DESENVOLVIMENTO DE SOFTWARE PARA WEB

CAPÍTULO 2 - COMO APLICAR PADRÕES DE PROJETO PARA A CRIAÇÃO DE PÁGINAS DINÂMICAS?

Fernando Cortez Sica

Introdução

Neste capítulo vamos compreender questões relativas à aplicação de padrões de projeto para que possamos criar páginas dinâmicas. Para isso, começamos com o questionamento mais comum: a aplicação de padrões de projeto serve apenas para criarmos páginas dinâmicas? Não, padrões também servem para sistematizar a criação das páginas, tornando-as mais fáceis de serem construídas, para gastarmos menos tempo, facilitar a abstração das demandas e recursos disponíveis e necessários, além de diversas outras vantagens.

Por onde começar para aplicar os padrões? Primeiro, teremos que nos aprofundar nas questões de interatividade, ou seja, conhecermos como as páginas funcionam. Por exemplo, como os campos de um formulário interagem com o servidor. A comunicação entre um formulário e o servidor é direta? Para que possamos possibilitar a interação entre esses dois elementos, teremos que executar *scripts,* por exemplo, escritos usando a linguagem PHP. Então será necessário saber PHP? Sim, PHP, junto com JavaScript e HTML, constituem as mais importantes linguagens (de programação e, no caso do HTML, de marcação) para a codificação de páginas.

É possível dar a dinamicidade da página, por meio da seleção de folhas CSS apropriadas? Sim, veremos como fazer isso neste capítulo, assim como conversaremos sobre uma das formas de se garantir uma página dinâmica, usando padrões de projeto.

Sendo assim, caminharemos, nesse capítulo, por alguns conceitos para que possamos amarrá-los e possibilitar a você construir páginas dinâmicas, com formulários validados.

Vamos lá então? Bons estudos!

2.1 Validação de formulários utilizando JavaScript

Uma página *web* não tem somente o objetivo de fornecer informações ao usuário. Ela também deve possibilitar a interatividade, na qual o usuário fornece informações que serão processadas no servidor. Mas, tem como sabermos que o usuário forneceu corretamente as informações? Para que não sejam enviadas informações mal formatadas, vamos implementar, na página, mecanismos de validação das informações. Mas, a validação das informações ocorre somente a nível de páginas?

Para responder essa pergunta, vamos falar um pouco de padrões de projeto. Os padrões de projeto são modelos que possuem o objetivo de direcionar a modelagem de um projeto, ou, neste caso, de uma página. Basicamente, existe um padrão denominado como modelo das três camadas (3 *tiers* – 3 níveis), mas estudaremos isso mais adiante.

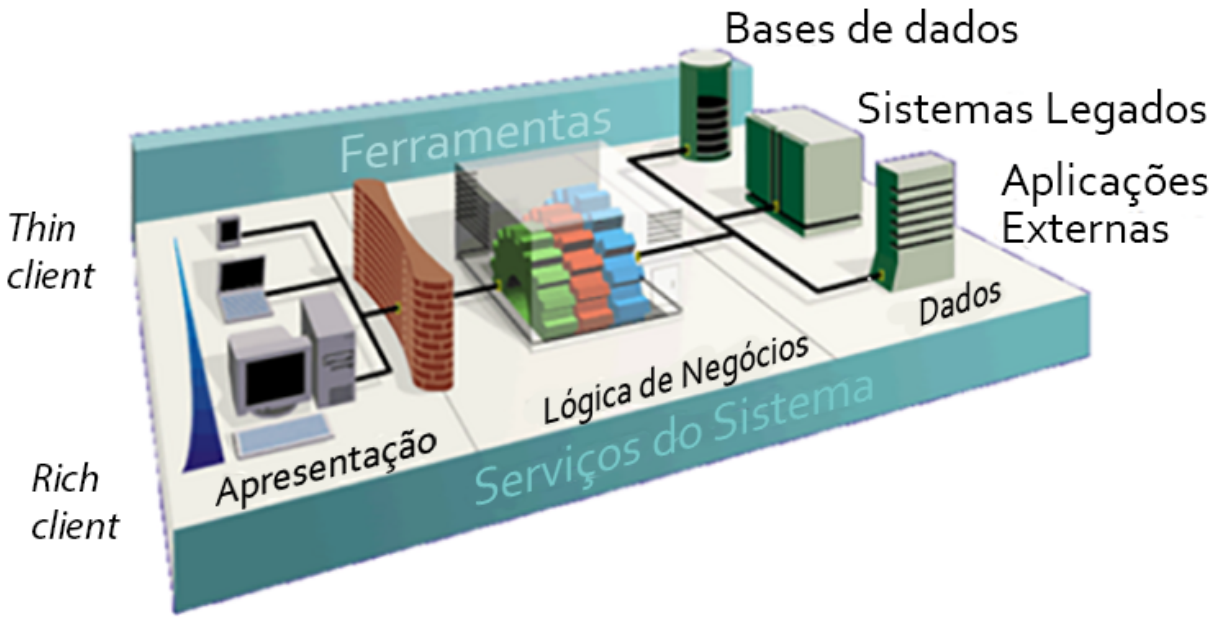

Figura 1 - Modelo de 3-camadas (3-tier): divisão entre os componentes de apresentação (presentation), lógica e regras de negócio (businness logic) e armazenamento de dados (data).
Fonte: CAMBIUCCI, 2008.

Na figura acima, cada camada desempenha uma funcionalidade, conforme a sumarização, que detalhamos a seguir (SOUZA, 2016).
Camada de apresentação (camada 1)
Consiste na camada mais próxima ao usuário – em tal camada encontra-se a interface que permite a interação entre o usuário e o sistema.
Regras de negócio (camada 2)
Contempla o processamento em si, ou seja, a manipulação das informações fornecidas pelo usuário e a montagem das informações que serão enviadas ao próprio usuário. Além de interfacear com a camada 1, também se comunica com a camada 3, de forma a poder solicitar a gravação ou recuperação das informações pelo banco de dados.
Camada de dados (camada 3)
Essa camada é responsável por abranger o sistema de gerenciamento de banco de dados.
Nesse capítulo, abordaremos a validação realizada na camada 1, ou seja, aplicação de regras por meio de *scripts,* para que as informações fornecidas pelo usuário possam ser validadas, a fim de serem enviadas à camada 2. Vamos então, aprender mais sobre validação e a biblioteca JQuery.

2.1.1 Validação

Como mencionamos a pouco, um sistema *web*, para ser mais organizado, tende a ser dividido em três camadas. Pensando em validação, vamos pensa em dois exemplos, a seguir, para que possamos diferenciar os tipos de validações, associando-os à camada 1 ou à camada 2.

- **Camada 1**

Nesta camada, a validação ocorre para que possamos certificar a corretude dos campos em relação à sua estrutura, como por exemplo, o formato correto de um CPF e, por exemplo, na reserva de uma passagem aérea, se a data da volta está posterior à data da ida.

- **Camada 2**
  Na camada de negócios, temos, por exemplo, a validação da idade do usuário. Lembrando que a faixa de idade pode ser específica para cada negócio/aplicação – por isso não é viável realizar essa validação junto à interface do sistema.

