



UC13: Executar os processos de codificação, manutenção e documentação de aplicativos computacionais para internet

CARGA HORÁRIA: 96 HORAS

Indicadores



1. Configura o ambiente de desenvolvimento conforme as funcionalidades e características do aplicativo computacional para WEB.
2. Desenvolve softwares de acordo com as melhores práticas da linguagem de programação selecionada.
3. Elabora código conforme as funcionalidades e características do aplicativo computacional para WEB.
4. Realiza a compilação e depuração do código de acordo com orientações técnicas da IDE utilizada.
5. Utiliza comandos de integração dos objetos de bancos de dados com o código construído para WEB de acordo com premissas do sistema operacional (servidor) de rede.
6. Elabora o manual do projeto de software desenvolvido conforme orientação técnica.

Elementos da competência



CONHECIMENTOS

- Ferramentas de desenvolvimento de programas para internet. Ferramentas de desenvolvimento colaborativo. Ferramentas de modelagem de software. Linguagens de programação. Ambientes de programação (IDE).
- Linguagem de programação orientada a objetos – Visão geral da linguagem de programação. Palavras reservadas. Application Program Interface (API). Plataforma de desenvolvimento: internet. Tipos de dados. Variáveis e constantes. Coleções: lista, conjunto e mapa. Operadores. Comandos condicionais. Comandos de repetição. Objetos, classes, interfaces, atributos, modificadores de acesso, métodos e propriedades. Herança, polimorfismo, encapsulamento e agregação. Tratamento de erros e exceções. Distribuição do aplicativo. Defeitos e falhas. Documentação de programas de computador.
- Controle de versão de software web – Segurança da informação. Instalação e configuração.

HABILIDADES

- Resolver problemas lógicos e aritméticos.
 - Representar expressões lógicas e matemáticas.
 - Interpretar textos técnicos.
 - Comunicar-se de maneira assertiva.
 - Mediar conflitos nas situações de trabalho.
 - Selecionar informações necessárias ao desenvolvimento do seu trabalho.
 - Analisar as etapas do processo de trabalho.
-

ATITUDES/VALORES

- Zelo na apresentação pessoal e postura profissional.
 - Sigilo no tratamento de dados e informações.
 - Zelo pela segurança e pela integridade dos dados.
 - Proatividade na resolução de problemas.
 - Colaboração no desenvolvimento do trabalho em equipe.
 - Cordialidade no trato com as pessoas.
 - Responsabilidade no uso dos recursos organizacionais e no descarte de lixo eletrônico.
-

1. 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.

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 e 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.

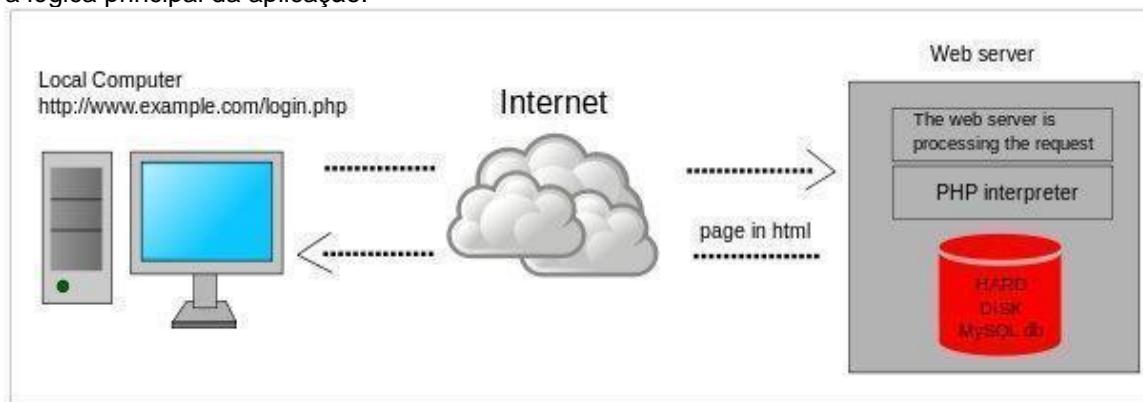
E como funciona uma página dinâmica?

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.

2. 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.

3. 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.

4. 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, verificar, 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.

5. 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.

5.1 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.

5.2 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.

5.3 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.

5.4 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.

6. 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.

7. 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), Wampserver, 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.

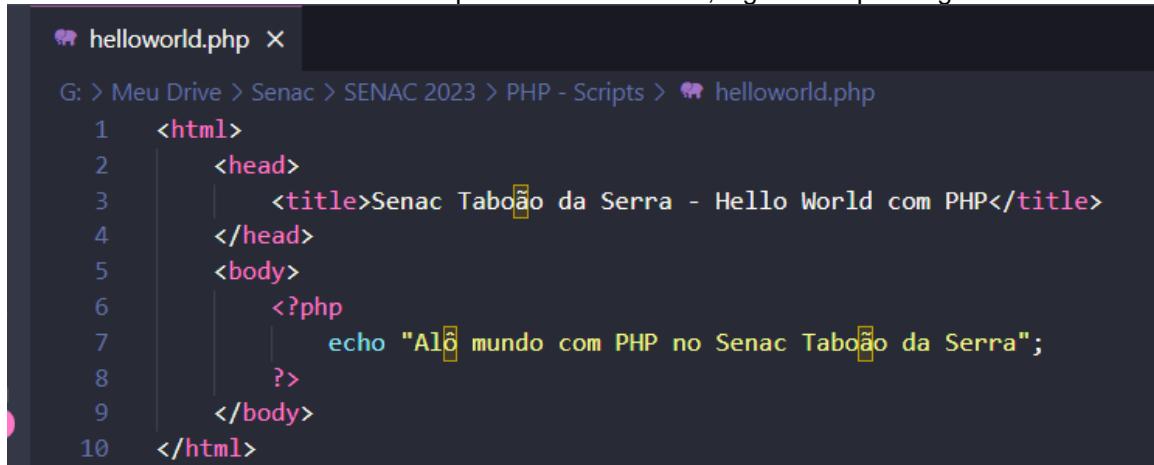
Para a instalação do PHP, os pacotes essenciais são os seguintes:

- 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.

8. Iniciando os trabalhos com o PHP

Vamos criar um pequeno script em PHP para testarmos e verificarmos se o PHP e o servidor Web foram instalados e configurados corretamente no seu computador.

Utilizando o um dos editores que citamos no item 6, digite o script a seguir.

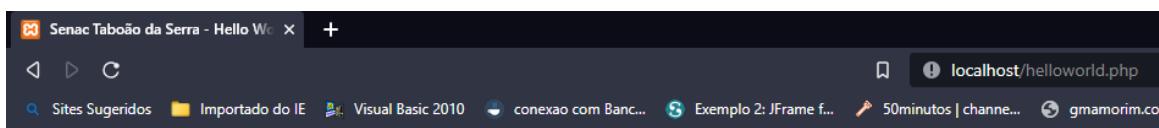


```
helloworld.php <input type="button" value="X">
G: > Meu Drive > Senac > SENAC 2023 > PHP - Scripts > helloworld.php
1   <html>
2     <head>
3       <title>Senac Taboão da Serra - Hello World com PHP</title>
4     </head>
5     <body>
6       <?php
7         echo "Alô mundo com PHP no Senac Taboão da Serra";
8       ?>
9     </body>
10    </html>
```

Salve o arquivo como “helloworld.php” na pasta C:\XAMPP\htdocs\aula1, esse é o endereço default que o servidor no caso o XAMPP busca os scripts PHP, a pasta **aula1**(você deve criar) é para uma questão de organização dos códigos que iremos trabalhar em nossas aulas, mas fiquem a vontade para modificar se for o caso.

Agora, abra o navegador e, na barra de endereços, digite o seguinte endereço:
<http://localhost/aula1/helloworld.php>.

Observe que o seu browser exibiu a frase “Alô mundo com PHP para não dar azar!!!”. Sinal de que o PHP e o servidor Web foram instalados e configurados corretamente no seu computador.



Alô mundo com PHP no Senac Taboão da Serra

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 caracter \$ (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

Strings (Atenção)

Strings podem ser atribuídas de duas maneiras:

- a) utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \'- ver exemplo abaixo)
- b) utilizando aspas duplas (") – Desta maneira, qualquer variável ou caractere de escape será expandido antes de ser atribuído.

Exemplo:

```
1 <?php
2     $teste = "Marcos";
3     $vivas = '---$teste--\n';
4     $texto = "2º DSN";
5     echo "$vivas";
6 ?>
```

A saída desse script será "---\$teste--\n".

```
1 <?php
2     $texto = "2º DSN"; Nova linha
3     $teste = "Marcos";
4     $vivas = "---$teste--\n";
5     echo "$vivas";
6 ?>
```

A saída desse script será "---Marcos--" (com uma quebra de linha no final).

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 ?>
```

Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas.

Através de listas é possível atribuir valores que estão num array para variáveis. Vejamos o exemplo:

```
1 <?php
2     list($a, $b, $c) = array("a", "b", "c");
3 ?>
```

O comando acima atribui valores às três variáveis simultaneamente. É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos. No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero.

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

- and “e” lógico
- or “ou” lógico
- xor ou exclusivo
- ! não (inversão)
- && “e” lógico
- || “ou” lógico

Existem dois operadores para “e” e para “ou” porque eles têm diferentes posições na ordem de precedência.

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

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, utilize 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 09/10/2024, quarta-feira.

12.3.1. Correção dos exercícios

Abaixo temos uma versão de correção para os exercícios realizados, lembrando que esta é uma forma de fazer os exercícios, correto?

- Exercício 1

```
<?php
    <html>
        <head>
            <meta charset="UTF-8">
            <title>Exercício 1</title>
        </head>
        <body>
            <h2>Exercício número 1</h2>
            <form method="post" action = "exercicio1.php">
                <label for="numero">Informe um número:</label>
                <input type="number" id="nro" name="nro" required><br><br>
                <input type="submit" value="Realizar verificação">
            </form>
        </body>
    </html>
```

Parte html

```
exercicio1.php
<?php
    $numero = $_POST['nro'];

    if ($numero > 0){
        $resultado = "Número positivo";
    }elseif ($numero == 0){
        $resultado = "Número igual a zero";
    }else{
        $resultado = "Número negativo";
    }
    echo $resultado;
    echo '<br> <a href= "exe1.html">Retornar' ;
?>
```

Parte PHP.

- Exercício 2

