

Sumário

1.0 ARQUITETURA WEB	2
1.1 CONCEITOS PRELIMINARES	2
1.2 PROTOCOLO E REQUISIÇÃO HTTP	2
1.2.1 CODIGOS DE RESPOSTAS REQUISIÇÃO HTTP	3
1.3 MÉTODOS HTTP	4
2. LINGUAGEM PHP	4
3. SERVIDOR APACHE	4
3.1 INSTALAÇÃO: APACHE / PHP / MYSQL	5
4. CODIFICAÇÃO – LINGUAGEM PHP / ESTRUTURADA	5
4.1 ESTRUTURA BÁSICA DE UMA CODIFICAÇÃO PHP	5
4.2 VARIÁVEIS PHP / SINTAXE	5
4.3 VARIÁVEIS PHP / IDENTIFICANDO O TIPO DO VALOR ARMAZENADO	6
4.4 VARIÁVEIS PHP / VERIFICANDO O TIPO DO VALOR ARMAZENADO	6
4.5 VARIÁVEIS DO TIPO ARRAY	7
4.5.1 ARRAY - DEFINIÇÃO EXPLÍCITA - SEM CHAVES	7
4.5.2 ARRAY - DEFINIÇÃO EXPLÍCITA - COM CHAVES	8
4.5.2.1 ARRAY - DEFINIÇÃO EXPLÍCITA - COM CHAVES - OUTRO EXEMPLO	8
4.5.3 ARRAY - DEFINIÇÃO IMPLÍCITA OU DIRETA	9
4.5.4 ARRAYS MULTIDIMENSIONAIS	9
4.5.5 FUNÇÕES PHP PARA MANIPULAÇÃO DE ARRAYS	10
4.6 CONSTANTES:	11
4.6.1 CONSTANTES PRÉ-DEFINIDAS:	12
4.7 VARIÁVEIS SUPERGLOBAIS (PRÉ-DEFINIDAS)	13
4.8 - FUNÇÕES - PHP	14
4.9 FORMULÁRIOS HTML – MÉTODOS POST E GET:	15
4.9.1 FORM HTML / POST	15
4.9.2 FORM HTML / GET	16
4.9.3 DADOS VIA POST - SIMPLIFICADO	17
4.9.4 DADOS VIA POST - DINÂMICA DE ROTAS	18
4.9.5 LEITURA DE ARQUIVOS TEXTO - PHP	20
4.9.6 ESCRITA EM ARQUIVOS TEXTO - PHP	21

1.0 Arquitetura Web

1.1 Conceitos Preliminares

A infra-estrutura da Internet é baseada no modelo cliente x servidor, onde os clientes efetuam requisições junto aos servidores no intuito obter determinados tipos de dados.

Tecnologias como HTML e CSS encontram-se no lado do cliente (*client side*), pois podem ser interpretadas e renderizadas pelo navegador executado no computador do usuário – não há necessidade de conexão com um servidor remoto.

Já tecnologias como PHP encontram-se no lado do servidor (*server side*) pois sua interpretação é feita remotamente, visto que o navegador não é capaz disso. Para tal é necessário que o cliente efetue requisições a um servidor (*Apache*, por exemplo), que após interpretar o script PHP retorna como resultado dados que podem ser manipulados e exibidos pelo navegador.

As requisições e respostas geradas precisam obedecer a um padrão, para que ambos os lados possam trocar informações e compreender o que está sendo requisitado e respondido. Por esse motivo o protocolo HTTP é utilizado.

1.2 Protocolo e Requisição HTTP

A sigla HTTP refere-se a Protocolo de Transferência de Hipertexto (*Hyper Text Transfer Protocol*). Tal protocolo define regras e padrões que permitem que clientes e servidores Web possam se comunicar adequadamente, efetuando requisições e obtendo suas respectivas respostas.

Por exemplo, quando um cliente acessa uma determinada URL (endereço web) através do navegador, uma requisição será efetuada a um determinado servidor, tendo como procedimentos, de modo mais simplista:

1. **Navegador** (cliente) efetua uma conexão com o servidor e envia uma solicitação HTTP para a página da web especificada;
2. **Servidor** recebe e verifica a solicitação, sendo adequada, o servidor devolve como resposta os dados para página especificada e um código indicando que a solicitação foi atendida corretamente. Caso o servidor, por algum motivo, não consiga atender a solicitação, enviará uma mensagem de erro juntamente com um código que permite sua identificação;
3. **Navegador** recebe a resposta do servidor (“página/código” ou “mensagem de erro / código”), e a conexão é finalizada;
4. **Navegador** analisa a resposta, caso indique que a solicitação foi atendida adequadamente o navegador adota os procedimentos necessários para exibir a página especificada na solicitação;

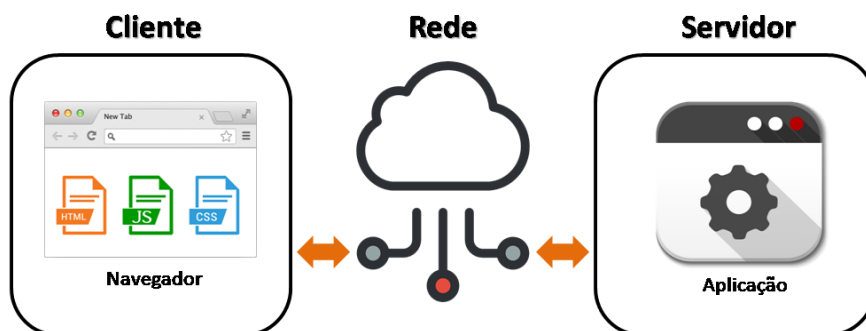


Figura 1: Comunicação processo a processo.

