                <li><a
href="index.php?controlador=produto&metodo=listar">Produtos</a></l
i>
                <li><a href="">Fabricantes</a></li>
                <li><a href="">Linhas</a></li>
                <li><a
href="index.php?controlador=login&metodo=sair">Sair</a></li>
                <!-- END BLOCK : menu-logado -->

                <!-- START BLOCK : menu-login -->
                <li><a
href="index.php?controlador=login">Login</a></li>
                <!-- END BLOCK : menu-login -->

          </ul>
     </div>
     <div id="conteudo">
          <!-- INCLUDE BLOCK : conteudo -->
     </div>

     <!-- START BLOCK : mensagem -->
     <script type="text/javascript">
          alert( '{mensagem}' );
     </script>
     <!-- END BLOCK : mensagem -->

</body>
</html>
```

Observe que neste código não há nenhuma "regra" de validação ou chamada a algum procedimento. Temos apenas as linhas de comando que servem como demarcadores para o Template Power:

```
<-- START BLOCK : xx -->
<-- END BLOCK : xx -->
```

Arquivo: app/view/login_form.htm

```
<br />
<br />
<br />
<form action="index.php" method="post">
<input type="hidden" name="controlador" value="login" />
<input type="hidden" name="metodo" value="validar" />
<table align="center">
	<tr>
		<th colspan="2">Acesso Restrito</th>
	</tr>
	<tr>
		<td>Usuario</td>
		<td><input type="text" name="usuario" /></td>
	</tr>
	<tr>
		<td>Senha</td>
		<td><input type="password" name="senha" /></td>
	</tr>
	<tr>
		<td></td>
		<td><input type="submit" value="entrar" /></td>
	</tr>
</table>
</form>
```

Arquivo: app/view/bemvindo.htm

```
<br />
<br />
<br />
<br />
<div id="bemvindo">
	<h2>Bem vindo ao administrador de notícias!</h2>
	<p>
	<small>COPYRIGHT 2011 &copy; Alfamídia</small>
```

```
	</p>
</div>
```

Arquivo: app/view/produto_editar.htm

```
<form name="frm" method="post" action="index.php"
enctype="multipart/form-data">
<input type="hidden" name="controlador" value="produto" />
<input type="hidden" name="metodo" value="salvar" />
<input type="hidden" name="id" value="{id}" />
<table>
	<tr>
		<td>Nome</td>
		<td>
		<input type="text" name="nome" size="50" value="{nome}"
/>
		</td>
	</tr>
	<tr>
		<td>Descrição</td>
		<td>
			<textarea name="descricao" rows="2"
cols="50">{descricao}</textarea>
		</td>
	</tr>
	<tr>
		<td>Linha</td>
		<td><select name="linha">
			<option value="">selecione uma linha</option>
			<!-- START BLOCK : item-linha -->
			<option value="{id}" {sel}>{nome}</option>
			<!-- END BLOCK : item-linha -->
		</select>
		</td>
	</tr>
	<tr>
		<td>Fabricante</td>
```

```
            <td><select name="fabricante">
                  <option value="">selecione um fabricante</option>
                  <!-- START BLOCK : item-fabricante -->
                  <option value="{id}" {sel}>{nome}</option>
                  <!-- END BLOCK : item-fabricante -->
            </select>
            </td>
      </tr>

      <tr>
            <td>Preço</td>
            <td>
            <input type="text" name="preco" value="{preco}" />
            </td>
      </tr>
      <tr>
            <td>Peso</td>
            <td>
            <input type="text" name="peso" value="{peso}" />
            </td>
      </tr>
      <tr>
            <td>Estoque</td>
            <td>
            <input type="text" name="qtd_estoque"
value="{qtd_estoque}" />
            </td>
      </tr>
      <tr>
            <td>Ativo</td>
            <td><input type="checkbox" name="ativo" {ativo} /></td>
      </tr>
      <tr>
            <td>Destaque</td>
            <td><input type="checkbox" name="destaque" {destaque}
/></td>
```

```
	</tr>
	<tr>
		<td>Imagem</td>
		<td>
		<input type="file" name="imagem" />
		</td>
	</tr>
	<!-- START BLOCK : imagem -->
	<tr>
		<td>Imagem</td>
		<td>
			<img src="images/{imagem}" height="80"
alt="imagem" />
		</td>
	</tr>
	<!-- END BLOCK : imagem -->