```
<?php
    <html>
        <head>
            <meta charset="UTF-8">
            <title>Exercício 2</title>
        </head>
        <body>
            <h2>Exercício número 2</h2>
            <form method="post" action = "exercicio2.php">
                <label for="pnumero">Informe o primeiro número:</label>
                <input type="number" id="pnro" name="pnro" required><br>
                <label for="snumero">Informe o segundo número:</label>
                <input type="number" id="snro" name="snro" required><br><br>
                <input type="submit" value="Realizar verificação">
            </form>
        </body>
    </html>
```

Parte html

```

    exercicio2.php X
    exercicio2.php
1   <?php
2   $primeiroNumero = $_POST['pnro'];
3   $segundoNumero = $_POST['snro'];
4
5   if ($primeiroNumero > $segundoNumero){
6       $resultado = $segundoNumero . " e " . $primeiroNumero;
7   }elseif ($primeiroNumero == $segundoNumero){
8       $resultado = "Os números são iguais -> " . $primeiroNumero;
9   }else{
10      $resultado = $primeiroNumero . " e " . $segundoNumero;
11  }
12 echo $resultado;
13 echo '<br> <a href= "exe2.html">Retornar';
14
15 ?>
16

```

Parte PHP

- Exercício 3

```

    exe3.html X
    exe3.html > ...
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <meta charset="UTF-8">
5       <title>Exercício 3</title>
6   </head>
7   <body>
8       <h2>Exercício número 3</h2>
9       <form method="post" action = "exercicio3.php">
10      <label for="pnnumero">Informe a primeira nota:</label>
11      <input type="number" id="pnnota" name="pnnota" step="0.010" required><br>
12      <label for="pnumero">Informe a segunda nota:</label>
13      <input type="number" id="snota" name="snota" step="0.010" required><br>
14      <label for="snumero">Informe a terceira nota:</label>
15      <input type="number" id="tnota" name="tnota" step="0.010" required><br><br>
16      <input type="submit" value="Realizar verificação">
17  </form>
18 </body>
19 </html>

```

Parte html

```

    exercicio3.php X
    exercicio3.php
1   <?php
2   $arrayNotas = array($_POST['pnnota'], $_POST['snota'], $_POST['tnota']);
3   //round -> faz o arredondamento da nota
4   //array_sum -> Soma de todos os elementos do Array
5   //$soma = $_POST['pnnota'] + $_POST['snota'] + $_POST['tnota'];
6   $media = round(array_sum($arrayNotas) / count($arrayNotas), 2);
7
8   if ($media >= 6){
9       $resultado = "Aluno aprovado, sua média foi -> ";
10  }else{
11      $resultado = "Aluno reprovado, sua média foi -> ";
12  }
13  echo $resultado . $media;
14  echo '<br> <a href= "exe3.html">Retornar';
15 ?>

```

Parte PHP

- Exercício 4

```

    < exe4.html >
    < exe4.html > ...
2   <html>
3     <head>
4       |   <title>Exercício 4</title>
5     </head>
6     <body>
7       |   <h2>Exercício número 4</h2>
8       |   <form method="post" action = "exercicio4.php">
9         |   <label for="pnumero">Informe o número do mês desejado:</label>
10        |   <input type="number" id="mes" name="mes" required><br><br>
11        |   <input type="submit" value="Realizar verificação">
12      </form>
13    </body>
14  </html>
15

```

Parte html

```

    exercicio4.php
    exercicio4.php
1  <?php
2   $arrayMeses = array(1 => "Janeiro", 2 => "Fevereiro", 3 => "Março", 4 => "Abril",
3   |   |   |   |   5 => "Maio", 6 => "Junho", 7 => "Julho", 8 => "Agosto",
4   |   |   |   |   9 => "Setembro", 10 => "Outubro", 11 => "Novembro", 12 => "Dezembro");
5
6   $mesInformado = $_POST['mes'];
7
8   if ($mesInformado < 1 || $mesInformado > 12 ){
9     $resultado = "O Mês informado é inválido!";
10  }else{
11    $resultado = "O mês informado é " . $arrayMeses[$mesInformado];
12  }
13 echo $resultado;
14 echo '<br> <a href= "exe4.html">Retornar';
15 ?>

```

Parte PHP

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)  
?>
```

15.5.3. 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 inicio 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>";
    }
?>
```

15.5.3. 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 é: ". $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 1000, reajuste de 50%
 - salários maiores que 1000, 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 1 e o número total, exemplo:

```
* string = "Gil Eduardo de Andrade"
* total_caracteres = 22
Imprime: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
```

Pessoal, criem os scripts .php e veremos em aula, fazer até 17/10/2023, irei corrigir em aula, de forma individual.

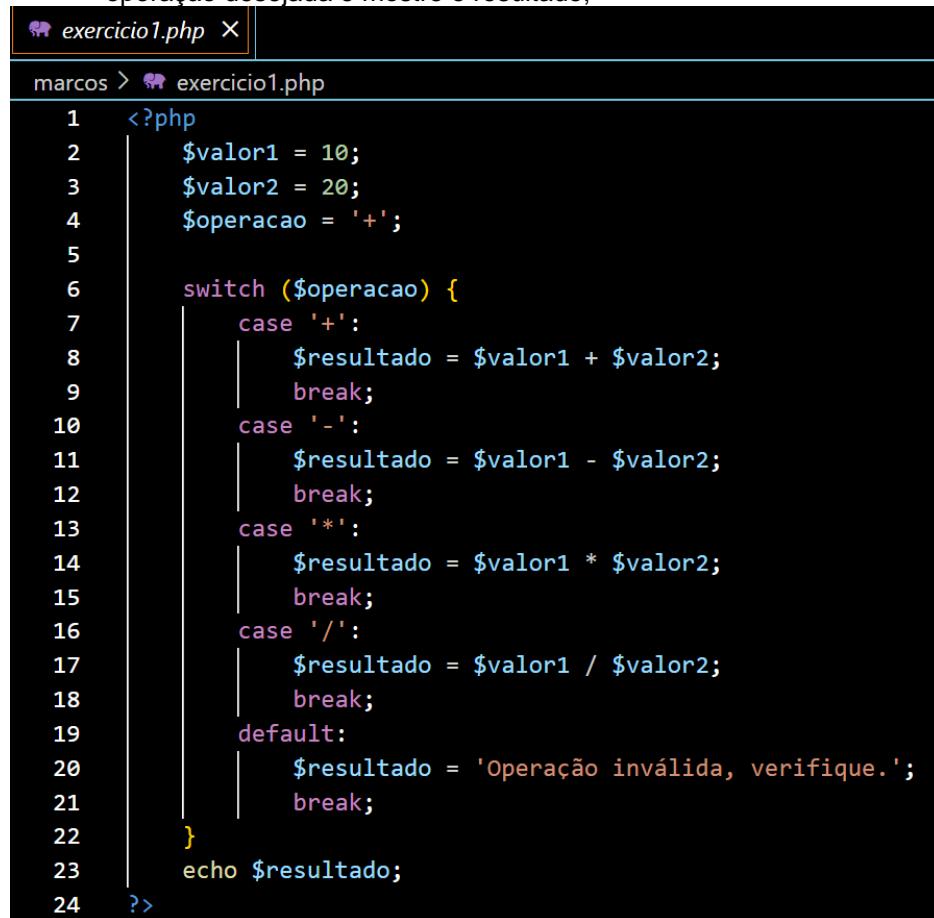
Com isso, finalizamos essa parte introdutória da linguagem PHP.

12.6.1 – Correção dos Exercícios de fixação:

Lembrando que temos várias formas de resolver os exercícios, abaixo temos uma delas, como falamos em aula, correto?

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;



```
exercicio1.php X
marcos > exercicio1.php

1  <?php
2      $valor1 = 10;
3      $valor2 = 20;
4      $operacao = '+';
5
6      switch ($operacao) {
7          case '+':
8              $resultado = $valor1 + $valor2;
9              break;
10         case '-':
11             $resultado = $valor1 - $valor2;
12             break;
13         case '*':
14             $resultado = $valor1 * $valor2;
15             break;
16         case '/':
17             $resultado = $valor1 / $valor2;
18             break;
19         default:
20             $resultado = 'Operação inválida, verifique.';
21             break;
22     }
23     echo $resultado;
24 ?>
```

- 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);

```

exercicio2.php X

marcos > exercicio2.php
1  <?php
2      $valorTotalCompra = 1000;
3      $tipoCliente = 6;
4
5      switch ($tipoCliente){
6          case 1:
7              $resultado = $valorTotalCompra;
8              break;
9          case 2:
10             $resultado = $valorTotalCompra - ($valorTotalCompra * 0.1);
11             break;
12         case 3:
13             $resultado = $valorTotalCompra - ($valorTotalCompra * 0.05);
14             break;
15         default:
16             $resultado = "Tipo de cliente inválido, besta!";
17     }
18
19     if ($tipoCliente > 3 || $tipoCliente < 1){
20         echo $resultado;
21     }else{
22         echo 'Valor da compra foi de: ' . $resultado;
23     }
24 ?>

```

- Faça um algoritmo PHP que calcule e imprima o salário reajustado de um funcionário de acordo com a seguinte regra:
 - salários até 1000, reajuste de 50%
 - salários maiores que 1000, reajuste de 30%.

