Capítulo 9

Desenvolvimento de Gerenciador de Conteúdo com Tecnologia Web 2.0

Cintia Carvalho Oliveira, Daniele Carvalho Oliveira, Cleber Ferreira Oliveira, Renan Gonçalves Cattelan, João Nunes de Souza

Abstract

This paper introduces the concepts and practices related to the development of content management system CMS considering interactive features. We present object-oriented technologies PHP, with MySQL database and Ajax with jQuery. This work consists of theoretical exposition and practical exercises, providing a comprehensive tool for developing Web 2.0 applications.

Resumo

Este trabalho tem como objetivo apresentar os conceitos e práticas relacionadas ao desenvolvimento de gerenciadores de conteúdo (CMS – Content Management System) com recursos interativos. Para tal, são apresentadas as tecnologias PHP orientada a objetos, com banco de dados MySQL e Ajax com jQuery. São apresentados, portanto, os conceitos fundamentais para o desenvolvimento de aplicativos Web 2.0.

9.1. Introdução

O objetivo deste trabalho é apresentar, de forma resumida, tecnologias que apóiam a criação de um CMS (*Content Management System* – Sistema Gerenciador de Conteúdo). Para atender tal objetivo, será criado um CMS, "simples", que consiste de um portal de receitas, no qual cada receita possui uma ferramenta de chat on-line para a participação dos leitores.

9.2. Fundamentos de PHP Orientado a Objetos com o Design Pattern MVC

Para desenvolver um CMS, será utilizada a linguagem PHP (PHP: *Hypertext Preprocessor)*. O PHP é uma linguagem de programação adequada para gerar conteúdo dinâmico na Web. Ela é uma linguagem de scripts, que é embutida em documentos HTML (*HyperText Markup Language*).

Além disso, a linguagem PHP é interpretada por servidores web, como o Apache, com módulo processador de PHP. Dessa forma, a maior parte das tarefas executadas por um programa PHP é invisível ao usuário final. O código da linguagem PHP é interpretado no servidor, gerando uma página HTML, que então é enviada ao usuário.

Finalmente, as razões da escolha da linguagem PHP são, fundamentalmente, as seguintes: Ela é uma linguagem open-source, que tem suporte a praticamente todos os bancos de dados existentes no mercado. Qualquer algoritmo desenvolvido utilizando CGI *(Common Gateway Interface)* pode, também, ser desenvolvidos utilizando PHP. Alguns exemplos são: coleta de dados de formulários, geração de páginas dinamicamente e envio e recebimento de *cookies*. Além disso, PHP é uma linguagem que suporta protocolos como: IMAP, SNMP, NNTP e POP3

Introdução ao PHP orientado a objetos

A orientação a objetos é um paradigma de programação diferente das linguagens estruturadas, onde o conjunto de procedimentos e variáveis nem sempre são agrupados de acordo com o contexto. O paradigma orientado a objetos lida com objetos, que são estruturas representativas de artefatos de nosso dia-a-dia.

Apesar de não ser o foco principal deste trabalho, alguns fundamentos da conceituação de orientação a objetos são apresentados a seguir. Isso, porque tais conceitos serão referenciados ao longo deste trabalho.

Para compreender a orientação a objetos, é fundamental entender o conceito de classes e objetos. Considere, por exemplo, alguns objetos como uma mesa, um computador ou um cachorro. Todos estes objetos compartilham duas características: propriedade e comportamento. O cachorro, por exemplo, tem como propriedades a sua cor, raça, e tamanho. Como comportamento ele come, dorme, late, etc. Objetos, então, são entidades (concretas ou conceituais) do mundo real que tem uma identidade. Ou seja, cada objeto é único, mesmo que outros objetos tenham as mesmas características. Todas as pessoas, por exemplo, possuem características iguais, mas cada uma tem uma identidade. Cada pessoa é um objeto diferente das demais, pois possui nome, CPF, endereço e seu comportamento pode diferir do comportamento das outras. A Figura 1 apresenta exemplos de objetos. O aluno de nome Fernando Soares é um objeto, assim como a professora de nome Regina Campos.

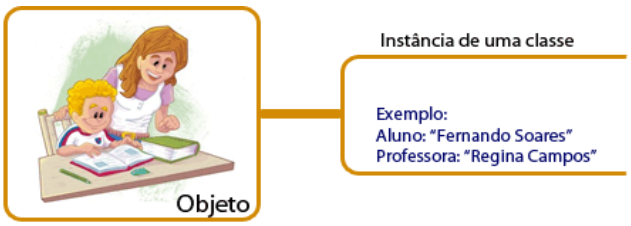

Figura 1 – Exemplo de Objetos

Portanto, existem muitos objetos similares, que possuem características e comportamento semelhantes. Como as pessoas, de modo geral, possuem comportamentos e propriedades semelhantes, é possível definir **Pessoas** como uma classe, e **Fernando Soares** uma instância pertencente à classe **Pessoas.** Nesse contexto,

uma instância é uma ocorrência particular de um objeto de uma classe. A instância possui um estado particular, independente de outras instâncias da mesma classe.

Uma classe é uma estrutura estática que define um tipo de dados. A classe pode conter atributos (variáveis, em analogia à programação estruturada) e funções (métodos) para manipular os atributos. O objeto instanciado contém exatamente a mesma estrutura e as propriedades de sua classe pai. No entanto, a sua estrutura é dinâmica, seus atributos podem mudar de valor durante a execução do programa. Ou seja, a classe é um molde, de onde os objetos são gerados. A Figura 2 apresenta exemplos de classes: a classe **Alunos** e a classe **Professores**.

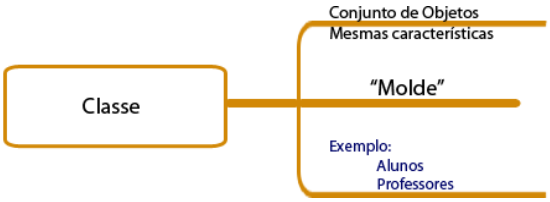

Figura 2 – Exemplos de classes

A Figura 3 mostra a classe **Funcionário,** implementada em PHP.

```php
<?php
class Funcionario {
    var $nome, $salario;

    function Funcionario($nome, $salario){
        $this->nome = $nome;
        $this->salario = $salario;
    }

    function aumentoSalario($valor){
        $this->salario += $valor;
    }
}
?>
```

Figura 3 – Classe Funcionário

E a Tabela 1 descreve os detalhes do código da Figura 3.

Tabela 1 - Código comentado

Linha(s)	Comentário
1 e 14	Abertura e fechamento do código PHP
2	Definição da Classe **Funcionário**. A palavra-chave `class` indica a declaração de classe
3	`var` declara os atributos da classe, a classe **Funcionário** possui dois atributos nome e salário
5	Método construtor da classe **Funcionário** deve possuir o mesmo nome da classe. Este método é executado sempre que uma instância da classe for criada.
6, 7 e 11	Quando a classe referencia seus próprios métodos e atributos, deve-se utilizar a palavra-chave `$this`. O operador `->` faz a referência ao método e atributo.

A Figura 4 apresenta, em PHP, como utilizar a classe criada na Figura 3, instanciando um objeto.

```php
<?php
    include "Funcionario.php";

    $func = new Funcionario("Pedro Cabral", 1200);
    echo("O funcionário ". $func->nome . "recebe R$". $func->salario);
    $func->aumentoSalario(200);
    echo ("Novo salário: " . $func->salario);
?>
```

Figura 4 – Instanciando um objeto da Classe Funcionário

A Tabela 2 comenta o código da Figura 4. O Resultado da execução do código da Figura 4 é:

O funcionário Pedro Cabral recebe R$1200
Novo salário R$1400

Tabela 2: Código comentado

Linha(s)	Comentário
2	A definição da classe deve estar disponível no script ou página PHP que utiliza a classe (comandos `include` ou `require` importam a classe para o script que fará uso dele)
4	Um objeto da classe deve ser instanciado pelo operador `new`.
6	Chamada ao método `aumentoSalario` do objeto `$func`.

Além disso, é possível criar constantes de classe e propriedades estáticas, que armazenam valores comuns a todos os objetos da mesma classe. As constantes são valores fixos, ou seja, não podem ser alterados. E as propriedades estáticas são atributos dinâmicos como as propriedades de um objeto, mas estão relacionados à classe. Esses atributos são acessados externamente pela sintaxe a seguir: `NomeDaClasse::NomeDaConstOuProp`, E dentro da classe, eles são referenciados da seguinte forma: `self::NomeDaConstOuProp`.

Também é possível criar métodos de classe (métodos estáticos), que podem ser chamados diretamente da classe, não sendo necessária a criação de objetos. Esses métodos não devem referenciar propriedades internas pelo operador `$this`. Eles podem apenas referenciar outros métodos estáticos ou propriedades estáticas. Os métodos de classe são referenciados da seguinte forma: `NomeDaClasse::NomeDoMétodo`. A Figura 5 apresenta um exemplo de um método estático. A Tabela 3 contém o comentário do código da Figura 5.

```php
<?php
class Funcionario {
    /* ... */

    static function listarFunc(){
        $bd = new BD();
        $funcs = array();
        $bd->consultar("SELECT * FROM tabela_funcionarios");

        while($linha = mysql_fetch_assoc($bd->resultadoSQL)){
            $funcs['nome_func']['salario_func'];
        }

        return $funcs;
    }
    /* ... */
}
?>
```

Figura 5 – Exemplo de Método Estático

Tabela 3: Código comentado

Linha(s)	Comentário
5	A definição do método estático deve possuir a palavra-chave `static`
6	A classe **BD** é responsável por fazer a conexão com o Banco de Dados e consultas
8	Método consultar da classe **BD** faz a pesquisa do SQL
10	Na variável da classe **BD** `resultadoSQL` está o resultado da consulta da linha 8.

A Figura 6 exemplifica o uso do método estático. A Tabela 4 o comentário do código.

```
1  <?php
2      include_once "funcionario.php";
3
4      $lista = Funcionario:listaFunc();
5
6      foreach($lista as $nome => $salario){
7          echo "O funcionário " . $nome . "recebe R$" . $salario . "<br />";
8      }
9  ?>
```

Figura 6 – Uso do Método Estático

Tabela 4: Código comentado

Linha(s)	Comentário
4	A chamada ao método estático é feita utilizando o operador `::`
6 a 8	O resultado da chamada do método estático é percorrido e impresso na tela

As classes são responsáveis por um assunto específico e possuem "responsabilidades" sobre o seu conteúdo. A classe deve proteger e gerenciar o seu conteúdo através de mecanismos como o encapsulamento, que provê a proteção de acesso ao conteúdo de um objeto. Dessa forma, como é responsabilidade dos métodos da classe tratar seus atributos, qualquer acesso a eles deve ser feito pelos métodos da classe. Assim, as propriedades da classe nunca são acessadas, diretamente, de fora dela. Tal propriedade é, também, chamada de ocultamento de informações. Nesse contexto, o uso do objeto é feito apenas pelo conhecimento da sua estrutura externa. E qualquer mudança da estrutura interna de um objeto não deve afetar o seu acesso externo.

A orientação a objetos e o encapsulamento de classes permitem uma maior organização do sistema. Considerando que cada classe representa uma característica distinta do sistema, então é possível a reutilização de partes do código. E reusar o código possibilita maior agilidade na produção de novos softwares, pois tal fato evita duplicações e reescritas desnecessárias.

