

▶ UC16: Desenvolver sistemas eletrônicos automatizados por software.

CARGA HORÁRIA: 48 HORAS

Indicadores

1. Elabora protótipos de circuitos eletrônicos para testes de automação de acordo com a teoria de circuitos eletrônica e normas de segurança física.
2. Cria circuitos eletrônicos para controle de sensores e atuadores em uma rede IoT de acordo com a necessidade do projeto.
3. Integra o circuito eletrônico ao software desenvolvido de acordo com as especificações da aplicação.
4. Realiza testes nos sistemas eletrônicos de acordo com orientações técnicas dos projetos.
5. Instala sistemas eletrônicos para IoT desenvolvidos de acordo com as necessidades dos clientes.

Elementos da competência

CONHECIMENTOS

- Eletricidade: grandezas elétricas (corrente, tensão, resistência, potência).
- Teste de grandezas elétricas em circuitos usando multímetro
- Componentes eletrônicos ativos e passivos: funcionamento e aplicações de resistores, capacitores, indutores, diodos transistores e circuitos integrados.
- Equipamentos de medição e teste eletrônicos: multímetros.
- Diagramas esquemáticos: leitura e elaboração usando softwares apropriados.
- Circuitos eletrônicos: projeto e testes de circuitos simples.
- Matriz de contatos: técnicas de montagem e elaboração de protótipos com componentes eletrônicos e jumpers de ligação.
- Circuitos eletrônicos customizados: integração com microcontroladores.

HABILIDADES

- Organizar materiais, ferramentas, instrumentos, documentos e local de trabalho.
- Identificar os componentes e os dispositivos eletrônicos necessários a um projeto.
- Planejar e implementar circuitos eletrônicos simples.
- Planejar e configurar sistemas embarcados.
- Elaborar documentos técnicos.
- Interpretar textos técnicos.
- Mediar conflitos nas situações de trabalho.
- Analisar as etapas do processo de trabalho.

- Comunicar-se de maneira assertiva.
- Interpretar e solucionar erros e problemas que impeçam o funcionamento de um circuito.
- Selecionar informações necessárias ao desenvolvimento do seu trabalho.

ATITUDES/VALORES

- Atitude colaborativa com membros da equipe, parceiros e clientes.
- Atuação com foco em segurança pessoal no uso de equipamentos, instrumentos e ferramentas.
- Cordialidade no trato com as pessoas.
- Proatividade na resolução de problemas.
- Sigilo no tratamento de dados e informações.
- Zelo na aplicação de práticas de desenvolvimento de circuitos.
- Zelo na apresentação pessoal e postura profissional.
- Zelo na execução de procedimentos técnicos.
- Zelo pela higiene, limpeza e conservação na utilização dos equipamentos, instrumentos e ferramentas.

1. Recomendações bibliográficas:

KARVINEN, Kimmo; KARVINEN, Tero. Primeiros Passos com Sensores. Primeira edição. São Paulo: Novatec, Outubro 2014.

BANZI, Massimo; SHILOH, Michael. Primeiros Passos com o Arduino. Segunda edição. São Paulo: Novatec, Maio 2015.

2. Introdução

Nessa fase do curso iremos abordar a ideia da programação no lado do servidor, o que podemos chamar de **backend**, para isso, utilizaremos a linguagem PHP, nossa preocupação não será o **frontend**, e sim entendermos o comportamento dessa linguagem dentro da programação Web, com o andamento do curso iremos refinando e melhorando os conceitos.

Mas antes vamos imaginar a internet na qual você pode apenas consumir conteúdos, como se fosse um jornal, uma revista, ou ainda, um programa na televisão. Chato, né? Mas quando se aprende as linguagens da web, como HTML e CSS, podemos montar sites que são como revistas e servem apenas para leitura, sem permitir interação com os internautas.

O segredo da famosa web 2.0 é a capacidade de interação entre as pessoas e os serviços online. Mas, para que esta interação seja possível, é necessário que os sites sejam capazes de receber informações dos internautas e também de exibir conteúdos personalizados para cada um, ou de mudar seu conteúdo automaticamente, sem que o desenvolvedor precise criar um novo HTML para isso.

- Planejar e configurar sistemas embarcados.
- Elaborar documentos técnicos.
- Interpretar textos técnicos.
- Mediar conflitos nas situações de trabalho.
- Analisar as etapas do processo de trabalho.
- Comunicar-se de maneira assertiva.

Estes dois tipos de sites são chamados de estático e dinâmico, respectivamente.

Você já parou para pensar em tudo o que acontece quando você digita um endereço em seu navegador web? A história toda é mais ou menos assim:

- O navegador vai até o servidor que responde no endereço solicitado e pede a página solicitada.

- O servidor verifica se o endereço existe e se a página também existe em seu sistema de arquivos e então retorna o arquivo para o navegador.

- Após receber o arquivo HTML, o navegador começa o trabalho de renderização, para exibir a página para o usuário. É neste momento que o navegador também requisita arquivos de estilos (css), imagens e outros arquivos necessários para a exibição da página.

Quando se desenvolve páginas estáticas, este é basicamente todo o processo necessário para que o navegador exiba a página para o usuário. Chamamos de estáticas as páginas web que não mudam seu conteúdo, mesmo em uma nova requisição ao servidor.

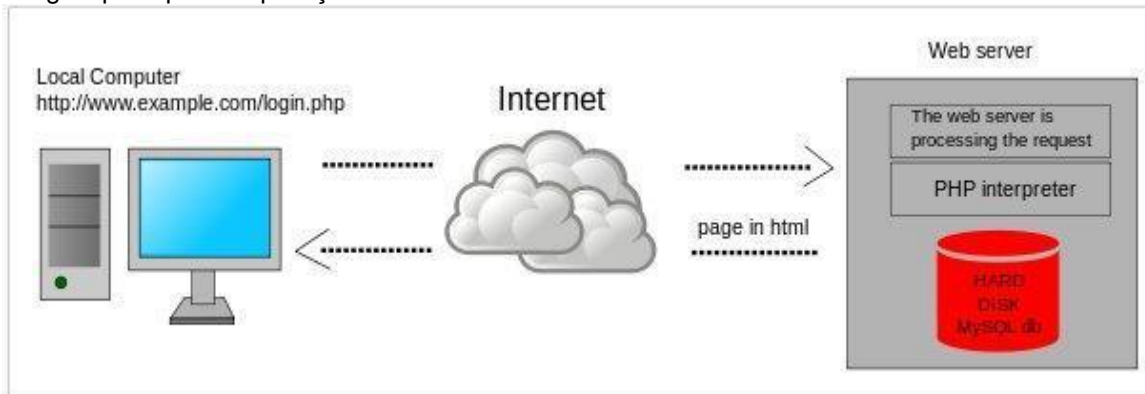
E como funciona uma página dinâmica?

O processo para páginas dinâmicas é muito parecido com o das páginas estáticas. A diferença é que a página será processada **no servidor** antes de ser enviada para o usuário. Este processamento no servidor é usado para alterar dinamicamente o conteúdo de uma página, seja ele HTML, CSS, imagens ou outros formatos.

Pense, por exemplo, em um site de um jornal. Em geral, este tipo de site contém algumas áreas destinadas às notícias de destaque, outras áreas para notícias gerais e ainda outras áreas para outros fins. Quando o navegador solicita a página para o servidor, ele irá montar o conteúdo antes de enviar para o navegador. Este conteúdo pode ser conseguido de algumas fontes, mas a mais comum é um banco de dados, onde, neste caso, as notícias ficam armazenadas para serem exibidas nas páginas quando necessário.

3. Scripting no lado do servidor (server-side)

A linguagem de servidor, ou *server-side scripting*, é a linguagem que vai rodar, fornecendo a lógica principal da aplicação.



Do lado do servidor significa toda a regra de negócio, acesso a banco de dados, a parte matemática e cálculos estarão armazenadas, enquanto que *local computer* será na máquina do usuário/cliente. Para a segurança e integridade dos dados, do lado do servidor geralmente é melhor, pois o usuário final não será capaz de ver ou manipular os dados que são enviados para o mesmo, e muito menos visualizar a regra de negócio, ou a forma de acesso ao banco de dados.

4. Scripting no lado do cliente (client-side)

As linguagens *client-side* são linguagens onde apenas o seu NAVEGADOR vai entender. Quem vai processar essa linguagem não é o servidor, mas o seu browser (Chrome, IE, Firefox, etc...). Significa "lado do cliente", ou seja, aplicações que rodam no computador do usuário sem necessidade de processamento de seu servidor (ou host) para efetuar determinada tarefa.

Basicamente, ao se falar de aplicações client-side na web, estamos falando de *JavaScript*, e *AJAX (Asynchronous Javascript And XML)*.

Existem vantagens e desvantagens ao utilizar o JavaScript e AJAX.

A principal vantagem está na possibilidade de você economizar bandwidth (largura de banda), e dar ao usuário uma resposta mais rápida de sua aplicação por não haver processamento externo.

Outra vantagem ao utilizar, agora o AJAX, seria o apelo visual de sua aplicação e rapidez de resposta. O que o AJAX faz é o processamento externo (server-side) parecendo ser interno (client-side). O usuário não percebe que houve um novo carregamento de página, pois ele busca informações no servidor e mostra rapidamente em um local específico da página através do JavaScript.

A principal desvantagem do *JavaScript* atualmente é que o usuário pode desativá-lo em seu navegador. Se a sua aplicação basear-se exclusivamente em JavaScript, nesse caso, ela simplesmente não vai funcionar.

5. A diferença entre o lado do cliente (*client-side*) e do lado do servidor (*server-side*) Scripting

Ao escrever aplicações para a web, você pode colocar os programas, ou scripts, ou no servidor da Web ou no navegador do cliente, a melhor abordagem combina uma mistura cuidadosa dos dois. Endereços de scripts do lado do servidor de gerenciamento e segurança de dados, enquanto que o script do lado do cliente se concentra principalmente na verificação de dados e layout de página.

Um servidor web é um computador separado e software com a sua própria conexão à Internet. Quando o navegador solicita uma página, o servidor recebe o seu pedido e envia o conteúdo do navegador. Um script de programa que executa no servidor web gera uma página com base na lógica do programa e envia para o navegador do usuário. O conteúdo pode ser texto e imagens padrão, ou pode incluir scripts do lado do cliente. Seu navegador executa os scripts do lado do cliente, o que pode animar imagens da página web, solicitar os dados do servidor ou executar outras tarefas.

Para um website ter uma sessão, onde você faz *login*, fazer compras e outros pedidos, o servidor precisa para identificar o computador. Milhares de usuários podem ser registrados em, ao mesmo tempo, o servidor tem de distingui-los. *Scripting* do lado do servidor mantém o controle da identidade de um usuário através de alguns mecanismos diferentes, tais como variáveis de sessão. Quando você fizer *login*, o script do servidor cria uma identificação de sessão exclusiva para você. O script pode armazenar informações em variáveis que duram o tempo que você ficar conectado.

Muitas páginas da web têm formulários para você preencher com seu nome, endereço e outras informações. Para certificar que os dados vão corretamente, os scripts de validação são capazes de verificar que as datas e códigos postais contêm apenas números e os estados têm certas combinações de duas letras. Este processo é mais eficaz quando o script é executado no lado do cliente. Caso contrário, o servidor tem que receber os dados, verificá-lo, e enviar-lhe uma mensagem de erro. Quando o navegador faz isso, você envia os dados de volta para o servidor somente uma vez.

Quando uma sessão de web envolve peneirar grandes quantidades de dados, um *script* do lado do servidor faz este trabalho melhor. Por exemplo, um banco pode ter um milhão de clientes. Quando você entrar, ele deve buscar o seu registro a partir deste arquivo grande. Ao invés de enviá-lo por toda a sua conexão de Internet para o seu navegador, o servidor web solicita informações de um servidor de dados perto dele. Além de aliviar a Internet do tráfego de dados desnecessários, isso também melhora a segurança.

Podemos encontrar uma maior variedade de linguagens de programação em servidores do que em navegadores. Os programadores fazem mais *scripting* do lado do cliente com a linguagem Javascript. No lado do servidor, você pode escrever em linguagens como PHP, VBScript ou ColdFusion. Enquanto alguns programadores escrever scripts do lado do cliente para executar fora do navegador, isso é arriscado, uma vez que pressupõe que o computador sabe que a linguagem.