```

exercicio3.php X

marcos > exercicio3.php
1  <?php
2      /*
3          * Faça um algoritmo que calcule e imprima o salário reajustado
4          de um funcionário de acordo com a seguinte regra:
5          *     • salários até 300, reajuste de 50%
6          *     • salários maiores que 300, reajuste de 30%.
7      */
8
9      $salario = 200;
10
11     if ($salario <= 1000){
12         $parametro = 1;
13     }else{
14         $parametro = 2;
15     }
16
17     switch($parametro){
18         case 1:
19             $novoSalario = $salario * 1.50;
20             break;
21         case 2:
22             $novoSalario = $salario * 1.30;
23             break;
24     }
25
26     echo 'O antigo salário era de ' . $salario .
27     | ' reais agora o novo será de ' . $novoSalario . ' reais.';
28 ?>

```

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

```

exercicio4.php x
marcos > exercicio4.php

1  <?php
2   /*
3    * Faça um algoritmo em que receba um valor qualquer e imprima os
4    * valores de 0 até o valor recebido, exemplo:
5    * • Valor recebido = 9
6    * • Impressão do programa - 0 1 2 3 4 5 6 7 8 9
7    */
8    $valor = 5;
9    $concatena = '';
10
11   for ($i = 0; $i <= $valor; $i++){
12       echo $i . '<br>'; //resultado na vertical
13       //concatenando para mostrar o resultado na horizontal
14       $concatena = $concatena . $i . ($i < $valor ? ',' : '.');
15   }
16
17   echo $concatena;
18 ?>

```

- Faça um algoritmo PHP que receba um valor qualquer e calcule o seu fatorial (!), sabendo que fatorial de um número é: $7! = 7*6*5*4*3*2*1$ $4! = 4*3*2*1$

```

exercicio5.php x
marcos > exercicio5.php

1  <?php
2   /*
3    * Faça um algoritmo que receba um valor qualquer e calcule
4    * o seu fatorial (!), sabendo que fatorial de um número é:
5    *  $7! = 7*6*5*4*3*2*1$   $4! = 4*3*2*1$ .
6    */
7
8    $valor = 7;
9    $referencia = $valor - 1; // 7- 1 = 6
10   $resultado = $valor;
11   $concatena = $valor;
12
13   do{
14       $resultado *= $referencia;
15       $concatena = $concatena . "*" . $referencia;
16       $referencia--;
17   }while($referencia >= 1);
18
19   echo 'O fatorial de ' . $valor . ' ! = ' . $concatena . ' -> ' . $resultado;
20 ?>

```

- 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

```

exercicio6.php x
marcos > exercicio6.php
1  <?php
2  /*
3   * Faça um algoritmo que receba dois valores quaisquer e
4   * imprime todos os valores intermediários a ele, veja exemplo:
5   * Primeiro Valor = 5 Segundo Valor = 15 Imprime: 6 7 8 9 10 11 12 13 14.
6   */
7
8  $primeiroValor = 6;
9  $segundoValor = 41;
10 $concatena = '';
11 $valorReferencia = $primeiroValor;
12
13
14  for ($i = 1; $i < ($segundoValor - $primeiroValor); $i++){
15      $valorReferencia = $primeiroValor + $i;
16      $concatena = $concatena . $valorReferencia . ' '; //concatenando para mostrar o resultado na horizontal
17  }
18
19  echo 'Primeiro valor = ' . $primeiroValor . ' Segundo Valor = ' . $segundoValor .
20  | ' Resultado -> ' . $concatena;
21 ?>

```

- **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 1 e o número total, exemplo:

```

* string = "Gil Eduardo de Andrade"
* total_caracteres = 22
Imprime: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

```

```

exercicio7.php x
marcos > exercicio7.php
1  <?php
2  /*
3   * Desafio, pesquise. Faça um algoritmo PHP que receba
4   * uma string, encontre o número total de caracteres
5   * desta e imprima todos os números que existem entre
6   * 0 e o número total, exemplo:
7   * * string = "Gil Eduardo de Andrade"
8   * * total_caracter = 22
9   * Imprime: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
10 */
11
12 $conteudo = "Marcão do Senac";
13 $qtdCaracteres = strlen($conteudo);
14 $concatena = '';
15
16 for ($i = 1; $i <= $qtdCaracteres; $i++){
17     echo $i . '<br>'; //resultado na vertical
18     $concatena = $concatena . $i . ' '; //concatenando para mostrar o resultado na horizontal
19 }
20
21 echo 'String escolhida -> ' . $conteudo . ' ; Total de caracteres -> '
22 | | . $qtdCaracteres . ' elementos -> ' . $concatena;
23 ?>

```

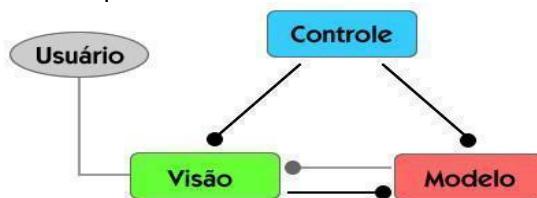
13. Padrão MVC

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.



13.1 Explicação do conceito do MVC

13.1.1 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.

13.1.2 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.

13.1.3 Então podemos concluir resumidamente

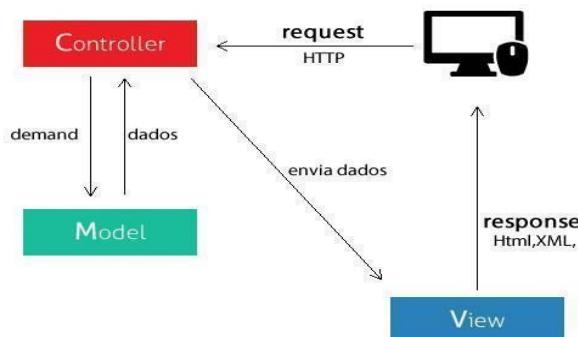
MVC é nada mais que um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário (*view*), a camada de manipulação dos dados(*model*) e a camada de controle(*controller*).

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)



13.1.4 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...

14. 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.

14.1 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).

14.2 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*

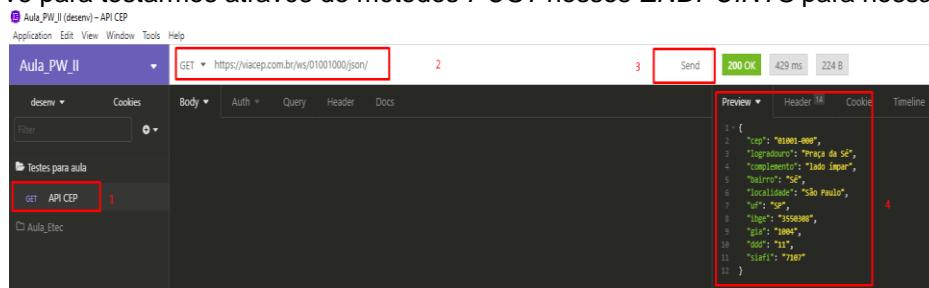
Até aqui, exploramos o que é *API* e as possibilidades de você criar a *API* para facilitar a integração com aplicações.

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, mas 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: "95010000" (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:

```

{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "35500004",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7007"
}

```

3º - Enviamos a requisição;

4º - Retorno da API no formato JSON

```

{
  "cep": "06756-030",
  "logradouro": "Rua Marino Martins de Oliveira",
  "complemento": "",
  "bairro": "Parque Monte Alegre",
  "localidade": "Taboão da Serra",
  "uf": "SP",
  "ibge": "35528009",
  "gia": "6750",
  "ddd": "11",
  "siafi": "7157"
}

```

14.3 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*.

15. JSON

Vamos trabalhar em nosso projeto a passagem e 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.

15.1. 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:

```
{  
}  
[]
```

15.2. 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  
}
```

16. 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).

16.1 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:

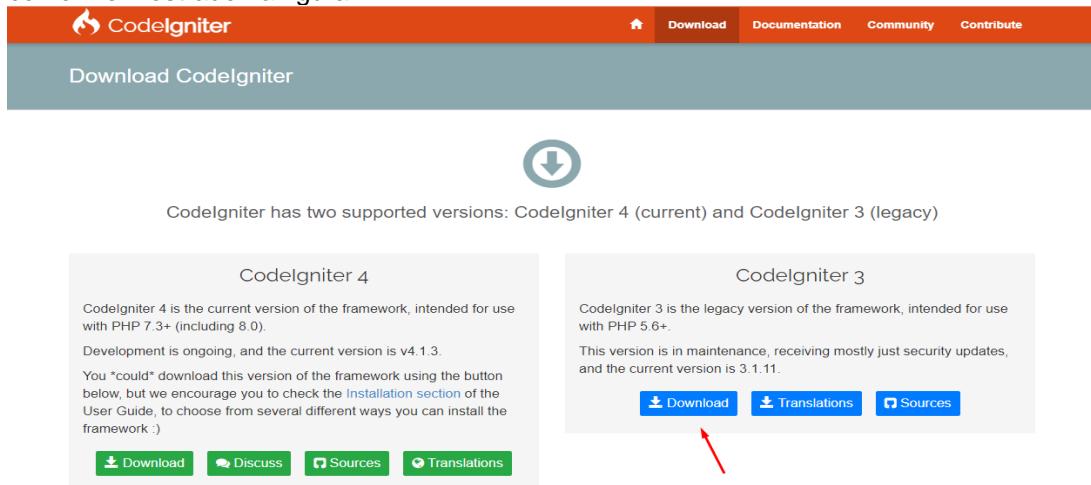
- MySQL(5.1+), mysqli e PDO drivers
- Oracle (drivers oci8 e PDO)
- PostgreSQL (postgre a PDO)
- MSSQL (mssql, sqlsrv e PDO)
- Interbase/Firebird (ibase e PDO)
- ODBC (odbc e PDO)

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.

16.2. 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:



The screenshot shows the official CodeIgniter website at <https://codeigniter.com>. At the top, there's a navigation bar with links for Home, Download, Documentation, Community, and Contribute. Below the navigation, a large button says "Download CodeIgniter". Underneath this button, there's a large blue circular icon with a white downward arrow. Below the icon, text reads "CodeIgniter has two supported versions: CodeIgniter 4 (current) and CodeIgniter 3 (legacy)".

On the left, a box for "CodeIgniter 4" states: "CodeIgniter 4 is the current version of the framework, intended for use with PHP 7.3+ (including 8.0). Development is ongoing, and the current version is v4.1.3. You *could* download this version of the framework using the button below, but we encourage you to check the [Installation section](#) of the User Guide, to choose from several different ways you can install the framework :)".

Below this box are four buttons: "Download" (green), "Discuss" (grey), "Sources" (grey), and "Translations" (grey).

On the right, a box for "CodeIgniter 3" states: "CodeIgniter 3 is the legacy version of the framework, intended for use with PHP 5.6+. This version is in maintenance, receiving mostly just security updates, and the current version is 3.1.11." Below this box are three buttons: "Download" (blue), "Translations" (blue), and "Sources" (blue).

A red arrow points to the "Download" button for CodeIgniter 3.

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:

The screenshot shows the GitHub repository page for `bcit-ci/Codeigniter`. At the top, there are tabs for Code, Issues (48), Pull requests (16), Wiki, and Pulse. Below that, it says "Open Source PHP Framework (originally from EllisLab) <http://codeigniter.com/>". The main area shows a list of commits. A green arrow points to the "Download ZIP" button at the top right of the commit list.