Outro recurso da orientação a objetos que, também, possibilita a reutilização de código é a herança. A herança é o compartilhamento de atributos e comportamentos entre as classes de uma mesma hierarquia. A classe **Pessoas**, por exemplo, pode ter atributos em comum com **Professores** e **Alunos**. Cada uma destas classes, que representam profissões, possui características distintas. O professor tem uma formação, ensina, aplica avaliações. E o aluno faz lições, presta atenção às aulas, faz provas. Mas, mesmo com estas diferenças, tais classes possuem semelhanças como: ter nome e endereço, andar, dormir, etc. A classe pai, **Pessoas**, e as classes filhas: **Professores,** e **Alunos**. Todas as características da classe pai serão herdadas pelas classes filhas, ou seja, os elementos da classe **Professores** têm nome, endereços, andam e dormem. Da mesma forma, os elementos de **Alunos**. A Figura 7 apresenta o uso da herança na criação da subclasse **Professor,** que herda as propriedades da classe **Funcionário.**

```php
<?php
include_once "Funcionario.php";

class Professor extends Funcionario{

    var $disciplina;

    function Professor($nome, $salario, $disciplina){
        $this->Funcionario($nome, $salario);
        $this->disciplina = $disciplina;
    }

    function alterarDisciplina($disciplina){
        $this->disciplina =  $disciplina;
    }
}
?>
```

Figura.7 – Subclasse Professor

O código apresentado na Figura 7 é comentado na Tabela 5.

Tabela 5: Código comentado

Linha(s)	Comentário
2	A definição da subclasse deve incluir a definição da superclasse; utilize o comando `include_once` (ou então `require_once`) para evitar problemas
4	`extends` informa que a classe `Professor` é uma subclasse de `Funcionario`
9	O construtor da superclasse deve ser chamado explicitamente pelo construtor da subclasse

Na Figura 8, é apresentado um código que exemplifica a utilização de uma subclasse. O resultado da execução do código da Figura 8 é:

O professor Regina Campos ministra a disciplina Engenharia de software e recebe R$2500
Novo salário R$2800
Nova disciplina: Banco de Dados

```php
<?php
    include_once "Funcionario.php";
    include_once "Professor.php";

    $prof = new Professor("Regina Campos",2500, "Engenharia de Software");
    echo("O professor ". $prof->nome . "ministra a disciplina" . $prof->
disciplina . " e recebe R$". $prof->salario ."<br>");
    $prof->aumentoSalario(300);
    echo ("Novo salário: " . $func->salario . "<br>");
    $prof->alterarDisciplina("Banco de Dados");
    echo ("Nova disciplina: " . $prof->salario);
?>
```

Figura 8 – Uso da Subclasse Professor

Na Tabela 6, o código da Figura 8 é comentado.

Tabela 6: Código comentado

Linha(s)	Comentário
2 e 3	A definição da superclasse e da subclasse devem estar disponíveis no script ou página PHP que utiliza a classe (`include_once` ou `require_once` para evitar problemas)
5	Um novo objeto `Professor` é criado, o método construtor da classe `Professor` é chamado com os parâmetros “Regina Campos”, 2500 e “Engenharia de Software”
6	Observe que o objeto `Professor` criado possui os atributos `nome` e `salário`, que estavam declarados na classe `Funcionário`
7	O objeto `Professor` criado também possui o método `aumentoSalario`

Como descrito nos exemplos anteriores, o uso de classes em PHP facilita a programação nos seguintes aspectos: permite utilizar uma sintaxe mais simples para operações complexas que requerem grande quantidade de argumentos; programas complexos são mais facilmente gerenciáveis e manuteníveis; fomenta a reutilização de código, por meio da criação de componentes genéricos; e permite a substituição de componentes de um sistema, através dos chamados plug-ins.

9.3. Design Pattern MVC (Model, View, Controller)

O MVC (*Model, View, Controller*) é um padrão de projeto que utiliza três camadas, cada uma responsável por uma função. São definidas as três camadas: Modelo, Visão e Controle, que são detalhadas mais adiante.

A Figura 9 apresenta um documento com programação PHP tradicional, onde o código PHP está misturado com marcações HTML. No exemplo, é executada uma consulta ao banco de dados em busca de uma lista dos visitantes cadastrados no portal e, então, o resultado é impresso. Nesse caso, se em outra página fosse necessário imprimir, novamente, uma lista de visitantes, seria necessário reescrever o trecho PHP/SQL. Tal fato torna difícil a manutenção do código e deve ser evitado.

```
1  <h1> Visitantes do Portal </h1>
2  <?php
3      $conexao = mysql_connect('dominio', 'usuario', 'senha');
4      mysql_select_db('banco_dados', $conexao);
5
6      $dados = mysql_query('SELECT nome FROM visitante');
7
8      echo "<ul>";
9      $i = 1;
10     while($usuario = mysql_fetch_array($dados)){
11         echo "<li> Visitante nº" . $i . ": " . $usuario['nome'] . "<//li>";
12         $i++;
13     }
14     echo "</ul>";
15 ?>
```

Figura 9 – Código PHP com HTML

A Tabela 7 apresenta o comentário do código da Figura 9.

Tabela 7: Código comentado da Figura 9.1.9

Linha(s)	Comentário
3 a 6	Conexão com banco de dados e consulta ao nome de todos os visitantes
10 a 12	Impressão dos nomes dos visitantes.

Para facilitar a manutenção e reaproveitamento do código, é necessário dividir o script. Nesse caso, é proposto um documento para a lógica de negócios e o outro para a apresentação. Então, cada arquivo se torna responsável por uma função.

A Figura 10(a) apresenta o arquivo *controle/lista_visitantes.php*, que estabelece a conexão com o banco de dados, consulta todos os nomes dos visitantes e preenche um arranjo (*array*) com os dados. Por outro lado, a Figura 10(b) apresenta o arquivo *visão/lista_visitantes.php*, no qual o arranjo preenchido pelo arquivo anterior é percorrido para a impressão. Nesse caso, esse arquivo se preocupa somente com a apresentação. O arranjo é o resultado da lógica de programação criado pelo arquivo *controle/lista_visitantes.php*.

Essa arquitetura é denominada ***View/Controller*** (*Visão/Controle*), na qual o arquivo *controle/lista_visitantes.php* é o controle e o arquivo *visão/lista_visitante.php* é a visão. A Figura 11 apresenta a arquitetura ***View/Controller*.**

```php
<?php
    $conexao = mysql_connect('dominio', 'usuario', 'senha');
    mysql_select_db('banco_dados', $conexao);

    $dados = mysql_query('SELECT nome FROM visitante');
    $visitante = array();

    while($usuario = mysql_fetch_array($dados)){
        $visitante[] = $usuario['nome'];
    }
?>
```

(a)

```php
<?php include_once "../controle/lista_visitantes.php"; ?>
<h1> Visitantes do Portal </h1>
<ul>
<?php
    foreach ($visitante as $i => $nome){
        echo "<li> Membro nº $i: $nome </li>";
    }
?>
</ul>
```

(b)

Figura 10 – Modelo *View/Control*

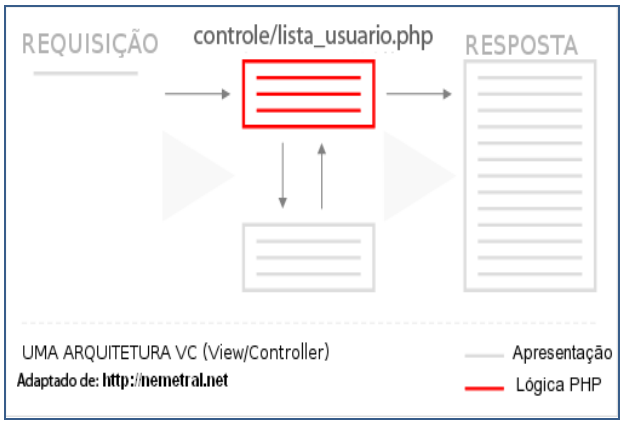

Figura 11 – Arquitetura *View/Controller*

O padrão de projeto **MVC** define a divisão das aplicações em 3 camadas, **Modelo**, **Visão** e **Controle**.

Modelo: O **Modelo** é a camada de acesso aos dados. Ela manipula os dados junto ao banco de dados, no qual consulta, adiciona, altera ou remove dados quando solicitado. A função `listaVisitantes()`, `por exemplo`, retorna um arranjo - como na Figura 10(a), nas linhas 8 a 10 - com todos os visitantes de um site, ou `cadastrarVisitante($nome, $email)` para adicionar um novo usuário no banco de dados.

Visão: A **Visão** é a camada responsável pela apresentação. É nessa camada que é gerada a saída. No exemplo, tal camada processa o arranjo que foi obtido na saída da função `listaVisitantes()`. Percorre o arranjo, imprimindo na tela o HTML final da página. Como resultado, são exibidos os nomes de todos os usuários cadastrados no portal, como indicado nas linhas 5 a 7 da Figura 10(b).

Controle: A camada de **Controle** se destina a coordenar todo o processo. Ela analisa as requisições e executa o modelo correspondente para obter os dados. No exemplo, ela chama a função `listaVisitantes()`, a qual retorna o arranjo com a lista dos visitantes. A camada de **Controle** passa o arranjo obtido para a camada de **Visão**, que o percorrerá imprimindo os nomes. Por fim o **Controle** exibe a saída.

A Figura 12 apresenta o esquema de funcionamento do MVC, no qual é explicitada a interação entre as camadas. Tal interação ocorre desde o momento que houve a requisição HTTP, por meio do navegador, até o retorno com a saída HTML.

Nesse sentido, a arquitetura MVC provê mais flexibilidade na construção da aplicação. Além disso, na arquitetura MVC, o acesso aos dados é de responsabilidade da camada Modelo. Dessa forma, a implementação de uma função, como, por exemplo, `listaVisitantes()`, que pertence à camada **Modelo**, executa uma consulta em banco de dados ou arquivos XML. Nesse contexto, tal função é solicitada pela camada **Controle**, que faz a chamada da referida função e, simplesmente, espera pelo resultado de sua execução. Nesse caso, a saída é um arranjo com os nomes dos visitantes.

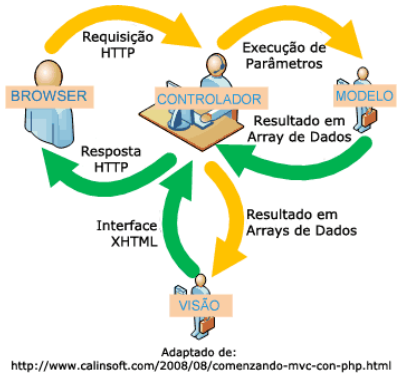

Figura 12 – Funcionamento MVC

A apresentação é responsabilidade da camada **Visão**. A apresentação pode ser, por exemplo, um *feed* RSS, ou uma página HTML.

Além disso, uma função na camada **Controle** pode escolher qualquer tipo de apresentação que a camada **Visão** oferece. Tal característica é uma vantagem da arquitetura MVC, pois apresenta aspectos de reusabilidade. Isso ocorre porque é possível reutilizar o mesmo modelo, com a mesma lógica e criar duas saídas diferentes (ambas imprimem a lista de visitantes, por exemplo).

Na arquitetura MVC, a camada **Controle** pode ser chamada diretamente. Nesse caso, o navegador referencia a função desejada, que pertence à camada **Controle.** Outra opção é aquela em que o navegador acessa um único ponto de entrada do sistema, que, geralmente, é denominado roteador. Nesse caso, o navegador informa ao roteador a função desejada pelo usuário. Então, o roteador escolhe a função, da camada **Controle,** adequada para atender tal solicitação.

A Figura 13 apresenta um esquema de funcionamento da arquitetura MVC com a utilização de um roteador.