	<tr><td></td><td><input type="submit" name="bt_salvar"
value="salvar" /></td></tr>

</form>
```

Arquivo: app/view/produto_listar.htm

```
<a href="index.php?controlador=produto&metodo=editar">Novo
produto</a><br />
Total de produtos cadastrados: {total}<br>
<table width="100%">
	<tr>
		<th width="60%">Descrição</th>
		<th>Preço</th>
		<th>Estoque</th>
		<th>Ações</th>
	</tr>
	<!-- START BLOCK : item -->
	<tr>
```

```
            <td>{nome}</td>
            <td>{preco}</td>
            <td>{qtd_estoque}</td>
            <td>
                  <a
href="index.php?controlador=produto&metodo=editar&id={id}">editar<
/a> ::
                  <a
href="index.php?controlador=produto&metodo=excluir&id={id}">exclui
r</a>
            </td>
      </tr>
      <!-- END BLOCK : item -->
      <!-- START BLOCK : semregistros -->
      <tr><td colspan="4" align="center">nenhum produto
cadastrado</td></tr>
      <!-- END BLOCK : semregistros -->
      <tr>
            <th></th>
            <th></th>
            <th></th>
            <th></th>
      </tr>
</table>
```

Arquivo: css/estilo.css

```
* { font-family: "Georgia", serif; }

body { margin: 0px; }

a {
      color: #00f;
      text-decoration: none;
}
th {
      color: white;
      background-color: black;
```

```
      padding: 3px;
}
#cabecalho { color: white; background-color: #006666; padding:
25px;  }
#cabecalho h1 { margin: 0px; }
#menu { border-bottom: solid 1px black; border-top: solid 1px
black; height: 30px; }
#menu ul { margin: 0px; padding: 0px; }
#menu li { list-style: none; float: left; padding: 5px 10px 2px
2px;}
#menu a:hover{background-color: #ccc;};
#conteudo{}

#bemvindo { border: solid 1px black; padding: 30px; width: 300px;
      margin-left: 30%; text-align: center;
}
#bemvindo h2 { text-align: center; font-size: 16px; }
```

Unidade 8
Camada de Modelo

O nome “Camada lógica da aplicação” ou Modelo (Model) deixa bem claro que tratasse do núcleo do sistema. É neste nível que armazenamos todas as ações que serão realizadas pela aplicação.

Além disso, aqui também fazemos o acesso e manipulação dos dados contidos na base de dados (persistência). Estas ações que a camada pode realizar são acionadas através da camada de controle, portanto, aqui apenas estará presente a lógica do negócio.

Com esta separação da regra de negócio das demais camadas, podemos pensar em desenvolver o sistema independente do módulo de visualização. Isto nos permite que o desenvolvimento ocorra de forma paralela, portanto mais rápida, e ainda nos dá a possibilidade de ajustar o layout sem a necessidade de alterar a regra de negócio da aplicação.

Esta característica de separação do negócio das demais camadas é o que chamamos de encapsulamento de dados, onde temos o comportamento independente da apresentação.

O primeiro arquivo que vamos criar é a classe mãe de todos os modelos de nossa aplicação. Todos os modelos criados deverão extender essa classe.

Todos os modelos da nossa aplicação deverão implementar um conjunto mínimo de funcionalidades:

- CarregarId(id) : carrega um registro do banco a partir do Id informado.
- CarregarTodas() : carrega todos os registros do banco.
- Ler(n) : lê o n-ésimo registro carregado.
- TotalLidas : total de linhas carregadas no modelo.

Opcionalmente serão implementados os métodos:

- Salvar() : salva o modelo no banco.
- Excluir() : exclui o modelo do banco.

Arquivo: system/class.Model.inc.php

```
<?php

abstract class Model
{
	protected $bd;
```

```
	public function __construct($id=0)
	{
		$this->bd = new BD(BD_SERVIDOR, BD_USUARIO, BD_SENHA, BD_BANCO,
BD_CHARSET);

		if($id>0)
		{
			$this->CarregarId($id);
		}
	}

	public function __destruct()
	{
		$this->bd->Fechar();
	}

	abstract function CarregarId($id);