6. A linguagem PHP

O PHP (um acrônimo recursivo para *PHP: Hypertext Preprocessor*) é uma linguagem de *script open source* de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML, é uma das linguagens mais utilizadas na web. Milhões de sites no mundo inteiro utilizam PHP. A principal diferença em relação às outras linguagens é a capacidade que o PHP tem de interagir com o mundo web, transformando totalmente os websites que possuem páginas estáticas. Imagine, por exemplo, um website que deseja exibir notícias em sua página principal, mostrando a cada dia, ou a cada hora, notícias diferentes. Seria inviável fazer isso utilizando apenas HTML. As páginas seriam estáticas, e a cada notícia nova que aparecesse no site a página deveria ser alterada manualmente, e logo após enviada ao servidor por FTP (*File Transfer Protocol*) para que as novas notícias fossem mostradas

no site. Com o PHP, tudo isso poderia ser feito automaticamente. Bastaria criar um banco de dados onde ficariam armazenadas as notícias e criar uma página que mostrasse essas notícias, “puxando-as” do banco de dados. Agora imagine um site que possui cerca de cem páginas. Suponha que no lado esquerdo das páginas há um menu com links para as seções do site. Se alguma seção for incluída ou excluída, o que você faria para atualizar as cem páginas, incluindo ou excluindo esse novo link? Alteraria uma a uma, manualmente? Com certeza, você demoraria horas para alterar todas as páginas. E isso deveria ser feito cada vez que houvesse alteração, inclusão ou exclusão de uma seção no site. Para resolver esse problema utilizando PHP é muito simples. Basta construir um único menu e fazer todas as cem páginas acessarem esse arquivo e mostrá-lo em sua parte da esquerda. Quando alguma alteração for necessária, basta alterar um único arquivo, e as cem páginas serão alteradas automaticamente, já que todas acessam o mesmo menu.

Essas são apenas algumas das inúmeras vantagens das páginas que utilizam PHP. Você acabou de conhecer dois exemplos de sites em que a principal característica é o dinamismo e a praticidade. Automatização de tarefas, economia de tempo e de mão de obra são características evidentes nos dois exemplos citados. Mais adiante, veremos como implementar programas como os que foram citados aqui.

6.1 Características do PHP

- É uma linguagem de fácil aprendizado;
- Tem suporte a um grande número de bancos de dados como: dBase, Interbase, mSQL, MySQL, Oracle, Sybase, PostgreSQL e vários outros.
- Tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP;
- É multiplataforma, tendo suporte aos sistemas Operacionais mais utilizados no mercado;
- Seu código é livre, não é preciso pagar por sua utilização e pode ser alterado pelo usuário na medida da necessidade de cada usuário
- Não precisa ser compilado.

6.2 Embutido no HTML

Outra característica do PHP é que ele é embutido no HTML. Veremos mais adiante as facilidades que isso pode nos trazer. Uma página que contém programação PHP normalmente possui extensão .php (isso depende da configuração do seu servidor web). Sempre que o servidor web receber solicitações de páginas que possuem essa extensão, ele saberá que essa página possui linhas de programação. Porém, você verá que o HTML e o PHP estão misturados, pois começamos a escrever em PHP, de repente escrevemos um trecho em HTML, depois voltamos para o PHP, e assim por diante.

6.3 Baseado no servidor

O *JavaScript* que vocês já viram anteriormente, consiste em scripts que também são colocados nas páginas web, no meio do HTML, mas essa é uma programação que é executada no lado do cliente. Você abre seu browser (navegador) e acessa uma página que possui JavaScript. Essa página é carregada na memória da sua máquina, e o código JavaScript é executado consumindo os recursos de processamento do seu computador. Além disso, a programação escrita em JavaScript pode ser vista e copiada por qualquer pessoa. Para isso, basta escolher Exibir > Código-fonte no menu do navegador. O PHP é exatamente o contrário, pois é executado no servidor. Quando você acessa uma página PHP por meio de seu navegador, todo o código PHP é executado no servidor, e os resultados são enviados para seu navegador.

Portanto, o navegador exibe a página já processada, sem consumir recursos de seu computador. As linhas de programação PHP não podem ser vistas por ninguém, já que elas são executadas no próprio servidor, e o que retorna é apenas o resultado do código executado.

Há um exemplo simples para facilitar a compreensão: você já deve ter visto alguns sites que exibem a data e a hora atual em suas páginas. Se essas informações forem escritas utilizando *JavaScript*, a data e a hora mostradas serão retiradas do relógio do seu computador. Ou seja, para cada pessoa que acessar, a data e a hora mostradas serão diferentes, pois nem todos os computadores marcam exatamente o mesmo horário. Agora, se a data e a hora forem escritas utilizando PHP, essas informações serão retiradas do relógio do servidor, ou seja, há um relógio único, e por isso todos que acessarem o site ao mesmo tempo verão a mesma data e a mesma hora.

6.4 Bancos de dados

Diversos bancos de dados são suportados pelo PHP, ou seja, o PHP possui código que executa funções de cada um. Entre eles, temos MySQL, PostgreSQL, Sybase, Oracle, SQL Server e muitos outros. Cada um dos bancos de dados suportados pelo PHP possui uma série de funções que você poderá usar em seus programas para aproveitar todos os recursos. Os bancos de dados não suportados diretamente pelo PHP podem ser acessados via ODBC. Veremos exemplos de utilização do MySQL.

7. Iniciando em PHP

Um programa PHP pode ser escrito em qualquer editor de texto, como um bloco de notas, porém existem diversos editores específicos para PHP, que exibem cada elemento (variáveis, textos, palavras reservadas etc.) com cores diferentes, para melhor visualização, podemos destacar o *Sublime Text*, *Visual Studio Code* e o *Atom*. Mas antes precisamos configurar o ambiente para que tudo funcione.

8. Obtendo o PHP

Antes da instalação do PHP, devemos obter o software, por meio do download gratuito a partir da página oficial www.php.net. Há a necessidade, apenas, de escolher o formato no qual realizar o download e, então, podemos instalar o PHP no sistema.

Existem diversos pacotes de instalação automatizados, os quais facilitam o processo de instalação, pois recomendo isso no início de vocês por ser mais fácil, além de oferecer versatilidade em relação ao sistema operacional utilizado e às ferramentas ou aos módulos associados. Entre eles, podemos citar **XAMPP** (geralmente é o que utilizamos), Wampsever, Easy PHP.

O site oficial do PHP disponibiliza documentação, informações e manuais de utilização, além de todo o código-fonte do PHP e também versões pré-compiladas para os sistemas operacionais mais usados, como Windows e Unix.

Para a instalação do PHP, os pacotes essenciais são os seguintes:

- Linguagem de programação: PHP;
- Servidor de Internet, sendo o Apache o mais indicado;
- SGBD (Sistema Gerenciador de Banco de Dados), sendo o MySQL o mais indicado.

9. Sintaxe do PHP

9.1. Delimitando o código PHP

O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php
    comandos;
?>
```

```
<script language="php">
    comandos;
</script>
```

```
<?
    Comandos;
?>
```

```
<%
    comandos;
%>
```

O tipo de *tags* mais utilizado é o terceiro, que consiste em uma “abreviação” do primeiro. Para utilizá-lo, é necessário habilitar a opção *short-open_tag* na configuração do PHP. O último tipo serve para facilitar o uso por programadores acostumados à sintaxe de ASP. Para utilizá-lo também é necessário habilitá-lo no PHP, através do arquivo de configuração `php.ini`. Em nossas aulas adotarei o primeiro, por uma questão de costume.

9.2. Separador de instruções

Entre cada instrução em PHP é preciso utilizar o ponto-e-vírgula, assim como em C, Arduino, Java e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto e vírgula, mas por questões estéticas recomenda-se o uso sempre.

9.3. Comentários

Há dois tipos de comentários em código PHP:

Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de código PHP – o que vier antes. Pode ser delimitado pelo caractere “#” ou por duas barras (//). O delimitador “//”, normalmente, é o mais utilizado.

Exemplo:

```
<?php
echo “teste”; // este teste é similar ao anterior
echo “teste”; # este teste é similar ao anterior
// sql “teste”;
?>
```

Comentários de mais de uma linha:

Tem como delimitadores os caracteres “/*” para o início do bloco e “*/” para o final do comentário. Se o delimitador de final de código PHP (?>) estiver dentro de um comentário, não será reconhecido pelo interpretador.

Exemplo:

```
1 <?php
2     echo “teste”;
3     /*
4     este é um comentário com mais de uma linha.
5     echo “teste. Este comando echo é ignorado pelo interpretador do PHP por ser um comentário.”
6     */
7 ?>
```

9.4. Imprimindo código html

Um script php geralmente tem como resultado uma página html, ou algum outro texto.

Para gerar esse resultado, deve ser utilizada uma das funções de impressão, echo e print.

Sintaxes:

```
1 <?php
2     print(argumento);
3     echo (argumento1, argumento2, ... );
4     echo argumento;
5 ?>
```

Exemplos:

```
1 <?php
2     $a = 10;
3     $b = 20;
4     $soma = $a + $b;
5     //Imprimindo a soma na tela com o comando print.
6     print ("A soma é: ". $soma."<p>");
7
8     //Imprimindo a soma na tela com o comando echo.
9     echo ("A soma é: ". $soma."<p>");
10    echo "<br>";
11    echo $a." ".$b;
12 ?>
```

10 – Variáveis

10.1. Nomes de variáveis

Toda variável em PHP tem seu nome composto pelo caractere \$(dólar) e uma string, que deve iniciar por uma letra ou o caractere “_”. O PHP é case sensitive, ou seja, as variáveis \$vivas e \$VIVAS são diferentes.

Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui algumas variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

Exemplo de variáveis:

- \$teste = nome correto de variável
- teste= nome errado de variável.
- _teste= nome correto de variável

Obs.: Normalmente, a variável é criada utilizando-se, na maioria das vezes, o caractere \$ (dólar).

10.2. Tipos Suportados

O PHP suporta os seguintes tipos de dados:

- Inteiro
- Ponto flutuante
- *String*
- *Array*
- Objeto

O PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução.

Inteiros (integer ou long)

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

- \$vivas = 1234; # inteiro positivo na base decimal;
- \$vivas = -234; # inteiro negativo na base decimal;
- \$vivas = 0234; # inteiro na base octal-simbolizado pelo 0 equivale a 156 decimal;
- \$vivas = 0x34; # inteiro na base hexadecimal(simbolizado pelo 0x) – equivale a 52 decimal.

A diferença entre inteiros simples e long está no número de bytes utilizados para armazenar a variável.

Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

Ponto Flutuante (double ou float)

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

- \$vivas = 1.234;
- \$vivas = 23e4; # equivale a 230.000

Arrays

Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados. Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso. Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros. Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo. Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

Exemplo:

```
1 <?php
2     $cor[0] = "amarelo";
3     $cor[1] = "vermelho";
4     $cor[2] = "verde";
5     $cor[3] = "azul";
6     $cor[4] = "anil";
7     $cor["teste"] = 1;
8 ?>
```


Equivalentemente, pode-se escrever:

```
1 <?php
2 $cor = array(0 => "amarelo", 1 => "vermelho", 2 => "verde", 3 => "azul", 4 => "anil", teste => 1);
3 ?>
```

Booleanos

O PHP não possui um tipo booleano, mas é capaz de avaliar expressões e retornar true ou false, através do tipo integer: é usado o valor 0 (zero) para representar o estado false, e qualquer valor diferente de zero (geralmente 1) para representar o estado true.

10.3. Transformação de tipos

A transformação de tipos em PHP pode ser feita das seguintes maneiras:

a) Coerções

Quando ocorrem determinadas operações (“+”, por exemplo) entre dois valores de tipos diferentes, o PHP converte o valor de um deles automaticamente (coerção). É interessante notar que se o operando for uma variável, seu valor não será alterado.

O tipo para o qual os valores dos operandos serão convertidos é determinado da seguinte forma: Se um dos operandos for float, o outro será convertido para float, senão, se um deles for integer, o outro será convertido para integer.

Exemplo:

```
<?php
//declaração da variável vivas
//-----
$vivas = "1"; // $vivas é a string "1"
$vivas = $vivas + 1; // $vivas é o integer 2
$vivas = $vivas + 3.7; // $vivas é o double 5.7
$vivas = 1 + 1.5; // $vivas é o double 2.5
echo "O valor de vivas é: $vivas";

?>
```

Como podemos notar, o PHP converte string para integer ou double mantendo o valor. O sistema utilizado pelo PHP para converter de strings para números é o seguinte:

- É analisado o início da string. Se contiver um número, ele será avaliado. Senão, o valor será 0 (zero);
- O número pode conter um sinal no início (“+” ou “-”);
- Se a string contiver um ponto em sua parte numérica a ser analisada, ele será considerado, e o valor obtido será double;
- Se a string contiver um “e” ou “E” em sua parte numérica a ser analisada, o valor seguinte será considerado como expoente da base 10, e o valor obtido será double;

Exemplos:

```
1 <?php
2 $vivas = 1 + "10.5"; // $vivas == 11.5
3 $vivas = 1 + "-1.3e3"; // $vivas == -1299
4 $vivas = 1 + "teste10.5"; // $vivas == 1
5 $vivas = 1 + "10testes"; // $vivas == 11
6 $vivas = 1 + " 10testes"; // $vivas == 11
7 $vivas = 1 + "+ 10testes"; // $vivas == 1
8 echo "O valor de vivas é: $vivas";
9 ?>
```

b) Transformação explícita de tipos

A sintaxe do typecast de PHP é semelhante ao C: basta escrever o tipo entre parênteses antes do valor.

Exemplo:

```
<?php
$vivas = 15; // $vivas é integer (15)
$vivas = (double) $vivas; // $vivas é double (15.0)
$vivas = 3.9; // $vivas é double (3.9)
$vivas = (int) $vivas; // $vivas é integer (3)
// o valor decimal é truncado
echo "O valor de vivas é: $vivas";
?>
```

Os tipos permitidos na Transformação explícita são:

- (int), (integer) ----- = muda para integer;
- (real), (double), (float) ----- = muda para float;
- (string)----- = muda para string;
- (array) ----- = muda para array;
- (object) ----- = muda para objeto.

Com a função settype

A função settype converte uma variável para o tipo especificado, que pode ser "integer", "double", "string", "array" ou "object".

Sintaxe:

Settype(nomedavariável,novo tipo da variável)

Exemplo:

```
<?php
$vivas = 15; // $vivas é integer
settype($vivas,double); // $vivas é double
$vivas = 15; // $vivas é integer
settype($vivas,string); // $vivas é string
echo "O valor de vivas é: $vivas";
?>
```

11. Operadores

11.1 Operadores Aritméticos

Só podem ser utilizados quando os operandos são números (integer ou float). Se forem de outro tipo, terão seus valores convertidos antes da realização da operação. São eles:

- + adição
- subtração
- * multiplicação
- / divisão
- % módulo

11.2. Operadores de Strings

Só há um operador exclusivo para strings:

• concatenação

Exemplo:

```
1 <?php
2     $nome = "Marcos";
3     $sobrenome = "Costa";
4     echo $nome." ".$sobrenome;
5 ?>
```

A saída do script acima será: Marcos Costa

11.3. Operadores de atribuição

Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído. No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro. A atribuição é sempre por valor, e não por referência.

- = atribuição simples
- += atribuição com adição
- = atribuição com subtração

*= atribuição com multiplicação
 /= atribuição com divisão
 %= atribuição com módulo
 .= atribuição com concatenação

Exemplo:

```
1 <?php
2     $a = 7;
3     $a += 2; // $a passa a conter o valor 9
4 ?>
```

11.4. Operadores Lógicos

Utilizados para inteiros representando valores booleanos

- ! não (inversão)
- && “e” lógico
- || “ou” lógico

11.5. Operadores de Comparação

As comparações são feitas entre os valores contidos nas variáveis, e não as referências. Sempre retornam um valor booleano.

- == igual a
- === igual (conteúdo) e também (tipo de dado);
- != diferente de
- < menor que
- > maior que
- <= menor ou igual a
- >= maior ou igual a

11.6. Operadores de incremento e decremento

- ++ incremento
- -- decremento

Podem ser utilizados de duas formas: antes ou depois da variável. Quando utilizado antes, retorna o valor da variável antes de incrementá-la ou decrementá-la.

Quando utilizado depois, retorna o valor da variável já incrementado ou decrementado.

Exemplo:

```
1 <?php
2     $a = $b = 10; // $a e $b recebem o valor 10
3     $c = $a++; // $c recebe 10 e $a passa a ter 11
4     $d = ++$b; // $d recebe 11, valor de $b já incrementado
5 ?>
```

12. Estruturas de Controle

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando, portanto, descrever a sintaxe de cada uma delas, resumindo o funcionamento.

12.1. Blocos

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em PHP são delimitados pelos caracteres “{” e “}”. A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador.

Exemplo:

```
1 <?php
2 if ($x == $y)
3     comando1;
4     comando2;
5 ?>
```

Para que comando2 esteja relacionado ao if é preciso utilizar um bloco:

```
1 <?php
2     if ($x == $y) {
3         comando1;
4         comando2;
5     }
6 ?>
```

12.2. Estrutura if / else / elseif

O mais trivial dos comandos condicionais é o if. Ele testa a condição e executa o comando indicado se o resultado for true (valor diferente de zero). Ele possui duas sintaxes:

Sintaxes:

```
if (expressão)
    comando;
```

```
if (expressão) {
    comando;
    ...
    comando;
}
```

Para incluir mais de um comando no if da primeira sintaxe, é preciso utilizar um bloco, demarcado por chaves.

Exemplo 1:

```
1 <?php
2     $a = 10;
3     $b = 20;
4     if ($a > $b)
5         echo "o Valor de a é: ".$a + $b;
6 ?>
```

Exemplo 2:

```
1 <?php
2     $a = 30;
3     $b = 20;
4     if ($a >= $b) {
5         $media = ($a + $b) / 2;
6         $resto = $a % $b; //calcula o resto da divisão de A por B
7         echo " o Valor de A é: ".$a."<br>";
8         echo " o Valor de B é: ".$b."<br>";
9         echo " o Valor da média é: ".$media."<br>";
10        echo " O resto da divisão de A por B é: ".$resto;
11    }
12 ?>
```

12.2.1 Else

O else é um complemento opcional para o if. Se utilizado, o comando será executado se a expressão retornar o valor false (zero). Suas duas sintaxes são:

Sintaxes:

```
if (expressão)
    comando;

else
    comando;
```

ou

```
if (expressão) {
    comando 1;
    comando n
} else {
    comando 1;
    comando n
}
```

Exemplos:

```
<?
    $a = 10;
    $b = 20;
    if ($a > $b) {
        $maior = $a;
    } else {
        $maior = $b;
    }
?>
```

O exemplo acima coloca em \$maior o maior valor entre \$a e \$b.

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
    comando1;
else
    if (expressao2)
        comando2;
    else
        if (expressao3)
            comando3;
        else
            comando4;
```

Foi criado o comando, também opcional elseif. Ele tem a mesma função de um else e um if usados sequencialmente, como no exemplo acima. Num mesmo if podem ser utilizados diversos elseif's, ficando essa utilização a critério do programador, que deve zelar pela legibilidade de seu script.

12.2.2 Elseif

O comando elseif também pode ser utilizado com dois tipos de sintaxe. Em resumo, a sintaxe geral do comando if fica das seguintes maneiras:

Sintaxe:

<pre>if (expressao1) comando; elseif (expressao2) comando; else comando;</pre>	ou	<pre>if (expressão 1) { comando 1; comando n; } elseif (expressão 2) { comando 1; comando n; } else { comando 1; comando n; }</pre>
--	----	---

Exemplo:

```
<?php
    $nota1 = 6;
    $nota2 = 8;
    $media = ($nota1 + $nota2) / 2;

    if ($media > 7) {
        echo "Média: ".$media."<br>";
        echo "Aluno aprovado.";
    }elseif ($media < 7) {
        echo "Média: ".$media."<br>";
        echo "Aluno reprovado.";
    }else {
        echo "Média: ".$media."<br>";
        echo "Aluno em recuperação.";
    }
?>
```


12.3 – Exercícios:

1- Crie um algoritmo que receba um número digitado pelo usuário e verifique se esse valor é positivo, negativo ou igual a zero. A saída deve ser: "Valor Positivo", "Valor Negativo" ou "Igual a Zero".

2 - Faça um algoritmo PHP que receba os valores A e B, imprima-os em ordem crescente em relação aos seus valores. Exemplo, para A=5, B=4. Você deve imprimir na tela: "4 5". Dica, utilizem concatenação.

3 - Crie um algoritmo para calcular a média final de um aluno, para isso, solicite a entrada de três notas e as insira em um *array*, por fim, calcule a média geral. Caso a média seja maior ou igual a seis, exiba aprovado, caso contrário, exiba reprovado. Exiba também a média final calculada.

Ex: N1 = 5 | N2 = 10 | N3 = 4 | MG = 6,33 [Aprovado].

4 - Ler um número inteiro entre 1 e 12 e escrever o mês correspondente. Caso o número seja fora desse intervalo, informar que não existe mês com este número. Exigência, resolva esse exercício utilizando array.

ATENÇÃO: Para cada exercício, criem um arquivo .php (Exemplo: Exercicio1.php) e a correção será no dia 10/04/2025.

12.4. Estrutura Switch Case

O comando switch atua de maneira semelhante a uma série de comandos if na mesma expressão.

Frequentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável. Quando isso for necessário, deve-se usar o comando switch.

Sintaxe:

```
switch ($valor){
    case "_____";
        comandos;
        comandos;
        break;
    case "_____";
        comandos;
        comandos;
        break;
    case "_____";
        comandos;
        comandos;
        break;
    default;
        comandos;
        comandos;
        break;
}
```

O exemplo seguinte mostra dois trechos de código que fazem a mesma coisa, sendo que o primeiro utiliza uma série de if's e o segundo utiliza switch:

```
<?
    $i = 1;
    if ($i == 0){
        print "i é igual a zero";
    }elseif ($i == 1){
        print "i é igual a um";
    }elseif ($i == 2){
        print "i é igual a dois";
    }
?>
```

Exemplo 2: Estrutura SWITCH

```
<?
    $i = 0;
    switch ($i) {
    case 0:
        print "i é igual a zero";
        break;
    case 1:
        print "i é igual a um";
        break;
    case 2:
        print "i é igual a dois";
        break;
    }
?>
```

É importante compreender o funcionamento do switch para não cometer enganos. O comando switch testa linha a linha os cases encontrados, e a partir do momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outro teste, até o fim do bloco.

Por isso usa-se o comando break, quebrando o fluxo e fazendo com que o código seja executado da maneira desejada. Veremos mais sobre o break mais adiante.

Em outras linguagens que implementam o comando switch, ou similar, os valores a serem testados só podem ser do tipo inteiro.

Em PHP é permitido usar valores do tipo String como elementos de teste do comando switch. O exemplo abaixo funciona perfeitamente:

```
<?
    $s = "casa";
    switch ($s) {
    case "casa":
        print "A casa é amarela";
        break;
    case "arvore":
        print "A árvore é bonita";
        break;
    }
?>
```

12.5. Estruturas de Repetição

As estruturas de repetição são utilizadas quando o programador precisa, por exemplo, repetir o mesmo comando várias vezes. Vamos ver as estruturas de repetição existentes.

12.5.1. While

O while é o comando de repetição (laço) mais simples. Ele testa uma condição e executa um comando, ou um bloco de comandos, até que a condição testada seja falsa. Assim como o if, o while também possui duas sintaxes alternativas:

Sintaxes:

```
While (condição)
    comando;
```

Ou

```
While (condição) {
    comandos;
    comandos;
}
```

Exemplo:

O exemplo a seguir mostra o uso do while para imprimir os números de 1 a 10:

```
<?php
    $i = 1;
    while ($i <=10){
        print $i++;
        print "<br>";
    }
?>
```

12.5.2. Do ... While

O laço do..while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos.

Sintaxe:

```
do {
    comando
    ...
    comando
}
while (<condição>);
```

Exemplo:

```
<?php
    $a = 1;
    do {
        echo $a++;
        echo "<br>";
    }
    while ($a <=10)
?>
```

12.5.3. For

O tipo de laço mais complexo é o for. Para os que programam em C, C++ ou Java, a assimilação do funcionamento do for é natural. Mas para aqueles que estão acostumados a linguagens como Pascal, há uma grande mudança para o uso do for.

Sintaxe:

```
for (<inicializacao>;<condicao>;<incremento>) {
    <comando>;
    ...
    <comando>;
}
```

As três expressões que ficam entre parênteses têm as seguintes finalidades:

- Inicialização: comando ou sequência de comandos a serem realizados antes do início do laço. Serve para inicializar variáveis.
- Condição: Expressão booleana que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira (valor diferente de zero) os comandos serão executados.
- Incremento: Comando executado ao final de cada execução do laço.