Falando especificamente sobre validação na camada 1, teremos a necessidade de incorporar um *script* junto ao código HTML. Esse script será ativado quando houver a manipulação do campo específico do formulário. O mais comum de se realizar a validação nos formulários é utilizando a linguagem de *script* JavaScript. Para tanto, o *script* deverá estar delimitado pelas *tags* de marcação de início <script> e de fechamento de bloco </script>.
Mas como usar o *script* para validação? No exemplo da figura a seguir, vamos começar a compreender como, finalmente, validar um formulário.

```
<!-- Numeração das linhas como comentários -->
<!--01--><html>
<!--02--> <head>
<!--03-->   <script language="JavaScript">
/* 04 */     function validaForm(f)
/* 05 */      {
/* 06 */       var reg = /^\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/
/* 07 */       if(f.nome.value == "" || f.nome.value == null ||
/* 08 */          f.nome.value.lenght < 3)
/* 09 */        {
/* 10 */          alert("Nome inválido.");
/* 11 */          f.nome.focus();
/* 12 */          return false;
/* 13 */        }
/* 14 */       if(!(reg.test(f.email.value)) ||
/* 15 */           f.email.value == null)
/* 16 */        {
/* 17 */          alert("E-mail inválido.");
/* 18 */          f.email.focus();
/* 19 */          return false;
/* 20 */        }
/* 21 */      }    //final validaForm
/* 22 */   </script>
<!--23-->   </head>
<!--24-->   <body>
<!--25-->     <form name="form1" method="post"
<!--26-->       onSubmit="return validaForm(this);">
<!--27-->         <p>Validação de formulário</p>
<!--28-->         <p>Nome: <input type="text" name="nome"></p>
<!--29-->         <p>Email: <input type="text" name="email"></p>
<!--30-->         <p><label><input type="submit" name="submeter"
<!--31-->                          value="Enviar"></label></p>
<!--32-->     </form>
<!--33-->   </body>
<!--34--></html>
```

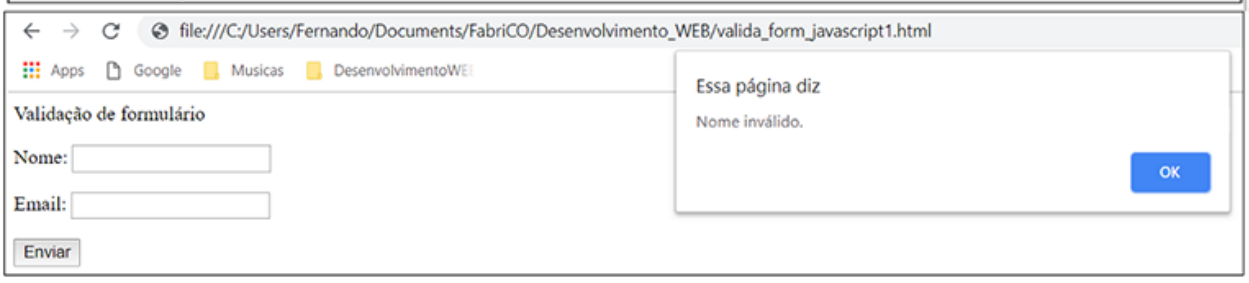

Figura 2 - Exemplo de codificação JavaScript para validação do formulário. Na figura, a parte superior contém o código e, na parte inferior, a janela aberta pela sua execução.
Fonte: Elaborado pelo autor, 2018.

Na figura acima, a cor vermelha (linhas 4 a 22) destaca o script escrito em JavaScript. Convém ressaltar que os scripts deverão ser colocados dentro da seção de cabeçalho da página (delimitada pelas tags <head> e </head>).
Na linha 3, encontramos a marcação de início de *script* (<script>), tendo como parâmetro language – informando ao navegador que será utilizada a linguagem JavaScript. A finalização do bloco de *script* encontra-se na linha 22 – explicitado pela *tag* de marcação de encerramento </script>. A partir da marcação de início de *script*, o navegador interpretará, de forma apropriada, a codificação da linguagem do *script* – sendo assim, inclusive, a notação de comentários deve seguir o formato imposto pela linguagem (no exemplo da figura acima, entre as linhas 04 e 22 usa-se "//" como comentários e não "<!-- ... -->" como no HTML.
Ainda no exemplo contido na figura anterior, temos o formulário, sendo passado como parâmetro à função "ValidaForm"- na figura, marcado na com azul. O formulário recebido pela função remete ao parâmetro "*this*" referenciado na linha 26. Toda a referência aos campos do formulário é usada como itens de estruturas de dados como, por exemplo, na linha 07, em que temos a utilização do item "nome". Tais campos devem seguir o parâmetro *name* pertencente à *tag* <input> – no caso, codificadas nas linhas 28 e 29. As cores diferenciadas permitem, na figura, diferenciar a utilização dos itens dentro do código do *script.*
Nas linhas 10 e 17, temos a função *alert* cujo objetivo é abrir uma caixa de diálogo para que o usuário seja avisado sobre a informação inconsistente. Quando a informação é inconsistente, ativa-se o campo correspondente à inconsistência para que o usuário possa entrar com a informação de forma correta – linhas 11 e 18 (ativação do método *focus*).
Finalmente, nas linhas 12 e 19, temos o retorno negativo da função, para que o evento de submissão seja cancelado, ou seja, as informações não sejam enviadas ao servidor.

VOCÊ QUER LER?

JavaScript é uma ferramenta bastante poderosa para a manipulação da identidade visual de uma página. Além de seu poder, uma outra vantagem consiste no fato de que a comunidade JavaScript é grande, produzindo, continuamente, vários trechos de código que poderão ser aproveitados em suas páginas (MOZILLA, 2018). Para que você possa iniciar no JavaScript, sugerimos que você acesse: <https://developer.mozilla.org/pt-BR/docs/Aprender/Getting_started_with_the_web/JavaScript_basico>.

Segundo Miletto e Bertagnolli (2014), há a possibilidade de realizar a validação pelo uso de expressões regulares, como usado no código da figura anterior para a validação do *e-mail*. Mas, o que vem a ser uma expressão regular? Define-se, como expressão regular, a representação de um padrão textual (EIS, 2016).
Ainda referenciando a figura anterior, foi colocada, na linha 06, um exemplo de uma expressão regular para a validação de *e-mail* (existem outras versões cobrindo uma gama maior de estruturação de *e-mails*). Em JavaScript, a expressão regular é iniciada por uma barra ("/").

VOCÊ SABIA?

Expressões regulares fazem parte de um tema muito importante em relação à verificação de formulários. Com o padrão representado pela expressão regular pode-se avaliar a corretude de sintaxe do valor inserido pelo usuário (LINS, 2007). Para ler mais sobre expressões regulares, acesse o artigo: <https://www.devmedia.com.br/iniciando-expressoes-regulares/6557>.

Por fim, a validação é feita pela utilização de máscaras. As máscaras auxiliam no preenchimento do campo, pois formata o campo em tempo de digitação. Além disso, esse tipo de validação tem condições de ser feita assim que cada caractere for digitado (MILETTO; BERTAGNOLLI, 2014). Para ilustrar a utilização de máscara, suponha a codificação ilustrada na figura a seguir.

```
<html>
<head>
  <title>Máscara CEP</title>
  <script language="JavaScript">
    function mascara(campo, mascara)
      {
/*01*/  if(event.keyCode<48 || event.keyCode>57)
        {
/*02*/    event.returnValue=false;
/*03*/    alert("Digite apenas números");
/*04*/    return false;
        }
/*05*/  var tam = campo.value.length;
/*06*/  var saida = mascara.substring(1,0);
/*07*/  var txt = mascara.substring(tam)
/*08*/  if (txt.substring(0,1) != saida)
        {
/*09*/    campo.value += txt.substring(0,1);
        }
      }
  </script>
  <body>
    <form action="#" method="post" onSubmit="return
     VerificaPreenchimento(this);">
      <p>CEP:<input label="CEP" type="text" name="cep"
/*10*/      onkeypress="mascara(this, '##.###-###')" maxlength="10"></p>
      <input type="submit" name="submeter" value="Enviar">
    </form>
  </body>
</html>
```

Figura 3 - Exemplo da utilização de máscara para facilitar o preenchimento dos campos pelo usuário. Além da máscara, no código é feita a verificação a cada tecla pressionada.
Fonte: Elaborada pelo autor, 2018.