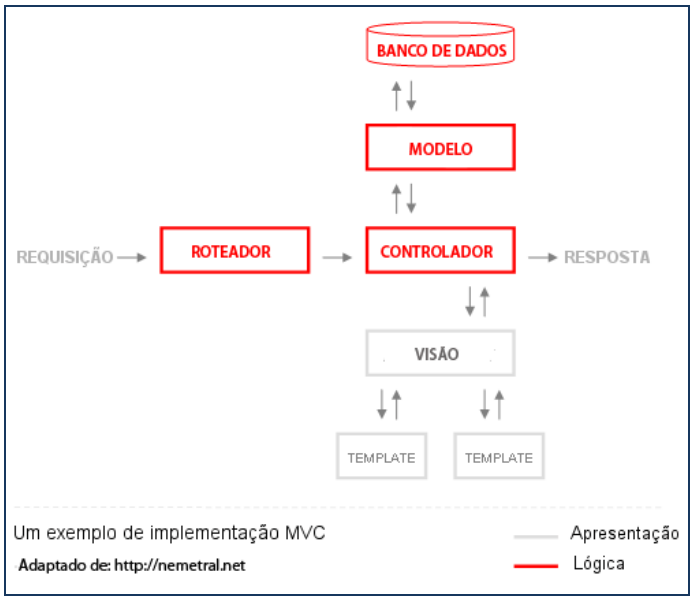

Figura 13 – Arquitetura MVC com o roteador

Programação da interface em camadas

Como a camada **Visão** contém as características: estilo, estrutura e comportamento, então a interface do sistema pode ser programada, também, em camadas.

Nesse contexto, o estilo é programado em CSS, a estrutura em HTML e o comportamento em JavaScript.

O HTML permite a inclusão de marcações e atributos para configuração de estilos. Um exemplo é adicionar alinhamento em uma tag de cabeçalho `<h1>`.

A Figura 14 apresenta um documento no qual o conteúdo está sendo formatado com estilos incluídos no próprio documento XHTML. A figura 15 é o resultado desse documento.

```
1  <html>
2  <head>
3      <title>Documento com Marcação de Apresentação</title>
4  </head>
5  <body>
6      <h1 align="right">Marcação de Apresentação</h1>
7      <p align="right"><font face="Arial, Helvetica, sans-serif" size="2" color="blue">Aqui o
   conteúdo será alinhado à direita, num tipo de letra sans-serif de tamanho 2, e a azul.</font></p>
8  </body>
9  </html>
```

Figura 14 - Documento XHTML com marcações de apresentação

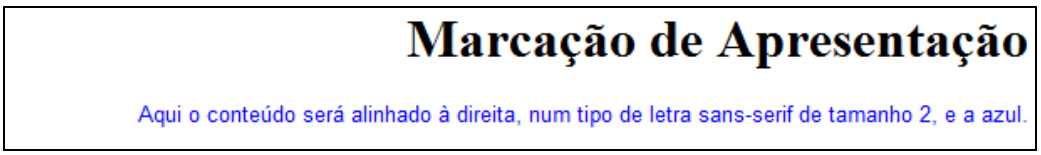

Figura 15 - Interface do Documento com marcações de apresentação

A Tabela 8 comenta o código da Figura 14.

Tabela 8 - Código comentado

Linha(s)	Comentário
1 e 9	Abertura e fechamento do documento HTML
2 a 4	Cabeçalho do documento com o título da página
5 e 8	Abertura e fechamento do corpo do documento HTML
6	O elemento `<h1>` define um cabeçalho com tipo de letra grande, e o atributo `align` define que o alinhamento do texto fica à direita
7	A marcação `<p>` define um parágrafo, e `align` um alinhamento à direita. E `<font>` é uma marcação em desuso, ela configura o estilo da fonte do texto demarcado por ela, o atributo face define o tipo de letra, `size` o tamanho e `color` a cor da fonte.

No caso do documento anterior, quanto maior for o documento, as estruturas e o estilo (toda marcação ou atributo que define o desenho, o estilo e não a estrutura) maior, mais complexo e confuso é o código. Nesse caso, é necessário um código fonte estruturado, que separa o conteúdo e sua estrutura, dos estilos e do comportamento.

A figura 16 apresenta um documento HTML, que contém somente a estrutura, sem nenhum estilo e com configurações padrões. Na Figura 17 é apresentada a interface definida pelo código da Figura 16.

```
1  <html>
2  <head>
3      <title>Documento com Estrutura</title>
4  </head>
5  <body>
6      <h1>Documento Estruturado</h1>
7      <p>A não ser que lhe seja aplicada uma folha de estilo, este parágrafo será apresentado com os
    valores default.</p>
8  </body>
9  </html>
```

Figura 16 - Documento com estrutura

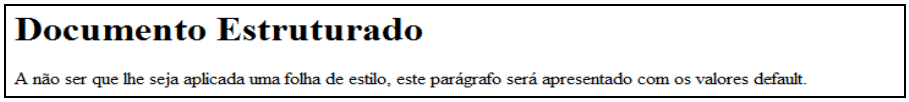

Figura 17 - Interface do documento

Na Figura 18 a estrutura e o estilo são separados. O resultado no navegador é mostrado na figura 19.

```
1  <html>
2  <head>
3      <title>Documento com estrutura e estilo no mesmo documento</title>
4  <style type="text/css">
5  <!--
6  h1 {
7      text-align: right;
8  }
9
10 p {
11     text-align: right;
12     font-family: Arial, Helvetica, sans-serif;
13     font-size: 16px;
14     color: blue;
15 }
16 -->
17 </style>
18 </head>
19 <body>
20     <h1>Documento com estrutura e estilo juntos</h1>
21     <p>Neste documento temos a estrutura da página e o estilo definidos no mesmo documento, porém
    separados.</p>
22 </body>
23 </html>
```

Figura 18 - Documento com estrutura e estilo no mesmo documento

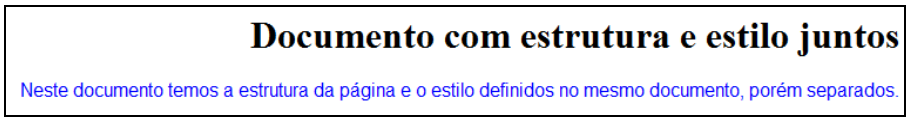

Figura 19 - Resultado no navegador

Na Tabela 9, o código da Figura 18 é comentado.

Tabela9 - Código comentado

Linha(s)	Comentário
4 e 17	Abertura e fechamento das definições de estilo em CSS
6 a 8	Customização do estilo das marcações `h1`, definindo alinhamento à direita.
10 a 15	Customização do estilo das marcações `p`, definindo um alinhamento à direita, tipo de fonte Arial, tamanho 16 e cor azul.

No documento anterior a estrutura e o estilo ainda não estão separados. Então, o próximo passo é separar estilo e estrutura em dois documentos diferentes. Procedendo dessa forma, são obtidos arquivos separados e especializados. Ou seja, a folha de estilo, Figura 20(a) se chama *apresentacao.css*. Nela só existem as definições de elementos e de como ficará sua apresentação. Por outro lado, no arquivo *apresentacao.html* da Figura 201(b) há somente a estrutura.

```
1  h1 {
2  text-align: right;
3  }
4
5  p {
6  text-align: right;
7  font-family: Arial, Helvetica, sans-serif;
8  font-size: 16px;
9  color: blue;
10 }
```

(a)

```
1  <html>
2  <head>
3      <title>Documento com estrutura e estilo</title>
4      <link type="text/css" rel="stylesheet" href="apresentacao.css">
5  </head>
6  <body>
7      <h1>Documento com estrutura e estilo em dois documentos</h1>
8      <p>Neste documento temos a estrutura da página em um arquivo e o
   estilo definidos em outro. </p>
9
10 </body>
11 </html>
```

(b)

Figura 20 – Estilo e estrutura em arquivos separados

Na Tabela 10 o código da Figura 20(b) é comentado.

Tabela 10: Código comentado

Linha(s)	Comentário
4	Aplicação da folha de estilo *apresentacao.css*.

Uma estrutura HTML pode ser apresentada de diversas maneiras, dependendo somente da folha de estilo aplicada. Na Figura 20(b), a linha 4 é um ligação com o arquivo *apresentacao.css*. Entretanto, no lugar desse arquivo pode haver outro que tem outros tipos de estilos para os elementos do documento *apresentacao.html*.

Como o comportamento é programado em JavaScript, neste trabalho é considerada a biblioteca jQuery para efetuar tal programação. A biblioteca jQuery customiza comportamentos para elementos HTML. Assim, como na programação do estilo, a programação do comportamento também pode ser separada da estrutura. Logo, para a programação da interface em camadas, é necessário separar em arquivos distintos a **Estrutura**, o **Estilo** e o **Comportamento**. A seguir, é descrito o que deve conter cada uma dessas camadas.

- **Estrutura**: Essa camada contém: marcações padrão de HTML, tais como `<html>`, `<head>`, `<title>`, `<body>`; marcações que representam estruturas, tais como parágrafos, quebras de linha, listas, divisões, tabelas e formulários.
- **Estilo**: Essa camada contém: tudo que diz respeito ao desenho e não à estrutura, tais como alinhamentos, cor, fontes, bordas etc.
- **Comportamento**: Essa camada contém: programas em JavaScript. Neste trabalho, é utilizada a biblioteca jQuery.

Dada a arquitetura MVC e a programação da interface em camadas, tais conceitos são utilizados a seguir para criar um Sistema Gerenciador de Conteúdo.

9.3. CMS (Content Management System)

Um sistema gerenciador de conteúdo (*CMS - Content Management System*) define uma forma de criar e manter páginas web. O CMS é constituído de um conjunto de ferramentas que conferem agilidade, segurança e confiabilidade requeridas para o tratamento da informação.

Ferramentas do CMS permitem o gerenciamento de conteúdo proveniente de diversas fontes. Elas podem ser destinadas a diversos dispositivos como *browsers* ou celulares. Além disso, elas possuem como vantagens a redução do custo com a atualização de conteúdo, aumento de canais de publicação de conteúdo, padronização da informação e aumento da eficiência das equipes de TI.

A idéia do CMS é separar o gerenciamento do conteúdo do *design* gráfico das páginas que o apresentam. Para isso, o design é visto como o molde, um esqueleto, enquanto o conteúdo é armazenado em banco de dados ou arquivos separados. Como o *design* é visto como um molde, o administrador, ou autor de um site, não precisa se preocupar com detalhes a respeito da interface. É necessário estar atento somente ao conteúdo que é, nele, exibido.

A ferramenta CMS permite que o autor insira conteúdo através de algum meio como: formulários, *upload* de arquivo etc. E tal informação deve ser armazenada em um repositório de dados. Assim, quando um usuário visita o site, o sistema requisita a informação na base de dados, formata, insere na interface e apresenta o resultado ao usuário. Além disso, o CMS permite a inserção, alteração, exclusão e busca de conteúdo em tempo real, sem necessidade de conhecimento de linguagem de programação (PHP, Java etc) de marcação (HTML), e nem de intermediadores para a inserção deste conteúdo.

Alguns tipos de CMS são, livremente, disponibilizados. Como exemplos, há o Drupal, Joomla e Wordpress. De acordo a literatura, o Wordpress é utilizado por pessoas que necessitam de ferramentas para publicar conteúdo e blog. Por outro lado, o Drupal permite extensa customização/programação. E o Joomla possui maior facilidade de personalização, além de possuir quantidade e qualidade de componentes e extensões.

O CMS possui duas sessões, o **Frontend** e o **Backend,** que são descritas na Figura 21.

1. **Frontend** – O **Frontend** atende os usuários do site
2. **Backend** – O **Backend** atende os administradores e editores.