Um comando for funciona de maneira semelhante a um while escrito da seguinte forma:

Exemplo:

```
<?php
    for ($a = 1; $a <= 10; $a++) {
        echo "O valor de A é: ". $a;
        echo "<br>";
    }
?>
```

12.5.3. Foreach

O FOREACH tem o mesmo funcionamento do FOR, porém, ele não precisa de contador. Ou seja, você vai correr o laço sabendo em que posição está. Veja abaixo o primeiro exemplo da sintaxe do FOREACH:

```
1 <?php
2     $frutas = array('maca','banana','melancia','melao',
3                     'abacaxi','laranja');
4     foreach($frutas as $fruta)
5     {
6         echo "A fruta e: ". $fruta . "<br>";
7     }
8 ?>
```

12.6 – Exercícios de fixação:

1 - Crie as soluções abaixo utilizando a estrutura switch case:

- Elabore um algoritmo que leia dois valores do usuário e a operação que ele deseja executar (Operações: soma, subtração, divisão, multiplicação). Execute a operação desejada e mostre o resultado;
- Faça um algoritmo que aborde a seguinte situação: Uma loja fornece 10% de desconto para funcionários e 5% de desconto para clientes vips. Faça um programa que calcule o valor total a ser pago por uma pessoa. O script deverá ler o valor total da compra e um código que identifique se o comprador é um cliente comum (1), funcionário (2) ou vip (3);
- Faça um algoritmo PHP que calcule e imprima o salário reajustado de um funcionário de acordo com a seguinte regra:
 - salários até 300, reajuste de 50%
 - salários maiores que 300, reajuste de 30%.

2 – Crie as soluções abaixo utilizando estruturas de repetição de acordo com sua escolha:

- Faça um algoritmo em PHP que receba um valor qualquer e imprima os valores de 0 até o valor recebido, exemplo:
 - Valor recebido = 9
 - Impressão do programa – 0 1 2 3 4 5 6 7 8 9
- Faça um algoritmo PHP que receba um valor qualquer e calcule o seu fatorial (!), sabendo que fatorial de um número é: 7! = 7*6*5*4*3*2*1 4! = 4*3*2*1
- Faça um algoritmo PHP que receba dois valores quaisquer e imprime todos os valores intermediários a ele, veja exemplo: Primeiro Valor = 5 Segundo Valor = 15 Imprime: 6 7 8 9 10 11 12 13 14
- **Desafio, pesquise.** Faça um algoritmo PHP que receba uma string, encontre o número total de caracteres desta e imprima todos os números que existem entre 0 e o número total, exemplo:

```
* string = "Gil Eduardo de Andrade"
* total_caracter = 22
```

Pessoal, criem os scripts .php e veremos em aula, fazer até 08/09/2022, para nota, irei corrigir em aula, poderá ser realizada em duplas.

Com isso, finalizamos essa parte introdutória da linguagem PHP.

13. O Banco de Dados *MySQL*

O *MySQL* é um dos sistemas de gerenciamento de banco de dados mais populares que existe e, por ser otimizado para aplicações Web, é amplamente utilizado na internet. É muito comum encontrar serviços de hospedagem de sites que oferecem o *MySQL* e a linguagem PHP, justamente porque ambos trabalham muito bem em conjunto.

Outro fator que ajuda na popularidade do *MySQL* é sua disponibilidade para praticamente qualquer sistema operacional, como *Linux*, *FreeBSD* (e outros sistemas baseados em Unix), *Windows* e *Mac OS X*. Além disso, o *MySQL* é um software livre sob licença GPL (*General Public License*), o que significa que qualquer um pode estudá-lo ou alterá-lo conforme a necessidade.

Entre as características técnicas do SGBD *MySQL*, estão:

- Alta compatibilidade com linguagens como PHP, Java, Python, C#, Ruby e C/C++;
- Baixa exigência de processamento (em comparação com outros SGBD);
- Vários sistemas de armazenamento de dados (database engine), como *MyISAM*, *MySQL Cluster*, *CSV*, *Merge*, *InnoDB*, entre outros;
- Recursos como *transactions* (transações), conectividade segura, indexação de campos de texto, replicação, etc;
- Instruções em *SQL* como indicam o nome.

13.1 Utilização

Uma característica fundamental do *MySQL*, quiçá na origem do seu sucesso, é ser desenvolvido em código aberto e funcionar num grande número de sistemas operacionais : *Windows*, *Linux*, *FreeBSD*, *BSDI*, *Solaris*, *Mac OS X*, *SunOS*, *SGI*, etc.

É reconhecido pelo seu desempenho e robustez e também por ser multitarefa e multiusuário. A própria Wikipédia, usando o programa *MediaWiki*, utiliza o *MySQL* para gerenciar seu banco de dados, demonstrando que é possível utilizá-lo em sistemas de produção de alta exigência e em aplicações sofisticadas.

No passado (até à versão 3.x), devido a não possuir funcionalidades consideradas essenciais em muitas áreas, como *stored procedures*, *two-phase commit*, *subselects*, *foreign keys* ou integridade referencial, era frequentemente considerado um sistema mais "leve" e para aplicações menos exigentes, sendo preterido por outros sistemas como o *PostgreSQL*.

13.2 Características do *MySQL*

- Banco de dados de código aberto e gratuito;
- As tabelas criadas podem ter tamanho de até 4 GB;
- Suporta diferentes plataformas: Win32, Linux, FreeBSD, Unix, etc;
- Suporte às API's das Seguintes linguagens: PHP, Perl, C,C++,Java, Pynthon, etc;
- Suporte a múltiplos processadores;
- Um sofisticado sistema de senhas criptografadas flexível e Seguro.
- Suporte à ODBC, você pode facilmente conectar o Access a um banco de dados do *MySQL*;
- Suporta até 16 índices por tabela;
- Código fonte escrito em C e C++ e testado com uma variedade de diferentes compiladores;
- O Cliente conecta no *MySQL* através de conexões TCP/IP;
- Capacidade para manipular bancos com até 50 milhões de registros;
- Reduz a administração, engenharia e a sustentação custa por até 50%.

13.3 O Que o *MySQL* Faz de Melhor

- Aplicações Web;
- Aplicações de nível corporativo;
- Suporte a código fonte aberto;
- Requisitos de sistema baixo;
- Tabelas com tamanho grande;
- Estabilidade.

13.4 Segurança no MySQL

O MySQL possui componentes de segurança contra ameaças externas como crackers e outros, e também proteger os dados dos próprios usuários. O mysql apresenta vários níveis de segurança em relação ao acesso. Todas as informações de segurança estão armazenadas no banco MySQL.

A filosofia de segurança em banco de dados refere-se a fornecer ao usuário apenas o que é essencial para o seu trabalho.

13.5 O MySQL é Gratuito?

Pessoas confundem "free" com "grátis" o que é comum aqui no Brasil. Mas em se tratando de software este "free" é de *open source* e não gratuito. Para poder utilizar o MySQL sob a licença GPL (*General Public License*) e não precisar pagar, o produto desenvolvido precisa ser GPL (*General Public License*) também, senão, orientamos a compra da licença comercial, com baixo custo, sendo comercializada por servidor, sem limites de usuários e processadores e ainda com garantia perpétua de atualização de versão para o resto da vida.

13.6 Principais Tipos de Dados

INT: Representa um valor inteiro. Pode ser com sinal ou sem sinal

REAL: Representa um valor com ponto flutuante. Oferece uma grande precisão e uma extensa faixa de valores

CHAR(n): Representa um valor caractere com tamanho fixo.

TEXT: Representa um valor para caractere com tamanho variável

DATE: Representa um valor de data padrão. Formato: YYYY-MM-DD (2001-01-01)

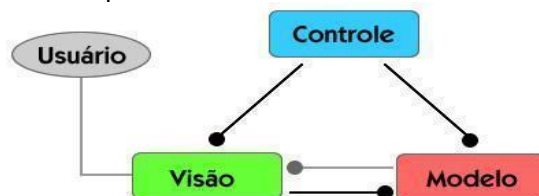
TIME: Representa um valor de tempo padrão. Armazena a hora de um dia independente de uma data particular. Formato : hh:mm:ss (06:00:00).

14. Padrão MVC

MVC é o acrônimo de *Model-View-Controller* (em português, Modelo-Visão-Controle) e que, como o próprio nome supõe, separa as camadas de uma aplicação em diferentes níveis. Ao contrário do que muitos desenvolvedores dizem, o MVC não é um Padrão de Projeto, mas sim um Padrão de Arquitetura de Software.

O MVC define a divisão de uma aplicação em três componentes: Modelo, Visão e Controle. Cada um destes componentes tem uma função específica e estão conectados entre si. O objetivo é separar a arquitetura do software para facilitar a compreensão e a manutenção.

A camada de Visão é relacionada ao visual da aplicação, ou seja, as telas que serão exibidas para o usuário. Nessa camada apenas os recursos visuais devem ser implementados, como janelas, botões e mensagens. Já a camada de Controle atua como intermediária entre as regras de negócio (camada Modelo) e a Visão, realizando o processamento de dados informados pelo usuário e repassando-os para as outras camadas. E por fim, temos a camada de Modelo, que consiste na essência das regras de negócio, envolvendo as classes do sistema e o acesso aos dados. Observe que cada camada tem uma tarefa distinta dentro do software.



14.1 Explicação do conceito do MVC

14.1.1 Certo, mas qual é a vantagem?

A princípio, pode-se dizer que o sistema fica mais organizado quando está estruturado com MVC. Um novo desenvolvedor que começar a trabalhar no projeto não terá grandes dificuldades em compreender a estrutura do código. Além disso, as exceções são mais fáceis de serem identificadas e tratadas. Ao invés de revirar o código atrás do erro, o MVC pode indicar em qual camada o erro ocorre. Outro ponto importante do MVC é a segurança da transição de dados. Através da camada de Controle, é possível evitar que qualquer dado inconsistente chegue à camada de Modelo para persistir no banco de dados. Imagine a camada de Controle como uma espécie de Firewall: o usuário entra com os dados e a camada de Controle se responsabiliza por

bloquear os dados que venham a causar inconsistências no banco de dados. Portanto, é correto afirmar que essa camada é muito importante.

14.1.2 Posso ter apenas três camadas no MVC?

Eis que essa é outra dúvida bastante questionada pelos desenvolvedores. Embora o MVC sugira a organização do sistema em três camadas, nada impede que você “estenda” essas camadas para expandir a dimensão do projeto. Por exemplo, vamos supor que um determinado projeto tenha um módulo web para consulta de dados. Aonde esse módulo se encaixaria dentro das três camadas?

Bem, este módulo poderia ser agrupado com os outros objetos da camada Visão, mas seria bem mais conveniente criar uma camada exclusiva (como “Web”) estendida da camada de Visão e agrupar todos os itens relacionados a esse módulo dentro dela. O mesmo funcionaria para um módulo Mobile ou Webservice. Neste caso, a nível de abstração, essas novas camadas estariam dentro da camada de Visão, mas são unidades estendidas.

14.1.3 Então podemos concluir resumidamente

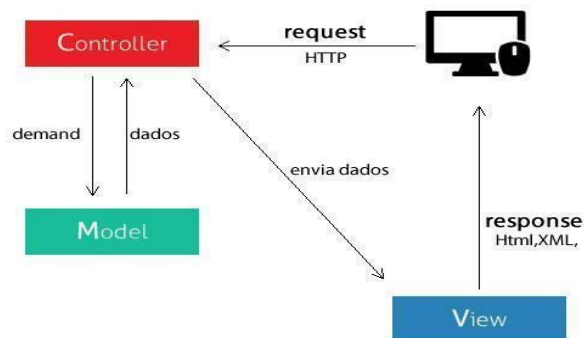
MVC é nada mais que um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário (*view*), a camada de manipulação dos dados (*model*) e a camada de controle (*controller*).

Model: quando você pensar em manipulação de dados, pense em model. Ele é responsável pela leitura e escrita de dados, e também de suas validações.

View: é a camada de interação com o usuário. Ela apenas faz a exibição dos dados.

Controller: é o responsável por receber todas as requisições do usuário. Seus métodos chamados *actions* são responsáveis por uma página, controlando qual *model* usar e qual *view* será mostrado ao usuário.

(A imagem a seguir representa o fluxo do MVC em um contexto de Internet, com uma requisição HTTP e resposta em formato HTML ou XML)



14.1.4 O diálogo das camadas na Web (Brincadeirinha)

View - Fala **Controller**! O usuário acabou de pedir para acessar o Facebook! Pega os dados de *login* dele aí.

Controller – Blz. Já te mando a resposta. Aí *model*, meu parceiro, toma esses dados de *login* e verifica se ele loga.

Model – Os dados são válidos. Mandando a resposta de login.

Controller – Blz. **View**, o usuário informou os dados corretos. Vou mandar pra vc os dados dele e você carrega a página de perfil.