Em relação à figura acima, temos dois aspectos distintos para a validação: a verificação a cada tecla pressionada (linhas 01 a 04) e o preenchimento automático do campo (linhas 05 a 09). Esses dois aspectos são ativados no momento de pressionamento de uma tecla pelo evento *onkeypress* da *tag* <input> (linha 10).
Na linha 01, da figura anterior, temos o *event.keyCode*. Esse evento corresponde à tecla pressionada. Os valores comparados, 48 e 57, correspondem aos valores ASCII (*American Standard Code for Information Interchange* – código padrão americano para intercâmbio de informações) das teclas 0 e 9, respectivamente.
Na linha 02, foi inserido o valor *false* para o campo *evento.returnValue*, para limpar a tecla digitada, a fim de que não seja carregada no campo sob digitação.

VOCÊ SABIA?

Você sabia que, para facilitar a validação com expressões regulares, o HTML5 adicionou, à *tag* <input>, o parâmetro *pattern* (padrão)? Para saber mais detalhe, acesse o artigo "Validação de formulários com HTML5" (FABENI, 2014): <https://tableless.com.br/validacao-de-formularios-com-html5/>.

Quando houver a necessidade de aproveitar o mesmo código em várias páginas, pode-se criar um arquivo externo contendo o código do *script* (MILETTO; BERTAGNOLLI, 2014). Nessa forma, define-se, fora da seção do cabeçalho, a referência ao arquivo externo (podendo ser um arquivo local, ou localizado em uma máquina remota). A sintaxe consiste em:
<script src="arquivo.js"></script>
No trecho de código acima, como mencionado anteriormente, o "arquivo.js" pode ser o nome de um arquivo ou a URL que contém a função cujo nome foi passado pelo parâmetro *onSubmit* da *tag* <form>.
Existe uma maneira de facilitar a criação de *scripts* JavaScript? Veremos, a seguir, uma das formas para fazer isso: a utilização do *framework* JQuery.

2.1.2 JQuery

Como mencionado a pouco, o JQuery consiste em um *framework* para JavaScript. Dessa forma, a criação de páginas que demandem a codificação JavaScript, fica facilitada com a utilização do JQuery. O JQuery tem como vantagens (TEIXEIRA, 2013):
Incorporar à página, efeitos tais como animações.
Utilizar *plugins* já desenvolvidos pela comunidade JQuery (JQuery segue o princípio de código aberto).
Exporta funcionalidades para serem usadas em conjunto com CSS.
Redução da codificação da página com a simplificação de escrita de funções JavaScript.
Inicialmente, para que possamos utilizar o JQuery em nosso projeto, temos que incorporá-lo à nossa página HTML com a *tag* <script> inserida no cabeçalho (entre os marcadores <head> e </head>):
<script type="text/javascript" src="jquery.js"></script>
De acordo com (JQUERY), a utilização do JQuery baseia-se na referenciação dos elementos HTML/CSS que ocorre da seguinte forma:
$(elemento_HTML_CSS)
Neste caso, o *elemento_HTML_CSS* consiste em um elemento HTML ou um seletor CSS – seletor tipo *TAG, ID* ou classe. Mas, como efetivamente utilizar o JQuery? Vamos responder a essa pergunta com o exemplo inserido na figura a seguir.

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Introdução jQuery </title>
    <!-- Referencia a biblioteca jquery (JS) -->
    <script type="text/javascript" src="jquery.js"></script>
    <!-- Funcionalidades da página -->
    <script type="text/javascript">
        $(document).ready(function(){
alert("página carregada")
        });
    </script>
</head>
<body>
    <h1>Página demonstração</h1>
    <h2>jQuery biblioteca javascript.</h2>
</body>
</html>
```

Figura 4 - Exemplo básico de utilização do framework JQuery.
Fonte: TEIXEIRA, 2013.

Na figura acima, foi utilizada uma referência ao próprio documento, no qual uma janela de diálogo é aberta em seu momento de carga – por intermédio do código *$(document).ready.* Ainda em relação a essa linha, podemos salientar que a implementação feita no momento de carga da página é realizada *inline,* ou seja, logo à frente do item *"function()".*
Assim, dessa forma, a implementação de eventos associados aos elementos HTML, ou CSS, torna-se mais fácil, além de se ter inúmeras outras vantagens, como por exemplo, uma melhor manipulação do AJAX (*Asynchronous JavaScript and XML* – JavaScript e XML assíncronos). Ajax consiste em uma plataforma que exposta diversas funcionalidades para a implementação de páginas.

2.2 Associação de CSS e JavaScript ao HTML

Em uma página HTML, as configurações de aparência visual podem ser baseadas na utilização de folhas de estilo (CSS). Você já viu como manipular o CSS de forma estática, ou seja, até aqui, a configuração do estilo era realizado no momento das definições da página ou dos elementos nela contidos. Mas, como podemos fazer algo mais dinâmico?
Respondendo positivamente à sua pergunta, uma primeira coisa a ser pensada é a possibilidade de criar páginas responsivas. Vamos relembrar: páginas responsivas são aquelas capazes de se adaptarem às variações de dispositivos nos quais a página será exibida – isso diante da diversidade de aparelhos que poderão ser utilizados (tais como *notebooks, smartphones* e *tablets*).
Porém, para que possamos construir uma página responsiva, devemos obter informações a respeito do ambiente, como por exemplo, o tipo do navegador e qual o tamanho de tela que o dispositivo apresenta. Para saber tais informações, a figura, a seguir, exemplifica um código que contempla essa questão.

```
<html>
  <head>
    <title>Inicia Página</title>
    <script language="JavaScript">
      function IniciaPagina()
        {
          setInterval(MudaCor, 1000);
          var texto ="Você está usando"+navigator.appVersion+" usando uma tela
de "+screen.width+"x"+screen.height;
          alert(texto);
        }   //final IniciaPagina