As requisições efetuadas por um cliente são denominadas *HTTP Request*.

Tais requisições são recebidas e processadas pelos servidores (como o Apache), que devolvem como resposta, tanto os conteúdos das páginas web solicitadas pelo usuário, quanto códigos que permitem ao cliente identificar se a requisição ocorreu como esperado.

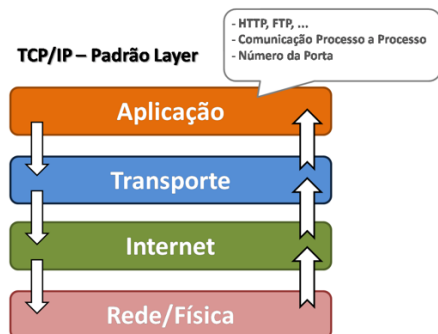


Figura 2: Protocolo TCP/IP -

1.2.1 Codigos de respostas Requisição HTTP

A seguir temos uma lista resumida com os principais códigos de resposta emitidos por um servidor quando uma requisição HTTP é recebida:

- 1XX: Informativo – solicitação foi aceita ou continua em andamento;
- 2XX: Confirmação – solicitação foi concluída ou entendida;
- 3XX: Redirecionamento – um ou mais procedimentos são necessários para atender a solicitação;

- 4XX: Erro/Cliente – solicitação não pode ser atendida ou contém erro de sintaxe;
- 5XX: Erro/Servidor – servidor falhou durante o atendimento da solicitação;

A lista completa pode ser encontrada em: <https://httpstatuses.com/>

1.3 Métodos HTTP

O HTTP possui um conjunto de métodos que podem ser utilizados quando uma solicitação é efetuada por um determinado cliente. Esses métodos definem o modo como os parâmetros são enviados quando uma requisição é efetuada ao servidor. Dentre esse conjunto de métodos destacam-se:

- **GET:** é o método padrão utilizado ao efetuar uma solicitação, nele os parâmetros são passados juntamente com cabeçalho da requisição HTTP, sendo possível vê-los na URI. Exemplo:

- www.pagina.com.br/cadastrar/cpf=00000000001

- **POST:** é um dos métodos que podem ser especificados no momento em que uma solicitação é efetuada. Diferentemente do método *GET*, o *POST* permite que os parâmetros sejam passados junto ao corpo da requisição, não sendo mais visíveis na URI.

Exemplo: - www.pagina.com.br/cadastrar

2. Linguagem PHP

Linguagem de script (interpretadas em tempo de execução – não são compiladas) que possui sua sintaxe baseada, em grande parte, nas linguagens C e Java, incluindo algumas características específicas.

O PHP possibilita o desenvolvimento de páginas dinâmicas, sua sigla significa Pré-processador de hipertexto (*Hypertext Preprocessor*). É uma linguagem scripts do tipo *backend*, ou seja, é executada no servidor antes de enviada ao cliente (navegador). Sendo assim, o servidor não envia código-fonte ao cliente, mas sim processa e transforma esse código em formato HTML, para após isso enviar ao cliente. Dentre os principais servidores web disponíveis, o Apache é o mais utilizado atualmente.

3. Servidor Apache

O servidor Apache ou também Apache HTTP Server dá suporte aproximadamente metade das aplicações web disponíveis atualmente. O Apache, assim como outros servidores web (IIS – *Internet Information Server*), tem como objetivo principal dar suporte adequado ao maior número de clientes simultaneamente. Ele é capaz de processar não apenas arquivos escritos em

PHP, mas também outras linguagens, tais como Python e Perl. O Apache pode ser visto, de modo simplificado, como um interpretador de código-fonte, capaz de transformar a codificação interpretada em conteúdo HTML.

3.1 Instalação: Apache / PHP / MySQL

Linux (Lamp):

<https://www.linuxbabe.com/ubuntu/install-lamp-stack-ubuntu-18-04-server-desktop>

Windows/Linux (Xampp):

https://www.apachefriends.org/pt_br/download.html

4. Codificação – Linguagem PHP / Estruturada

4.1 Estrutura básica de uma codificação PHP

```
<?php
    echo "Hello";
    print "World!";
?>
```

Estrutura Básica - PHP

[<?php] - define o início de um novo trecho de código em linguagem PHP
[echo][print] - comandos de saída, permitem apresentar dados no navegador
[?>] - define o término do trecho de código em linguagem PHP

localhost/php/aula01/estrutura_basica.php

HelloWorld!