}
```

Arquivo: app/model/class.FabricanteModel.inc.php

```
<?php
class FabricanteModel extends Model
{

	public $Id;
	public $Nome;
	public $TotalLidas;
	private $c;

	public function Ler($n)
	{
```

```
            // limpa
            $this->Id = null;
            $this->Nome = null;
            if($n<$this->c->Linhas())
            {
                  $this->Id = $this->c->Campo($n, 'id');
                  $this->Nome = $this->c->Campo($n, 'nome');
            }

      }

      public function CarregarId($id)
      {
            $id = (int)$id;
            if($id>0)
            {
                  $sql = "SELECT * FROM fabricante WHERE id=$id";
                  $this->c = new Consulta($sql, $this->bd);
                  if($this->c->Linhas()>0)
                  {
                        $this->TotalLidas=1;
                        $this->Ler(0);
                  }
            }
      }

      public function CarregarTodas()
      {
            $sql = "SELECT * FROM fabricante ORDER BY nome";
            $this->c = new Consulta($sql, $this->bd);
            $this->TotalLidas = $this->c->Linhas();
            if($this->c->Linhas()>0)
            {
                  $this->c->Linhas();
                  // lê o primeiro registro
```

```
                $this->Ler(0);
            }
        }
}
```

Arquivo: app/model/class.LinhaModel.inc.php

```
<?php
class LinhaModel extends Model
{

    public $Id;
    public $Nome;
    public $TotalLidas;
    private $c;

    public function Ler($n)
    {
        // limpa
        $this->Id = null;
        $this->Nome = null;
        if($n<$this->c->Linhas())
        {
            $this->Id = $this->c->Campo($n, 'id');
            $this->Nome = $this->c->Campo($n, 'nome');
        }

    }

    public function CarregarId($id)
    {
        $id = (int)$id;
        if($id>0)
```

```
            {
                  $sql = "SELECT * FROM linha WHERE id=$id";
                  $this->c = new Consulta($sql, $this->bd);
                  if($this->c->Linhas()>0)
                  {
                        $this->TotalLidas=1;
                        $this->Ler(0);
                  }
            }
      }

      public function CarregarTodas()
      {
            $sql = "SELECT * FROM linha ORDER BY nome";
            $this->c = new Consulta($sql, $this->bd);
            $this->TotalLidas = $this->c->Linhas();
            if($this->c->Linhas()>0)
            {
                  $this->c->Linhas();
                  // lê o primeiro registro
                  $this->Ler(0);
            }
      }
}
```

Arquivo: app/model/class.LoginModel.inc.php

```
<?php

class LoginModel extends Model
{
      public function Validar($usu, $sen)
      {
            $usu = addslashes($usu);
            $sen = md5($sen);
```

```
            $sql = "SELECT * FROM usuario WHERE login='$usu' AND
senha='$sen'";
            $c = new Consulta($sql, $this->bd);
            if($c->Linhas()>0)
            {
                  return true;
            }
            return false;
      }

      public function CarregarId($id)
      {
      }
}
```

Arquivo: app/model/class.ProdutoModel.inc.php

```
<?php
class ProdutoModel extends Model
{
      public $Id;
      public $Nome;
      public $Descricao;
      public $Preco;
      public $Peso;
      public $Destaque;
      public $QtdEstoque;
      public $Fabricante;
      public $Linha;
      public $Imagem;
      public $Ativo;

      public $TotalLidas=0;
      private $c;

      public function Ler($n)
```

```
    {
        if($n>$this->TotalLidas)
        {
            return false;
        }

        $this->Id = $this->c->Campo($n, 'id');
        $this->Nome = $this->c->Campo($n, 'nome');
        $this->Descricao = $this->c->Campo($n, 'descricao');
        $this->Preco = $this->c->Campo($n, 'preco');
        $this->Peso= $this->c->Campo($n, 'peso');
        $this->Destaque = $this->c->Campo($n, 'destaque');
        $this->QtdEstoque = $this->c->Campo($n, 'qtd_estoque');
        $this->Imagem = $this->c->Campo($n, 'imagem');
        $this->Ativo = $this->c->Campo($n, 'ativo');