Commit	Message	Date
application	[ci skip] Polish changes from PR #42460	10 days ago
system	Merge pull request #4225 from zhenghongyi/loader-test	8 days ago
tests	Merge pull request #4225 from zhenghongyi/loader-test	15 days ago
user_guide_src	Merge pull request #4226 from galidolopatch-11	8 days ago
gitattribute	Renaming folders to directories []	7 months ago
.gitignore	Should ignore PHPStorm and Sublime Text project files.	7 months ago
.travis.yml	Enable Travis builds for 3.0-stable branch	3 months ago
DCO.txt	Adding contribution guidelines to user guide, including new Developer...	3 years ago
composer.json	[ci skip] Remove double description from composer.json	7 months ago
contributing.md	[ci skip] Make it clear that PHP <5.5 usage is discouraged	13 days ago

16.3. 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.

17. Mão à obra

Em nossas aulas iremos criar um sistema, levando em consideração a parte de Orientação a Objetos que fizemos na primeira parte do semestre e mais algumas técnicas novas.

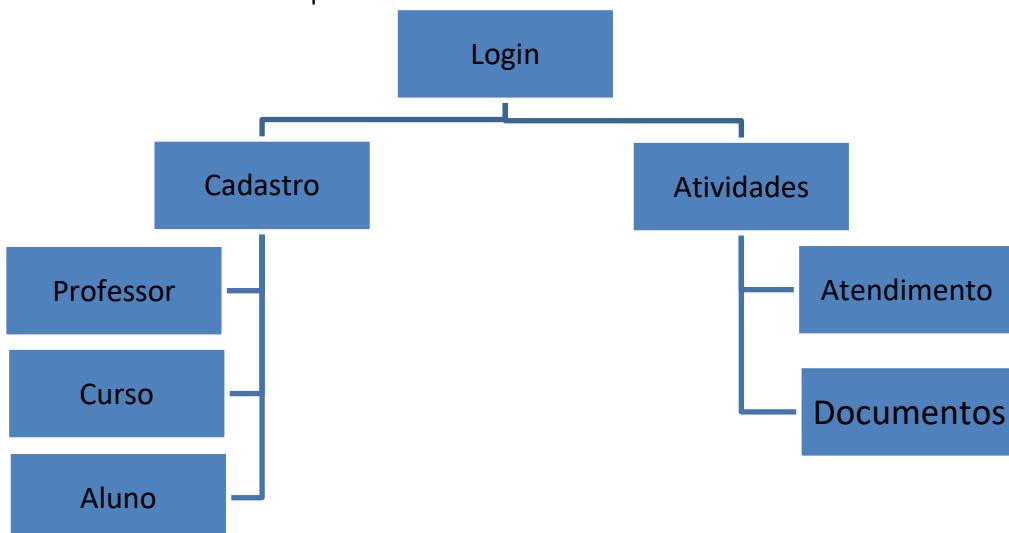
Para isso vamos levar em consideração um sistema que precisa ser feito que é relatado a seguir:

17.2. Cenário:

Em uma determinada faculdade, existe um professor que representa uma disciplina de Estágio, ele precisa administrar os alunos que requerem informações sobre estágio, seria uma gestão do atendimento ao discente, e mais, ele administra toda a documentação entregue por esse aluno, já em situação de desespero, esse professor pede auxílio ao time do curso de TI do Senac Taboão da Serra, para confecção de uma solução, onde possa armazenar todos esses atendimentos e gestão documental.

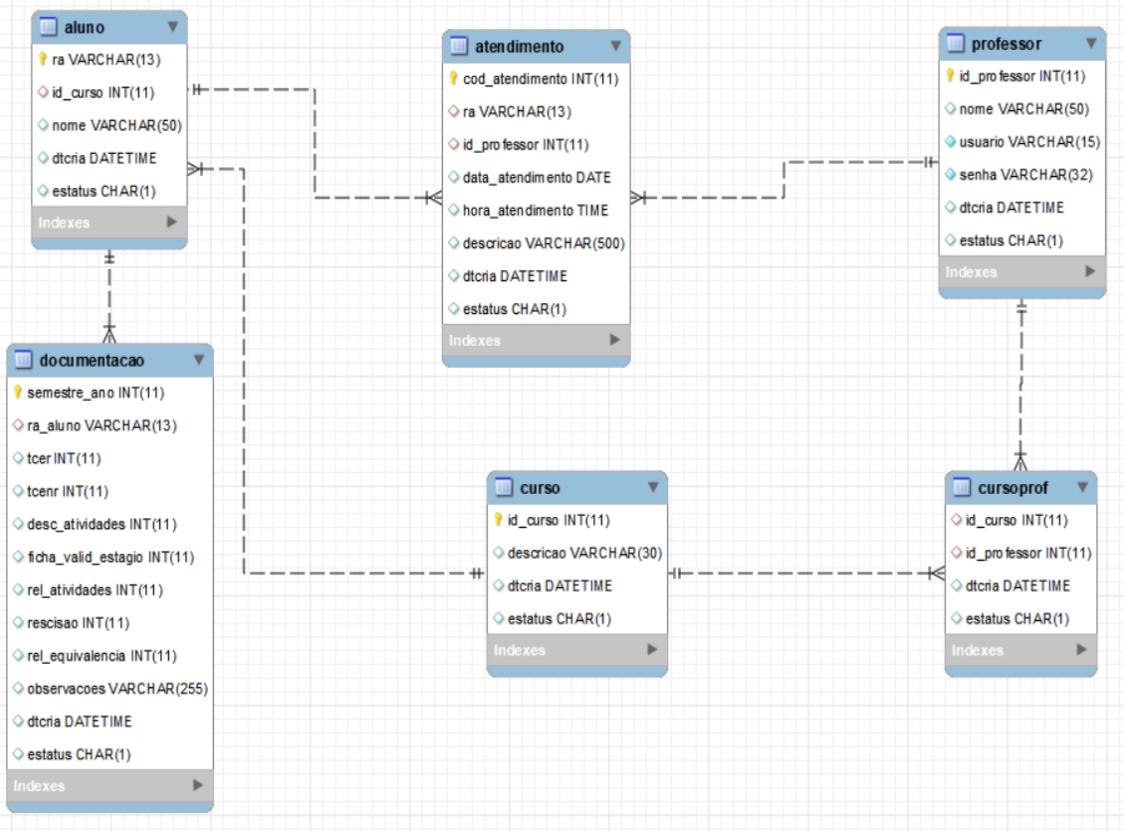
17.3. Decomposição funcional

Nosso sistema deve seguir a seguinte decomposição funcional, que foi realizada depois de um levantamento de requisitos.



17.4. Banco de Dados

Para essa atividade foi idealizada o seguinte Diagrama de Entidade e Relacionamento:



Que refletiu da seguinte codificação:

```

    DDL BD x
    /* Criação do banco de dados */
    create database bd_estagio;

    /* Selecionar o banco de dados */
    use bd_estagio;
    
```

```
11  /*
12   Criar a tabela de curso
13  */
14 • ⊖ create table curso(
15     id_curso int auto_increment primary key,
16     descricao varchar(30) default '',
17     dtcria    datetime default now(),
18     estatus   char(01) default ''
19 );
20
21  /*
22   Criar a tabela de aluno
23  */
24 • ⊖ create table aluno(
25     ra        varchar(13) primary key,
26     id_curso int,
27     nome      varchar(50) default '',
28     dtcria    datetime default now(),
29     estatus   char(01) default '',
30
31     foreign key (id_curso) references curso(id_curso)
32 );
33
34  /*
35   Criar a tabela de professor
36  */
37 • ⊖ create table professor(
38     id_professor int primary key,
39     nome        varchar(50),
40     usuario     varchar(15) not null,
41     senha       varchar(32) not null,
42     dtcria    datetime default now(),
43     estatus   char(01) default ''
44 );
```

```
45
46  /*
47   Criar a tabela de relação curso x professor
48  */
49 •◦ create table cursoprof(
50     id_curso      int,
51     id_professor int,
52     dtcria        datetime default now(),
53     estatus       char(01) default '',
54
55     foreign key(id_curso) references curso(id_curso),
56     foreign key(id_professor) references professor(id_professor)
57 );
58
59  /*
60   Criação da tabela de documentação
61  */
62 •◦ create table documentacao(
63     semestre_ano      int primary key,
64     ra_aluno          varchar(13) default '',
65     tcer               int default 0,
66     tcenr              int default 0,
67     desc_atividades    int default 0,
68     ficha_valid_estagio int default 0,
69     rel_atividades     int default 0,
70     rescisao           int default 0,
71     rel_equivalecia    int default 0,
72     observacoes        varchar(255) default '',
73     dtcria             datetime default now(),
74     estatus            char(01) default '',
75
76     foreign key(ra_aluno) references aluno(ra)
77 );
78
79  /*
80   Criação da tabela de atendimento
81  */
82 •◦ create table atendimento(
83     cod_atendimento   int primary key,
84     ra                 varchar(13) default '',
85     id_professor       int default 0,
```

```
86     data_atendimento date,  
87     hora_atendimento time,  
88     descricao      varchar(500),  
89     dtcria        datetime default now(),  
90     estatus       char(01) default '',  
91  
92     foreign key(ra) references aluno(ra),  
93     foreign key(id_professor) references professor(id_professor)  
94 );
```

17.5. Vamos codificar

Nesse momento iremos começar a codificar em PHP com o *Framework CodeIgniter*, para isso vamos colocar nosso projeto na pasta base do htdocs do XAMP, irei fornecer a base do projeto que estará com todos os arquivos necessários para a base de nosso projeto, inclusive o .htaccess.