View – Vlw. Mostrando ao usuário...

15. Introdução ao CodeIgniter

O *CodeIgniter* (ou CI, como muitos chamam, e nós também) é um *framework* MVC *open source*, escrito em PHP e mantido atualmente pelo *British Columbia Institute of Technology* e por uma grande comunidade de desenvolvedores ao redor do mundo. Sua simplicidade faz dele um dos mais utilizados e com uma curva de aprendizado bem pequena.

Sua documentação é bem completa e detalhada, facilitando o processo de pesquisa sobre determinada biblioteca ou funcionalidade. Com isso, o tempo gasto com a leitura da documentação diminui, e você pode passar mais tempo trabalhando no código do seu projeto. Com o CI, é possível desenvolver sites, APIs e sistemas das mais diversas complexidades, tudo de forma otimizada, organizada e rápida. Suas bibliotecas nativas facilitam ainda mais o processo de desenvolvimento, e ainda permitem ser estendidas para que o funcionamento se adapte à necessidade de cada projeto. Diversas bibliotecas de terceiros (*third-party*) estão disponíveis no *GitHub*, *Composer* e em outros repositórios de arquivos, e podem ser muito úteis.

Mais detalhes sobre o CI podem ser vistos no site do *framework* (<https://codeigniter.com>) e em sua documentação (https://codeigniter.com/user_guide).

15.1 Requisitos mínimos

Para um melhor aproveitamento, é recomendado o uso de PHP 5.4 ou mais recente. Ele até funciona com PHP 5.2, mas é recomendado que você não utilize versões antigas do PHP, por questões de segurança e desempenho potenciais, bem como recursos ausentes.

Algumas aplicações fazem uso de banco de dados, e o *CodeIgniter* suporta atualmente:

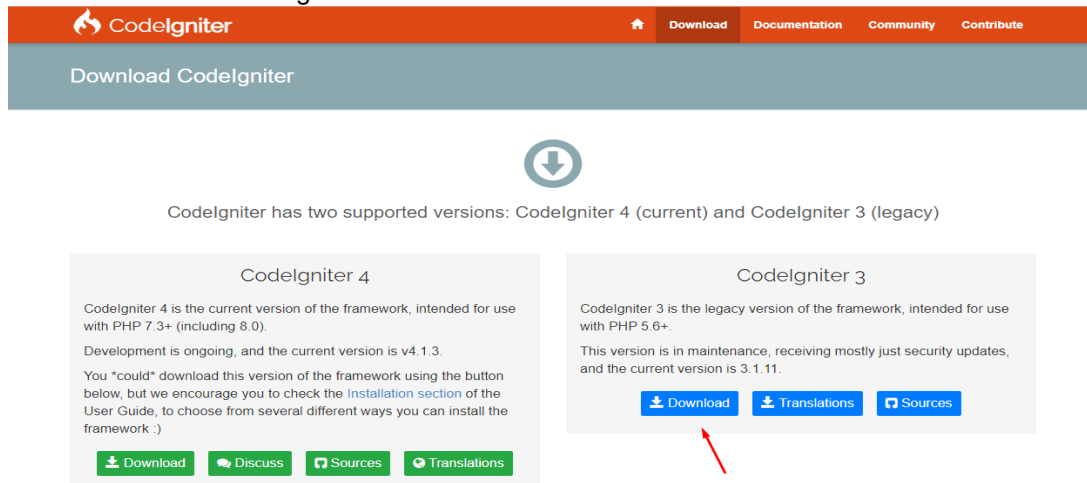
- MySQL(5.1+), mysqli e PDO drivers
- Oracle (drivers oci8 e PDO)
- PostgreSQL (postgre a PDO)
- MSSQL (mssql, sqlsrv e PDO)
- Interbase/Firebird (ibase e PDO)
- ODBC (odbc e PDO)

O CI pode ser instalado em sistemas operacionais UNIX (Linux e Mac OS X) e Windows, desde que o ambiente de desenvolvimento com Apache ou Nginx (UNIX) e IIS (Windows) estejam devidamente montados e funcionando.

Para nossas aulas trabalharemos com a versão do *CodeIgniter* 3, você pode fazer o *download* em <https://codeigniter.com/download> , mas irei disponibilizar o arquivo .zip no *Teams*.

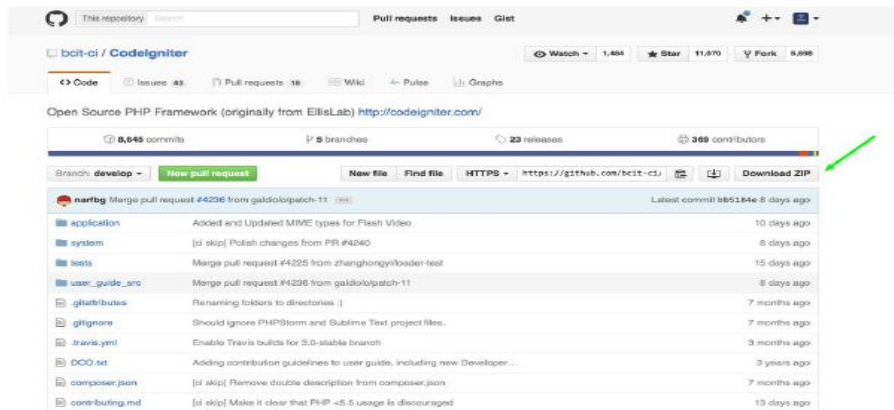
15.2 Instalando o CodeIgniter

O processo de instalação do CI é muito simples e fácil de ser executado. O primeiro passo é fazer o *download* do *framework* em nosso caso irei subir o arquivo no *Microsoft Teams* para *download*. Para isso, você tem duas possibilidades: Clique no link que passei acima, e faça conforme mostrado na figura:



Nossas aulas nesse momento estão baseadas no *CodeIgniter* 3.

Acesse o repositório do projeto no *GitHub* (<https://github.com/bcit-ci/CodeIgniter>) e faça o *download* clicando em *Download ZIP*, conforme a figura a seguir:



15.3 Estrutura de arquivos e diretórios do CodeIgniter

Agora que o *CodeIgniter* já está instalado e funcionando, mostrarei com mais detalhes a estrutura de arquivos e diretórios. É importante conhecê-la para que possa organizar o seu código da melhor maneira possível, e fazer o uso correto dos recursos que o CI disponibiliza.

Diretório *application*

Esse é o diretório da aplicação, onde ficarão todos os arquivos relacionados ao projeto.

Ele é composto de 12 diretórios, que farão com que os arquivos do projeto fiquem bem divididos e separados dos arquivos do "core" do CI. Assim, você poderá atualizar a versão do CI sem ter de mudar os arquivos do projeto de diretório ou estrutura.

- **cache** — Diretório que armazena os arquivos que são colocados em cache.
- **config** — Diretório que armazena os arquivos de configuração do CI, como por exemplo, *database.php*, *constants.php*, *routes.php*, entre outros que veremos no decorrer das aulas.
- **controllers** — Diretório que armazena os arquivos com os *controllers* do projeto. Ao instalar o CI, ele já vem com um *controller* criado, que é o *Welcome.php*.
- **core** — Diretório usado para estender classes e funcionalidades do core do CI, adaptando-se às necessidades do projeto.
- **helpers** — Diretório que armazena os arquivos com funções que funcionarão como assistentes durante o desenvolvimento. Por exemplo, você pode criar um helper (assistente) para determinado grupo de tarefas realizadas com frequência dentro do projeto, ou então estender as funcionalidades dos *helpers* nativos do CI.
- **hooks** — Diretório que armazena os arquivos que também estendem funcionalidades padrão do CI. Você pode criar um hook para alterar uma funcionalidade padrão, como por exemplo, minificar o código HTML de saída quando o método *load->view()* for executado.
- **language** — Diretório que armazena os arquivos com os dicionários de idiomas, assim você pode desenvolver projetos multi-idiomas de forma fácil, e também utilizar os outros idiomas dos pacotes de tradução oficiais do CI, que podem ser encontrados em <https://github.com/bcit-ci/codeigniter3-translations>.
- **libraries** — Diretório que armazena as *libraries* (bibliotecas) criadas para o projeto, ou *libraries* estendidas do core do CI.
- **logs** — Diretório que armazena os arquivos de log, que são configurados em *application/config/config.php*.
- **models** — Diretório que armazena os arquivos onde ficarão as regras de negócio do projeto.
- **third_party** — Diretório que armazena código de terceiros, como classes que podem auxiliar em alguma rotina do projeto e que foram desenvolvidas por outros programadores e/ou não possuem os padrões do CI.
- **views** — Diretório que armazena arquivos que serão carregados no *browser*. Você pode inserir outros diretórios dentro dele para organizar os arquivos da melhor maneira possível.

Nem sempre você fará uso de todos os diretórios dentro de *application*, os mais utilizados são: **config**, **controllers**, **libraries**, **logs**, **models** e **views**. Mas o uso fica a critério do desenvolvedor e da necessidade do projeto.

Diretório system

Esse diretório armazena o *core* do CI, e o conteúdo dos arquivos contidos nesse diretório não devem ser alterados. Isso porque, para manter o *CodeIgniter* atualizado, na maioria das versões basta substituir o conteúdo desse diretório pelo conteúdo do mesmo diretório na versão mais recente.

Ele possui apenas seis outros diretórios, muito bem divididos, e com os arquivos nomeados de forma bem intuitiva. Assim, caso queira estudar mais a fundo o funcionamento do CI, você pode fazê-lo analisando o código-fonte dos arquivos desse diretório.

Arquivo index.php

O arquivo *index.php* é o arquivo base de um projeto feito com CI. Ele carrega o arquivo de *core* necessário para a carga das *libraries*, *helpers*, classes, entre outros arquivos do framework e execução do código escrito por você.

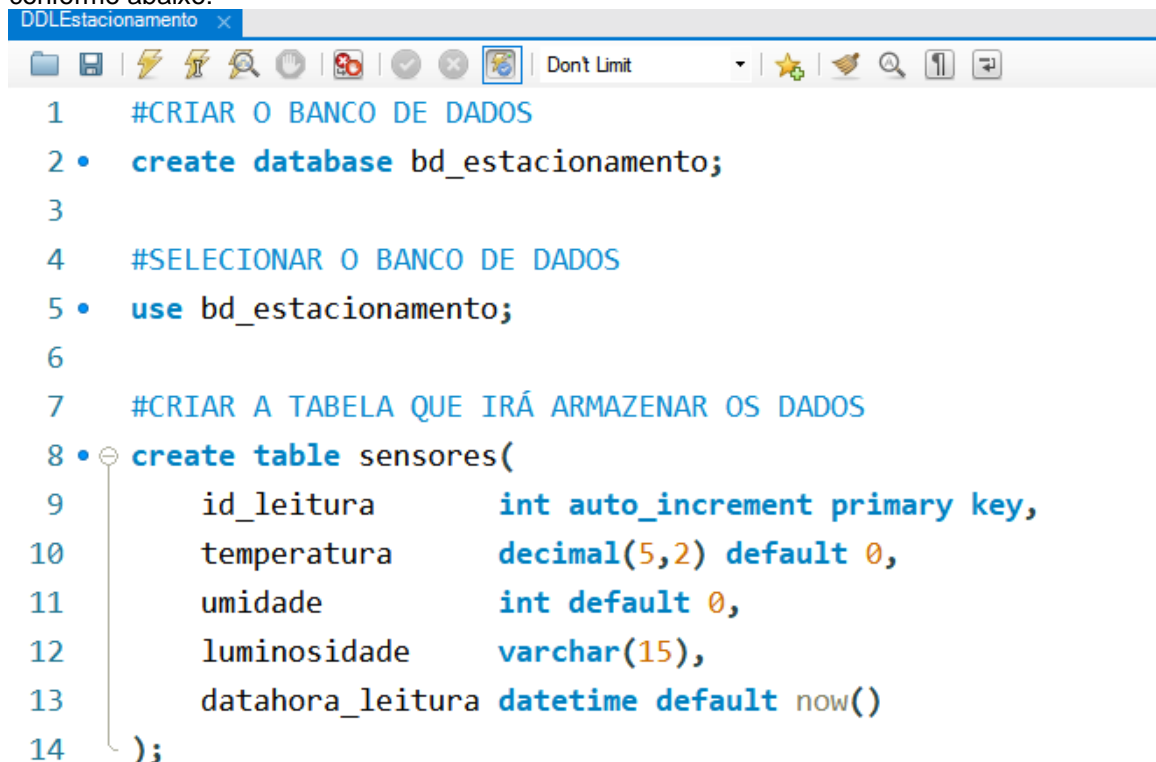
Nesse arquivo, você pode alterar a localização dos diretórios *system* e *application*, configurar as exibições de erro para os ambientes do projeto (desenvolvimento, teste e produção, por exemplo), customizar valores de configurações, entre outras possibilidades. Quase não se faz alteração nesse arquivo, mas durante as aulas serão feitas algumas, para que possam atender aos requisitos dos exemplos.