        $fabId = $this->c->Campo($n, 'fabricante_id');
        $this->Fabricante = new FabricanteModel($fabId);

        $linId = $this->c->Campo($n, 'fabricante_id');
        $this->Linha = new LinhaModel($linId);

        return true;
    }

    /**
     * Carrega um registro.
     * @param int $id Id a ser carregado.
     */
    public function CarregarId($id)
    {
        $id = (int)$id;
        if($id>0)
        {
            $sql = "SELECT * FROM produto WHERE id=$id";
```

```
                $this->c = new Consulta($sql, $this->bd);
                if($this->c->Linhas()>0)
                {
                      $this->TotalLidas=1;
                      $this->Ler(0);
                }
          }
     }

     public function CarregarTodas()
     {
          $sql = "SELECT * FROM produto ORDER BY nome";
          $this->c = new Consulta($sql, $this->bd);
          if($this->c->Linhas()>0)
          {
                // lê o primeiro registro
                $this->TotalLidas=$this->c->Linhas();
                $this->Ler(0);
          }
     }

     public function Salvar()
     {
          $nom = addslashes($this->Nome);
          $des = addslashes($this->Descricao);
          $pre = (float)$this->Preco;
          $pes = (float)$this->Peso;
          $est = (int)$this->QtdEstoque;
          $dtq = (int)$this->Destaque;
          $ati = (int)$this->Ativo;
          $img = addslashes($this->Imagem);
          $fab = (int)$this->Fabricante->Id;
          $lin = (int)$this->Linha->Id;
          if($this->Id>0)
          {
```

```
                $sql = "UPDATE produto SET "
                        . "nome='$nom' , "
                        . "descricao='$des' , "
                        . "fabricante_id=$fab , "
                        . "linha_id=$lin , "
                        . "preco=$pre , "
                        . "peso=$pes , "
                        . "qtd_estoque=$est , "
                        . "destaque=$dtq , "
                        . "ativo=$ati , "
                        . "imagem='$img' "
                        . "WHERE id=".$this->Id;
            }else{
                $sql = "INSERT INTO produto "
                        ."(nome, descricao, fabricante_id, linha_id,
preco, peso, qtd estoque, destaque, ativo, imagem) VALUES "
                        . "( '$nom' , '$des' , $fab, $lin, $pre,
$pes, $est, $dtq, $ati, '$img' )";
            }
            $c = new Consulta($sql, $this->bd);
        }

        public function Excluir()
        {
            $id = (int)$this->Id;
            if($id>0)
            {
                $sql = "DELETE FROM produto WHERE id=$id";
                $c = new Consulta($sql, $this->bd);
            }
        }

}
```

Unidade 9
Camada de Controle

Nesta camada fazemos o controle do fluxo da aplicação, determinando o que deve ser acessado e exibido. Ela serve como uma camada intermediária entre a camada de apresentação e a lógica.

Teremos aqui um arquivo chamado produtos.controller.php que acionará métodos e propriedades contidas nas classes da camada de modelo e por fim enviará uma solicitação de exibição destes dados para o arquivo de template da camada de visão.

Esta camada nos permite a modificação de forma simples e rápida a sequencia de exibições e manipulações que o sistema fará. Com isto ganhamos em tempo de implantação, mas principalmente no processo de manutenção do código.

O primeiro arquivo a ser criado será a classe mãe de todos os controladores. Todos os controladores do nosso sistema deverão estender a classe Controller.

Os métodos implementados irão variar de acordo com as ações que serão disponibilizadas para cada controlador. O único método obrigatório será o método Index(), que é o método default de cada controlador e será usado quando nenhum método for informado.

No nosso sistema, também faremos uso dos métodos comuns a todos controladores:

- InicializaView() : inicializa a template Power com a Master Page e o arquivo de include necessário.
- Exibe() : envia a página para o usuário.
- RestringeAcesso() : quando for acionado, esse método verificará se o usuário se logou, caso não tenha feito, negará o acesso à página e encaminhará para a página de login. Só será usado nas páginas em que isso for necessário.

Arquivo: system/class.Controller.inc.php

```
<?php
/**
 * Classe mãe para os Controladores.
 * Aqui vão as definições comuns a todos os controladores.
 * @author Alfamidia
 *
 */

class Controller {
```

```
	public $tpl;