     function MudaCor()
       {
         if(document.body.style.backgroundColor == "white")
          {
          document.body.style.backgroundColor="blue";
          }   //final if...
        else
          {
          document.body.style.backgroundColor="white";
          }   //final else...
       }   //final MudarCor
    </script>
  </head>
  <body onload="IniciaPagina()">
     <p>Apenas um exemplo para o onload e setInterval...</p>
  </body>
</html>
```

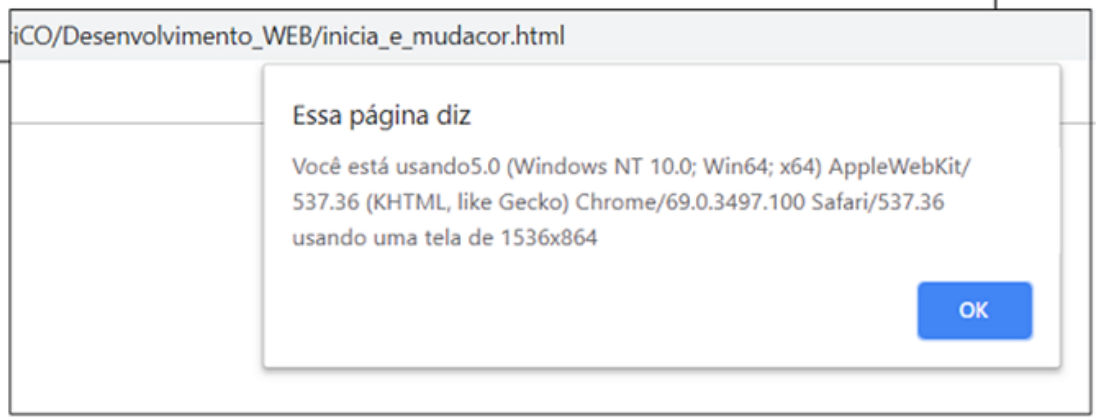

Figura 5 - Exemplo de código para detectar a configuração do sistema (navegador e tamanho da tela do dispositivo) e para mostrar o uso da função setInterval.
Fonte: Elaborada pelo autor, 2018.

Na figura acima, nos deparamos com dois trechos de código, diferenciados pelas cores vermelha e azul. O trecho em vermelho é responsável pela execução do código assim que a página é carregada. A função JavaScript é passada como argumento ao parâmetro *onload* – localizado no início da área delimitada pelas tags </body> e < /body>.

Ainda em relação à codificação marcada na cor vermelha da figura acima, temos, na função intitulada como *IniciaPágina*, a utilização do método *setInterval* para que, a cada 1s (1000ms), ocorra a troca da cor de fundo da página. A função responsável pela troca da cor da página está representada, no código, em azul, que será detalhada em breve. Como parâmetros do *setInterval*, temos a referência da função que será executada a cada segundo (no caso, a função *MudaCor*) e o intervalo de tempo entre as execuções da referida função, caso você queira que a temporização ocorra somente uma vez, como, por exemplo, criar um evento de *timeout*, você poderá utilizar o método *setTimeout()*. O outro objetivo do trecho vermelho consiste em coletar as informações do ambiente, navegador e dimensões da tela. No caso, as informações coletadas pelos atributos *navigator. appversion, screen.height* e *screen.width* são, posteriormente, exibidas ao usuário pela abertura de uma caixa de diálogo feita pelo método *alert*. Porém essas informações poderiam ser utilizadas para, por exemplo, o

redimensionamento de figuras e o reposicionamento ou a reconfiguração de campos . Tal verificação dos atributos CSS é extremamente útil pois permite coletar valores que podem sofrer variações em função das diversas implementações dos navegadores nos vários ambientes nos quais a página poderá ser aberta.

Por sua vez, o trecho marcado em azul refere-se à alteração de cor da página. Para tanto, é usado o campo *backgroundColor* pertinente ao estilo (*style*) do corpo da página. Esse campo pode ser usado para se verificar a cor corrente quanto para setar a cor desejada.

Voltando à página responsiva, falamos que uma das soluções é redimensionar ou reposicionar os elementos da página. Na figura anterior, já começamos a falar um pouco sobre isso quando as informações do ambiente foram detectadas e quando o estilo da página foi alterado, interagindo o JavaScript com o CSS.

Vamos falar mais um pouco sobre essa interação do JavaScript com o CSS, analisando a figura a seguir.

```
<html>
  <head>
   <title>Busca</title>
   <script language = "JavaScript">

        function valida()
          {
              document.formulario.busca.style.borderColor =  "#FFFFFF";
              document.formulario.busca.style.backgroundColor = "#FFFFFF";
              if (document.formulario.busca.value.length < 3)
               {
                 alert("Digite ao menos 3 letras ...");
                 document.formulario.busca.style.borderColor = "#FF4500";
                 document.formulario.busca.style.backgroundColor = "#FFFFE0";
                 document.formulario.busca.focus();
                 return false;
               }
              return true;
            }

           var entrou = 0;
           var saiu = 0;

           function Entrou()
              {
                entrou+=1;
                var texto = "Entrou = "+entrou+"   Saiu = "+saiu";
                document.getElementById("info").innerHTML = texto;
                document.getElementById("div1").style.background = "red";
              }
           function Saiu()
              {
                saiu+=1;
                var texto = "Entrou = "+entrou+"   Saiu = "+saiu";
                document.getElementById("info").innerHTML = texto;
                document.getElementById("div1").style.background = "white";
              }

  </script>
</head>
<body>
       <form name = "formulario" action="busca.php" onSubmit="return valida(); " >
              Procurar: <input type = "text" name="busca" required
                                onmouseenter="Entrou()"  onmouseleave="Saiu()" ><br>
              <input type="submit">
            </form>
           <div id="div1"   style="max.max-width: 100vw; width:150px;">
                 <span id="info">Entradas e saídas</span>
           </div>
   </body>
</html>
```

Figura 6 - Exemplo de código para modificar atributos e conteúdo de elementos da página usando JavaScript associado a CSS.
Fonte: Elaborada pelo autor, 2018.

Na figura acima, temos o código para alteração de elementos da página, usando uma interação entre JavaScript e CSS. Observando sequencialmente o código a partir da entrada <body>, temos o primeiro ponto a ser mencionado: a referência da função valida() ao parâmetro *onSubmit*. Analisando o código associado à função, encontramos linhas que visam alterar as cores de borda e de fundo da caixa de texto. Para tanto, foram selecionados campos pertencentes ao *style* do CSS dentro do código JavaScript.

Continuando na figura acima, encontramos a associação das funções Entrou() e Saiu() aos eventos *onmouseenter* e *onmouseleave* da caixa de diálogo textual, respectivamente. Esses eventos permitem que funções sejam executadas em todos os momentos nos quais o *mouse* entra em uma certa região (*onmouseenter*) e nos momentos de saída da referida região (*onmouseleave*).

Em ambas as funções (Entrou() e Saiu()) temos alterações tanto nas cores de fundo da caixa de diálogo quanto em seu conteúdo. Como mencionado anteriormente, alterações na cor de fundo podem ser alcançadas com a modificação de valor do campo *style.background.* Por sua vez, alterações de conteúdo podem ser realizadas por intermédio da alteração do conteúdo do campo *innerHTML.*

Convém mencionar que, para realizar as alterações citadas, utilizou-se a associação do elemento sob alteração por intermédio do método *getElementById*. Tal método facilita a codificação, pois pode-se passar o identificador como parâmetro da função JavaScript. Caso haja a necessidade de associar pela classe ou pelo nome, pode-se usar, respectivamente, os métodos *getElementByClassName* e *getElementByName.*

De acordo com Walsh (2014), existem diversas maneiras de interação entre o JavaScript e o CSS. Dentre as quais podemos destacar: obtenção das propriedades de pseudo-elementos, adição e remoção de regras CSS e realizar a carga de arquivos CSS.

A obtenção de propriedades de pseudo-elementos funciona de forma análoga à obtenção das propriedades de estilo (*style*) (WALSH, 2014). Por exemplo, para coletar a cor do pseudo-elemento *.element:before*, podemos usar a seguinte linha de código:

```
var cor=window.getComputedStyle(document.querySelector('.element'),
':before').getPropertyValue('color');
```

Em relação à linha de código acima, para coletar informações a respeito de outras propriedades, basta trocar o nome passado como parâmetro à *getPropertyValue* – por exemplo, caso necessite acessar o conteúdo, ficaria: *getPropertyValue('content').*

Agora, para a adição e remoção de regras CSS, pode-se usar os métodos *sheet.insertRule* ou *sheet.addRule*. O momento para o uso de uma ou de outra está relacionado com a compatibilidade frente ao tipo e versão do navegador. Para usar a inserção de regras é conveniente realizar um teste antes, pela referência do próprio método como parâmetro ao teste. Por exemplo:

```
if(obj.insertRule){obj.insertRule(seletor,def_do_estilo,índice);}
```

Na linha acima, temos:

- obj: refere-se à uma folha de estilo (*sheet*) na qual se deseja adicionar a regra;
- seletor: nome do novo seletor CSS;
- def_do_estilo: especifica as definições para o novo estilo adicionado;
- índice: esse parâmetro é opcional e serve para indicar a posição da nova regra dentro da coleção de regras.

Você, além de adicionar regras à uma folha de estilo (*sheet*), poderá, também, criar uma nova folha de estilo com o método *document.createElement("style")* e, depois, anexar a folha recém-criada ao cabeçalho, invocando o método *head.appendChild*(folha_recém_criada).

Caso necessário, você pode usar o processo de criação de uma nova folha de estilo para carregar um arquivo CSS proveniente de uma máquina remota. Para tanto, deve seguir os seguintes passos:

a

Criar um elemento do tipo *'link':*

```
var meu_link = document.createElement('link');
```

b

Setar o atributo "*type*" do elemento recém-criado:

```
meu_link.setAttribute('type',"text/css");
```

c

setar o atributo "*rel*" do elemento recém-criado:

```
meu_link.setAttribute('rel',"stylesheet");
```

d

Setar o atributo "*href*" do elemento recém-criado:
meu_link.setAttribute('href',url_do_arquivo_CSS);
e
Anexar o elemento recém-criado:
document.getElementsByTagName("head").item(0).appendChild(meu_link).
Assista ao vídeo abaixo e aprenda mais sobre o tema.
https://cdnapisec.kaltura.com/p/1972831/sp/197283100/embedIframeJs/uiconf_id/30443981/partner_id/1972831?iframeembed=true&playerId=kaltura_player_1547283050&entry_id=1_ugwuerou
A interação entre CSS e JavaScript é muito rica em recursos. Apresentamos, aqui, uma pequena fatia do que você poderá fazer em sua página para deixá-la mais dinâmica e atrativa, mas você poderá aplicar muito mais.
Até o momento, conversamos sobre os aspectos visuais da página, ou seja, o *frontend* (*layout* da página) e formulários para a interação com o usuário. Mas, como enviar, efetivamente, as informações ao servidor? Ao abordarmos o assunto relacionado aos formulários, nos deparamos com os métodos *get* e *post*. Como tratar tais métodos? Como introduzir a linguagem PHP à sua página? Esse o assunto que veremos a seguir.

2.3 Introdução ao Desenvolvimento *Web* com PHP

Como já mencionamos, uma página pode conter elementos que serão responsáveis por fazer a interação com o usuário (*frontend*) e elementos que estarão vinculados à máquina servidora (*backend*). Como já conversamos, no *frontend* encontramos, por exemplo, a linguagem de marcação HTML, configurações CSS e scripts JavaScript. No *backend* encontramos linguagens de programação, tais como o PHP.

VOCÊ QUER LER?

Além do código HTML, a sua página poderá ser associada à código escrito em PHP. Dessa forma, PHP poderá estar presente não somente quando algo deverá ser processado e despachado, ou carregado, do servidor. Sendo assim, é importante conhecer a linguagem PHP. Para saber mais, recomendamos o artigo "PH tutorial" (ANTÔNIO, 2015), que aborda uma introdução para a programação em PHP: <https://www.devmedia.com.br/php-tutorial/32540>.

Mas, como incorporar o PHP às páginas? Como desenvolver o código, associado ao botão de submissão nos formulários? O que podemos fazer com o PHP, além do código para submissão? Responderemos a essas questões a seguir, quando conversaremos mais especificamente sobre a linguagem PHP no desenvolvimento das páginas.

VOCÊ QUER VER?

A utilização de PHP está presente em várias frentes em relação ao desenvolvimento de páginas, desde o *layout* até a interação com a máquina servidora. Desta forma, para que possamos desenvolver nossas páginas e testá-las, temos que, dentre outras coisas, instalar o PHP. Recomendamos um vídeo tutorial, para você aprender como estruturar o PHP para criar um *site* responsivo completo (HXTUTORS, 2016): <https://www.youtube.com/watch?v=_rzQPa_ZwX4>.

Para começar, um script PHP pode atuar nos dois sentidos: recebendo informações da página, para que sejam processadas e, porventura, acionar a máquina servidora; ou gerar informações para que sejam enviadas à página (MILETTO; BERTAGNOLLI, 2014). Resumidamente, algumas funcionalidades básicas da utilização de PHP junto às páginas HTML, são listadas a seguir:

- processar informações dos formulários, assim que o botão de submissão for acionado;
- intercalar HTML com PHP para facilitar, por exemplo, a coleta de informações de configuração do ambiente;
- executar o script PHP localmente para, por exemplo, enviar um *e-mail*;
- geração de códigos tanto HTML quanto JavaScript, assim como gerar documentos, figuras e demais elementos que poderão ser exibidos na página ou, simplesmente, armazenados na máquina servidora.

Porque optar pelo uso do PHP? A opção de utilização do PHP é baseada nas seguintes características favoráveis (MILETTO; BERTAGNOLLI, 2014):

- exporta suporte a uma vasta gama de gerenciadores de banco de dados, dentre os quais podemos destacar: dBase, MySQL, PostgresSQL, IBM DB2 e Sybase;
- suporte a LDAP, IMAP, SNMP, POP3, SSH2;
- possibilidade de manipular *sockets*;
- boa interatividade com o servidor APACHE.

Para que possamos ver, por exemplo, a manipulação de formulários com PHP, transcrevemos, na figura abaixo, um código disponibilizado em Teixeira (2015), que implementa o envio de e-mails cujo destinatário e conteúdo são fornecidos, pelo usuário, por meio de caixas de texto.

```html
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf8">
        <title>Contato</title>
    </head>
    <body>
        <form action="mail_send.php" method="post">
            <fieldset>
                <label for="email">E-mail: </label>
                <input required name="email" type="email">
            </fieldset>
            <fieldset>
                <label for="mensagem">Mensagem: </label>
                <textarea required name="mensagem"></textarea>
            </fieldset>
            <fieldset>
                <button type="submit">Enviar</button>
            </fieldset>
        </form>
    </body>
</html>
```

(a) index.php

```html
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf8">
        <title>Erro</title>
    </head>
    <body>
        <h1>Erro</h1>

        <hr>

        <p>Houve um erro no envio do e-mail.
            <ahref="index.php">Tentar novamente</a>.</p>
    </body>
</html>
```

(b) mail_error.php

```html
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf8">
        <title>Sucesso</title>
    </head>
    <body>
        <h1>Sucesso</h1>

        <hr>

        <p>O e-mail foi enviado com sucesso.</p>
    </body>
</html>
```

(c) mail_ok.php

```php
<?php

function pegaValor($valor) {
    return isset($_POST[$valor]) ? $_POST[$valor] : '';
}

function validaEmail($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL);
}

function enviaEmail($de, $assunto, $mensagem, $para, $email_servidor)
{
    $headers = "From: $email_servidor\r\n" .
               "Reply-To: $de\r\n" .
               "X-Mailer: PHP/" . phpversion() . "\r\n";
    $headers .= "MIME-Version: 1.0\r\n";
    $headers .= "Content-Type: text/html; charset=ISO-8859-1\r\n";
  mail($para, $assunto, nl2br($mensagem), $headers);
}

$email_servidor = "email@servidor.com";
$para = "seu@email.com";
$de = pegaValor("email");
$mensagem = pegaValor("mensagem");
$assunto = "Assunto da mensagem";

if (validaEmail($de) && $mensagem) {
    enviaEmail($de, $assunto, $mensagem, $para, $email_servidor);
    $pagina = "mail_ok.php";
} else {
    $pagina = "mail_error.php";
}

header("location:$pagina");