4.2 Variáveis PHP / Sintaxe

```
<?php
$curso = "WEB Médio - Aluno 10!";
$disciplina = "PSW1";
$turma = 2022;
echo "[CURSO]: $curso<br>";
echo "[DISCIPLINA]: $disciplina<br>";
echo "[TURMA]: $turma";
?>
```

Variáveis - PHP

[Scurso] - variáveis php sempre começam pelo caractere especial “\$”

[Sdisciplina] - variáveis php não precisam ser declaradas, basta usá-las

[Sturma] - PHP é dito “fracamente tipado”, variáveis podem receber qualquer tipo de valor

← → ↻  localhost/php/aula01/variaveis.php

[CURSO]: Graduação em Análise e Desenvolvimento de Sistemas

[DISCIPLINA]: Desenvolvimento Web II

[TURMA]: 2015

4.3 Variáveis PHP / Identificando o tipo do valor armazenado

```
<?php
    $var = "Bill Gates";    // String
    $tipo = gettype($var);
    echo "$var ($tipo)<br>";
    $var = 12;              // Inteiro
    $tipo = gettype($var);
    echo "$var ($tipo)<br>";
    $var = 3.1415;          // Float/Double
    $tipo = gettype($var);
    echo "$var ($tipo)<br>";
    $var = true;            // Booleano
    $tipo = gettype($var);
    echo "$var ($tipo)<br>";

?>
```

Identificando Tipo Armazenado

[\$gettype(\$var)] - permite obter o tipo do valor armazenado em “\$var” Observe que a variável “\$var” pode receber valores dos mais variados tipos

4.4 Variáveis PHP / Verificando o tipo do valor armazenado

```
<?php
    if(is_null($var)) {
        echo "Variável \ $var é nula/vazia!<br>";
    }
    $var = "Bill Gates";
    if(is_string($var)) {
        echo "Variável \ $var é uma string!<br>";
    }
    $var = 12;
    if(is_integer($var)) {
        echo "Variável \ $var é um inteiro!<br>";
    }
    $var = false;
    if(is_bool($var)) {
        echo "Variável \ $var é booleana!";
    }
    // is_float(), is_array(), is_object()

?>
```

Verificando Tipo Armazenado

`[is_null()]` - função que verificar se uma variável está vazia (null)
`[is_string()]` - função que verificar se uma variável contém um valor do tipo "string"
`[is_integer()]` - função que verificar se uma variável contém um valor inteiro
`[is_bool()]` - função que verificar se uma variável contém um valor booleano

← → ↻  localhost/php/aula01/funcoes_tipos.php

Variável \$var é nula/vazia!
Variável \$var é uma string!
Variável \$var é um inteiro!
Variável \$var é booleana!

4.5 Variáveis do tipo Array

Arrays (em PHP) são mapas ordenados de chaves e valores, ou seja, é possível atribuir a um elemento do array uma chave e um valor. Existem duas formas de definir um *array* em PHP:

- Explícita: através do construtor **`array()`**. ○ *Ex.:* `Array([chave] => valor, ...)` ;
- Implícita: sem utilização do construtor **`array()`**. ○ *Ex.:* `$array_exemplo[chave] = valor;`

4.5.1 Array - Definição Explícita - Sem chaves

```
<?php

// Array: Definição Explícita (sem chave)
$var = array(1, 2, 3, 4);
echo "[for]: ";
for($a=0; $a<count($var); $a++) {
    echo "$var[$a] ";
}
echo "<br>[foreach]: ";
foreach ($var as $dado) {
    echo "$dado ";
}

?>
```

Definição Explícita - Sem Chaves

`[count($var)]` - função que obtém a quantidade de elementos contidos num "array"
`[$var[$a]]` - como não há chave, os elementos do array são acessados via índice, partindo do "0"

← → ↻  localhost/php/aula01/arrays.php

[for]: 1 2 3 4
[foreach]: 1 2 3 4

4.5.2 Array - Definição Explícita - Com chaves

```
<?php

// Array: Definição Explícita (com chave)
$var = array( "Maria" => 25,
             "João"  => 44,

             "José" => 12,

             "Neusa" => 73
            );
foreach ($var as $chave => $valor) {
    echo "$chave: $valor<br>";
}
echo "<br>";
print($var);

?>
```

Definição Explícita - Com Chaves

[array("Maria" => 25)] - Define a chave "Maria" para o valor "25"
["João" => 44] - Define a chave "João" para o valor "44"
[print_r] - comando de saída que apresenta o valor da variável de forma legível

← → ↻  localhost/php/aula01/arrays02.php

Maria: 25
João: 44
José: 12
Neusa: 73

Array ([Maria] => 25 [João] => 44 [José] => 12 [Neusa] => 73)