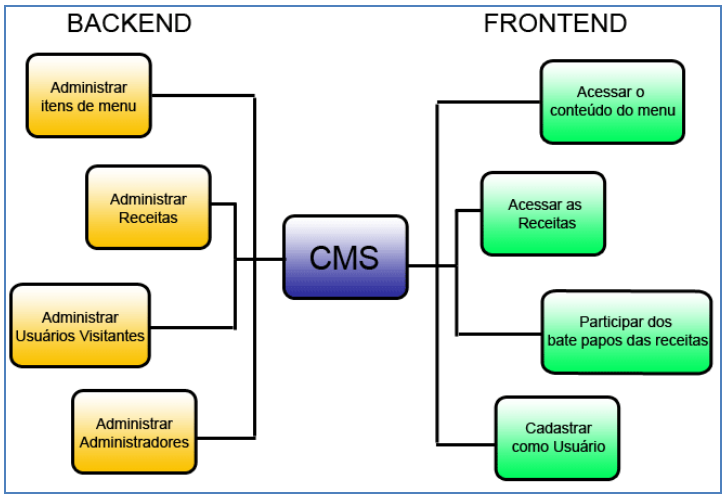

Figura 21 – CMS de Portal de Receitas

Conforme a Figura 21, o **Backend** oferece ferramentas para o administrador, como é detalhado a seguir:

- Administrar itens de menu: Essa funcionalidade permite adicionar/alterar/remover itens de menu. Eles podem ser públicos ou privados;
- Administrar Receitas: Essa funcionalidade permite adicionar/alterar/remover receitas que podem ser públicas ou privadas;
- Administrar Usuários Visitantes: Essa funcionalidade permite verificar novas inscrições de visitantes, podendo excluir visitantes;
- Administrar Administradores: Essa funcionalidade permite adicionar/remover novos administradores do portal.

Conforme a Figura 21, o **Frontend** oferece os recursos que ficarão acessíveis para o usuário, como é detalhado a seguir:

- Acessar o conteúdo do menu: Essa funcionalidade permite acesso ao conteúdo do menu. Se ele for privado, o usuário poderá acessar se ele estiver autenticando no site,
- Acessar as receitas: Essa funcionalidade permite acessar as receitas públicas. Se o usuário estiver autenticado no site, ele poderá acessar as receitas privadas.
- Participar dos bate papos das receitas: Essa funcionalidade permite ao usuário participar de um bate papo
- Cadastrar como Usuário: Essa funcionalidade permite acessar conteúdo privado.

Usabilidade em CMS

A usabilidade de alguma coisa é a característica que permite a sua utilização eficiente, efetiva e satisfatória. No CMS, o objetivo é criar um sistema que seja fácil de entender e simples de usar, mesmo por usuários leigos. Além disso, deve ser uma ferramenta intuitiva e que guie o usuário no seu uso.

Inicialmente, no projeto de um CMS, para que ele seja usável, é necessário definir o objetivo do sistema. Nesse casso ele pode ser utilizado atendendo uma, ou mais, das características as seguir.

- Escrita e estruturação de conteúdo de forma consistente;
- Customizar conteúdo para diferentes usuários;
- Customizar conteúdo para diferentes mídias;
- Distribuir conteúdo dinamicamente;
- Armazenar conteúdo e acessá-lo;
- Recuperação e reuso de conteúdo

Após a definição do objetivo do sistema, é necessário, ainda, responder algumas questões:

- Como o conteúdo será acessado?
- Como o conteúdo deverá ser identificado para que os usuários encontrem o que procuram?
- Como o conteúdo deve ser estruturado para corresponder aos objetivos do CMS?

Por fim para que um CMS seja usável é necessário avaliar as necessidades do sistema, seus usuários e o seu conteúdo, como definido a seguir:

- Avaliação das necessidades: As necessidades de um CMS são obtidas respondendo a questões do tipo: Por que um CMS é necessário? O que gerenciar conteúdo significa na presente situação e o que se deseja que ele faça por você? Quais problemas devem ser resolvidos por ele? Quais oportunidades deseja aproveitar?
- Avaliação dos usuários: Os usuários de um sistema devem ser identificados. É útil criar uma matriz usuário/tarefa para definir os tipos de usuários que utilizarão o sistema. A partir dessa matriz é possível traçar estratégias que definirão como os usuários executarão suas respectivas tarefas.
- Avaliação do conteúdo: A avaliação do conteúdo consiste em estruturá-lo de forma a atender às necessidades do usuário. Por exemplo, em uma loja virtual ao acessar um produto quais informações o usuário busca? Em qual ordem elas devem aparecer?

Esses passos permitem ao desenvolvedor do CMS compreender os objetivos e características que o sistema deve possuir, para assim guiar a criação de um sistema usável.

9.4. Web 2.0

A Web 2.0 é um conceito que possibilita expressar o comportamento dos usuários. Essa Web não se trata somente de novas tecnologias, mas do modo como as pessoas se relacionam e como a tecnologia suporta esse comportamento. Para suportar esta interação e colaboração, são necessárias novas técnicas e métodos computacionais. Assim os novos sistemas web devem ser voltados para que os usuários, hoje ativos e participantes, tenham meios de colaborar com o site.

9.5. Notas Bibliográficas sobre as seções 9.1, 9.2, 9.3 e 9.4

No site *http://php.net* existe uma completa documentação da linguagem de programação PHP. Uma abordagem sobre orientação a objetos em PHP pode ser encontrada no livro ***PHP: Programando com Orientação a Objetos*** de Pablo Dall'Oglio.

A base para a criação da seção sobre o padrão de projeto MVC foram três artigos sobre o tema publicados no site *http://nemetral.net/* do autor Nemetral. O site *http://www.calinsoft.com* também trás uma abordagem para o ensino do padrão, com referência a alguns frameworks que o implementam tais como o CakePHP e o CodeIgniter, entre outros.

Os autores Júlio Pereira e Marcello Bax no artigo intitulado ***Introdução à Gestão de Conteúdos*** abordam sobre o uso de Gerenciadores de Conteúdo e as suas vantagens.

Em *http://www.portal2web.com.br/software-livre/drupal-x-joomla-x-wordpress-qual-o-melhor-cms.html* temos uma comparação entre alguns dos CMS mais utilizados atualmente.

O Dominic Kristaly em seu artigo intitulado ***Web 2.0 technologies in web application development*** aborda a respeito do conceito de Web 2.0, sua aplicação em CMS, e descreve um sistema gerenciador de conteúdo desenvolvido chamada Butterfly. Pamela Kostur em seu trabalho ***Incorporating Usability into Content Management*** **explica** o que é usabilidade e como ela se aplica no planejamento e desenvolvimento de um sistema gerenciador de conteúdo.

Sobre Web 2.0 o presente trabalho se baseou nas obras de Tim Berners-Lee, Tim O'Reilly e Paul Anderson, ***Weaving the Web: the past, present and future of the World Wide Web by its inventor, What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*** *e* ***What is Web 2.0? Ideas, Technologies and implications for education***, respectivamente.

9.6. jQuery

Introdução

jQuery é uma biblioteca JavaScript que simplifica a manipulação de documentos HTML, o uso de animações e requisições Ajax, propiciando um rápido desenvolvimento de aplicações web. Seu criador foi o desenvolvedor americano John Resig.

O jQuery tem como objetivos facilitar o uso do JavaScript para realizar as seguintes tarefas:

- Adicionar efeitos visuais e animações;
- Acessar e manipular o documento HTML;
- Alterar o conteúdo de uma página sem recarregá-la;
- Prover interatividade;
- Modificar apresentação e estilização de elementos HTML;
- Simplificar tarefas específicas do JavaScript.

A biblioteca jQuery atende os Padrões Web, o que significa que ela é compatível com qualquer sistema operacional e navegador. Essa biblioteca possui as seguintes características:

- Utiliza seletores CSS para localizar elementos em um documento HTML;

- Possibilita a instalação de plug-ins e extensões;
- É indiferente às inconsistências de renderização de navegadores;
- Localiza elementos dentro de um documento de forma precisa;
- Admite programação encadeada;
- É extensível, admitindo a inserção de novas funcionalidades na biblioteca.

Além disso, a biblioteca jQuery é um arquivo JavaScript, que deverá ser linkado à página web que utilizar programação jQuery. A Figura 22 apresenta a linha que faz o link com a biblioteca. Tal link dever ser colocado no cabeçalho do documento, ou seja, entre as marcações `<head></head>`. Já o atributo `src`, da marcação `<Script>`, indica o caminho no qual o arquivo JavaScript está gravado.

```
...
<head>
...
    <script type='text/javascript' src='jquery.js'></script>
...
</head>
...
```

Figura 22 – Link com a biblioteca jQuery

Na Figura 23 é apresentado um código com JavaScript in-line (script dentro de marcação HTML). Tal escrita de código contraria um dos princípios fundamentais dos padrões Web, que trata da separação total entre estrutura, estilo e comportamento.

```
1  ...
2      <style type="text/css" media="all">
3          h1 { color:#090; } /* verde */
4      </style>
5  </head>
6  <body>
7      <h1 id="cor"> Cabeçalho muda de cor </h1>
8      <button type="button" onclick="document.getElementById('cor').style.color = '#F00';">
9          Vermelho
10     </button>
11 ...
```

Figura 23 – JavaScript in-line

A Figura 24 apresenta a interface que corresponde ao código da Figura 23. Quando o botão de nome “Vermelha” é clicado, o método `onclick` do elemento `button` (linha 8) é acionado. Então, o código JavaScript é executado, alterando a cor do elemento cujo `id` é cor, ou seja, a marcação `<h1>` (linha 7).

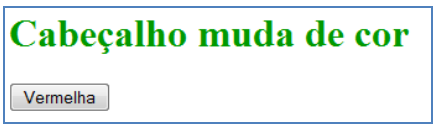

Figura 24 - Interface do arquivo da Figura 23

Essa mesma funcionalidade é representada na Figura 25, com uma marcação HTML isenta de código JavaScript. Isso é desejável, porém seria melhor ainda se fossem separados o estilo, comportamento e conteúdo em documentos diferentes.

A tabela 11 comenta o código da Figura 25.

```
1  ...
2      <style type="text/css" media="all">
3          h1 { color:#090; } /* verde */
4      </style>
5      <script type="text/javascript">
6          window.onload = function(){
7              document.getElementById('btn-vermelho').onclick = mudaCor()
8          };
9          function mudaCor(){
10             document.getElementById('cor').style.color = '#F00'; /* vermelho */
11         };
12     </script>
13 </head>
14 <body>
15     <h1 id="cor"> Cabeçalho muda de cor </h1>
16     <button type="button" id="btn-vermelho">Vermelho</button>
17 ...
```

Figura 25 – Código JavaScript separado do HTML

Tabela 11 – Comentários a respeito do código da Figura 25

Linha(s)	Comentário
6 a 8	Assim que o arquivo terminar de ser carregado (linha 6) será atribuído um evento `onclick` ao elemento que possui como `id` `btn-vermelho`, quando este for clicado a função `mudaCor` é chamada.
9 a 11	Função `mudaCor` quando acionada alterará a cor do elemento cujo `id` for `cor`.

A Figura 26 apresenta o script programado, utilizando a biblioteca jQuery. Como é indicado, o código gerado tem a mesma função do código da Figura 25. Porém, ele é mais simples e, conseqüentemente, mais rápido de programar.

```
1  ...
2      <style type="text/css" media="all">
3          h1 { color:#090; } /* verde */
4      </style>
5      <script type="text/javascript" src="jquery.js"></script>
6      <script type="text/javascript">
7          $(document).ready(function(){
8              $("#btn-vermelho").click(function(){
9                  $("#cor").css("color","#FF0000");
10             });
11         });
12     </script>
13 </head>
14 <body>
15     <h1 id="cor"> Cabeçalho muda de cor </h1>
16     <button type="button" id="btn-vermelho">Vermelho</button>
17 ...
```

Figura 26 – Código jQuery

A tabela 12 comenta o código da Figura 26 e explica algumas das principais estruturas da biblioteca.

Tabela 12 – Código da Figura 26 comentado

Linha(s)	Comentário
5	Link com o arquivo da biblioteca jQuery para poder utilizar a sintaxe jQuery.
6	O `$()` é a função construtora da biblioteca, ela é responsável por encontrar os elementos em um documento. Na linha 6 ela está referenciando o documento como um todo. `ready()` é um método similar ao `window.onload`. Assim `$(document).ready` irá executar quando a estrutura (árvore) do documento terminar de carregar.
8	A função `$()` encontra o elemento cujo `id` (referenciamos o `id` por #) é `btn-vermelho` e atribui um evento `click` a ele.
9	Quando o botão for clicado (linha 8) a função `$()` encontra o elemento cujo `id` for `cor` e modifica um atributo CSS, no caso a cor, colocando o código referente à cor vermelha.