?>
```

(d) mail_send.php

Figura 7 - Código PHP vinculado à um formulário HTML. Em (a) temos a própria página de entrada cujo formulário evoca o script contido em (d) que usa códigos auxiliares (b) e (d).
Fonte: TEIXEIRA, 2015.

Na figura acima, temos o código para manipular o envio de e-mails separado em quatro blocos, assim identificados:

- *index.php*

  Corresponde à própria entrada da página. Esse arquivo é evocado pelo servidor de páginas. Para tanto, deverá ser instalado um servidor de páginas como o Apache e o Chrome Server. Temos, nesse arquivo, toda a configuração visual da página assim como a implementação do formulário. O formulário está associado à função "mail_send.php", contido no bloco (d) da figura acima

- *mail_error.php* e *mail_ok.php*

  Esses arquivos PHP implementam apenas a interações com o usuário, no caso de erro, ou sucesso de envio, respectivamente.

- *mail_send.php*

Esse bloco representa a parte principal para o envio do e-mail, que explanaremos a seguir.

Como mencionamos, bloco *mail_send.php*, identificado por (d), contém a codificação responsável pelo envio do e-mail. Para tanto, temos a entrada do script identificada pela instrução *if (validaEmail($de) && $mensagem)*. No trecho que contém essa linha de código, temos as chamadas para as funções de validação (*validaEmavalidaEmail*) e o envio (*enviaEmail*) do e-mail. Caso a validação tenha ocorrido com sucesso, a referência do script que contém a mensagem de sucesso (*mail_ok.php*) será atribuída à variável $pagina para que possa ser aberta pela evocação do método *header(location).* Esse método é evocado quando a variável referenciar à página de insucesso da validação (*mail_error.php*).
Neste código, temos uma importante passagem denotada pela linha:
return isset($_POST[$valor]) ? $_POST[$valor] : '';
Essa linha, pertence à função *pegaValor* que é evocada em dois momentos distintos: as instanciações da variável *$de* e da variável *$mensagem*. Na referida linha, temos a presença da variável *$_POST*. Mas, na definição do formulário, no arquivo *index.php*, não aparece a palavra post na instanciação do parâmetro método (*method*) do formulário (*form*)? Sim, essas duas ocorrências são interrelacionadas e veremos isso a seguir, quando abordaremos a variável *POST* e a variável *GET* – o outro valor que poderemos associar ao parâmetro *method.*

2.3.1 Variáveis POST

Já conversamos um pouco sobre a forma de como um formulário passa seus parâmetros para um script PHP responsável pelo processamento das informações fornecidas. Na ocasião, mencionamos que é possível usar duas formas (métodos): *post* e *get.* Mas, como o *script* coleta de forma correta todos os itens explicitados no formulário? Para responder a isso, vamos falar um pouco de variáveis superglobais. Variáveis superglobais são aquelas que pertencem ao próprio PHP, ou seja, são variáveis nativas – poderão ser acessadas dentro de qualquer escopo. Como exemplos de variáveis superglobais temos: *$_POST, $_GET* e *$_SERVER.*
Mais especificamente falando sobre a variável *$_POST*, de acordo com Arrigoni (2013), a coleta das informações contidas na variável *$_POST* é realizada da seguinte maneira, por exemplo:
$variável = $_POST[“nome_do_campo”];
No caso, o item *nome_do_campo* denota o valor passado ao parâmetro *name* da *tag* <input>, presente no formulário. Salientamos que nem todas as informações passadas à variável *$_POST* foram fornecidas pelo usuário. Pode-se definir algumas variáveis ocultas (*hidden*) de forma que o usuário não poderá alterá-las. Por exemplo, um campo interno do sistema, tal como o número de cadastro de um usuário. Na listagem presente na figura a seguir, temos essa situação.

arquivo html

```
<form action="hidden.php" method="post">
<input type=hidden name=escondido value="valor do escondido">
<input type=hidden name=id value="111">
<input type=submit>
</form>
```

arquivo hidden.php

```
<?php
echo "Campo Hidden: " . $_POST["escondido"];
echo "<br>Oi, seu ID é: " . $_POST["id"];
?>
```

Figura 8 - Definição e utilização de variáveis ocultas ao usuário. No caso, temos dois campos ocultos denominados como escondido e id.
Fonte: ARRIGONI, 2013.

No caso da figura acima, temos duas variáveis escondidas: *escondido* e *id.* Tais variáveis poderiam ser utilizadas normalmente entre os diversos outros tipos de entrada (*input*). A diferenciação é realizada simplesmente por meio de seus nomes atribuídos ao parâmetro *name.*

2.3.2 Variáveis GET

Como já mencionado em outra oportunidade, o método *post* garante uma maior segurança pois os dados a serem transmitidos são encapsulados na mensagem enviada do cliente ao servidor. Por outro lado, o método *get* insere os dados a serem transmitidos na própria URL.
Porém, a forma de coleta das informações do método *get* é exatamente igual à coleta pelo método *post*. Sendo assim, a linha do *script* PHP responsável pela extração das informações é exemplificada a seguir:
$variável = $_GET[“nome_do_campo”];
É extremamente aconselhável que sejam utilizadas as variáveis superglobais que mencionamos e não as variáveis declaradas nos próprios formulários. Caso você for utilizar as variáveis declaradas no formulário (na forma, *$campo1*), deve-se garantir que a diretiva *register_globals* do PHP esteja setada para que as variáveis dos campos do formulário possam ser sincronizadas. Uma solução para garantir que as variáveis não estejam vazias é utilizar a função *import_request_variables* na introdução de seus códigos de *script*, para garantir que suas variáveis não permaneçam com valores nulos (vazios). A seguir, segue a sintaxe da função *import_request_variables* (ARRIGONI, 2013):
import_request_variables(“argumentos”)
Onde, argumentos podem ser:

- P: sincronizar variáveis relativas a *$_POST*;
- G: sincronizar variáveis relativas a *$_GET*;
- C: sincronizar variáveis relativas a *$_COOKIE.*

Sendo, assim, por exemplo, se tivermos:
import_request_variables("pc")
estaríamos referenciando as variáveis *post* e *cookies.*
Segundo Miletto e Bertagnolli (2014), as variáveis passadas pelo formulário, tanto no método *post,* quanto do método *get*, poderão ser acessadas, também, pela variável superglobal *$_REQUEST*. A variável mantém tanto a lista *$_POST* quanto a lista *$_GET*.

2.4 Padrão MVC com PHP

Como conversamos um pouco no início deste capítulo, para facilitar a modelagem do sistema, desenvolvimento e integração com as bases de dados, pode-se aplicar ferramentas e padrões computacionais (ou modelos computacionais). Um dos padrões existentes, e aqui abordado, consiste no "3-camadas" (3-tier). Um outro padrão amplamente utilizado é o MVC (*Model, View, Controller* – Modelo, Visão, Controlador).
Agora, vamos entender mais detalhes e vantagens de se utilizar padrões de projeto para que, depois, possamos nos aprofundar no padrão MVC.

2.4.1 Padrões de Projeto

Para o desenvolvimento de projetos de *software*, dentre os quais estão inclusos os projetos de páginas e *sites*, é interessante aplicar um padrão de projeto (*design pattern*). Dentre as vantagens, podemos destacar:

- reutilização de código, diminuindo o tempo e o esforço para o desenvolvimento de componentes de *software*;
- melhor estruturação do sistema, tornando-o mais legível, facilitando a abstração e manutenção;
- facilitar a portabilidade para outros ambientes computacionais pois, com a adoção de padrões, pode-se projetar o sistema em camadas funcionais, separando aquelas que são dependentes daquelas independentes do ambiente no qual o sistema será implantado.

Na prática, existem duas grandes famílias de padrões de projeto: GRASP (*General Responsibility Assignment Software Patterns* – Padrões de *Software* de Atribuição Geral de Responsabilidade) e GOF (*Gang of Four* – Bando dos quatro).
De acordo com Palmeira (2013), a característica básica do GRASP é a sua centralização em critérios de responsabilidade, ou seja, com o GRASP, facilita-se a atribuição de responsabilidade a classes e objetos que comporão o sistema computacional. Por sua vez, os padrões baseados em GOF, possuem outros níveis de abstrações para que seja possível atuar no projeto dos sistemas computacionais.
Para possibilitar vários níveis de abstrações, o padrão GOF é o preferido para a implementação de páginas *Web*. Por esse motivo, vamos focar nesta linha, para que possamos entender como aplicá-la em conjunto com PHP.
Os padrões baseados em GOF podem ser agrupados em três grupos, conforme a descrição a seguir (PALMEIRA, 2013):
Padrões de criação
Esses padrões são responsáveis pela definição pelo processo de criação. Constituem os padrões de criação: *Factory Method, Abstract Factory, Singleton, Builder, Prototype.*
Padrões estruturais
Definem a forma de associação e organização entre os objetos e classes de forma a possibilitar a criação de estruturais complexas por meio da composição, herança e associação entre as classes. Fazem parte dos padrões estruturais: *Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.*
Padrões comportamentais

Voltados para a definição de como os objetos/classes de comunicam e se interagem, buscando um baixo acoplamento para permitir, assim, uma flexibilidade e reuso mais fácil dos componentes vinculados. Esse grupo de padrões é formado pelos seguintes padrões: *Chain of Responsability, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template, Method, Visitor.*

Para exemplificar, esses padrões, tomemos por base os exemplos encontrados em Mourão (2009): o *Singleton* e *Factory Method*.

O padrão *Singleton* é usado quando é necessário garantir que haja apenas um objeto de uma classe especifica instanciado. Lembrando que a instanciação se faz durante a manipulação da página permitindo-se, portanto, a dinamicidade da página. A figura a seguir ilustra o uso do padrão *Singleton* usando PHP.