16 Mãos à obra

Agora vamos implementar efetivamente nosso código para fazermos nossa comunicação entre nosso servidor, o esp32, nosso protótipo, para isso teremos que ter um computador, que fará o papel do servidor, assim, ele terá que ter os seguintes programas instalados: XAMPP (Para fazer o papel do servidor APACHE para o PHP e o Servidor MySQL).

16.1 Banco de dados

Vamos criar o banco de dados com uma tabela que terá a incumbência de armazenar os dados lidos pelos sensores de nossa maquete, vocês definiram os mesmos e iremos codificar conforme abaixo:



```

DDL Estacionamento
1  #CRIAR O BANCO DE DADOS
2  • create database bd_estacionamento;
3
4  #SELECIONAR O BANCO DE DADOS
5  • use bd_estacionamento;
6
7  #CRIAR A TABELA QUE IRÁ ARMAZENAR OS DADOS
8  • create table sensores(
9      id_leitura      int auto_increment primary key,
10     temperatura     decimal(5,2) default 0,
11     umidade         int default 0,
12     luminosidade    varchar(15),
13     datahora_leitura datetime default now()
14 );
  
```

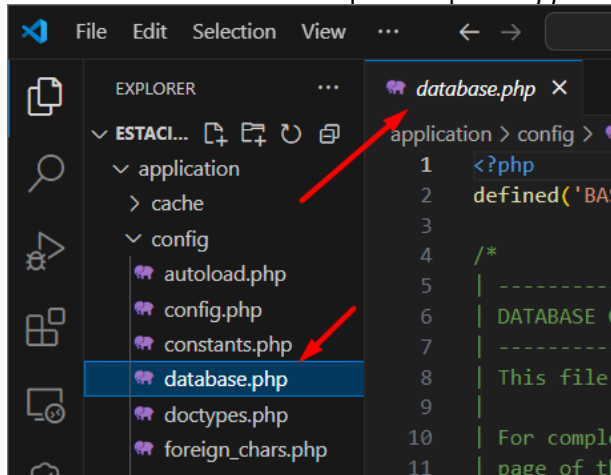
16.2 Configurando o CodeIgniter para nosso projeto

Nesse momento teremos que configurar nosso *framework* para a estrutura de nosso projeto, irei disponibilizar a pastaBase do projeto que chamaremos de Estacionamento que deverá estar na Pasta C:\XAMP\htdocs, e após isso, iremos configurar alterando os seguintes arquivos na nossa estrutura:

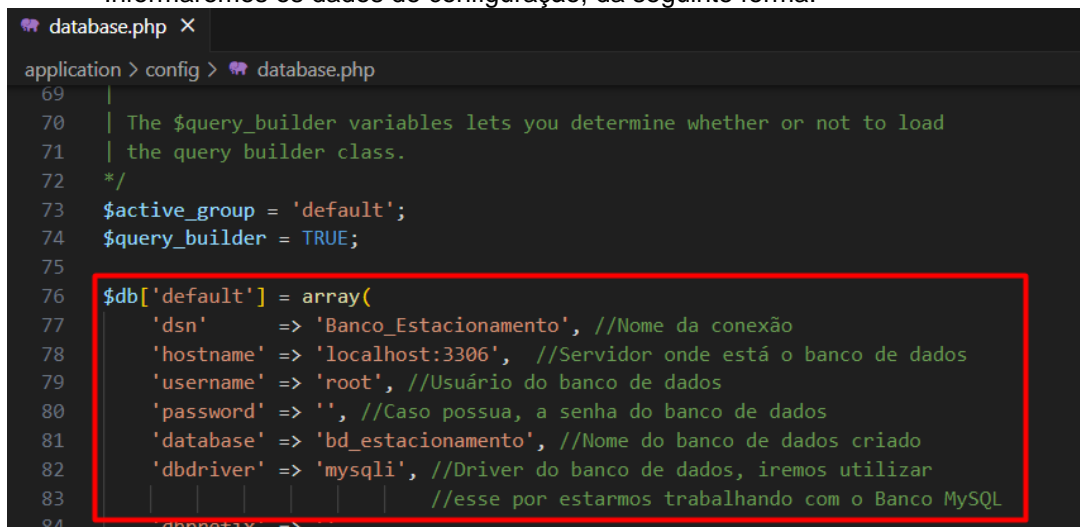
- database.php (configuraremos o banco de dados de estágio);
- autoload.php (configuraremos a chamada automática do banco de dados);
- routes.php (configuraremos a rota de nossa *controller* inicial do projeto);
- config.php (configuraremos nossa *base_url()*);
- htaccess (URLs amigáveis).

16.3 Configurando o arquivo *databases.php*

Vamos acessar o arquivo na pasta *application/config*, assim:

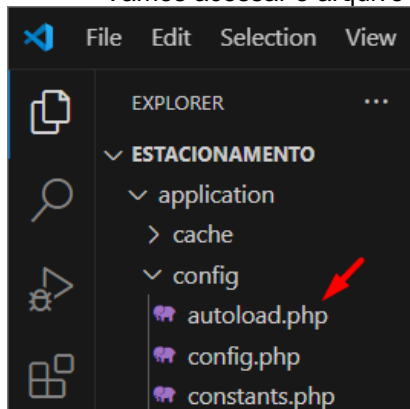


Informaremos os dados de configuração, da seguinte forma:

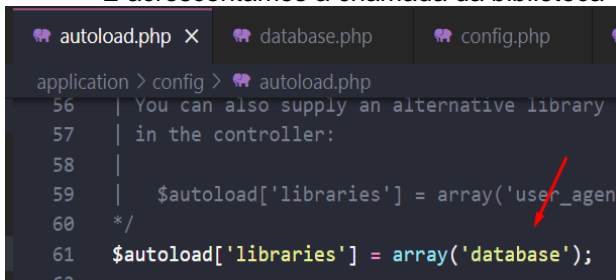


16.4 Configurando o arquivo *autoload.php*

Vamos acessar o arquivo na pasta *application/config*, assim:



E acrescentamos a chamada da biblioteca *database*, conforme abaixo:



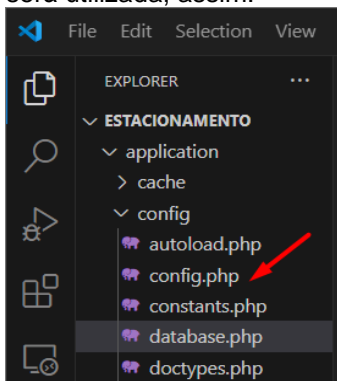
```

application > config > autoload.php
56 | You can also supply an alternative library
57 | in the controller:
58 |
59 | $autoload['libraries'] = array('user_agent
60 | */
61 | $autoload['libraries'] = array('database');

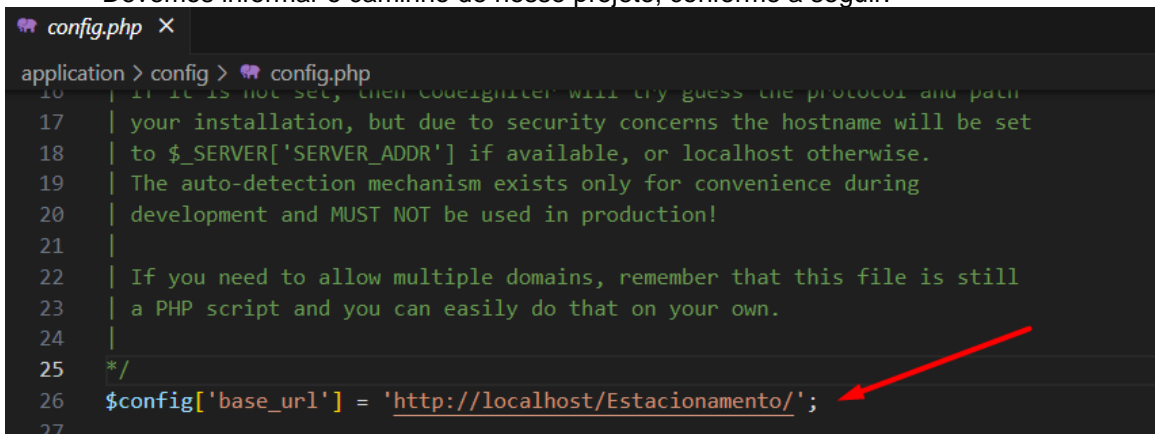
```

16.5 Configurando o arquivo *config.php*

Vamos acessar o arquivo na pasta *application/config*, para vamos configurar a URL que será utilizada, assim:



Devemos informar o caminho de nosso projeto, conforme a seguir:



```

application > config > config.php
16 | If it is not set, then CodeIgniter will try guess the protocol and path
17 | your installation, but due to security concerns the hostname will be set
18 | to $_SERVER['SERVER_ADDR'] if available, or localhost otherwise.
19 | The auto-detection mechanism exists only for convenience during
20 | development and MUST NOT be used in production!
21 |
22 | If you need to allow multiple domains, remember that this file is still
23 | a PHP script and you can easily do that on your own.
24 |
25 | */
26 | $config['base_url'] = 'http://localhost/Estacionamento/';
27 |

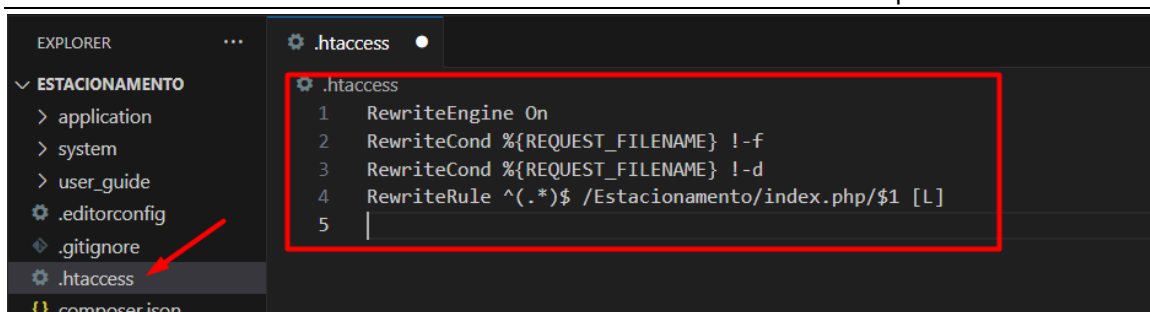
```

16.6 Configurando o arquivo *htaccess*

O CodeIgniter por padrão já dá suporte a URL amigáveis, como os grandes frameworks em PHP, o grande problema é que ele ainda insiste em mostrar o arquivo *index.php* na URL o que deixa toda requisição horrível. Veja o padrão de URL na instalação padrão do CodeIgniter: **<http://seusite.com.br/index.php/controller/method/parameter>**

Como deveria ser: **<http://seusite.com.br/controller/method/parameter>**

Como estamos usando o Apache como nosso servidor *web*, criamos um arquivo *.htaccess* no caminho "C:\XAMP\HTDOCS\Estagio com o seguinte conteúdo (irei fornecer a pasta para vocês, mas irei mostrar como o arquivo é composto, conforme a seguir:



```

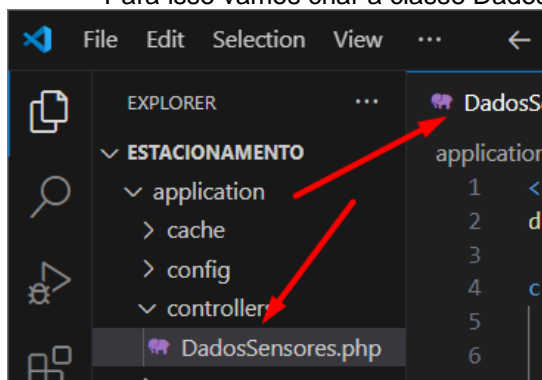
1 RewriteEngine On
2 RewriteCond %{REQUEST_FILENAME} !-f
3 RewriteCond %{REQUEST_FILENAME} !-d
4 RewriteRule ^(.*)$ /Estacionamento/index.php/$1 [L]
5

```

Com isso terminamos a configuração do projeto Estacionamento, vamos dizer que é a base para que o projeto “rode”.