Conforme os exemplos anteriores, uma das estruturas mais básicas do jQuery é a função construtora `$()`. Ela é responsável pela localização dos elementos dentro da árvore do documento.

Assim, caso seja necessário, por exemplo, alterar a cor de fundo de todos os elementos `<div>` para a cor vermelha, então a sintaxe a ser utilizada deve ser aquela apresentada na Figura 27.

```
$("div").css("background","red");
```

Figura 27 – Todos os elementos com fundo vermelho

Mas, encontrar todos os elementos de um tipo pode não ser aquilo desejado pelo programador. Nesse caso, pode ser que o programador deseja localizar apenas uma ocorrência específica de um elemento. Se há um elemento, como na Figura 28, cujo `id` é `xyz`, o jQuery o localiza, como mostrado na Figura 29.

```
<p id="xyz"> Texto do parágrafo </p>
```

Figura 28 – elemento <p> identificado por xyz

```
$('#xyz');
```

Figura 29 – Função construtora localiza elemento com id igual à xyz

Encadeamento de funções

O encadeamento de funções da biblioteca jQuery é o ordenamento em uma única linha de várias funções. Um exemplo desse encadeamento é mostrado na Figura 30.

```
$('div').children('p').fadeOut().addClass('xyz');
```

Figura 30 – Exemplo de encadeamento

A sintaxe da Figura 30 diz o seguinte:

"Construtor, encontre todos os elementos `parágrafos` que sejam filhos dos elementos `div`, neles aplique um efeito de esmaecimento (`fadeOut`) e adicione a classe CSS `xyz`."

Em jQuery, o encadeamento é possível porque cada método retorna um objeto, que por sua vez pode ser processado pelo método seguinte.

Funções-padrão e seletores jQuery

A biblioteca jQuery possui funções-padrão que podem ser utilizadas em programação, com vários objetivos. Algumas delas são:

» **`.css`**

A função .css adiciona, ou altera, um estilo do elemento associado. Na Figura 31, o primeiro parâmetro indica a propriedade CSS do elemento que se deseja alterar. O

segundo parâmetro é a quantificação ou característica da propriedade. Na Figura 31, seu código altera o tamanho da caixa do bate papo para 200 pixels.

```
$('#chat').css('height','200px');
```

Figura 31 – Função .css

» **$( )**

A função $() é a principal função jQuery. Ela aceita dois parâmetros, como mostrado na Figura 32.

```
$('p',$('#chat')).css('color','green');
```

Figura 32 – Construtor $()

Na Figura 31 o construtor `$()` possui somente um parâmetro (`#chat`) que indica a busca do elemento cujo `id` é `chat`. Na Figura 32 há um segundo parâmetro que é denominado contexto. Ele é utilizado para localizar (no caso do exemplo) elementos `<p>` que estejam dentro de elementos, cujo `id` é `chat`. Se o segundo argumento é omitido, então todos os parágrafos do documento são estilizados na cor verde. O resultado da aplicação da função da Figura 32 na Figura 33 faz com que todos os parágrafos dentro de chat fiquem verdes.

```
<div id="chat">
    <p> Primeiro Parágrafo. </p>
    <p> Segundo Parágrafo. </p>
</div>
```

Figura 33 – Exemplo de documento HTML

As funções a seguir selecionam elementos dentro do documento. Existem várias formas de efetuar esta seleção de acordo com o objetivo do script.

» **$( )**

A função construtora `$()` pode ser empregada como uma função seletora. Quando utilizada na forma da Figura 34 seleciona um elemento cujo atributo `id` é igual a `receita` (elemento presente na linha 3 da Figura 35) e altera seu estilo.

```
$('#receita').css('background','red');
```

Figura 34 - Selecionando elemento com id igual à receita

```
1  <h1> Título da Receita </h1>
2
3  <div id="receita">
4      <p> Corpo da Receita. </p>
5      <p> Autor. </p>
6  </div>
7
8  <form class="form_verde" id="contato">
9      Nome: <input type="text" id="nome" /> <br />
10     E-mail: <input type="text" id="email" /> <br />
11     Texto: <input type="text" id="email" /> <br />
12     <input type="submit" value="Enviar" />
13 </form>
```

Figura 35 - Exemplo de um documento HTML com um formulário

Na Figura 36, o parâmetro da função construtora seleciona um elemento cujo atributo `class` é igual à `form_verde` (elemento presente na linha 8 da Figura 9.2.15) e altera seu estilo.

```
$('.form_verde').css('background','#60F');
```

Figura 36 – Selecionando elemento com class igual à form_verde

O parâmetro da função construtora escrito como no código da Figura 37 seleciona todos os elementos `input` do documento (elementos presentes nas linhas 9 a 12 da Figura 35) e altera seu estilo.

```
$('input').css('background','#FFC');
```

Figura 37 – Seleciona todos os elementos input

A função `$()` escrita como no código da Figura 38 permite definir um conjunto de seletores como argumento.

```
$('#receita, input, .form_verde').css('background','red');
```

Figura 38 – Vários elementos sendo selecionados

» **`:first`**

O seletor `:first` cuja sintaxe é apresentada na Figura 39, seleciona a primeira ocorrência do elemento associado a ele, no caso a linha 2 da Figura 40.

```
$('li:first').css('background','#60F');
```

Figura 39 – Seletor filtro :first

```
1  <ul>
2      <li> Primeiro item </li>
3      <li> Segundo item </li>
4      <li> Terceito item </li>
5      <li> Quarto item </li>
6      <li> Quinto item </li>
7  </ul>
```

Figura 40 – Exemplo de documento HTML com uma lista não ordenada

» **`:last`**

O seletor `:last` seleciona a última ocorrência do elemento associado a ele. A sintaxe do código da Figura 41 altera o estilo do elemento da linha 6 da Figura 40.

```
$('li:last').css('background','green');
```

Figura 41 – Seletor filtro :last

» **`:even`**

O seletor `:even` seleciona as ocorrências de ordem par do elemento associado a ele O código da Figura 42 altera o estilo dos elementos das linhas 2, 4 e 6 da Figura 40, que correspondem aos índices 0, 2 e 4 (a contagem inicia no 0).

```
$('li:even').css('background', 'red');
```

Figura 42 – Seletor filtro :even

» **`:odd`**

O seletor `:odd` seleciona as ocorrências de ordem ímpar do elemento associado a ele. O código da Figura 43 altera o estilo dos elementos das linhas 3 e 5, que correspondem aos índices 1 e 3 (a contagem inicia no 0) da Figura 40.

```
$('li:odd').css('background', 'green');
```

Figura 43 – Seletor filtro :odd

» **`:eq`**

O seletor `:eq` seleciona a última ocorrência do elemento associado a ele. O código da Figura 44 altera o estilo do quarto elemento representado na linha 5 da Figura 40.

```
$('li:eq(3)').css('background', 'yellow');
```

Figura 44 – Seletor filtro :eq

Manipulação de elementos da árvore do documento HTML

A seguir são apresentadas funções jQuery que definem atributos e valores a elementos de um documento HTML.

» **`.attr`**

Se um documento HTML possui uma imagem como no código da Figura 45, a sintaxe do método `.attr` mostrado na Figura 46 captura o valor do atributo `src` da imagem e a armazena na variável `arquivo`.

```
<img src="foto.jpg">
```

Figura 45 – HTML com uma imagem

```
var arquivo = $('img').attr('src');
```

Figura 46 – Método .attr

O método `attr` é utilizado, também, para adicionar atributos e valores a um elemento. Na Figura 47 a sintaxe do método `attr` adiciona ao elemento `img` vários atributos com seus respectivos valores. Na Figura 48 a sintaxe do método `attr` adiciona ao elemento `<img>` somente um atributo e seu valor.

```
$('img').attr({
    border:"10px",
    src: "foto2.jpg"
});
```

Figura 47 – Método .attr adicionando múltiplos atributos

```
$('img').attr('border','10px');
```

Figura 48 – Método .attr adicionando somente um atributo

» **`.removeAttr`**

A função `.removeAttr` acessa o elemento associado a ele e remove o atributo definido em seu parâmetro. No código da Figura 49, o método `removeAttr` remove o atributo `src` do elemento `img`.

```
$('img').removeAttr('src');
```

Figura 49 – Método .removeAttr

» **`.addClass`**

O método `.addClass` atribui ao elemento associado a ele uma classe CSS já definida. Na Figura 50, se a classe CSS `form_verde` existe, então o método `.addClass` atribuirá ao formulário do documento esse estilo.

```
$('form').addClass('form_verde');
```

Figura 50 – Método .addClass

» **`.html`**

A função `.html` captura ou atribui código HTML a um elemento. Na Figura 51 o método `.html` captura o conteúdo do elemento `p` e o atribui à variável `conteudo`. Na Figura 52 a função `.html` atribui o conteúdo da variável `dado` (a variável contém código HTML) a um elemento `div` (exemplo retirado do bate papo do CMS).

```
var conteudo = $('p').html();
```

Figura 51 – Capturando conteúdo html de um elemento

```
$('#chat').html(dado);
```

Figura 52 – Atribuindo conteúdo html a um elemento

Na biblioteca jQuery existem ainda os métodos `.text` e `.val` que são semelhantes ao método `.html`.

Eventos

Eventos são ações realizadas pelos usuários no navegador. Quando o usuário abre uma página, clica em um botão, seleciona uma opção em um campo ou preenche um formulário ocorre um evento. A seguir são apresentados alguns eventos jQuery:

» **`.blur`**

O evento `.blur` é chamado quando um elemento perde o foco. Por exemplo, um usuário ao preencher um formulário segue de campo em campo preenchendo-o com informações, quando ele termina de preencher um campo e passa para outro o campo anterior perde o foco, cedendo-o ao próximo elemento. Assim o evento `.blur` é muito usado em formulários para validar a entrada de um dado em um campo. Na Figura 53 quando o campo perde o foco uma caixa de alerta é acionada.

```
$('input').blur(function() {
    alert("Você acionou o evento blur");
});
```

Figura 53 – Evento .blur

» **`.click`**

O evento `.click` é acionado quando algum elemento de um documento HTML é clicado. A Figura 54 apresenta um trecho do código do bate papo do CMS, esse trecho será executado quando um usuário clicar em um elemento com `id` igual a `ok`. A Figura 55 apresenta o elemento cujo evento de click dispara o evento mostrado na Figura 54. No bate papo quando o usuário preencher uma mensagem ele a enviará ao servidor através do evento `.click` do botão `"Enviar"`, esse evento processará o envio da mensagem.

```
$('#ok').click(function(){
    enviar();
});
```

Figura 54 – Evento .click

```
<input id="ok" type="button" value="Enviar">
```

Figura 55 – Botão de envio do bate papo

» **`.dblclick`**

O evento `.dblclick` é disparado quando um elemento é clicado duas vezes pelo usuário.