	/**
	 * Inicializa a VIEW do Controlador
	 * @param string $master Nome da Master Page.
	 * @param string $include Nome do include block se houver.
	 */
	public function InicializaView($include="",
$master="master.htm") {
		$this->tpl = new TemplatePower ( "app/view/$master" );
		if ($include != "") {
			$this->tpl->assignInclude ( "conteudo",
"app/view/$include" );
		}
		$this->tpl->prepare();

		if (isset ( $_SESSION ['logado'] )) {
			$this->tpl->newBlock( 'menu-logado' );
		} else {
			$this->tpl->newBlock( 'menu-login' );
		}

		if (isset ( $_SESSION ['mensagem'] )) {
			$this->tpl->newBlock( 'mensagem' );
			$this->tpl->assign('mensagem', $_SESSION
['mensagem']);
			unset($_SESSION ['mensagem']);
		}

	}

	/**
	 * Responsável por exibir a VIEW
	 */
	public function Exibe() {
		$this->tpl->printToScreen ();
```

```
    }

    /**
     * Restringe o acesso, se o usuário não estiver logado
     * exibe a tela de login e suspende a execução do restante
     * do programa.
     */
    public function RestringeAcesso()
    {
        if(!isset($_SESSION['logado']))
        {
            $log = new LoginController();
            $log->Index();
            exit;
        }
    }

    /**
     * Converte um valor no formato '1.234,56' em FLOAT
     * @param string $valor
     */
    public function ConverteParaFloat($valor)
    {
        // supondo que a entrada seja 3.400,25
        // remove o ponto => 3400,25
        $nro = str_replace('.', '', $valor);
        // troca a virgula por ponto => 3400.25
        $nro = str_replace(',', '.', $nro);
        // converte para float
        $nro = (float)$valor;

        return $nro;
    }

    /**
```

```
        * Converte um float em uma string no formato '1.234,56'
        * @param float $valor Valor a ser convertido
        * @param int $casas decimais
        */
      public function FormataNumero($valor, $casas=2)
      {
            return number_format($valor, $casas, ',', '.');
      }
}
```

Arquivo: app/controller/class.IndexController.inc.php

```
<?php

class IndexController extends Controller
{

      public function Index()
      {
            $this->InicializaView();
            $this->Exibe();

      }
}
```

Arquivo: app/controller/class.LoginController.inc.php

```
<?php
class LoginController extends Controller
{
      public function Index()
      {
            $this->InicializaView('login_form.htm');
            $this->Exibe();
      }

      public function Validar()
```

```
	{
		if(isset($_POST['usuario']) && isset($_POST['senha']))
		{
			$login = new LoginModel();
			if($login->Validar($_POST['usuario'],
$_POST['senha']))
			{
				$_SESSION['logado'] = $_POST['usuario'];
				header("Location:
index.php?controlador=login&metodo=bemvindo");
				return;
			}
		}
		$_SESSION['mensagem']= "Usuário ou senha inválida!";
		header("Location: index.php?controlador=login");
	}

	// exemplo de método/página com acesso restrito usuários
	// logados no sistema
	public function Bemvindo()
	{
		// isso garante que se o usuário não estiver logado
		// a página não será exibida
		if( ! isset($_SESSION['logado']))
		{
			header("Location: index.php?controller=login");
		}

		$this->InicializaView('bemvindo.htm');
		$this->Exibe();
	}

	public function Sair()
	{
		// destroi a sessao e volta a tela de login
		session_destroy();
```

```
            unset($_SESSION['logado']);
            header("Location: index.php?controlador=login");
      }


}
```

Arquivo: app/controller/class.ProdutoController.inc.php

```
<?php

class ProdutoController extends Controller
{

      public function Index()
      {
            $this->Listar();
      }

      public function Listar()
      {
            // somente para usuários logados
            $this->RestringeAcesso();

            $this->InicializaView("produto_listar.htm");

            $pro = new ProdutoModel();
            $pro->CarregarTodas();