```php
class Usuario{
        static private $instance = null;
        private $nome;
        private $senha;
        private function __construct(){

        }

        static function getinstance(){
                if (self::$instance == null){
                    self::$instance = new Usuario;
                }
                    return self::$instance;
        }
```

Figura 9 - Utilização do padrão Singleton usando PHP. A garantia de existência de apenas uma instância é garantida pela definição do método construtor como privado (private).
Fonte: MOURÃO, 2009.

No caso da figura acima, o método construtor (*__construct())* foi definido como privado (*private*) para garantir que a instanciação de objetos seja realizada apenas pelo método *getInstance().*

Por sua vez, no *Factory Method,* ilustrado na figura a seguir, implementa-se uma classe que tem a funcionalidade de criar objetos. O tipo de objeto a ser criado é definido sob demanda, ou seja, mais uma funcionalidade útil para deixar as suas páginas dinâmicas.

```php
interface ICarro{
   function Ligar();
}

class Gol implements ICarro{
      function Ligar(){echo "Gol Ligado !!!!"; }
}

class Palio implements ICarro{
      function Ligar(){echo "Palio Ligado !!!!"; }
}
```

```php
class FactoryCar{
   static function CriarCarro($marca){
     switch ($marca){
        case "Gol":  $obj = new Gol;
                     $obj->Ligar();
        break;
        case "Palio": $obj = new Palio;
                      $obj->Ligar();
     }
   }
}
```