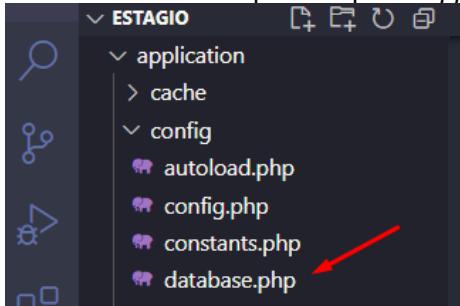
17.5.1. Configurando o *CodeIgniter* para nosso projeto

Nesse momento teremos que configurar nosso *framework* para a estrutura de nosso projeto, iremos configurar nesse alterando os seguintes arquivos em nossa estrutura:

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

17.5.1.1. Configurando o arquivo *databases.php*

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



Informaremos os dados de configuração, da seguinte forma:

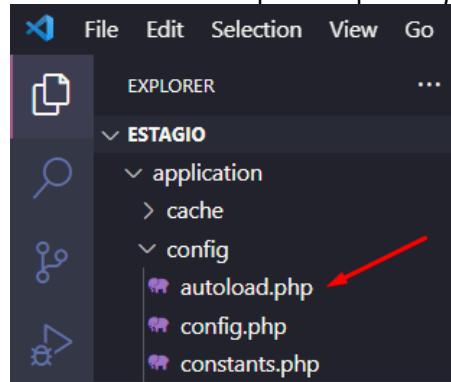
```

    application > config > database.php
    75
    76 //Conexão default, seria o banco de produção de nosso projeto
    77 $db['default'] = array(
    78     'dsn'      => 'Estagio', //Nome da conexão
    79     'hostname' => 'localhost:3306', //Servidor onde está o banco de dados
    80     'username' => 'root', //Usuário do banco de dados
    81     'password' => '', //Caso possua, a senha do banco de dados
    82     'database' => 'bd_estagio', //Nome do banco de dados criado
    83     'dbdriver'  => 'mysqli', //Driver do banco de dados, iremos utilizar esse por estarmos
    84     //trabalhando com o Banco MySQL
    85     'dbprefix' => '',
    86     'pconnect' => FALSE,
    87     'db_debug' => ($ENVIRONMENT !== 'production'),
    88     'cache_on' => FALSE,
    89     'cachedir' => '',
    90     'char_set' => 'utf8',
    91     'dbcollat' => 'utf8_general_ci',
    92     'swap_pre' => '',
    93     'encrypt' => FALSE,
    94     'compress' => FALSE,
    95     'stricton' => FALSE,
    96     'failover' => array(),
    97     'save_queries' => TRUE
    98 );

```

17.5.1.2. Configurando o arquivo *autoload.php*

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



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

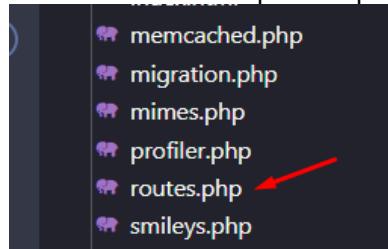
```

    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');
    62

```

17.5.1.3. Configurando o arquivo *routes.php*

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



Informaremos a *controller* que dará início ao sistema, da seguinte forma:

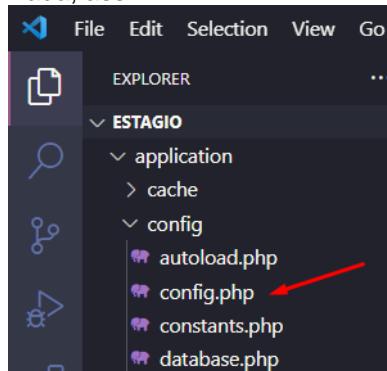
```

50 |     my-controller/my-method -> my_controller
51 */
52 $route['default_controller'] = 'curso'; ----->
53 $route['404_override'] = '';
54 $route['translate_uri_dashes'] = FALSE;
55 |

```

17.5.1.4. 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:

```

config.php x
application > config > config.php
19 | file 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/estagio/'; ----->
27

```

Também nesse mesmo arquivo vamos retirar o *index.php* para trabalharmos com URL amigáveis.

```

30 | Index File
31 | -----
32 |
33 | Typically this will be your index.php file, unless you've renamed it to
34 | something else. If you are using mod_rewrite to remove the page set this
35 | variable so that it is blank.
36 |
37 */
38 $config['index_page'] = ''; ----->
39

```

17.5.1.5. 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):

```

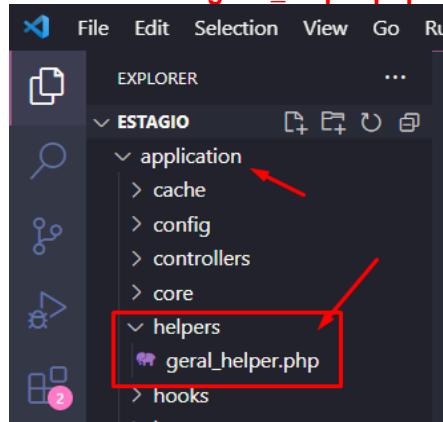
        File Edit Selection View Go Run Terminal Help .htaccess - estagio - Visual Studio Code
        EXPLORER ...
        ESTAGIO
        > application
        > system
        > user_guide
        .editorconfig
        .gitignore
        .htaccess
        composer.json
        .htaccess x
        .htaccess
        1 RewriteEngine On
        2 RewriteCond %{REQUEST_FILENAME} !-f
        3 RewriteCond %{REQUEST_FILENAME} !-d
        4 RewriteRule ^(.*)$ /estagio/index.php/$1 [L]
    
```

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

17.6. Utilização da *Helpers* no *Codeigniter*

Faremos agora a utilização do **HELPERS** no *Codeigniter*, como falamos no início da abordagem desse framework, não é nada mais que um ajudante, um grupo de funções onde você poderá chamar no decorrer de seu projeto, assim que precisar, em nosso caso faremos essa função inicialmente, e caso precisemos, criaremos outras na sequência de nossa práxis.

Para criar um *helper* você terá que criar um arquivo no formato **nomedoarquivo_helper.php**. Então toda vez que você criar um *helper* utilize o sufixo **_helper**. Esse arquivo deve ser colocado na pasta “*helpers*” localizado na sua pasta “*application*”. Nosso implemento se chamará **geral_helper.php**.



A seguir vamos programar nossa função no **HELPER**, que será bem simples, somente será uma rotina para verificar se os campos passados pelo *FrontEnd* serão os mesmos utilizados dentro das *Controllers*, isso foi idealizado em nosso projeto para evitar erros no lado do *BackEnd*,

vejam:



```
geral_helper.php X
application > helpers > geral_helper.php
1  <?php
2  defined('BASEPATH') or exit('No direct script access allowed');
3
4  /*
5   * Função para verificar os parâmetros vindos do FrontEnd
6   */
7  function verificarParam($atributos, $lista){
8      //1º -Verificar se os elementos do Front estão nos atributos necessários
9      foreach($lista as $key => $value){
10          if(array_key_exists($key, get_object_vars($atributos))){
11              $estatus = 1;
12          }else{
13              $estatus = 0;
14              break;
15          }
16      }
17
18      // 2º - Verificando a quantidade de elementos
19      if (count(get_object_vars($atributos)) != count($lista)){
20          $estatus = 0;
21      }
22
23      return $estatus;
24  }
25
26 ?>
```

Acima vemos dois comandos interessantes, que nos auxiliam a verificar os arrays necessários, em um deles temos a necessidade de converter objetos em array, para que a comparação seja realizada.

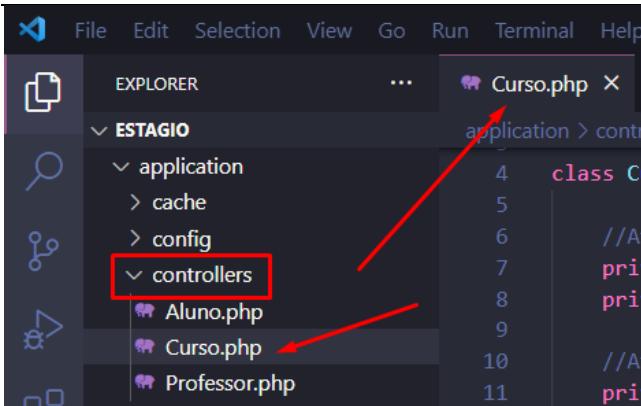
- `array_key_exists()` — Checa se uma chave ou índice existe em um array;
- `get_object_vars` — Retorna um array associativo das definidas acessíveis propriedades do objeto especificado por `object`.

17.7. Objeto Curso

Vamos programar nossa primeira classe do projeto, iremos fazer um **CRUD** (*Create, Read, Update and Delete*) para os cursos da instituição.

17.7.1. Controller de Curso (métodos inserir, atualizar, consultar e apagar)

Vamos programar nossa primeira classe, lembrando que utilizaremos conceitos de utilização de Orientação a Objetos, a seguir iremos estruturar nossa classe com atributos privados e criação dos `getters` e `setters` que utilizaremos, vejam:



A seguir a codificação:

```

class Curso extends CI_Controller{
    private $json;
    private $resultado;
    private $idcurso;
    private $descricao;
    private $estatus;

    public function getIdCurso(){
        return $this->idcurso;
    }

    public function getDescricao(){
        return $this->descricao;
    }

    public function getEstatus(){
        return $this->estatus;
    }

    public function setIdCurso($idCursoFront){
        $this->idcurso = $idCursoFront;
    }

    public function setDescricao($descFront){
        $this->descricao = $descFront;
    }

    public function setEstatus($estatusFront){
        $this->estatus = $estatusFront;
    }
}

```

Na sequência, vamos observar e programar o método inserirCurso(), vejam:

```

40
41     //Métodos da classe
42     //Cadastrar o Curso
43     public function inserirCurso(){
44         //Dados vindos do FrontEnd
45         $json = file_get_contents('php://input');
46         $resultado = json_decode($json);
47
48         //Array com os dados que deverão vir do Front
49         $lista = array("descricao" => '0',
50                         "estatus"   => '0');
51
52         //Verificação dos parâmetros passados na Helper
53         if (verificarParam($resultado, $lista) == 1){
54             //Fazemos os setters
55             $this->setDescricao($resultado->descricao);
56             $this->setEstatus($resultado->estatus);
57
58             /*
59              ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
60              01 - CURSO CADASTRADO CORRETAMENTE (MODEL)
61              02 - HOUVE ALGUM PROBLEMA DE SALVAMENTO DOS DADOS (MODEL)
62              03 - DESCRIÇÃO NÃO INFORMADA (CONTROLLER)
63              04 - STATUS NÃO CONDIZ COM O PERMITIDO (CONTROLLER)
64              99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
65
66
67             if (trim($this->getDescricao()) == ""){
68                 $retorno = array('codigo' => 3,
69                                 'msg'      => 'Descrição não informada.');
70             elseif ($this->getEstatus() != "D" && $this->getEstatus() != ""){
71                 $retorno = array('codigo' => 4,
72                                 'msg'      => 'Status não condiz com o permitido.');
73             }else{
74                 //Realizo a instância da Model
75                 $this->load->model('M_curso');
76
77                 //Atributo $retorno recebe array com informações da validação do acesso
78                 $retorno = $this->M_curso->inserirCurso($this->getDescricao(), $this->getEstatus());
79             }
80         }else{
81             $retorno = array('codigo' => 99,
82                             'msg'      => 'Os campos vindos do FrontEnd não representam o método de inserção,
83                                         | verifique.');
84         }
85
86         echo json_encode($retorno);
87     }

```

Acima temos três considerações:

- 1- São dados recebidos que serão decodificados do JSON para um array, onde colocamos em \$resultado;
- 2- Um array onde colocaremos sempre os dados que iremos tratar dentro de nossa Controller; definiremos na regra de negócio, seria uma espécie de controle do lado do BackEnd;
- 3- Chamada da função da Helper para validarmos o que recebemos e o que necessitamos em nosso método.