            $this->tpl->assign("total", $pro->TotalLidas);
            for($i=0; $i<$pro->TotalLidas; $i++)
            {
                  $pro->Ler($i);
                  $this->tpl->newBlock("item");
                  $this->tpl->assign("id", $pro->Id);
                  $this->tpl->assign("nome", $pro->Nome);
```

```
                $this->tpl->assign("preco", $pro->Preco);
                $this->tpl->assign("qtd_estoque", $pro-
>QtdEstoque);
            }

            if($pro->TotalLidas==0)
            {
                $this->tpl->newBlock("semregistros");
            }
            $this->Exibe();
        }

        public function Editar()
        {
            // somente para usuários logados
            $this->RestringeAcesso();


            $this->InicializaView("produto_editar.htm");
            // inicializa
            $fabId = 0;
            $linId = 0;

            // se for edição, carrega produto salvo
            if(isset($_GET['id']))
            {
                $proId = (int)$_GET['id'];
                $pro = new ProdutoModel($proId);
                $this->tpl->gotoBlock('_ROOT');
                $this->tpl->assign('id', $proId);
                $this->tpl->assign('nome', $pro->Nome);
                $this->tpl->assign('descricao', $pro->Descricao);
                $this->tpl->assign('preco', $pro->Preco);
                $this->tpl->assign('peso', $pro->Peso);
                $this->tpl->assign('qtd_estoque', $pro-
>QtdEstoque);
```

```
                $this->tpl->assign('ativo', ($pro->Ativo==1 ?
'checked="checked"' : "" ));
                $this->tpl->assign('destaque', ($pro->Destaque==1
? 'checked="checked"' : "" ));
                $fabId = $pro->Fabricante->Id;
                $linId = $pro->Linha->Id;

                if($pro->Imagem!="")
                {
                      $this->tpl->newBlock('imagem');
                      $this->tpl->assign('imagem', $pro->Imagem);
                }
          }

          // preenche selects
          $fab = new FabricanteModel();
          $fab->CarregarTodas();
          for($i=0; $i<$fab->TotalLidas; $i++)
          {
                $fab->Ler($i);
                $this->tpl->newBlock('item-fabricante');
                $this->tpl->assign('id', $fab->Id);
                $this->tpl->assign('nome', $fab->Nome);
                // se for edição, marca o item selecionado
                if($fab->Id==$fabId) $this->tpl->assign('sel',
'selected');
          }

          // preenche selects
          $lin = new LinhaModel();
          $lin->CarregarTodas();
          for($i=0; $i<$lin->TotalLidas; $i++)
          {
                $lin->Ler($i);
                $this->tpl->newBlock('item-linha');
                $this->tpl->assign('id', $lin->Id);
```

```
                $this->tpl->assign('nome', $lin->Nome);
                // se for edição, marca o item selecionado
                if($lin->Id==$linId) $this->tpl->assign('sel',
'selected');
            }

            $this->Exibe();
        }

        public function Salvar()
        {
            // somente para usuários logados
            $this->RestringeAcesso();

            if(isset($_POST['bt_salvar']))
            {
                $id = (int)$_POST['id'];

                $mod = new ProdutoModel($id);
                $mod->Nome = $_POST['nome'];
                $mod->Descricao = $_POST['descricao'];
                $mod->Fabricante = new
FabricanteModel((int)$_POST['fabricante']);
                $mod->Linha = new
LinhaModel((int)$_POST['linha']);
                $mod->QtdEstoque = $this-
>ConverteParaFloat($_POST['qtd_estoque']);
                $mod->Peso = $this-
>ConverteParaFloat($_POST['peso']);
                $mod->Preco = $this-
>ConverteParaFloat($_POST['preco']);
                $mod->Ativo = (isset($_POST['ativo']) ? 1 : 0);
                $mod->Destaque = (isset($_POST['destaque']) ? 1 :
0);

                $up = new Upload('imagem', 'images', false);
                if($up->ehUpload)
```

```
                {
                        $mod->Imagem = $up->Arquivo;
                }

                $mod->Salvar();
        }
        $this->Listar();
    }

    public function Excluir()
    {
        // somente para usuários logados
        $this->RestringeAcesso();

        if(isset($_GET['id']))
        {
            $id = (int)$_GET['id'];
            $mod = new ProdutoModel($id);
            $mod->Excluir();

            $this->Listar();
        }

    }
}
```