4.5.2.1 Array - Definição Explícita - Com chaves - Outro Exemplo

```
<?php

// Array: Definição Explícita (com chave)
$var = array( "000.000.000-00" => "Maria",
             "000.000.000-01" => "João"
            );
foreach ($var as $chave => $valor) {
    echo "($chave: $valor) ";
}
$var = array( 12 => "Maria",
             21 => "João"
            );
```



```
    );  
    echo "<br>";  
    foreach ($var as $chave => $valor) {  
        echo "($chave: $valor) ";  
    }  
  
?>
```

Definição Explícita - Com Chaves

[array("000.000.000.00" => "Maria")] - Define a chave "000.000.000.00" para o valor "Maria"
[array(12 => "Maria")] - Define a chave "12" para o valor "Maria"

← → ↺ 📄 localhost/php/aula01/arrays03.php

(000.000.000-00: Maria) (000.000.000-01: João)
(12: Maria) (21: João)

4.5.3 Array - Definição Implícita ou Direta

```
<?php  
  
// Array: Definição Direta (sem chave)  
$var[0] = "Desenvolvimento";  
$var[1] = "Web";  
$var[2] = "II";  
echo "[for]: ";  
for($a=0; $a<count($var); $a++) {  
    echo "$var[$a] ";  
}  
echo "<br>[foreach]: ";  
foreach ($var as $dado) {  
    echo "$dado ";  
}  
  
?>
```

Definição Implícita ou Direta

[\$var[1] = "Web"] - Construtor "array" não é utilizado, o valor é atribuído diretamente

← → ↺ 📄 localhost/php/aula01/arrays04.php

[for]: Desenvolvimento Web II
[foreach]: Desenvolvimento Web II

4.5.4 Arrays Multidimensionais

```
<?php  
//Array Multidimensional: Definição Explícita  
$arr =
```

```
array("Maria" => array(
    "endereco" => "Rua Chile 1046",
    "bairro" => "Rebouças",
),
    "João" => array(
        "endereco" => "Rua Iapó 234",
        "bairro" => "Prado Velho",
    )
);
foreach($arr as $chave => $aux) {
    echo strtoupper($chave).": <br>";
    foreach($aux as $chave => $valor) {
        echo "    - $valor<br>";
    }
    echo "<br>";
}
```

Array Multidimensional

[array("Maria" => array())] - o valor da chave "Maria" é outro array, com outras chaves

← → ↻ 📄 localhost/php/aula01/arrays_multi.php

MARIA:
- Rua Chile 1046
- Rebouças

JOÃO:
- Rua Iapó 234
- Prado Velho

4.5.5 Funções PHP para Manipulação de Arrays

(https://www.php.net/manual/pt_BR/ref.array.php)

Os códigos-fonte que exemplificam as funções apresentadas a seguir estão disponíveis juntos com os arquivos-fonte de exemplo da aula.

- **array array_key(*arr*)**: retorna todas as chaves do array *arr*;
- **array array_values(*arr*)**: retorna todos os valores do array *arr*;
- **String array_search(*val*, *arr*)**: busca pelo valor *val* no array *arr* e retorna a respectiva chave;
- **bool array_key_exists(*key*, *arr*)**: verifica se uma chave ou índice *key* existe para um array *arr*;
- **bool in_array(*val*, *arr*)**: verifica se um valor *val* existe em um array *arr*;
- **bool isset(*var*)**: verifica se a variável *var* foi inicializada;
- **void unset(*var*)**: destrói a variável *var*;
- **bool empty(*var*)**: verifica se *var* está vazia;

- **int array_push(arr, ele[]):** adiciona um ou mais elementos **ele[]** no final do array **arr**;
- **String array_pop(arr):** extrai um elemento do final do array **arr**;
- **mixed array_shift(arr):** remove o primeiro elemento do array **arr**. O retorno pode ser um conjunto de tipos;
- **String array_unshift(arr):** adiciona um ou mais elementos no início do array **arr**;
- **int count(var):** Conta o número de elementos da variável **var**, ou propriedades do objeto **var**;
- **array explode(del, str):** retorna uma matriz de strings, dividindo **str** de acordo com **del**;
- **String implode(str, arr):** retorna uma string contendo os elementos do array **arr** concatenados pela string **str**;
- **array array_combine(key, val):** Cria um array usando o array **key** para chaves e o array **val** para os valores;
- **array array_diff(arr1, arr2):** encontra a diferença entre os arrays **arr1** e **arr2** (elementos que existem em **arr1** e não existem em **arr2**);
- **array array_intersect (arr1, arr2):** encontra a intersecção entre os arrays **arr1** e **arr2** (elementos que existem tanto em **arr1** quanto em **arr2**);

4.6 Constantes:

Para definir uma constante utilizamos o comando **define()**. Após sua definição, uma constante não pode ser alterada nem removida.

```
<?php
```

```
define("PI", 3.1415);
```

```
echo PI; ?>
```

Constantes - PHP

[define("PI", 3.1415)] - define a constante "PI" com valor "3.1415"

Por convenção (e boas práticas) utilizamos nomes de constantes com letras maiúsculas. Por padrão, constantes PHP são "case sensitive", mas isso pode ser configurado.



localhost/php/aula01/constantes.php

3.1415

```
<?php
```

```
define("PI", 3.1415, true);
```

```
echo PI;  
echo "<br>";  
echo pi;  
  
?>
```

Constantes - PHP

[define(..., true)] - o parâmetro "true" indica que a constante definida NÃO é "case sensitive"

← → ↻ ⓘ localhost/php/aula01/define02.php

3.1415
3.1415

4.6.1 Constantes Pré-definidas:

A linguagem PHP disponibiliza um conjunto de constantes pré-definidas, que costumam ser muito úteis durante o desenvolvimento de aplicações.