» **.change**

O evento .change é disparado quando um elemento muda de valor. Esse evento é muito utilizado em campos de formulário do tipo `select`. A Figura 56 apresenta um exemplo do uso do evento `.change` em um `select` que lista os estados brasileiros.

```
$('select').change(function() {
    alert("Você escolheu uma UF");
});
```

Figura 56 – Evento .change

» **.focus**

O evento `.focus` ocorre quando um elemento recebe foco, por exemplo, quando uma caixa de texto é selecionada. Na Figura 57 quando o usuário clica sobre um campo de formulário para digitar seu CPF, uma janela de alerta é exibida com informações.

```
$('input').focus(function() {
    alert("Use apenas números para informar seu CPF");
});
```

Figura 57 – Evento .focus

» **.keydown**

O evento `.keydown` é disparado quando o usuário pressiona uma tecla qualquer do teclado. A Figura 58 apresenta um exemplo de seu uso.

```
$('input').keydown(function() {
    alert("Uma tecla foi pressionada");
});
```

Figura 58 – Evento .keydown

» **.keyup**

O evento `.keyup` ocorre quando o usuário solta uma tecla do teclado que antes estava pressionada. Um exemplo de evento é apresentado na Figura 59.

```
$('input').keyup(function() {
    alert("Uma tecla deixou de ser pressionada");
});
```

Figura 59 – Evento .keyup

» **.load**

O evento `.load` é disparado quando o elemento ao qual ele se referencia termina de ser carregado na página HTML. Esse evento quando relacionado ao objeto `window` será disparado após o carregamento total da página. Na Figura 60 é exemplificado seu uso com o elemento `<body>` do HTML.

```
$('body').load(function() {
    alert("O corpo do HTML foi carregado");
});
```

Figura 60 – Evento .load

» **`.mouseover`**

O evento `.mouseover` ocorre quando o usuário passa o mouse por cima de um elemento. A Figura 61 exemplifica o uso desse evento..

```
$('input').mouseover(function() {
    alert("Passou o mouse por cima do elemento");
});
```

Figura 61– Evento .mouseover

» **`.mouseout`**

Quando o mouse sai de cima de um elemento o evento `.mouseout` é disparado. A Figura 62 a sintaxe do evento é apresentada.

```
$('input').mouseover(function() {
    alert("Passou o mouse por cima do elemento");
});
```

Figura 62 – Evento .mouseout

» **`.select`**

Quando o usuário seleciona um texto ou fragmento de texto de um elemento o evento `.select` é processado. Esse evento está disponível somente para elementos destinados à entrada de textos, como `input` e `textarea`. Na Figura 63 um exemplo da sintaxe do evento `.select` é mostrado..

```
$('input').select(function() {
    alert("Você selecionou um texto");
});
```

Figura 63 – Evento .select

» **`.submit`**

Quando um formulário é enviado o evento `.submit` é disparado. Ele é freqüentemente utilizado por programador para verificar ser processado quando um formulário é enviado para verificar se dados inseridos nos campos estão corretos. A Figura 64 exemplifica o uso do evento `.submit`.

```
$('input').submit(function() {
    alert("Dados do Formulário enviados");
});
```

Figura 64 – Evento .submit

Efeitos

Simplificar a implementação de efeitos é um dos objetivos do jQuery. O programador consegue enriquecer a experiência do usuário no uso de interfaces através de efeitos como esconder e revelar conteúdos ou um menu que mostra seus subitens de forma visualmente agradável através de suaves transições. A biblioteca jQuery permite implementar esses efeitos sem o detrimento da usabilidade da interface ou bloqueio do acesso de uma página. A seguir são apresentados alguns métodos para obtenção de efeitos em elementos HTML.

» **`.show`**

A sintaxe do método `.show` na sintaxe mostrada na Figura 65 revela o elemento associado a ele (elemento `<div>`) de maneira abrupta.

```
$('div').show();
```

Figura 65 – Método show sem parâmetro

O método `.show` pode possuir parâmetros. No código da Figura 66 o método `.show` apresenta o parâmetro `slow`, que indica que o elemento irá se revelar de forma lenta. Ao invés do parâmetro `slow` outros valores são possíveis tais como `normal` ou `fast` (velocidade normal e rápida respectivamente). A Figura 67 exemplifica o uso do método `.show` com um parâmetro numérico, esse parâmetro indica a velocidade em milissegundos com que o elemento irá aparecer na página.

```
$('div').show('slow');
```

Figura 66 – Revelação lenta do elemento DIV

```
$('div').show(3000);
```

Figura 67 – Revelação em 3 segundo do elemento DIV

O método show permite definir uma função que será executada quando o efeito terminar. Na Figura 68 assim que o elemento for revelado um alerta aparecerá.

```
$('div').show('normal', function(){
    alert("Elemento DIV revelado");
});
```

Figura 68 – Função com execução subseqüente ao efeito

» **`.hide`**

O efeito `.hide` tem um funcionamento contrário do método `.show`, mas possui sintaxe semelhante. O efeito `.hide` esconde o elemento associado a ele. A Figura 69 apresenta as sintaxes possíveis do método `.hide`.

```
$('input').hide();

$('table').hide('slow');

$('form').hide(5000);

$('div').hide('fast', function(){
    alert("Elemento DIV escondido");
});
```

Figura 69 – Sintaxes do método hide

» **`.toggle`**

Com o uso do método `.toggle` se um elemento está visível ele fica invisível, e vice versa. A Figura 70 apresenta a sintaxe do método.

```
$('div').toggle();
```

Figura 70 – Método toggle

» **`.slideDown`**

O método .slideDown se associado a um elemento escondido fará com que ele se revele suavemente. O método faz a transição do invisível para o visível por meio do aumento gradativo da altura do elemento. Os parâmetros possíveis são `slow`, `normal` ou `fast`. Na Figura 71 um exemplo de sintaxe é apresentado.

```
$('div').slideDown('fast');
```

Figura 71 – Método slideDown

» **`.fadeIn`**

O efeito `.fadeIn` é destinado para a criação de um efeito de revelar o elemento associado a ele que está escondido através do aumento gradativo da opacidade, saindo da opacidade 0 (invisível) para a opacidade 100% (visível). Os parâmetros possíveis são `slow`, `normal` ou `fast` ou um valor numérico que indica o tempo em milissegundos que durara o efeito. Uma função ainda poderá ser definida para ser executada assim que o elemento reaparecer. Na Figura 72 alguns exemplos de sintaxe são apresentados.

```
$('input').fadeIn('fast');

$('table').fadeIn(3000);

$('div').fadeIn('slow', function(){
    alert("O elemento DIV reapareceu gradativamente");
});
```

Figura 72 – Sintaxes do método fadeIn

» **`.fadeOut`**

O efeito `.fadeOut` tem funcionamento contrário ao criado pelo método `.fadeIn`, tendo sintaxe semelhante. A aplicação dessa função cria o efeito de esconder

um elemento através da diminuição gradativa da opacidade. A Figura 73 apresenta alguns exemplos de sintaxe.

```
$('input').fadeOut('slow');

$('table').fadeOut(5000);

$('div').fadeOut('fast', function(){
    alert("O elemento DIV desapareceu gradativamente");
});
```

Figura 73 – Sintaxes do método fadeOut

9.7. Notas Bibliográficas sobre a seção 9.6

Essa seção foi baseada no livro ***jQuery – A biblioteca do programador JavaScript***, um excelente livro para se aprofundar em jQuery, possui diversos exemplos e uma ampla documentação. O autor é o conceituado Maurício Samy Silva, que também desenvolve o site Maujor (*www.maujor.com*), referência na área de desenvolvimento para a web.

9.8. Ajax com jQuery

Introdução

AJAX é a sigla em inglês para *Asynchoronous JavaScript and* XML. Trata-se de uma técnica de carregamento de conteúdos em uma página Web com o uso de JavaScript e XML. O objeto `XMLHttpRequest` é o responsável pelo funcionamento do AJAX, que pode ser "traduzir" por requisição XML com o uso do protocolo HTTP. É uma técnica empregada para carregar conteúdo de um servidor (armazenada em um banco de dados ou não), sem necessidade de haver *reload*, ou seja sem necessidade de haver o recarregamento de toda a página para ocorrer a sua atualização (como ocorre com as aplicações Web tradicionais).

Ajax sem jQuery

Uma revisão do uso tradicional do Ajax é interessante para conhecer o seu mecanismo de funcionamento, pois no uso com jQuery algumas funções ficam transparentes ao desenvolvedor. O objeto `XMLHttpRequest` é o responsável pelo funcionamento do Ajax,e para a criação de uma instância desse objeto a sintaxe é como a apresentada na Figura 74:

```
var req = new XMLHttpRequest();
```

Figura 74 – Instanciando o objeto

A criação deste objeto varia de acordo com alguns navegadores. Caso o navegador seja o Internet Explorer nas versões 5 e 6 a instanciação do objeto se dá como mostrado na Figura 75.

```
var req = new ActiveXObject("Microsoft.XMLHTTP");
```

Figura 75 – Instanciando o objeto no Internet Explorer

Para a criação do objeto `XMLHttpRequest` nas versões mais antigas do Internet Explorer é utilizada a sintaxe `Microsoft.XMLHTTP`. Nas versões mais recentes a sintaxe `Msxml2.XMLHTTP` é empregada. Nas versões 7 e 8 do Internet Explorer o objeto `XMLHttpRequest` se tornou um objeto nativo do navegador, podendo ser instanciado como mostrado na Figura 74.

Instintivamente criar-se-ia um código que teste primeiramente o navegador para assim criar o objeto utilizando a sintaxe mais apropriada, porém este modo não é muito utilizado. A criação é feita através de tentativa e erro. Tenta-se criar o objeto, caso ocorra um erro é feita uma tentativa de criação por outra sintaxe e assim sucessivamente. A Figura 76 apresenta a criação do objeto XMLHttpRequest utilizando a sintaxe `try...catch`, ou seja, o que se segue a `try` é uma tentativa de executar algo, caso ocorra algum erro esse é "capturado" pelo `catch` e tratado.

```
1  function inicia_ajax(){
2      var req;
3
4      try {
5          req = new ActiveXObject("Microsoft.XMLHTTP");
6      } catch(e) {
7          try {
8              req = new ActiveXObject("Msxml2.XMLHTTP");
9          } catch(ex) {
10             try {
11                 req = new XMLHttpRequest();
12             } catch(exc) {
13                 alert("Esse browser não suporta Ajax");
14                 req = false;
15             }
16         }
17     }
18     return req;
19 }
```

Figura 76: Interface do Documento com marcações de apresentação

Na Tabela 13 acompanha-se o código da Figura 76 comentado.

Tabela 13: Código comentado

Linha(s)	Comentário
1	Criação da função que será sempre chamada para criar o objeto `XMLHttpRequest`
5	Cria o objeto se o navegador do usuário é o Internet Explorer 5 até 6
8	Cria o objeto se o navegador do usuário possui versões mais recentes do Internet Explorer
11	Sintaxe padrão para a criação do objeto `XMLHttpRequest` para os demais navegadores.
13 e 14	Caso o navegador não suporte Ajax uma caixa de alerta é apresentada.
18	A função retorna o objeto criado na variável `req`

Depois de criada a instância do objeto `XMLHttpRequest` o próximo passo é estabelecer a comunicação com o servidor, no qual são requisitadas informações ou arquivos. Os seguintes métodos cumprirão essa função:

- Ação disparadora de evento `onreadystatechange`
- Método `open`
- Método `send`
- Método `setRequestHeader`

» onreadystatechange

Uma ação disparadora de evento é uma ação iniciada por um usuário (como um evento `onclick` em um elemento da interface), mas não exclusiva dele, por exemplo, quando uma página é carregada um evento pode ser disparado através da sintaxe `window.onload`.

A ação `onreadystatechange` é disparada pelo servidor quando ele envia para o cliente uma informação de que alguma atualização ou mudança ocorreu nele. A ação é executada sempre que ocorre uma mudança no valor da propriedade `readyState` do objeto `XMLHttpRequest` A propriedade readyState pode assumir cinco valores, como mostrado na Tabela 14:

Tabela 14: Valores da propriedade readyState

Valor	Status/Interpretação
0	`UNSENT/` Comunicação não iniciada
1	`OPENED/` Início da comunicação. A comunicação foi iniciada sem envio de dados.
2	`HEADERS RECEIVED/` Carregamento finalizado
3	`LOADING/` Servidor em processo de envio da resposta à requisição.
4	`DONE/` Servidor acaba de enviar a resposta à requisição.

A Figura 77 apresenta a implementação da ação disparadora do servidor.

```
1  var req = inicia_ajax();
2  if(req){
3      req.onreadystatechange=function(){
4          if(req.readyState == 4){
5              if(req.status == 200){
6                  document.getElementById("chat").innerHTML = req.responseText;
7              }
8          }
9      };
10 }
```

Figura 77 – Ação disparadora do servidor do bate papo

Comentários a respeito do código da Figura 77 são mostrados na Tabela 15..