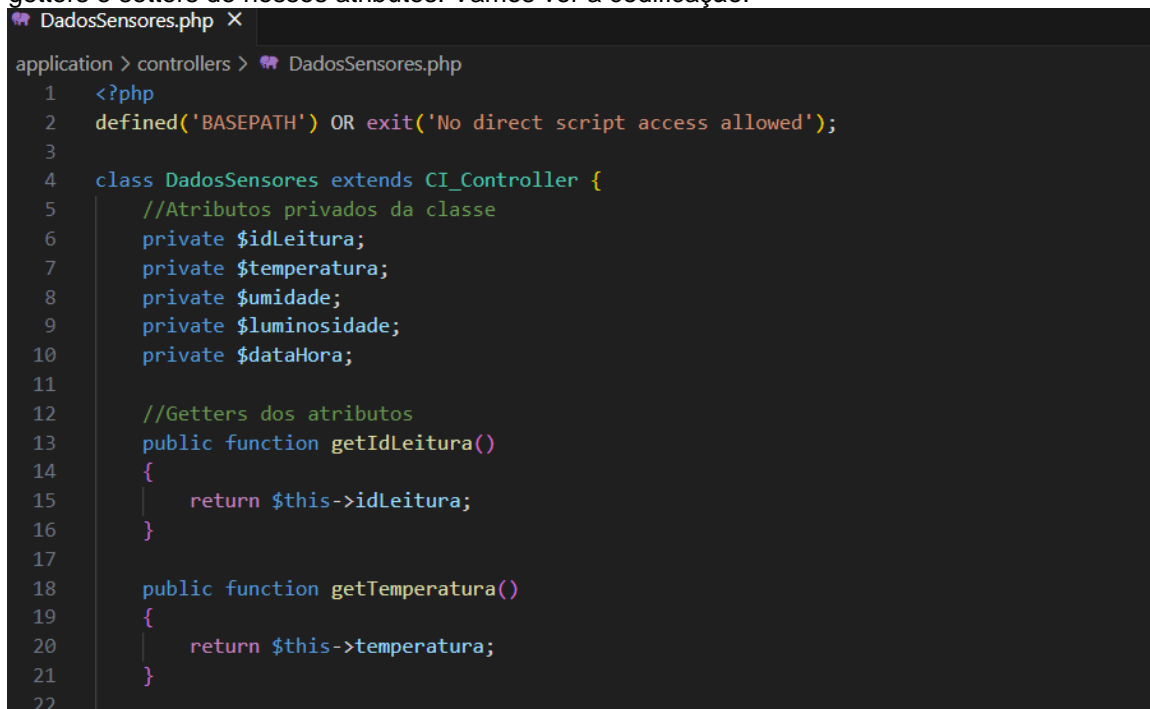
16.7 Montando a *controller* que irá receber os dados do esp32

Para isso vamos criar a classe DadosSensores, que será nosso objeto neste projeto:



Você verá em nosso código o try-catch, que é uma estrutura utilizada para lidar com **exceções** (erros que ocorrem durante a execução de um programa). Ele permite que você **trate erros** de maneira controlada, em vez de deixar que o programa simplesmente termine abruptamente. Assim, você pode executar ações específicas quando uma exceção ocorre, como mostrar mensagens de erro ou tentar uma recuperação.

Vamos trabalhar de forma orientada a objetos, e vamos utilizar o encapsulamento com getters e setters de nossos atributos. Vamos ver a codificação:



```

1 <?php
2 defined('BASEPATH') OR exit('No direct script access allowed');
3
4 class DadosSensores extends CI_Controller {
5     //Atributos privados da classe
6     private $idLeitura;
7     private $temperatura;
8     private $umidade;
9     private $luminosidade;
10    private $dataHora;
11
12    //Getters dos atributos
13    public function getIdLeitura()
14    {
15        return $this->idLeitura;
16    }
17
18    public function getTemperatura()
19    {
20        return $this->temperatura;
21    }
22

```

```
23     public function getUmidade()
24     {
25         return $this->umidade;
26     }
27
28     public function getLuminosidade()
29     {
30         return $this->luminosidade;
31     }
32
33     public function getDataHora()
34     {
35         return $this->dataHora;
36     }
37
38     //Setters dos atributos
39     public function setIdLeitura($idLeituraFront)
40     {
41         $this->idLeitura = $idLeituraFront;
42     }
43
44     public function setTemperatura($temperaturaFront)
45     {
46         $this->temperatura = $temperaturaFront;
47     }
48
49     public function setUmidade($umidadeFront)
50     {
51         $this->umidade = $umidadeFront;
52     }
53
54     public function setLuminosidade($luminosidadeFront)
55     {
56         $this->luminosidade = $luminosidadeFront;
57     }
58
59     public function setDataHora($dataHoraFront)
60     {
61         $this->dataHora = $dataHoraFront;
62     }
63
64     public function inserir(){
65         //Temperatura, umidade e luminosidade
66         //recebidos via POST e colocados em variáveis
67         //Retornos possíveis:
68         //1 - Dados salvo corretamente
69         //2 - Faltou a temperatura
70         //3 - Faltou a umidade
71         //4 - Faltou a luminosidade
72
73         try{
74             if ($_SERVER["REQUEST_METHOD"] == "POST") {
75                 //Fazendo os setters
76                 $this->setTemperatura($_POST['temperatura']);
77                 $this->setUmidade($_POST['umidade']);
78                 $this->setLuminosidade($_POST['luminosidade']);
79
80                 //Faremos uma validação para sabermos
81                 //se todos os dados foram enviados
82                 if ($this->getTemperatura() == '' || $this->getTemperatura() == 0){
83                     $retorno = array('codigo' => 2,
84                                     'msg' => 'Temperatura não informada.');
```

```

86         $retorno = array('codigo' => 3,
87                           'msg' => 'Umidade não informada.');
```

```

88     }elseif ($this->getLuminosidade() == ''){
89         $retorno = array('codigo' => 4,
90                           'msg' => 'Luminosidade não informada.');
```

```

91     }else{
92         //Realizo a instância da Model
93         $this->load->model('M_dadosSensores');
```

```

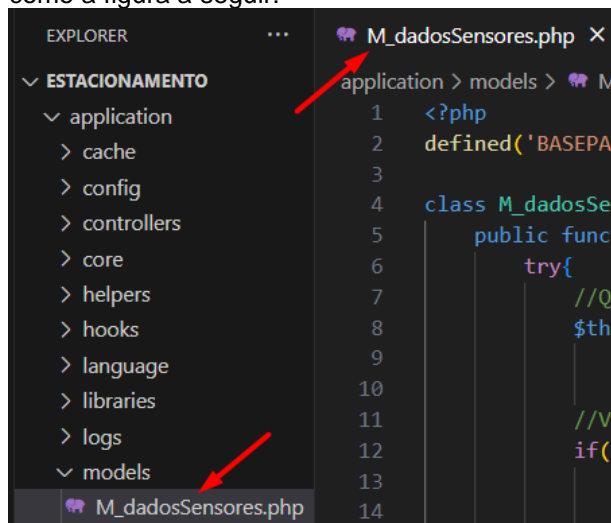
94
95         //Atributo $retorno recebe array
96         //com informações da validação do acesso
97         $retorno = $this->M_dadosSensores->inserir($this->getTemperatura(),
98                                                    $this->getUmidade(),
99                                                    $this->getLuminosidade());
100     }
101
102     }else{
103         $retorno = array('codigo' => 99,
104                           'msg' => 'ATENÇÃO: Requisição inválida.');
```

```

105     }
106 }catch (Exception $e) {
107     $retorno = array('codigo' => 0,
108                       'msg' => 'ATENÇÃO: O seguinte erro aconteceu -> ',
109                       $e->getMessage());
110 }
111
112 //Retorno no formato JSON
113 echo json_encode($retorno);
114 }
115 }
```

16.9 Montando a model

Agora vamos implementar a camada de modelo, ou seja, nossa model, assim sendo, vamos criar um arquivo na pasta models de nossa estrutura, o arquivo M_dadosSensores.php como a figura a seguir:



E vamos codificar da forma abaixo, vamos fazer a camada de modelo com o método inserir que será o responsável pela comunicação da model com o banco de dados de nossa aplicação:


```

M_dadosSensores.php X
application > models > M_dadosSensores.php
1  <?php
2  defined('BASEPATH') or exit('No direct script access allowed');
3
4  class M_dadosSensores extends CI_Model {
5      public function inserir($temperatura, $umidade, $luminosidade){
6          try{
7              //Query de inserção dos dados
8              $this->db->query("insert into sensores (temperatura, umidade, luminosidade)
9                  values ($temperatura, $umidade, '$luminosidade')");
10
11              //Verificar se a inserção ocorreu com sucesso
12              if($this->db->affected_rows() > 0){
13                  $dados = array('codigo' => 1,
14                      'msg' => 'Dados cadastrados corretamente.');

```

Com isso, finalizamos a parte de implementação da parte de inserção, agora vamos ver novos conceitos para testarmos os *endpoints* e verificar se está tudo certo com nosso código.

17 API: conceito, exemplos de uso e importância da integração para desenvolvedores

A integração a APIs pode facilitar demais o trabalho de quem trabalha com desenvolvimento. Entenda melhor o conceito de API e veja exemplos de como elas funcionam.

A sigla API é uma abreviação para *Application Programming Interface*, ou, em português, interface de programação de aplicação.

17.1 API: Conceito

Em uma definição formal, o conceito de API está relacionado a um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por outros aplicativos.

O conceito de API nada mais é do que uma forma de comunicação entre sistemas. Elas permitem a integração entre dois sistemas, em que um deles fornece informações e serviços que podem ser utilizados pelo outro, sem a necessidade de o sistema que consome a API conhecer detalhes de implementação do software.

Metaforicamente, podemos pensar no que é API como um garçom. Quando estamos em um restaurante, buscamos o que desejamos no menu e solicitamos ao garçom. O garçom encaminha esse pedido à cozinha, que prepara o pedido. No fim, o garçom traz o prato pronto até a gente. Não temos detalhes de como esse prato foi preparado, apenas recebemos o que solicitamos.

Com os exemplos de API disponíveis, a ideia é a mesma do garçom. Ela vai receber seu pedido, levar até o sistema responsável pelo tratamento e te devolver o que solicitou (o que pode ser uma informação, ou o resultado do processamento de alguma tarefa, por exemplo).

17.2 A API de integração são grandes aliadas dos devs e das empresas.

Importância das APIs:

Atualmente, quando temos um mundo tão conectado, as APIs são essenciais para a entrega de produtos cada vez mais ricos para os usuários.

Independente do produto que se deseja oferecer, seja ele um site, um aplicativo, um *bot*, é muito importante a utilização do conceito de *APIs* integradas a sistemas para trazer uma gama de funcionalidades para sua aplicação.

Podemos pensar na utilidade do que é *API* por dois pontos de vista: como produtor ou consumidor.

Quando você produz, você está criando *APIs* para que outras aplicações ou sistemas possam se integrar ao seu sistema. Isso não significa que você irá criar uma *API* apenas para expor seu sistema a terceiros. Vai depender do seu propósito, mas a *API* também pode ser apenas para seu uso, como de uma interface sua para os clientes.

Seguindo nessa linha de raciocínio, imagine que você possua um sistema de comércio. Ao criar uma *API* de acesso ao seu sistema, você possibilita a oferta de seus produtos nas interfaces de seu interesse. O que você vai expor de seu sistema na *API* depende do que deseja oferecer como funcionalidade, mas poderia, por exemplo:

- Listar produtos;
- Ofertar promoções;
- Efetivar vendas;
- Realizar cobrança do pedido.

Algumas aplicações do conceito de *API*

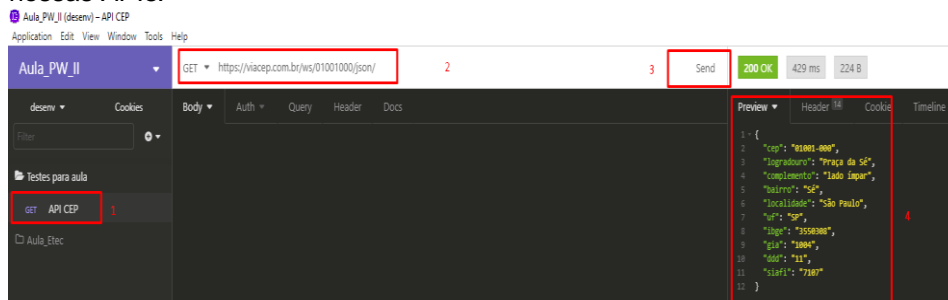
Até aqui, exploramos o que é *API* e as possibilidades de você criar a *API* para facilitar a integração com aplicações.