- `__FILE__`: contém o nome do arquivo (script) que está sendo executado;
- `__DIR__`: contém o diretório do script que está sendo executado;
- `__LINE__`: contém o número da linha atual;
- `__FUNCTION__`: contém o nome da função que está sendo executada;
- `__CLASS__`: contém o nome da classe;
- `__METHOD__`: contém o nome do método da classe;

```
<?php  
function funcConsts() {  
    echo "ARQUIVO: " . __FILE__ . "<br>";  
    echo "DIRETÓRIO: " . __DIR__ . "<br>";  
    echo "LINHA: " . __LINE__ . "<br>";  
    echo "FUNÇÃO: " . __FUNCTION__ . "<br>";  
}  
  
funcConsts();  
?>
```

Constantes Pré-definidas - PHP

[function] - palavra reservada utilizada para definir uma função

← → ↺ 📄 localhost/php/aula01/constantes_pre.php

ARQUIVO: /var/www/html/php/aula01/constantes_pre.php
DIRETÓRIO: /var/www/html/php/aula01
LINHA: 6
FUNÇÃO: funcConsts

```
class veiculo {  
    private $marca;  
    function __construct() {  
        echo "CLASSE: " . __CLASS__ . "<br>";  
    }  
  
    function setMarca($marca) { $this->marca = $marca;  
        echo "MÉTODO: " . __METHOD__ . "<br>";  
    }  
}  
  
$obj = new veiculo();  
  
$obj->setMarca("Wolksvagem");  
?>
```

Constantes Pré-definidas - PHP

[class] - palavra reservada utilizada para definir uma class
O conceito de classe e orientação a objeto no PHP será abordado em detalhes na próxima

← → ↺ 📄 localhost/php/aula01/constantes_pre02.php

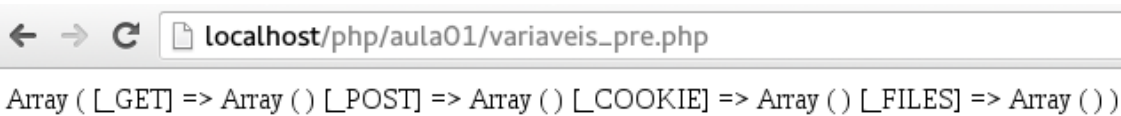
CLASSE: veiculo
MÉTODO: veiculo::setMarca

4.7 Variáveis Superglobais (pré-definidas)

O PHP também disponibiliza um conjunto de variáveis já pré-definidas acessíveis por qualquer script, conhecidas como super globais. Essas variáveis dependem do ambiente e módulo PHP que estão carregados, e podem ser obtidas através da função **get_defined_vars()**.

```
<?php  
$vars_pre = get_defined_vars();  
print_r($vars_pre);  
  
?>
```

Variáveis Superglobais



Array ([_GET] => Array () [_POST] => Array () [_COOKIE] => Array () [_FILES] => Array ())

A seguir uma listagem contendo as principais variáveis superglobais:

- **\$GLOBALS**: retorna um array para todas as variáveis que estão atualmente disponíveis no escopo global;
- **\$_SERVER**: array contendo informações sobre o servidor web e o ambiente de execução;
- **\$_GET**: array contendo todas as variáveis enviadas via método GET (mais detalhes a seguir);
- **\$_POST**: array contendo todas as variáveis enviadas via método POST (mais detalhes a seguir);
- **\$_COOKIE**: array contendo todas as variáveis especiais que são gravadas na máquina do usuário e recuperadas pelo navegador;
- **\$_FILES**: array contendo informações sobre arquivos enviados do computador do cliente para o servidor web – upload;
- **\$_ENV**: array contendo as variáveis de ambiente disponíveis no momento;
- **\$_REQUEST**: array contendo o todas as variáveis do \$_GET, \$_POST e \$_COOKIE;
- **\$_SESSION**: array contendo registradas na seção corrente

4.8 - Funções - PHP

Uma função pode ser definida e invocada através da seguinte sintaxe:

```
<?php
function nome_funcao($par_1 = 0, $par_2 = "vazio") {
    echo "Código da Função <br>";
    return "Dado de Retorno: ".$par_1."/".$par_2;
}
$retorno = nome_funcao();
echo $retorno."<br><br>";
$retorno = nome_funcao(12, "Bill Gates");
echo $retorno."<br><br>";

?>
```

Funções - Parâmetro com valor "default"

[function nome_funcao] - palavra reservada para criar uma função / nome da função
[\$par_1 = 0] - parâmetro da função, com valor default "0" pré-definido

[\$retorno = nome_funcao()] - função é invocada sem passagem de parâmetro

[\$retorno = nome_funcao(12, "Bill Gates")] - função é invocada com passagem de parâmetro

4.9 Formulários HTML – Métodos POST e GET:

Formulários HTML são interfaces criadas, do lado cliente, para que os usuários possam inserir informações, que posteriormente serão tratadas por algum script no lado do servidor (PHP, no nosso caso).