Tabela 15 – Figura 77 comentada

Linha(s)	Comentário
1	Iniciamos o objeto `XMLHttpRequest`, que será referenciado por `req`.
2	Testamos se o objeto foi criado.
3	`onreadystatechange` definirá a função que será chamada quando o servidor alterar a propriedade `readyState` do objeto `req`
4	Se a propriedade `readyState` indicar que certa ação está com o status `DONE`, ou seja, o servidor enviou a resposta a alguma requisição efetuada, ele executará as linhas 5 a 7.
5	O servidor envia um cabeçalho HTTP alterando a propriedade status com um valor numérico de 3 dígitos. Quando o valor da propriedade status é 200 indica que a requisição efetuada foi bem sucedida.
6	A propriedade `responseText` retorna os dados enviados pelo servidor como resposta à requisição, foi feita (em outra parte do código não presente na Figura 77) uma requisição pelas últimas 30 mensagens do bate papo, a propriedade `responseText` conterá o HTML das mensagens enviadas pelo servidor ao navegador. Este HTML enviado será colocado no elemento `<div id="chat">` que criamos.

» **Método open**

O método `open` é utilizado para informar ao servidor qual o endereço do arquivo ao qual ele requisita. A Figura 78 mostra a sintaxe do método que foi utilizado no bate papo.

```
req.open('GET', "msg.php", true);
```

Figura 78 – Sintaxe do método open

O primeiro parâmetro do método `open` indica que método de envio dos dados pode ocorrer por `GET` ou `POST`. Na Figura 78 é utilizado o `GET`. O segundo parâmetro indica a `url` do arquivo que se deseja obter, e o terceiro é um parâmetro facultativo, que quando está com o valor `true` indica que a comunicação com o servidor é assíncrona.

» **Método send**

O método `send` inicia a requisição que foi definida pelo método `open`. Ela admite somente um parâmetro que pode ser um conjunto de dados que se deseja enviar ao servidor ou o valor `null` que significa que a requisição não envia dados (como na Figura 79).

```
req.send(null);
```

Figura 79 – Sintaxe do método send

A Figura 80 apresenta o código da interface do bate papo. A apresentação da interface do bate papo no navegador é mostrada na Figura 81.

```
1  <html>
2      <head>
3          <title>Bate Papo Naturalista</title>
4          <script type="text/javascript" src="ajax.js" ></script>
5          <link rel="stylesheet" href="estilo.css" />
6      </head>
7      <body>
8          <div id="pagina_chat">
9              <div id="chat"> </div>
10                 <form  id="chat">
11                     <input id="msg" type="text" size="20">
12                     <input id="ok" type="button" value="Enviar" onClick="enviar()">
13                 </form>
14             </div>
15         </div>
16     </body>
17 </html>
```

80 – Código HTML da interface do chat

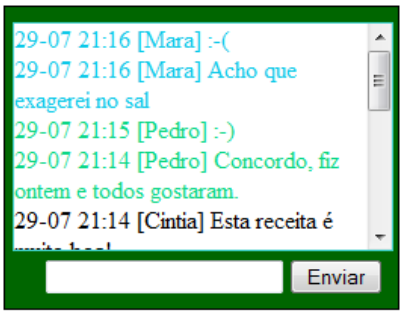

Figura 81 – Interface do bate papo

O comportamento Ajax é apresentado na Figura 82.

```
1  function inicia_ajax(){
2      /* ... criação do objeto XMLHttpRequest ... */
3  }
4
5  function enviar() {
6      var req = inicia_ajax();
7      if(req){
8          req.onreadystatechange= function(){ /* implementação da função disparadora */ };
9          var mensagem = document.getElementById("msg").value;
10         var url = "msg.php?msg=" + escape(mensagem);
11         req.open('GET',url, true);
12         req.send(null);
13     }
14     document.getElementById("msg").value = "";
15 }
16
17 function ler() {
18     var tempoChat = 2000;
19     var req = inicia_ajax();
20     if(req){
21         req.onreadystatechange=function(){
22             /* implementação da função disparadora */
23             setTimeout("ler()", tempoChat);
24         };
25         req.open('GET', "msg.php?", true);
26         req.send(null);
27     }
28 }
29
30 window.onload = ler();
```

Figura 82 – Implementação do comportamento Ajax do bate papo

Na Tabela 15 acompanha-se o código da Figura 82 comentado.

Tabela 15 – Código da Figura 82 comentado

Linha(s)	Comentário
1 a 3	Cria o objeto Ajax
5 a 15	Quando um usuário envia uma mensagem, a função `enviar()` é chamada. Ela inicia o objeto Ajax, define a função disparadora do servidor (`onreadystatechange`), captura a mensagem digitada (linha 9), prepara a url que será requerida no servidor (linha 10), informa ao servidor qual o arquivo e em seguida efetua a requisição (linhas 11 e 12). Ao final limpa o campo de mensagem (linha 14).
17 a 28	A função `ler()` é chamada de 2 em 2 segundos, ela é responsável por buscar no servidor as mensagens. Na linha 23 a sintaxe `setTimeout` chamará a função `ler()` no tempo definido. Em seguida definimos novamente o arquivo que será requisitado e executamos a requisição (linhas 25 e 26).
30	Quando o documento terminar de ser carregado no navegador do usuário a função `ler()` é chamada, é nele que o script inicia, sem interferência do usuário.

A página *msg.php* que é chamada nos exemplos do bate papo desse trabalho é apresentada na Figura 83.

```
1  <?php
2      $conexao = mysql_connect("endereco","usuario", "senha").
3      $db = mysql_select_db("bd");
4
5      if($_GET["msg"]){
6          $data = time();
7          $usuario = $_SESSION["nome"];
8          $mensagem = $_GET["msg"];
9
10         $sql = "INSERT INTO batepapo (data, usuario, mensagem)
11                 VALUES '$data', '$usuario', '$mensagem')";
12         $resultado = mysql_query($sql) or die (mysql_error());
13     }
14
15     $sql = "SELECT * FROM batepapo ORDER BY id DESC LIMIT 20";
16     $resultado = mysql_query($sql) or die (mysql_error());
17
18     while ($linha_bp = mysql_fetch_array($resultado)){
19         $data = date('d-m H:i',$linha["data"]);
20         $usuario = $linha["usuario"];
21         $mensagem = $linha["mensagem"];
22
23         echo "$data [$usuario] $mensagem <br />";
24     }
25 ?>
```

Figura 83 – msg.php, arquivo que grava e imprime as mensagens

Na Tabela 16 acompanha-se o código da Figura 83 comentado.

Tabela 16 – Código da Figura 83 comentado

Linha(s)	Comentário
2 e 3	Conexão ao banco de dados
5 a 13	Caso o usuário tenha enviado uma mensagem a assertiva `$_GET[“msg”]` será verdadeira e o código será executado. A data e hora do envio será armazenada na variável `data` (linha 6), o nome do usuário é capturado da variável de sessão (que é global enquanto o usuário estiver logado no site, linha 7) e a variável `mensagem` armazena o que foi digitado pelo usuário (linha 8). A `data`, `usuário` e `mensagem` são armazenados na tabela `bate-papo` (linhas 10, 11 e 12)
10 e 11	São consultadas as últimas 20 mensagens armazenadas.
18 a 24	O resultado da consulta é percorrido e impresso.

Ajax com jQuery

Na seção 9.6 é visto que jQuery facilita a implementação do código JavaScript, nessa seção veremos a estrutura que o jQuery oferece para implementar aplicações Ajax.

Funções de requisição

A seguir serão apresentadas as funções jQuery para realizar requisições Ajax, serão vistos sua sintaxe, aplicações e funcionamento. O coração do funcionamento do Ajax é o objeto `XMLHttpRequest` mas no jQuery a instanciação deste objeto é transparente para o programador, além de serem transparentes também alguns métodos que o Ajax utiliza na programação tradicional.

» **$.get**

A função $.get é uma requisição HTTP com o uso do método GET. Ela admite somente uma função que será chamada ao final da requisição. Se forem necessárias funções para tratamento de erros então é feito o uso da função $.ajax. A Figura 84 e Figura 85 apresentam a requisição HTTP utilizada no bate papo.

```
1  $.get('msg.php',
2        function(dado){ $('#chat').html(dado); },
3        'html');
```

Figura 84– Função $.get

A função $.get admite 4 parâmetros, na Figura 84 temos 3, e no código da Figura 85 temos todos os 4 parâmetros, sendo que o primeiro é sempre obrigatório. O código da Figura 84 é utilizado no exemplo do CMS do Portal de Receitas para a consulta das últimas mensagens adicionadas no banco de dados. O código da Figura 85 envia a mensagem que o usuário digita no bate papo, a armazena no banco de dados e por fim retorna as últimas mensagens dos usuários. A Tabela 17 apresenta o código da Figura 84 comentado.

Tabela 17 – Código da Figura 84 comentado

Linha(s)	Comentário
1	Função de requisição Ajax $.get possui no exemplo 3 parâmetros, o arquivo a ser requisitado, uma função para ser executada ao final da requisição e o tipo do dado que está transitando. O primeiro parâmetro, o url, é obrigatória na função $.get e está definida na linha 1. O URL do arquivo requisitado, no caso do exemplo é uma página com o resultado em HTML de uma consulta ao banco de dados das últimas mensagens enviadas pelos usuários.
2	Definição da função de retorno. A variável dado que é o parâmetro da função, recebe o conteúdo de retorno da requisição, no caso o html com as últimas mensagens do bate papo, que está sendo colocado no elemento DIV através da sintaxe $('#chat').html(dado).
3	Tipo de dado que está transitando na requisição, os valores possíveis para este parâmetro são: "XML", "html", "script", "json" ou "text"

Na Figura 85 o código da função $.get possui 4 parâmetros. O segundo parâmetro é um conjunto de pares variável/valor que é utilizado para enviar dados ao servidor..

```
1  var mensagem = $('#msg').val();
2
3  $.get('msg.php',
4        {msg: mensagem},
5        function(dado){ $('#chat').html(dado); },
6        'html');
```

Figura 85 – Função $.get enviando dado para o arquivo

Na Tabela 18 acompanhamos o código da Figura 85 comentado. Na Figura 86 temos o código que determina o comportamento do nosso bate papo.

Tabela 18 – Código da Figura 85 comentado

Linha(s)	Comentário
1	A variável mensagem captura o valor que o elemento de `id` igual a `msg` possui, no caso este elemento é um campo de texto no qual o usuário do bate papo digita a mensagem.
2	Definição da função de retorno. A variável `dado` que é o parâmetro da função, recebe o conteúdo de retorno da requisição, no caso o html com as últimas mensagens do bate papo, que está sendo colocado no elemento DIV através da sintaxe `$('#chat').html(dado)`.
3	Tipo de dado que está transitando na requisição, os valores possíveis para este parâmetro são: "`XML`", "`html`", "`script`", "`json`" ou "`text`"

```
1  $(document).ready(function(){
2      ler();
3      $('#msg').focus();
4      $('#ok').click(function(){
5          enviar();
6      });
7  });
8
9  var ler = function() {
10     $.get('msg.php',
11           function(dado){ $('#chat').html(dado); },
12           'html');
13     setTimeout('ler()', 2000);
14 }
15
16 var enviar = function() {
17     var mensagem = $('#msg').val();
18     $.get('msg.php',
19           {msg: mensagem},
20           function(dado){ $('#chat').html(dado); },
21           'html');
22     $('#msg').val('');
23     $('#msg').focus();
24 }
```

Figura 86 – Implementação do comportamento Ajax com jQuery do bate papo

» **`$.post`**

A função `$.post` é uma requisição HTTP com o uso do método `POST`. A sintaxe da função é similar a da função `$.get`, onde consta o termo `get` substituímos por `post`.