```php
FactoryCar::CriarCarro("Palio");  ou FactoryCar::CriarCarro("Gol");
```

Figura 10 - Utilização do padrão Factory Method usando PHP. A classe FactoryCar objetiva instanciar um objetivo cujo tipo é passado por parâmetro em função da demanda corrente.
Fonte: MOURÃO, 2009.

Na figura acima, temos a possibilidade de instanciar objetos de duas classes distintas, que, porém, possuem a mesma interface (ambas apresentam o método *Ligar()*). O tipo do objeto é passado como parâmetro – desta forma, a classe a ser instanciada é determinada em função da demanda corrente, ou seja, como o usuário está a manipular com a página em questão. Para tanto, evoca-se o método *CriarCarro()* pertencente, no caso do exemplo, à classe *FactoryCar.*
Em projetos de páginas, ou outros sistemas computacionais, separar código em diversos objetos, para atender tarefas extremamente específicas, se torna uma tarefa árdua. Para facilitar esse ponto de vista, pode-se usar o padrão MVC (*Model, View, Controller* – Modelo, Visão, Controlador). Veremos, a seguir, esse padrão.

2.4.2 MVC

O padrão MVC, pelo fato de permitir diversos níveis de abstrações e organizações se tornou o principal padrão no desenvolvimento de código PHP. Segundo (GAMMA, 2000), ele integra outros padrões pertencentes ao mundo GOF. Dentre os quais podemos citar: *Factory Method, Observer, Composite* e *Strategy.*

VOCÊ O CONHECE?

O início do padrão MVC é datado no ano de 1979, quando Trygve Reenskaug, funcionário da empresa Xerox PARC, apresentou o seu trabalho "*Applications Programming in Smalltalk-80: How to use Model-View-Controller*" (Programando Aplicações em Smalltalk-80: Como usar o Modelo-Visão-Controlador". Recomendamos o artigo "Introdução ao Padrão MVC", para entender o histórico do padrão (MEDEIROS, 2013): <https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>.

Como já mencionado, o padrão MVC contempla três camadas. Confira a seguir.

- Camada de Modelo

  Representa o próprio processamento, ou seja, codifica as regras de negócio e gerencia o processo de armazenamento e recuperação de objetos persistentes, por intermédio da manipulação de sistemas de gerenciadores de bancos de dados (SGBD). A camada de modelo manipula os dados sem ter o conhecimento de quais, especificamente, serão utilizados. Essa tarefa é atribuída à camada controladora.

- Camada de Visão

  A camada de visão objetiva o interfaceamento com o usuário. A interface pode apresentar mudanças em função das alterações do estado da aplicação.

- Camada Controladora

  Tem a função de controlar e intermediar as ações entre as outras duas camadas, ou seja, define o comportamento do sistema em função da demanda do usuário.

Cabe salientar que a utilização de padrões, tal como o MVC, acarreta em muitos benefícios à produção das páginas, como a possibilidade de reutilização de código (diminuindo-se, assim, o tempo de codificação), melhor abstração do sistema de modo a permitir uma criação e uma manutenção mais eficientes e divisão das responsabilidades entre os colaboradores do desenvolvimento, de forma mais clara.

CASO

Atualmente, muita importância se dá aos sistemas baseados em Web, tanto para o gerenciamento e comunicação interna da empresa, quanto para o relacionamento com o cliente. Implementar páginas que se adequem bem à situação e à demanda é a meta de qualquer desenvolvedor. A utilização de padrões, tais como o MVC vem ao encontro dessa necessidade, tendo, ainda, como consequência, uma maior facilidade de manutenção e uma melhora na organização e manipulação das informações. No trabalho "Análise e desenvolvimento de um sistema CRM para concessionárias de veículos" (GROTH, 2015), acompanhamos um estudo de caso para a implementação de um sistema para uma concessionária de veículos. Para a implementação, utilizou-se o MVC. Você pode ler a publicação completa em: <https://painel.passofundo.ifsul.edu.br/uploads/arq/20160331190703476695280.pdf>.

Nos dias de hoje, a quantidade de informações que necessitam de uma manipulação eficiente, rápida e objetiva, faz com que tenhamos a necessidade de refinar com bastante cautela o sistema computacional sob desenvolvimento. À essa necessidade, alia-se o fato de que, por outro lado, podemos contar com uma rede de cooperação e possibilidade de reaproveitamento de código, que não tínhamos a poucos anos atrás. Esses vieses vão ao encontro da necessidade de utilizarmos padrões de projeto, para implementar nossos sistemas computacionais (abrangendo, também, nossas páginas). Tente sempre aplicar padrões, tal como o MVC em seus projetos para diminuir os custos financeiros e temporais, assim como permitir a produção mais adequada e eficaz às suas respectivas demandas.

Síntese

Chegamos ao fim deste capítulo. Tivemos a oportunidade de entender como deixar as páginas dinâmicas. Para isso, passamos por alguns conceitos relativos aos padrões de projeto, utilização de linguagens (como JavaScript e PHP), para validar formulários, atuar na apresentação visual de sua página, assim como realizar o interfaceamento com o servidor.

Como foi abordado neste capítulo, construção de páginas não é um trabalho trivial, deve-se analisar muito bem o ambiente onde o sistema irá interagir, assim como as suas regras de negócio. Para facilitar, aplica-se padrões de projeto, para que o seu trabalho fique menos oneroso e com mais qualidade.

Com os pontos pelos quais caminhamos neste capítulo, esperamos que você continue incrementando as funcionalidades de suas páginas, para torná-las mais acessíveis, dinâmicas e independentes de tecnologias representadas pelos dispositivos que servirão para acessá-las.

Neste capítulo, você teve a oportunidade de:

- ter contato com padrões de projeto de modo que você possa aplicá-los no desenvolvimento de suas páginas;
- analisar e empregar JavaScript para a validação de formulários;
- esboçar e desenvolver soluções baseadas na utilização da linguagem PHP;
- compor um sistema/páginas pela aplicação de MVC;
- aplicar os conceitos para o desenvolvimento de páginas responsivas e acessíveis.

Bibliografia

ANTÔNIO, G. **PHP tutorial.** Portal Devmedia, publicado em 30/04/2015. Disponível em: <https://www.devmedia.com.br/php-tutorial/32540>. Acesso em 28/10/2018.

ARRIGONI, R. **PHP forms: manipulando dados de formulários**. Portal Devmedia, publicado em 21/10/2013. Disponível em: <https://www.devmedia.com.br/php-forms-manipulando-dados-de-formularios/29392>. Acesso em 28/10/2018.

CAMBIUCCI, W. **Do Windows DNA para o mundo orientado a serviços: cada caso é um caso?** Portal Microsoft Developer Network, publicado em 12/04/2008. Disponível em: <https://blogs.msdn.microsoft.com/wcamb/2008/04/12/do-windows-dna-para-o-mundo-orientado-a-servios-cada-caso-um-caso/>. Acesso em: 26/11/2018.

EIS, D. **O básico sobre expressões regulares.** Portal Tableless, publicado em 15/06/2016. Disponível em: < https://tableless.com.br/o-basico-sobre-expressoes-regulares/>. Acesso em: 26/11/2018.

FABENI, R. **Validação de formulário com HTML5.** Portal Tableless, publicado em 01/07/2014. Disponível em: < https://tableless.com.br/validacao-de-formularios-com-html5/>. Acesso em: 26/11/2018.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Padrões de projeto – soluções reutilizáveis de *softwares* orientado a objeto.** Porto Alegre: Artmed Editora S.A, 2000. [Recurso eletrônico, Minha Biblioteca]

GROTH, V. J. **Análise e desenvolvimento de um sistema CRM para concessionárias de veículos.** Monografia (tecnólogo em sistemas para Internet). Instituto Federal Sul-Rio-Grandense, campus de Passo Fundo, Passo Fundo, 2015. Disponível em: <https://painel.passofundo.ifsul.edu.br/uploads/arq/20160331190703476695280.pdf>. Acesso em: 26/11/2018.

HXTUTORS. **#19 Estruturando o PHP – criando um site responsivo do começo ao fim**. Canal HxTutors, YouTube, publicado em 19/10/2016. Disponível em: <https://www.youtube.com/watch?v=_rzQPa_ZwX4>. Acesso em: 26/11/2018.

JQUERY. **jQuery API Documentation. Portal jQuery.** Disponível em: <http://api.jquery.com/>. Acesso em: 26 /11/2018.

LEMAY, L.; COLBURN, R.; TYLER, D. **Aprenda a criar páginas Web com HTML e XHTML em 21 dias.** São Paulo: Pearson Education do Brasil, 2002. [Recurso eletrônico, Biblioteca Virtual Universitária]

LINS, K. **Iniciando expressões regulares**. Portal Devmedia, publicado em 14/09/2007. Disponível em: < https://www.devmedia.com.br/iniciando-expressoes-regulares/6557>. Acesso em: 26/11/2018.

MEDEIROS, H. **Introdução ao padrão MVC.** Portal Devmedia, publicado em 10/10/2013. Disponível em: < https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>. Acesso em: 26/11/2018.

MILETTO, E. M.; BERTAGNOLLI, S. C. **Desenvolvimento de software II: introdução ao desenvolvimento web com html, css, javascript e php**. Porto Alegre, Bookman, 2014.

MOURÃO, R. C. **Quick tips: padrões de projeto no PHP**. Portal Devmedia, publicado em 30/09/2009. Disponível em: <https://www.devmedia.com.br/quick-tips-padroes-de-projeto-no-php/14452>. Acesso em: 26 /11/2018.

MOZILLA. **JavaScript básico.** Portal MDN webdocs, publicado em 18/09/2018. Disponível em: < https://developer.mozilla.org/pt-BR/docs/Aprender/Getting_started_with_the_web/JavaScript_basico>. Acesso em: 26/11/2018.

PALMEIRA, T. V. V. **Entendendo os conceitos dos padrões de projetos em Java.** Portal Devmedia, publicado em 09/07/2013. Disponível em: <https://www.devmedia.com.br/entendendo-os-conceitos-dos-padroes-de-projetos-em-java/29083>. Acesso em: 26/11/2018.

SOUZA, R. **Arquitetura em camadas, uma definição simples**. Portal Linkedin, publicado em 29/06/2016. Disponível em: <https://www.linkedin.com/pulse/arquitetura-em-camadas-uma-defini%C3%A7%C3%A3o-simples-robson-souza>. Acesso em: 26/11/2018.

TEIXEIRA, J. R. **JQuery Tutorial.** Portal Devmedia, publicado em 23/02/2013. Disponível em: <https://www.devmedia.com.br/jquery-tutorial/27299>. Acesso em: 26/11/2018.
TEIXEIRA, F. **Formulário de e-mail e envio com PHP.** Portal Tableless, publicado em 18/08/2015. Disponível em: <https://tableless.com.br/formulario-de-e-mail-e-envio-com-php/>. Acesso em: 26/11/2018.
WALSH, D. **5 formas de interação de CSS e JavaScript que você provavelmente não conhece.** Portal iMasters, publicado em 03/06/2014. Disponível em: <https://imasters.com.br/css/5-formas-de-interacao-de-css-e-javascript-que-voce-provavelmente-nao-conhece>. Acesso em: 26/11/2018.