4.9.1 FORM HTML / POST

```
<html lang="pt-br"> <head>

    <title> Formulário - POST </title>

<meta charset="utf-8"> </head>

<body>
<form action="recebe_post.php" method="post">

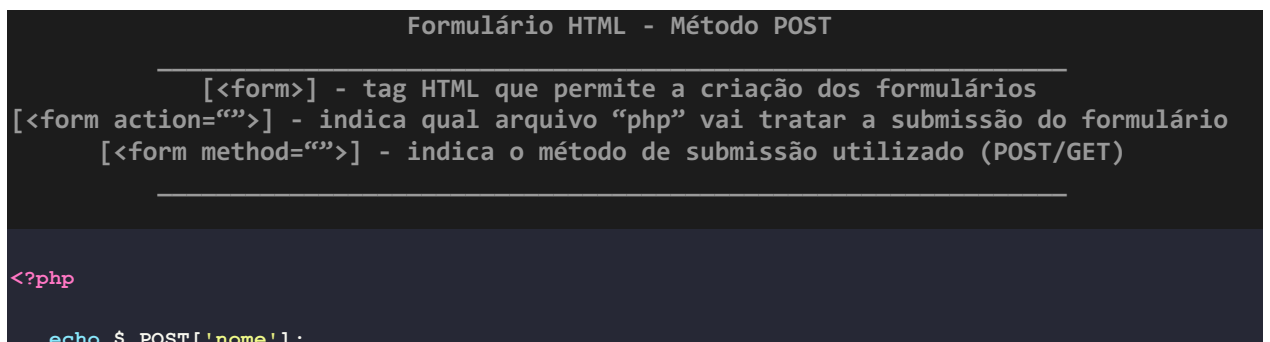
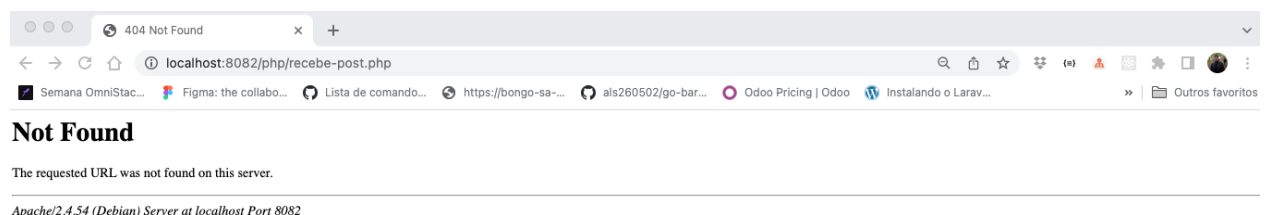
    <p> Aluno: <input type="text" name="aluno"/> </p>

    <p> Turma: <input type="text" name="turma"/> </p>

    <p> <input type="submit" value="Enviar!"/> </p>

    </form>
</body>

</html>
```

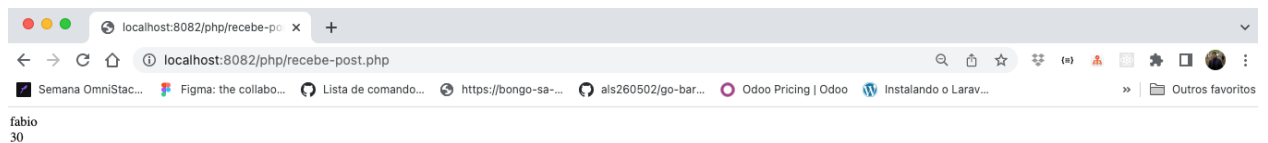


```
echo "<br>";  
echo $_POST['idade'];  
  
?>
```

Código PHP que Trata a Submissão do Formulário

[\$_POST[]] - variável superglobal que contém os dados submetidos via POST

(Arquivo-fonte: recebe_post.php)



4.9.2 FORM HTML / GET

```
<!DOCTYPE html>  
<html lang="pt-br">  
<head>  
  <meta charset="UTF-8">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>formulário get</title>  
</head>  
<body>  
  <form action="recebe-get.php" method="get">  
    <p> Nome: <input type="text" name="nome" /> </p>  
    <p> Idade: <input type="text" name="idade" /> </p>  
    <p> <input type="submit" value="Enviar"> </p>  
  
  </form>  
  
</body>  
</html>
```