» **`$.getScript`**

A função `$.getScript` é utilizada para realizar requisições de arquivos JavaScript que se encontram em um domínio remoto (domínio distinto da página que faz a requisição).

A Figura 87 apresenta um documento HTML com um elemento `div`, que será estilizado através do script importado pela função `$.getScript`. O arquivo JavaScript para a qual a função `.getScript` faz requisição é um plugin que arredonda as bordas de alguns elementos HTML.

```
1  $.getScript('http://github.com/malsup/corner/raw/master/jquery.corner.js',
2              function() {
3                  $('div').corner();
4              });
```

Figura 87 – Função $.getScript

Na Tabela 19 são mostrados os comentários a respeito do código da Figura 87.

Tabela 19 – Código da Figura 87 comentado

Linha(s)	Comentário
1	O primeiro parâmetro é a requisição do arquivo JavaScript remoto. O arquivo é o *jquery.corner.js* hospedado em *http://github.com/malsup/corner/raw/master/jquery.corner.js*. É um plugin para estilização de cantos cuja documentação encontra-se em *http://jquery.malsup.com/corner/*.
2	Definição da função de retorno, que arredondará todos os elementos `div` da página.

» **`.load`**

O método `.load` é uma função jQuery simplificada que é utilizada para realizar requisições HTTP com o uso do método `GET` por padrão. Caso sejam passados dados pelo método a requisição assumirá um envio HTTP pelo método `POST`. Na Figura 88 há um exemplo de sintaxe do método `.load` que possui a mesma função da função `$.get` apresentada na Figura 84.

```
$('#chat').load('msg.php');
```

Figura 88 – Método .load

» **`$.ajax`**

A função `$.ajax` carrega dados do servidor utilizando uma requisição HTTP. As funções vistas anteriormente também podem ser feitas com essa função, porém pode ela pode ser configurada com muito mais parâmetros que aquelas.

A função `$.ajax` admite um parâmetro. Esse parâmetro é um objeto constituído de um conjunto de pares chave/valor que serão usados para iniciar e manipular a requisição.

Na tabela 20 temos algumas das possíveis chaves/valores do objeto que é o parâmetro da função `$.ajax`. No exemplo da Figura 89 utilizam-se algumas chaves, tais como `url`, `dataType`, `data`, `type`, `success` e `error`.

Tabela 20 – Algumas chave/valor que constituem o objeto da função $.ajax

Chave	Valor
`cache`	Default `true` que evita que a página requisitada vá para o cachê do navegador.
`complete`	Função chamada quando a requisição termina. Possui dois argumentos, o objeto `XMLHttpRequest` e uma string que define o status de como terminou a requisição.
`contentType`	String que descreve o tipo de conteúdo que está sendo enviado ao servidor, por exemplo "`text/xml`".
`data`	Os dados que serão enviados ao servidor.
`dataType`	O tipo de dado que se espera receber do servidor, os tipos possíveis são: `XML`, `html`, `script`, `json`, `jsonp` e `text`. Este tipo deve ser definido corretamente, caso contrário não será possível manipular os dados retornados.
`error`	Função chamada quando ocorre um erro na requisição. Possui dois argumentos, o objeto `XMLHttpRequest` e uma string que descreve o tipo de erro que ocorreu.
`sucess`	Função que é chamada quando a requisição foi executada com sucesso. Possui dois argumentos, os dados retornados e uma string descrevendo o status.
`type`	O valor default é `GET` e define o tipo de requisição. Os tipos normais são `GET` e `POST`, porém outros são possíveis como `PUT` ou `DELETE`.
`url`	O arquivo para qual a requisição é feita.

Na Figura 89 há um exemplo de sintaxe para a função `$.ajax` que realiza a mesma função do código da Figura 84, porém há o acréscimo de uma função que será requisitada caso ocorra qualquer erro durante a requisição.

```
$.ajax({
    url: 'msg.php',
    dataType: 'html',
    data: {msg: mensagem},
    type: 'GET',
    success: function(dado, statusReq){
        $('#chat').html(dado);
    },
    error: function(xhr, tipo){
        $('#chat').html('Erro: ' + xhr.status + ' - '
        + xhr.statusText + ' <br /> Tipo de erro: ' + tipo);
    }
});
```

Figura 89 – Função $.ajax

Requisições XML

A biblioteca jQuery oferece suporte para extrair conteúdo de um arquivo XML. A Figura 90 apresenta o arquivo *mousse.xml*, que armazena os dados de uma receita de cozinha de nosso CMS.

```
<?xml version="1.0" encoding="utf-8"?>
<receita>
    <titulo> Mousse de Maracujá </titulo>
    <imagem> imagens/mousse_maracuja.jpg</imagem>
    <ingredientes>
        <![CDATA[
        <ul>
            <li> 2 latas de leite condensado </li>
            <li> 4 maracujás </li>
        </ul>
        ]]>
    </ingredientes>
    <preparo>
        <![CDATA[
        <p>Coloque o leite condensado em uma vasilha e reserve.</p>
        <p>Retire a polpa dos maracujás e com a ajuda de uma peneira retire o suco.</p>
        <p>Junte o leite condensado e o suco do maracujá. Misture até ficar homogêneo.</p>
        <p>Leve a geladeira. Enfeite com sementes de maracujá, cerejas e folhas de hortelã.</p>
        ]]>
    </preparo>
</receita>
```

Figura 90 – Arquivo XML

Ao se extrair as informações do documento da Figura 90, serão apresentados os conteúdos a respeito de uma receita com título, imagem de apresentação, ingredientes e modo de preparo. A Figura 91 apresenta o código Ajax que extrai o conteúdo do arquivo XML e o imprime.

```
$.ajax({
    url: 'receita.xml',
    dataType: 'xml',
    type: 'POST',
    success: function(dado, textStatus){
        var html = '<h1>' + $(dado).find('titulo').text(); + '</h1>';
        html += '<p><img src="' + $(dado).find('imagem').text() + '" /> </p>';
        html += '<h2> Ingredientes </h2>';
        html += '<p>' + $(dado).find('ingredientes').text() + '</p>';
        html += '<h2> Modo de Preparo </h2>';
        html += '<p>' + $(dado).find('preparo').text() + '</p>';
        $('#rec').html(html);
    },
    error: function(xhr, erro){
        $('#rec').html(xhr.statusText + erro);
    }
});
```

Figura 91 - Requisição XML

Na Tabela 21 os comentários a respeito do código da Figura 91.

Tabela 21 – Código da Figura 91 comentado

Linha(s)	Comentário
2	Definimos o arquivo XML
3	Informamos que o tipo de dado que o servidor retornará será do tipo XML
4 a 13	Definição da função que será executado caso a requisição seja concluída com sucesso. A variável `dado` contém os dados enviados pelo servidor. A variável `html` conterá os dados que estão na estrutura XML formatados com marcações HTML que será exibido em um `<div>` com `id` igual a `rec` (linha 12 da Figura). A sintaxe `$('dado').find('titulo').text()` define o seguinte: percorra o conjunto de elementos de `$('dado')`, encontre os elementos `titulo` e deles extraia seus textos. Como o elemento `titulo` é único o resultado é um texto somente.
14 a 16	Função que será chamada quando ocorrer um erro na requisição;

Acessibilidade Ajax

Um dos grandes problemas no uso do Ajax é acessibilidade. Muitos navegadores suportam JavaScript, mas algumas tecnologias portáteis e navegadores somente textuais não suportam. O W3C (*World Wide Web Consortion*) diz no WCAG 1.0 (*Web Content Accessibility Guidelines* 1.0) que um site deve ser navegável mesmo quando o JavaScript do navegador estiver desativado.

Alguns desenvolvedores tem por prática utilizar requisições Ajax em todo site: em links, em menus etc. Mas isso não é necessário e também torna o site inacessível principalmente para os motores de buscas, pois esses sistemas não indexam dados que não estejam no código fonte do site. Dados requisitados através de requisições Ajax não são impressas no código fonte de páginas HTML.

É importante usar Ajax com moderação, e respeitando a acessibilidade. Um site que possui elementos Ajax deve funcionar em navegadores sem suporte a JavaScript, então ao desenvolver um site deve-se programar inicialmente sem JavaScript. Depois de concluído funcionalidades podem ser acrescentadas, porém, sempre respeitando questões como usabilidade e acessibilidade.

9.8. Notas Bibliográficas sobre a seção 9.8

A seção 9.8 se baseou no livro ***Ajax com jQuery – Requisições Ajax com a simplicidade de jQuery***, outro excelente livro que possui diversos exemplos e uma ampla documentação. O autor é também o conceituado Maurício Samy Silva, que também desenvolve o site Maujor (*www.maujor.com*), referência na área de desenvolvimento para a web.

Em *http://msdn.microsoft.com/pt-br/library/cc534581%28VS.85%29.aspx* é descrito algumas novas propriedades do Ajax criadas pela Microsoft no Internet Explorer 8, onde há comentários a respeito do objeto `XMLHttpRequest` já estar nativo nos navegadores Internet Explorer 7 e 8. Nesse endereço está sendo descrito algumas novas funções que estão implementadas no objeto, com novas possibilidades.

Existem muitos sites na web que possuem artigos sobre a acessibilidade em sites que utilizam Ajax, neste trabalho foram utilizados os sites *http://juliogreff.net* e *http://www.designacessivel.net.*

Referências Bibliográficas

Anderson, P. (2008). What is Web 2.0? - Ideas, technologies and implications for education. Disponível em <*www.jisc.ac.uk/techwatch*>. Acesso em: 10/04/2010.

Berners-Lee, T. (1999). "Weaving the Web: the past, present and future of the World Wide Web by its inventor". London : Orion Business.

Coelho, D. V. (2009). "Drupal x Joomla x Wordpress, qual o melhor CMS". Disponível em: <*http://www.portal2web.com.br/software-livre/drupal-x-joomla-x-wordpress-qual-o-melhor-cms.html*>. Acesso em: 15/07/2010.

Dall'Oglio, P. (2007). "PHP: Programando com Orientação a Objetos". Editora Novatec, São Paulo.

Design Acessível. (2007). "Acessibilidade com Ajax". Disponível em <*http://www.designacessivel.net/wiki/acessibilidade-com-ajax*>. Acesso em: 01/08/2010

Greff, J. (2006). "Ajax vs. Acessibilidade". Disponível em: <*http://juliogreff.net/ajax-vs-acessibilidade*>. Acesso em: 02/08/2010.

Kristaly, D. M., Sisak F., Truican, I., Moraru, S., Sandu, F. (2001). "Web 2.0 technologies in web application development".

Kostur, P. (2006). "Incorporating Usability into Content Management".

Microsoft. (2010). "Aprimoramentos de XMLHttpRequest no Internet Explorer 8". Disponível em <*http://msdn.microsoft.com/pt-br/library/cc534581%28VS.85%29.aspx*>. Acesso em 29/07/2010.

Montalvo, C. (2008). "Comezando MVC com php". Disponível em: <*http://www.calinsoft.com/2008/08/comenzando-mvc-con-php.html*>. Acesso em: 28/07/2010.

Nemetral. (2008). "A gentle introduction to MVC". Disponível em: <*http://nemetral.net/2008/07/31/a-gentle-introduction-to-mvc-part-1/*>. Acesso em: 10/07/2010.

O' Reilly, T. (2007). "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software". Communications & Strategies. Disponível em <*http://ssrn.com/abstract=1008839*>. Acesso em 01/4/2010.

Pereira, J. C. L., Bax, M. P. (2002). "Introdução à Gestão de Conteúdos".

Silva, M. S. (2008). "jQuery: A Biblioteca do Programador JavaScript", Editora Novatec, São Paulo.

Silva, M. S. (2009). "Ajax com jQuery: Requisições Ajax com a simplicidade de jQuery", Editora Novatec, São Paulo.