No entanto, hoje em dia já possuímos uma quantidade enormes de *APIs* que podemos utilizar para enriquecer nossas aplicações, e são de todos os tipos. Vamos citar um exemplo de verificação de CEP.

Em nosso exemplo vamos utilizar o ViaCep que é um webservice que serve para consulta de endereços no Brasil, ele é gratuito, para consultar o cep basta fazermos uma requisição *http* para a *API* do ViaCep, e então obter o retorno com informações como CEP, nome da cidade, código do município, UF e mais.

Para verificar como fazemos essa parte, o fornecedora da *API* pode disponibilizar a documentação da mesma, nesse caso acessando a url: <https://viacep.com.br/> nos é relatado como fazermos a consulta dos dados.

Para esse exemplo e vermos o retorno dos dados, vamos utilizar o *INSOMNIA* (<https://insomnia.rest/>) para testarmos a requisição, que é uma ferramenta popular de *API Client* usada para testar, enviar e depurar requisições HTTP e APIs REST, e que também utilizaremos em nossas aulas, que serve para testarmos através de métodos *POST* nossos *ENDPOINTS* para nossas *APIs*.



Passos:

- 1º - Criamos um nome para nossa requisição;
- 2º - Passamos a URL da API de acordo com a documentação vista no ViaCep.

Validação do CEP

Quando consultado um CEP de formato inválido, por exemplo: "950100100" (9 dígitos), "95010A10" (alfanumérico), "95010 10" (espaço), o código de retorno da consulta será um **400** (Bad Request). Antes de acessar o webservice, valide o formato do CEP e certifique-se que o mesmo possua **(8)** dígitos. Exemplo de como validar o formato do CEP em javascript está disponível nos exemplos abaixo.

Quando consultado um CEP de formato válido, porém inexistente, por exemplo: "99999999", o retorno conterá um valor de "erro" igual a "true". Isso significa que o CEP consultado não foi encontrado na base de dados. Veja como manipular este "erro" em javascript nos exemplos abaixo.

Formatos de Retorno

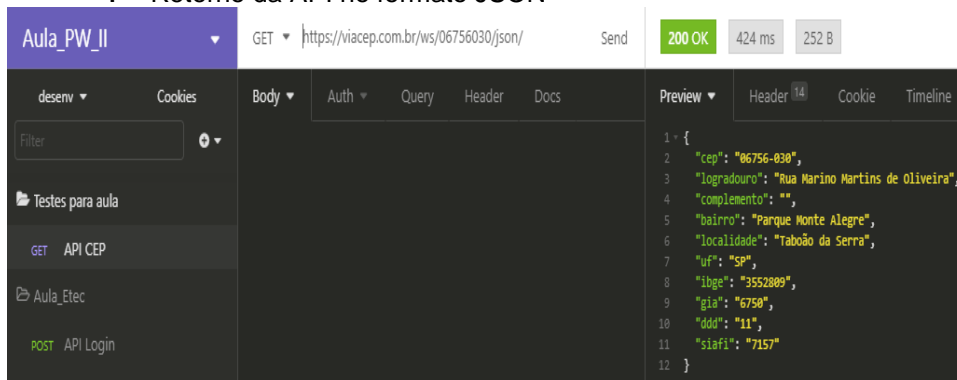
Veja exemplos de acesso ao webservice e os diferentes tipos de retorno:

```
JSON
URL: viacep.com.br/ws/01001000/json/
UNICODE: viacep.com.br/ws/01001000/json/unicode/

{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3550308",
  "gia": "1000",
  "ddd": "11",
  "siafi": "7107"
}
```

3º - Enviamos a requisição;

4º - Retorno da API no formato JSON



17.3 API REST

Representational State Transfer (REST), ou Transferência de Estado Representacional. Lá vem a Ciência da Computação, com suas siglas e nomes complexos.

Vamos simplificar um pouco. *API REST* é oriundo de uma dissertação de doutorado de Roy Fielding, nos anos 2000.

De maneira bem resumida, ele orienta a usar o protocolo HTTP para representar o estado/operação sobre a informação.

Em suma, o que precisamos saber é *API REST* é um serviço sem estado (*cookies* ou *session*) que utiliza corretamente os verbos HTTP:

- **POST**: para criar um objeto no servidor
- **DELETE**: para apagar um objeto do servidor.
- **PUT**: para atualizar um objeto no servidor.
- **GET**: para obter um ou mais objetos do servidor.

Ou seja, uma série de instruções que permitem a criação de um serviço com uma interface bem definida. Para isso, há uma série de recomendações com relação a rotas das *URLs*, dentre outras coisas.

Quando os princípios listados nessa recomendação são adotados em sua totalidade, temos uma *API RESTful*.

18 JSON

Vamos trabalhar em nosso projeto o retorno dos dados pelas *controllers* em formato *JSON*, mas afinal... o que é isso?

JSON é um formato de representação de dados baseado na linguagem de programação *Javascript*, daí o nome *JavaScript Object Notation*. Ou seja, Notação de Objeto em Javascript.

Vamos pensar no exemplo de um objeto pessoa com nome Pedro e altura 1,90. A representação deste objeto em *JSON* ficaria assim:

```
{  
  "nome": "Pedro",  
  "altura": 1.90  
}
```

Este é um exemplo bem simples, mas podemos observar que o JSON não vai muito além disso.

O mais importante que você precisa saber sobre o *JSON* é que ele é composto por chave/valor (ou no inglês *key/value*), as chaves representam os nomes dos atributos da classe e os valores, bem, são os valores do objeto.

No exemplo acima, temos as chaves nome e altura e os valores Pedro e 1.90, respectivamente.

18.1 A sintaxe básica do JSON

A sintaxe de um JSON é bem simples como já falamos, mas agora vamos aprofundar um pouco mais.

Vejamos os elementos básicos do *JSON*.

- { e } - delimita um objeto.
- [e] - delimita um *array*.
- : - separa chaves (atributos) de valores.
- , - separa os atributos chave/valor.

Entendendo estes quatro elementos básicos você já será capaz de ler um *JSON* com mais facilidade.

A leitura simples de um JSON é que ele representa um objeto que tem atributos e cada atributo tem valores.

Os *JSONs* são estruturados em objetos e/ou *arrays* (ou listas). Os objetos são representados por chaves { } e os *arrays* são representados por colchetes [].

Essa é a primeira coisa que você deve olhar no *JSON*: Onde estão as chaves e os colchetes?

Identificando estes dois tipos de estruturas você já sabe se os dados serão acessados por chaves (quando for um objeto), ou por índices/números (quando for um *array*).

Além disso, tem algumas outras regras, nos objetos os atributos devem seguir de um caractere : (dois-pontos) e o valor do atributo. E os atributos devem ser separados por vírgulas.

Já os *arrays* só podem ser de um determinado tipo de dados (veja a próxima sessão), não pode misturar texto com número, por exemplo.

>> O que são Vetores e Matrizes (*arrays*)

Veja abaixo um exemplo de objeto e um exemplo de *array*.

```
{  
  "atributo1": "valor1",  
  "atributo2": 2,  
  "atributo3": true  
}
```

[2,4,5,6]

Outro ponto interessante é que um objeto *JSON* pode ter atributos do tipo *array* e um *array* pode ser do tipo objeto ou *array*. Confuso? Veja o exemplo abaixo para entender melhor.

```
{  
  "atributoDoTipoArray": [1,2,3,54]  
}
```

```
[{  
  "a":1  
},{  
  "b":1  
}]
```

Uma observação é que tanto *array* quanto objeto podem ser vazios em JSON. Assim:

```
{
}
```

18.2 Os tipos de dados do JSON

Além de objeto e *array* serem considerados os tipos de dados principais. O JSON também tem os tipos de dados primitivos que nós já falamos aqui no { Dicas de Programação }.

Os tipos de dados básicos do JSON são:

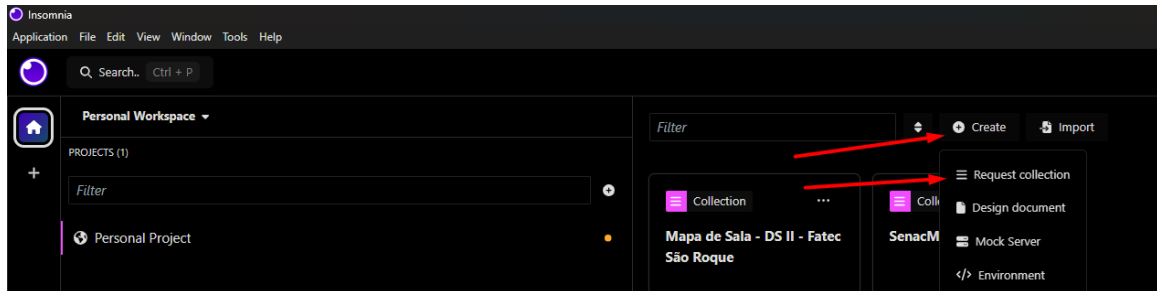
- **String** - separados por aspas (duplas ou simples). Ex. "Brasil" ou 'Brasil'
- **Número** - sem aspas e pode ser inteiro ou real, quando for do tipo real deve-se usar o caractere . (ponto) para separar a parte inteira das casas decimais. Ex. 1 (inteiro) ou 23.454 (real)
- **Booleano** - tipo lógico normal, pode assumir valores *true* ou *false*.
- **Nulo** - este é o valor nulo mesmo. Ex. { "nome" : *null* }

Veja abaixo um exemplo de objeto JSON com todos estes tipos de dados.

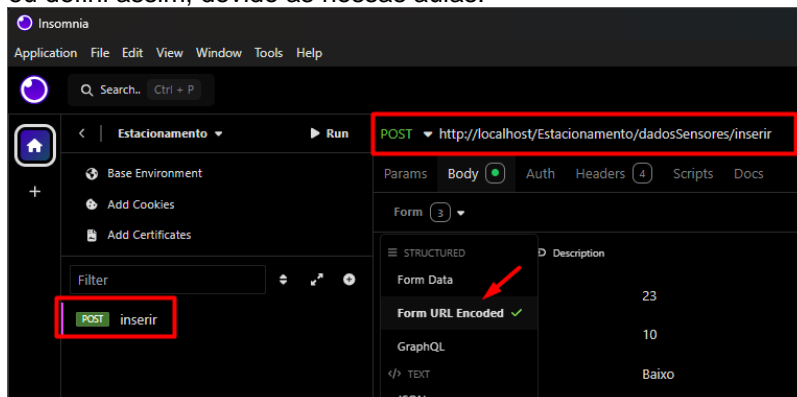
```
{
  "texto" : "Brasil",
  "numero" : 23,
  "numeroReal" : 54.87,
  "booleano": true,
  "nulo": null
}
```

19 Objeto DadosSensores no Insomnia

Vamos configurar nosso ambiente para o projeto de nossas aulas, você pode baixar o Insomnia em <https://insomnia.rest/download>, que é uma ferramenta avançada para desenvolvimento e teste de APIs (Application Programming Interfaces). Ele ajuda desenvolvedores a criar, testar e depurar requisições HTTP e HTTPS, como GET, POST, PUT, DELETE, entre outras. É amplamente utilizado para validar endpoints, simular conexões e garantir que APIs funcionem conforme esperado, recomendo vocês darem uma olhada materiais a respeito, aqui na aula farei de forma direta, vamos criar uma coleção de requisições, conforme imagem abaixo:



Após isso, será mostrada esta tela para nós, o nome da coleção pode ser qualquer um, eu defini assim, devido as nossas aulas:



POST ▼ http://localhost/Estacionamento/dadosSensores/inserir Send

Params Body ● Auth Headers (4) Scripts Docs

Form (3)

+ Add 🗑 Delete all 📄 Description

temperatura	23	▼ ✓ 🗑
umidade	10	▼ ✓ 🗑
luminosidade	Baixo	▼ ✓ 🗑

E finalizando testamos o endpoint:

POST ▼ http://localhost/Estacionamento/dadosSensores/inserir Send 200 OK 58 ms 53 B 33 Minutes Ago

Params Body ● Auth Headers (4) Scripts Docs

Form (3)

+ Add 🗑 Delete all 📄 Description

temperatura	23	▼ ✓ 🗑
umidade	10	▼ ✓ 🗑
luminosidade	Baixo	▼ ✓ 🗑

Preview Headers (6) Cookies Tests 0/0 → Mock Console

```

1 - {
2   "codigo": 1,
3   "msg": "Dados cadastrados corretamente."
4 }
  
```