Formulário HTML - Método GET

[<form>] - tag HTML que permite a criação dos formulários

[<form action="">] - indica qual arquivo "php" vai tratar a submissão do formulário

[<form method="">] - indica o método de submissão utilizado (POST/GET)

(Arquivo-fonte: form-get.php)

```
<?php  
  
echo $_GET['aluno'];  
echo "<br>";  
echo $_GET['idade'];
```

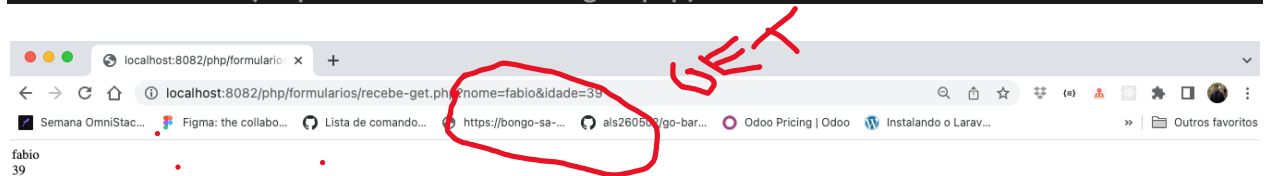


```
?>

Código PHP que Trata a Submissão do Formulário

[$_GET[]] - variável superglobal que contém os dados submetidos via GET

(Arquivo-fonte: recebe-get.php)
```



OBS.: repare que os dados enviados via GET são passados via URL, em contrapartida ao método POST, onde os dados são anexados ao corpo da requisição.

4.9.3 Dados via POST - Simplificado

(*"exemplo-post-include-hidden-action-mesmo-form styles-in-line-parametro-POST-alert"*)

Arquivo: *exemplo.post*

```
<?php
    error_reporting(E_ALL);
    ini_set("display_errors", 1);
    include_once ("exemplo-post-dados.php");
    if( !empty($_POST['form_submit']) ) {
        obterDados($_POST);
    }
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>exemplo-post</title>
</head>
<body>
<form style="background-color:yellow" class="form" method="post" action="exemplo-
post.php">
    <p> FORMULÁRIO EXEMPLO POST</p>
    <p> cpf: <input type="text" name="cpf" /> </p>
    <p> nome: <input type="text" name="nome" /> </p>
    <input TYPE="hidden" NAME="form_submit" VALUE="OK">
```

```
<br><br>
<button type="submit" class="btn">
    <b>Obter Dados Post</b>
</button>

</body>
</html>
```

Exemplo-post-dados.php

```
<?php
echo "<div style='background-color:red'>";
    echo "formulario - post dados";
echo "</div>";
function obterDados($post) {
    $dados = $post['cpf']." - ".$post['nome'];
    echo "<script> alert('".$dados."') </script>";
}
?>
```

Função de Manipulação dos Dados - Método POST

[obterDados(\$post)] - recebe a superglobal “\$_POST[]” como parâmetro

4.9.4 Dados via POST - Dinâmica de Rotas

(Arquivo: “viewRoute.php”)

```
<?php
error_reporting(E_ALL);
ini_set("display_errors", 1);
include_once ("route.php");
if( !empty($_POST['form_submit']) ) {
    rotas($_POST['acao']);
}
?>

<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!--bootstrap -->
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css">
  <script
src="https://cdn.jsdelivr.net/npm/jquery@3.6.1/dist/jquery.slim.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"></s
cript>
  <title>Document</title>
</head>
<body>
<form action="viewRoute.php" method="post">
  <input TYPE="hidden" NAME="form_submit" VALUE="OK">
  <div class='col-sm-4'>
    <button type="submit" name="acao" value="cadastar/0" class="btn btn-
primary btn-block">
      <b>Cadastar</b> </button>
    </div>
    <div class='col-sm-4'>
      <button type="submit" name="acao" value="alterar/1" class="btn btn-success
btn-block">
        <b>Alterar</b> </button>
      </div>
      <div class='col-sm-4'>
        <button type="submit" name="acao" value="remover/2" class="btn btn-danger
btn-block">
          <b>Remover</b> </button>
        </div>
  </form>
</body>
</html>
```

Formulário HTML - Método POST - Dinâmica de Rotas

[<button value="">] - observe que o valor atribuído ao "button" indica sua ação/rota

(Arquivo-fonte: viewRoute.php)

(Arquivo: "route.php")

```
<?php
function rotas($url) {
    $dados = explode("/", $url);
    // CADASTRAR
    if(strcmp($dados[0], "cadastar") == 0) {
        echo "<script> alert('CADASTRAR') </script>";
    }

    // ALTERAR

    else if(strcmp($dados[0], "alterar") == 0) {
        echo "<script> alert('ALTERAR') </script>";
    }

    // REMOVER

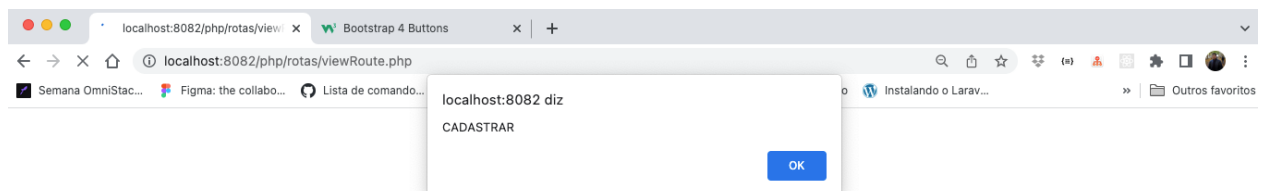
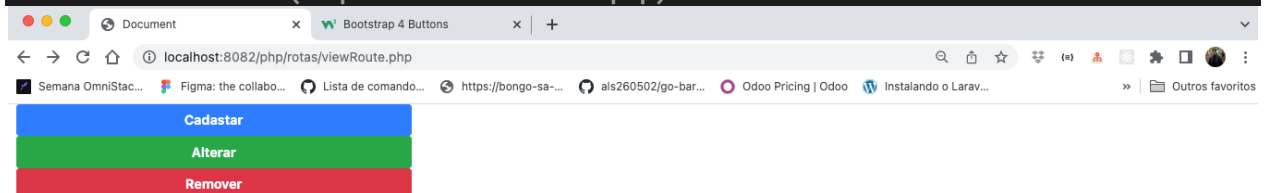
    else if(strcmp($dados[0], "remover") == 0){
        echo "<script> alert('REMOVER') </script>";
    } }

?>
```

Função de Manipulação dos Dados - Método POST - Dinâmica de Rotas

[explode("/")] - quebra o valor recebido de acordo com o "button" pressionado
[if(strcmp())] - verifica a ação que foi selecionada e efetua o procedimento

(Arquivo-fonte: route.php)



4.9.5 Leitura de Arquivos Texto - PHP

(Arquivo: "view-ler.php")

```
<?php
error_reporting(E_ALL);
ini_set("display_errors", 1);
include_once('lerArquivo.php');
ler();
```

?>

HTML - Invocando Função PHP para Leitura de Arquivo Texto

[ler()] - invoca a função PHP que efetua a leitura do arquivo texto

(Arquivo-fonte: view-ler.php)

(Arquivo: "lerArquivo.php")

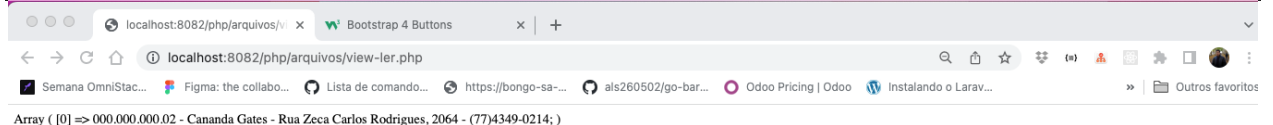
```
<?php
function ler() {
    $dados = array();
    $fp = fopen('alunos.txt', 'r');

    if ($fp) {
        while(!feof($fp)) {
            $cpf = fgets($fp);
            $linha = fgets($fp);
            if(!empty($linha)) {
                $dados = explode("#", $linha);
                print_r($dados);
                echo "<br>";
            }
        }

        fclose($fp);
    }
}
```

Rotina PHP para Leitura do Arquivo Texto

(Arquivo-fonte: lerArquivo.php)



4.9.6 Escrita em Arquivos Texto - PHP

(Arquivo: "view-escrever.php")

```
<?php
error_reporting(E_ALL);
ini_set("display_errors", 1);
include_once('escreverArquivoArray.php');
$peessoas = array(
    "000.000.000.01" => array(
        "nome" => "Thanos Gates",
        "endereco" => "Rua Manuel Viana, 200",
```

```
        "telefone" => "(77)3422-2829",
    ),
    "000.000.000.02" => array(
        "nome" => "Cananda Gates",
        "endereco" => "Rua Zeca Carlos Rodrigues, 2064",
        "telefone" => "(77)4349-0214",
    ) );
    escreverArquivoArray($pessoas);
```

?>

HTML - Invocando Função PHP para Escrita no Arquivo Texto

[escreverArquivoArray()] - invoca a função PHP que efetua a escrita no arquivo texto

(Arquivo-fonte: view-escrever.php)

```
<?php
function escreverArquivoArray($arr) {
    $fp = fopen('alunos.txt', 'a+');

    if ($fp) {
        foreach($arr as $cpf => $dados) {
            if(!empty($dados)) {
                $linha = $cpf." - ".$dados['nome']." - ".$dados['endereco']." - 
".$dados['telefone'];
                fputs($fp, "$linha\n");
            }
        }
        fclose($fp);
    }
    echo "[OK] Dados escritos com Sucesso!";
}
?>
```

Rotina PHP para Escrita no Arquivo Texto

(Arquivo-fonte: escreverArquivoArray.php)