Ressalto que iremos utilizar esses conceitos e estruturas estudadas agora nos demais métodos de nosso projeto, independente da classe que estejamos trabalhando, fazemos também as validações necessárias de acordo com cada atributo recebido, e consequentemente o retorno de um erro, ou, seguimos até a chamada da Model, ainda não fizemos a model, mas a chamada é o método estão sendo contemplados no código.

Na sequência, vamos ver agora o método consultarCurso(), vejam:

```
89
90     //Consultar Curso(s)
91     public function consultarCurso(){
92         //Dados vindos do FrontEnd
93         $json = file_get_contents('php://input');
94         $resultado = json_decode($json);
95
96         //Array com os dados que deverão vir do Front
97         $lista = array("idCurso" => '0',
98                         "descricao" => '0',
99                         "estatus" => '0');
100
101        //Verificação dos parâmetros passados na Helper
102        if (verificarParam($resultado, $lista) == 1){
103            //Fazendo os setters
104            $this->setIdCurso($resultado->idCurso);
105            $this->setDescricao($resultado->descricao);
106            $this->setEstatus($resultado->estatus);
107
108            /*
109             ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
110             01 - CURSO CONSULTADO CORRETAMENTE (MODEL)
111             02 - NÃO HOUVE RETORNO NA CONSULTA DOS DADOS (MODEL)
112             04 - STATUS NÃO CONDIZ COM O PERMITIDO (CONTROLLER)
113             99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
114
115
116            if ($this->getEstatus() != "D" && $this->getEstatus() != ""){
117                $retorno = array('codigo' => 4,
118                                'msg' => 'Status não condiz com o permitido.');
119            }else{
120                //Realizo a instância da Model
121                $this->load->model('M_curso');
122
123                //Atributo $retorno recebe array com informações da validação do acesso
124                $retorno = $this->M_curso->consultarCurso($this->getIdCurso(), $this->getDescricao(),
125                                                $this->getEstatus());
126            }
127        }else{
128            $retorno = array('codigo' => 99,
129                            'msg' => 'Os campos vindos do FrontEnd não representam o método de consulta,
130                            verifique.');
131        }
132
133        echo json_encode($retorno);
134
135    }
```

Na sequência, vamos ver agora o método alterarCurso(), vejam:

```
136 //Na sequência, vamos ver agora o método alterarCurso(), vejam...
137
138     //Alterar Curso
139     public function alterarCurso(){
140         //Dados vindos do FrontEnd
141         $json = file_get_contents('php://input');
142         $resultado = json_decode($json);
143
144         //Array com os dados que deverão vir do Front
145         $lista = array("idCurso" => '0',
146                         "descricao" => '0');
147
148         //Verificação dos parâmetros passados na Helper
149         if (verificarParam($resultado, $lista) == 1){
150             //Fazendo os setters
151             $this->setIdCurso($resultado->idCurso);
152             $this->setDescricao($resultado->descricao);
153
154             /*
155                 ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
156                 01 - CURSO ALTERADO CORRETAMENTE (MODEL)
157                 02 - HOUVE ALGUM PROBLEMA DE ALTERAÇÃO DOS DADOS (MODEL)
158                 03 - ID DO CURSO NÃO INFORMADO OU ZERADO (CONTROLLER)
159                 04 - DESCRIÇÃO DO CURSO NÃO INFORMADA (CONTROLLER)
160                 05 - O ID DO CURSO INFORMADO NÃO ESTA CADASTRADO NA BASE DE DADOS (MODEL)
161                 99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
162             */
163         }
164     }
```

```

162
163     if ($this->getIdCurso() == "" || $this->getIdCurso() == 0){
164         $retorno = array('codigo' => 3,
165                         'msg'      => 'ID do curso não informado ou zerado.');
166     }elseif(strlen($this->getDescricao()) == 0){
167         $retorno = array('codigo' => 4,
168                         'msg'      => 'Descrição do curso não informada.');
169     }else{
170         //Realizo a instância da Model
171         $this->load->model('M_curso');
172
173         //Atributo $retorno recebe array com informações da validação do acesso
174         $retorno = $this->M_curso->alterarCurso($this->getIdCurso(), $this->getDescricao());
175     }
176 }else{
177     $retorno = array('codigo' => 99,
178                     'msg'      => 'Os campos vindos do FrontEnd não representam o método de consulta,
179                               verifique.');
180 }
181
182 echo json_encode($retorno);
183
184 }

```

E por último, mas não menos importante, o método de apagarCurso(), vejam:

```

186 //Apagar Curso
187 public function apagarCurso(){
188     //Dados vindos do FrontEnd
189     $json = file_get_contents('php://input');
190     $resultado = json_decode($json);
191
192     //Array com os dados que deverão vir do Front
193     $lista = array("idCurso" => '0');
194
195     //Verificação dos parâmetros passados na Helper
196     if (verificarParam($resultado, $lista) == 1){
197         //Fazendo os setters
198         $this->setIdCurso($resultado->idCurso);
199
200         /*
201          ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
202          01 - CURSO "APAGADO" CORRETAMENTE (MODEL)
203          02 - HOUVE ALGUM PROBLEMA NA "EXCLUSÃO" DO CURSO (MODEL)
204          03 - ID DO CURSO NÃO INFORMADO OU ZERADO (CONTROLLER)
205          04 - O ID DO CURSO INFORMADO NÃO ESTA CADASTRADO NA BASE DE DADOS (MODEL)
206          99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
207
208
209         if ($this->getIdCurso() == "" || $this->getIdCurso() == 0){
210             $retorno = array('codigo' => 3,
211                             'msg'      => 'ID do curso não informado ou zerado.');
212         }else{
213             //Realizo a instância da Model
214             $this->load->model('M_curso');
215
216             //Atributo $retorno recebe array com informações da validação do acesso
217             $retorno = $this->M_curso->apagarCurso($this->getIdCurso());
218         }
219     }else{
220         $retorno = array('codigo' => 99,
221                         'msg'      => 'Os campos vindos do FrontEnd não representam o método de consulta,
222                               verifique.');
223     }
224
225     echo json_encode($retorno);
226
227 }
228 }

```

26.8.2. Model de Curso (métodos inserir, atualizar, consultar e apagar)

Após a criação da camada *Controller*, com seus devidos métodos, abordando validações de entrada de dados, quantidade de elementos e regras de negócio, vamos fazer a camada de modelo, reforço a utilização de conceitos da Orientação a Objetos, vamos primeiramente criar o arquivo *M_curso*, vejam:

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... M_curso.php X
ESTAGIO application > models >
    > cache
    > config
    > controllers
    > core
    > helpers
    > hooks
    > language
    > libraries
    > logs
    > models
        M_aluno.php
        M_curso.php
        M_professor.php
    
```

A seguir, vamos fazer o primeiro método, o **inserirCurso()**, onde o mesmo receberá os atributos, **\$descricao** e **\$estatus** que serão armazenados no banco de dados.

```

M_curso.php X
application > models > M_curso.php
1 <?php
2 defined('BASEPATH') or exit('No direct script access allowed');
3
4 class M_curso extends CI_Model {
5     public function inserirCurso($descricao, $estatus){
6         //Instrução de inserção dos dados
7         $sql = "insert into curso (descricao, estatus)
8             values ('$descricao', '$estatus')";
9
10        $this->db->query($sql);
11
12        //Verificar se a inserção ocorreu com sucesso
13        if($this->db->affected_rows() > 0){
14            $dados = array('codigo' => 1,
15                           'msg'      => 'Curso cadastrado corretamente.');
16        }else{
17            $dados = array('codigo' => 2,
18                           'msg'      => 'Houve algum problema na inserção na tabela de curso.');
19        }
20        //Envia o array $dados com as informações tratadas
21        //acima pela estrutura de decisão if
22        return $dados;
23    }
24 }

```

Na sequência, iremos fazer a consulta do curso, no método consultarCurso(), iremos receber os atributos **\$idCurso**, **\$descricao** e **\$estatus**, e faremos uma **Query** dinâmica, onde retornaremos somente o que for mencionando por esses atributos, vejam:

```

25 public function consultarCurso($idCurso, $descricao, $estatus){
26     //-----
27     //Função que servirá para quatro tipos de consulta:
28     // * Para todos os cursos;
29     // * Para um determinado curso;
30     // * Para um determinado Status;
31     // * Para nomes de cursos;
32     //-----
33
34     //Query para consultar dados de acordo com parâmetros passados
35     $sql = "select * from curso
36             where estatus = '$estatus' ";
37
38     if(trim($idCurso) != '' && trim($idCurso) != '0') {
39         $sql = $sql . "and id_curso = '$idCurso' ";
40     }
41
42     if(trim($descricao) != ''){
43         $sql = $sql . "and descricao like '%$descricao%' ";
44     }
45

```

```

46     $retorno = $this->db->query($sql);
47
48     //Verificar se a consulta ocorreu com sucesso
49     if($retorno->num_rows() > 0){
50         $dados = array('codigo' => 1,
51                       'msg' => 'Consulta efetuada com sucesso.',
52                       'dados' => $retorno->result());
53
54     }else{
55         $dados = array('codigo' => 2,
56                       'msg' => 'Dados não encontrados.');
57     }
58
59     //Envia o array $dados com as informações tratadas
60     //acima pela estrutura de decisão if
61     return $dados;
62 }
```

A seguir temos um método que chamei de `consultarSoCurso()`, onde é recebido o atributo `$idCurso`, ele se fez necessário para quando formos fazer uma simples consulta de existência do mesmo na base dados, onde poderemos utilizar em outras partes de nosso projeto, um ponto a ser mencionado aqui, é que poderíamos utilizar a sobreposição (*overloading*) de método que foi visto em Orientação a Objetos, só mudando a assinatura do mesmo, porém, o PHP não dá suporte a esse recurso, alegando ser uma linguagem fracamente tipada, e considerar isso uma vantagem, então para continuarmos nosso desenvolvimento foi criado esse método, vejam:

```

63
64     public function consultarSoCurso($idCurso){
65         //-----
66         //Função que servirá somente para verificar se o curso está na base de dados
67
68         //Query para consultar dados de acordo com parâmetros passados
69         $sql = "select * from curso
70             where id_curso = '$idCurso'
71             and estatus = ''";
72
73         $retorno = $this->db->query($sql);
74
75         //Verificar se a consulta ocorreu com sucesso
76         if($retorno->num_rows() > 0){
77             $dados = array('codigo' => 1,
78                           'msg' => 'Consulta efetuada com sucesso.');
79
80         }else{
81             $dados = array('codigo' => 2,
82                           'msg' => 'Dados não encontrados.');
83         }
84
85         //Envia o array $dados com as informações tratadas
86         //acima pela estrutura de decisão if
87         return $dados;
88     }
89 }
```

A seguir temos o método `alterarCurso()`, onde são recebidos os atributos `$idCurso` e `$descricao`, reparem na linha 92, fazemos a chamada do método `consultarSoCurso()`, onde passamos o atributo `$idCurso`, fazemos isso para validar se o curso que será alterado, efetivamente está cadastrado, vejam:

```

90     public function alterarCurso($idCurso, $descricao){
91         //Verificar se o curso está cadastrado na base de dados
92         $retornoCurso = $this->consultarSoCurso($idCurso);
93
94         if ($retornoCurso['codigo'] == 1){
95
96             //Instrução de inserção dos dados
97             $sql = "update curso set descricao = '$descricao'
98                 where id_curso = $idCurso";
99
100            $this->db->query($sql);
101        }
102    }
103 }
```

```

102     //Verificar se a atualização ocorreu com sucesso
103     if($this->db->affected_rows() > 0){
104         $dados = array('codigo' => 1,
105                     'msg'      => 'Descrição do curso atualizada corretamente.');
106
107     }else{
108         $dados = array('codigo' => 2,
109                     'msg'      => 'Houve algum problema na atualização na descrição do curso.');
110     }
111 }else{
112     $dados = array('codigo' => 5,
113                     'msg'      => 'O ID do curso passado não está cadastrado na base de dados.');
114 }
115 //Envia o array $dados com as informações tratadas
116 //acima pela estrutura de decisão if
117 return $dados;
118 }

```

E para finalizar, temos o método apagarCurso(), onde é passado o atributo `$idCurso`, nesse caso, vamos reparar na linha122, que novamente fazemos a chamada do método consultaSoCurso(), passando o atributo `$idCurso`, para validarmos que estaremos “apagando” um curso que se encontra em nosso banco de dados, vejam:

```

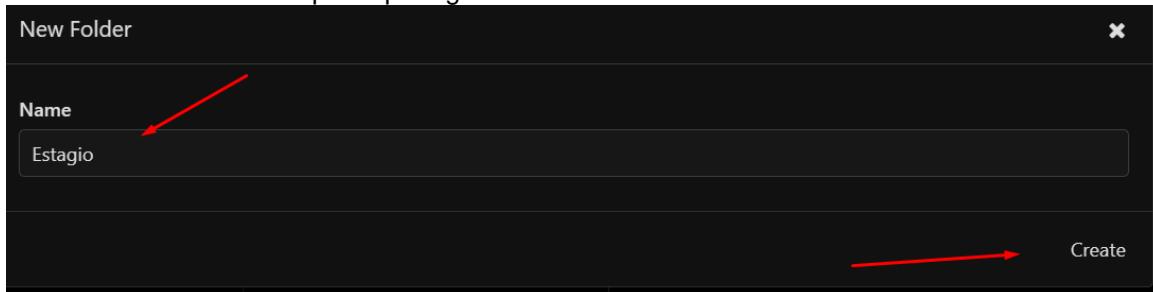
120 public function apagarCurso($idCurso){
121     //Verificar se o curso está cadastrado na base de dados
122     $retornoCurso = $this->consultarSoCurso($idCurso);
123
124     if ($retornoCurso['codigo'] == 1){
125
126         //Instrução de inserção dos dados
127         $sql = "update curso set estatus = 'D'
128             where id_curso = $idCurso";
129
130         $this->db->query($sql);
131
132         //Verificar se a atualização ocorreu com sucesso
133         if($this->db->affected_rows() > 0){
134             $dados = array('codigo' => 1,
135                         'msg'      => 'Curso desativado corretamente.');
136
137         }else{
138             $dados = array('codigo' => 2,
139                         'msg'      => 'Houve algum problema na desativação do curso.');
140         }
141     }else{
142         $dados = array('codigo' => 5,
143                         'msg'      => 'O ID do curso passado não está cadastrado na base de dados.');
144     }
145     //Envia o array $dados com as informações tratadas
146     //acima pela estrutura de decisão if
147     return $dados;
148 }
149 }

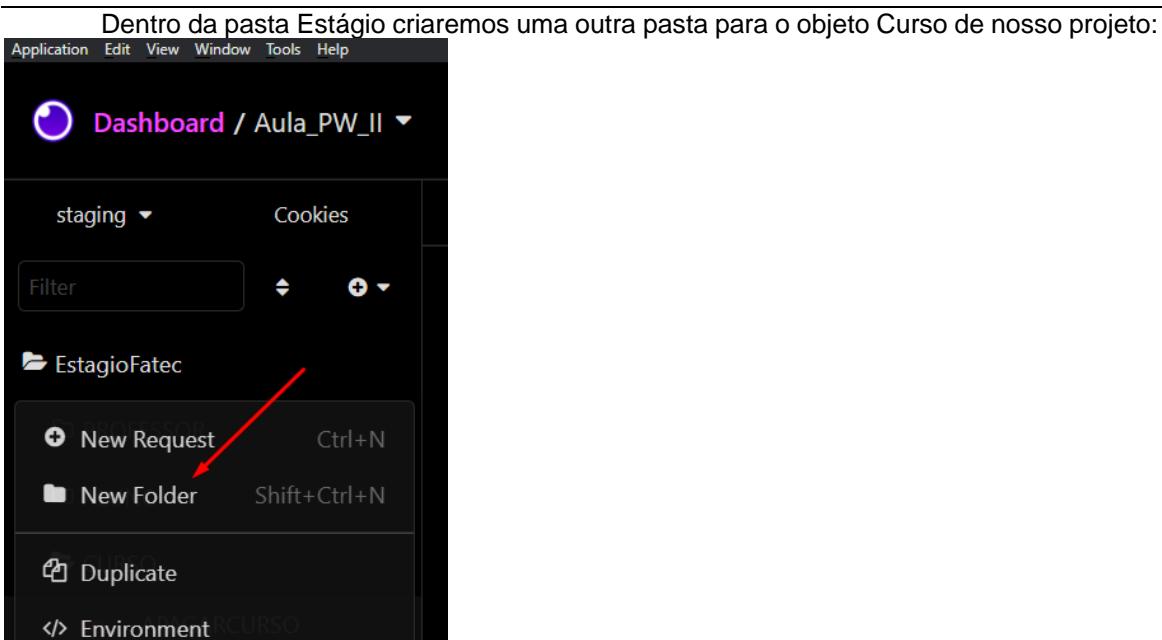
```

26.8.3. Colocando nossa classe no *Insomnia*

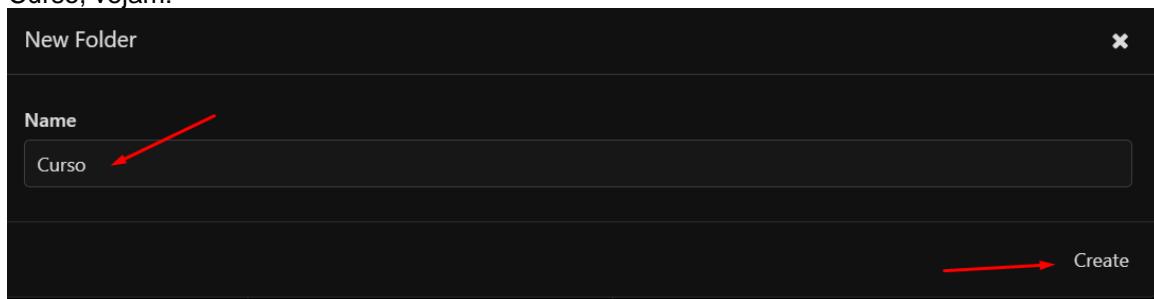
Vamos configurar nosso ambiente para o projeto de nossas aulas, vamos criar uma pasta conforme imagem abaixo:

Definiremos uma pasta para guardar os *ENDPOINTS*:



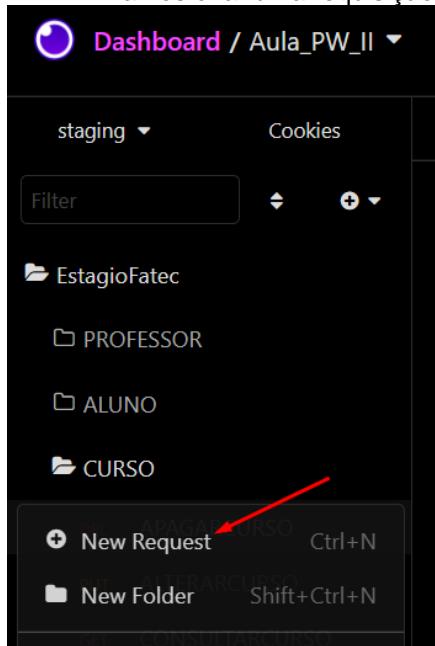


Vamos criar uma pasta Curso, a ideia é armazenarmos os *ENDPOINTS* relacionados a Curso, vejam:



26.8.3.1 Método inserirCurso()

Vamos criar uma requisição, onde vamos invocar nosso método, vejam:



Iremos colocar o nome do *ENDPOINT*:

The screenshot shows the 'New Request' section of the POSTMAN interface. A red box highlights the 'Name' input field containing 'INSERIRCURSO'. Another red box highlights the 'Method' dropdown set to 'POST'. A third red box highlights the 'Format' dropdown set to 'JSON'. Below the input fields, a note says '* Tip: paste Curl command into URL afterwards to import it'.

Sendo: 1 – Nome do *ENDPOINT*; 2 – *POST* (Criar); 3 – Formato (*JSON*)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

The screenshot shows a POST request to 'http://localhost/estagio/curso/inserirCurso'. The method is set to 'POST' and the URL is 'http://localhost/estagio/curso/inserirCurso'. The 'Header' tab shows a value of '1'. The 'Body' tab contains the following JSON data:

```

1 ▶ {
2   "descricao": "Sistemas para Internet",
3   "estatus": ""
4 }
    
```

Após isso enviamos a requisição e observamos o retorno com a resposta.

The screenshot shows a successful POST request with a status of '200 OK' and a response time of '422 ms'. The 'Header' tab shows a value of '1'. The 'Body' tab contains the following JSON data:

```

1 ▶ {
2   "codigo": 1,
3   "msg": "Curso cadastrado corretamente."
4 }
    
```

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

The screenshot shows a POST request with a status of '200 OK' and a response time of '60 ms'. The 'Header' tab shows a value of '1'. The 'Body' tab contains the following JSON data:

```

1 ▶ {
2   "descricao_curso": "",
3   "estatus": ""
4 }
    
```

The 'Preview' tab shows an error message: '1 ▶ { 2 "codigo": 99, 3 "msg": "Os campos vindos do FrontEnd não representam o método de inserção, verifique." 4 }'

Atributos diferentes de acordo com solicitado.

The screenshot shows a POST request with a status of '200 OK' and a response time of '32.9 ms'. The 'Header' tab shows a value of '1'. The 'Body' tab contains the following JSON data:

```

1 ▶ {
2   "descricao": "",
3   "estatus": ""
4 }
    
```

The 'Preview' tab shows an error message: '1 ▶ { 2 "codigo": 3, 3 "msg": "Descrição não informada." 4 }'

Descrição não informada

POST ▼ http://localhost/estagio/curso/inserirCurso Send 200 OK 46.1 ms 62 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
1 ↴ { 2 "descricao": "Sistemas para Internet", 3 "estatus": "Q" 4 }					1 ↴ { 2 "codigo": 4, 3 "msg": "Status não condiz com o permitido." 4 }			

Status diferente dos permitidos

Com resultado, podemos consultar no banco de dados e ver se refletimos a ação, vejam:

```
1 • select * from curso;
```

The result grid shows one row of data:

	id_curso	descricao	dtcria	estatus
▶	1	Sistemas para Internet	2023-06-09 19:12:22	NULL
*	NULL	NULL	NULL	NULL

26.8.3.2 Método consultarCurso()

Vamos criar uma requisição, onde vamos invocar nosso método, vejam:

The sidebar shows the following structure:

- staging ▾
- Cookies
- Filter
- EstagioFatec
- PROFESSOR
- ALUNO
- CURSO**
 - + New Request Ctrl+N
 - New Folder Shift+Ctrl+N

Iremos colocar o nome do *ENDPOINT*:

New Request

Name (defaults to your request URL if left empty)

CONSULTARCURSO

1

2 GET ▾

* Tip: paste Curl command into URL afterwards to import it

Create

Sendo: 1 – Nome do *ENDPOINT*; 2 – GET (Consultar)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

```
GET ▼ http://localhost/estagio/curso/consultarCurso Send
```

JSON ▼ Auth ▼ Query Header 1 Docs

```
1 ▶ {
  2   "idCurso": "",
  3   "descricao": "",
  4   "estatus": ""
  5 }
```

Após isso enviamos a requisição e observamos o retorno com a resposta.

```
GET ▼ http://localhost/estagio/curso/consultarCurso Send 200 OK 20.3 ms 161 5 Minutes Ago
```

JSON ▼ Auth ▼ Query Header 1 Docs Preview ▼ Header 6 Cookie Timeline

```
1 ▶ {
  2   "idCurso": "",
  3   "descricao": "",
  4   "estatus": ""
  5 }
```

```
1 ▶ {
  2   "codigo": 1,
  3   "msg": "Consulta efetuada com sucesso.",
  4   "dados": [
  5     {
  6       "id_curso": "1",
  7       "descricao": "Sistemas para Internet",
  8       "dtcria": "2023-06-09 19:12:22",
  9       "estatus": ""
  10    }
  11  ]
  12 }
```

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

```
GET ▼ http://localhost/estagio/curso/consultarCurso Send 200 OK 16.4 ms 111 B Just Now
```

JSON ▼ Auth ▼ Query Header 1 Docs Preview ▼ Header 6 Cookie Timeline

```
1 ▶ {
  2   "idCurso": "",
  3   "descricao": ""
  4 }
```

```
1 ▶ {
  2   "codigo": 99,
  3   "msg": "Os campos vindos do FrontEnd não representam o método de consulta, verifique."
  4 }
```

Atributos diferentes de acordo com solicitado.

```
GET ▼ http://localhost/estagio/curso/consultarCurso Send 200 OK 51.4 ms 62 B Just Now
```

JSON ▼ Auth ▼ Query Header 1 Docs Preview ▼ Header 6 Cookie Timeline

```
1 ▶ {
  2   "idCurso": "",
  3   "descricao": "",
  4   "estatus": "Z"
  5 }
```

```
1 ▶ {
  2   "codigo": 4,
  3   "msg": "Status não condiz com o permitido."
  4 }
```

Status diferente dos permitidos

26.8.3.3 Método alterarCurso()

Vamos criar uma requisição, onde vamos invocar nosso método, vejamos:

The screenshot shows a dark-themed dashboard interface. On the left, there's a sidebar with categories: 'staging ▾', 'Cookies', 'Filter', 'EstagioFatec', 'PROFESSOR', 'ALUNO', and 'CURSO'. A context menu is open over the 'CURSO' item, containing 'New Request' (with a red arrow pointing to it), 'New Folder', and 'GET CONSULTARCURSO'. At the bottom of the menu, there are 'Ctrl+N' and 'Shift+Ctrl+N' keyboard shortcuts.

Iremos colocar o nome do *ENDPOINT*:

The screenshot shows a 'New Request' dialog box. The 'Name' field (1) contains 'ALTERARCURSO'. To the right, there are 'PUT' (2) and 'JSON' (3) dropdown menus. Below the dialog, a tip says: '* Tip: paste Curl command into URL afterwards to import it'.

Sendo: 1 – Nome do *ENDPOINT*; 2 – *PUT* (Atualizar); 3 – Formato (*JSON*)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

The screenshot shows a POSTMAN-style interface. The 'Method' dropdown is set to 'PUT' and the 'URL' field contains 'http://localhost/estagio/curso/alterarCurso'. The 'Body' tab is selected and shows the following JSON data:

```

1 ▶ {
2   "idCurso": "1",
3   "descricao": "Sistemas para Internet Fatec"
4 }

```

Após isso enviamos a requisição e observamos o retorno com a resposta.

The screenshot shows the API response in the POSTMAN tool. The status is '200 OK' and the response body is:

```

1 ▶ {
2   "codigo": 1,
3   "msg": "Descrição do curso atualizada corretamente."
4 }

```

Refletindo no banco de dados, vejam:

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it, a query editor window displays the SQL command: `select * from curso;`. The results are shown in a 'Result Grid' table with four columns: `id_curso`, `descricao`, `dtrcia`, and `estatus`. A single row is present with values: `1`, `Sistemas para Internet Fatec`, `2023-06-09 19:12:22`, and `NULL`.

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

The screenshot shows a Postman request for `http://localhost/estagio/curso/alterarCurso`. The 'Send' button is highlighted. The response status is `200 OK` with a time of `21.1 ms` and a size of `111 B`. The 'Preview' tab shows the response body: `1 { 2 "codigo": 99, 3 "msg": "Os campos vindos do FrontEnd não representam o método de consulta, verifique." 4 }`.

Atributos diferentes de acordo com solicitado.

The screenshot shows a Postman request for `http://localhost/estagio/curso/alterarCurso`. The 'Send' button is highlighted. The response status is `200 OK` with a time of `40.5 ms` and a size of `64 B`. The 'Preview' tab shows the response body: `1 { 2 "codigo": 3, 3 "msg": "ID do curso não informado ou zerado." 4 }`.

ID não informado

The screenshot shows a Postman request for `http://localhost/estagio/curso/alterarCurso`. The 'Send' button is highlighted. The response status is `200 OK` with a time of `44.8 ms` and a size of `71 B`. The 'Preview' tab shows the response body: `1 { 2 "codigo": 4, 3 "msg": "Descrição do curso não informada." 4 }`.

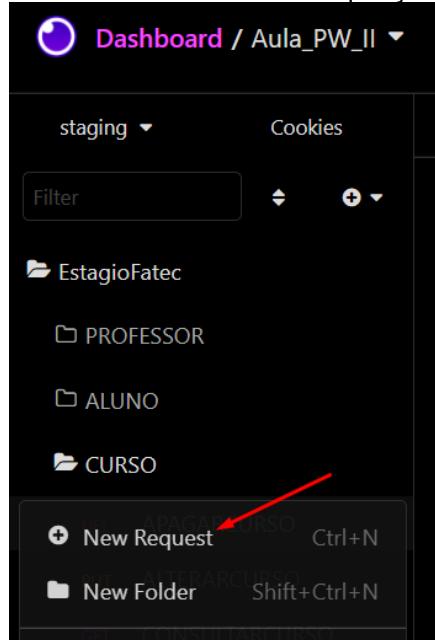
Descrição não informada

The screenshot shows a Postman request for `http://localhost/estagio/curso/alterarCurso`. The 'Send' button is highlighted. The response status is `200 OK` with a time of `38.3 ms` and a size of `92 B`. The 'Preview' tab shows the response body: `1 { 2 "codigo": 5, 3 "msg": "O ID do curso passado não está cadastrado na base de dados." 4 }`.

ID do curso a ser alterado não se encontra cadastrado no banco de dados

26.8.3.4 Método apagarCurso()

Vamos criar uma requisição, onde vamos invocar nosso método, vejam:



Iremos colocar o nome do *ENDPOINT*:

New Request

Name (defaults to your request URL if left empty)

APAGARCURSO 1

2 DELETE ▾

* Tip: paste Curl command into URL afterwards to import it

Create

Sendo: 1 – Nome do *ENDPOINT*; 2 – *DELETE* (Apagar)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

The screenshot shows a REST API testing interface. The top bar has a red box around the URL "DELETE http://localhost/estagio/curso/apagarCurso". To the right is a pink "Send" button. Below the URL are tabs: "JSON" (selected), "Auth", "Query", "Header 1", and "Docs". A red box highlights the JSON body area, which contains the following code:

```
1 {  
2   "idCurso": "1"  
3 }
```

Após isso enviamos a requisição e observamos o retorno com a resposta.

A screenshot of a POST request in Postman. The URL is `http://localhost/estagio/curso/apagarCurso`. The method is `DELETE`. The status code is `200 OK` and the time taken is `18.5 ms`. The response was received `Just Now`. The request body is JSON with the value `{"idCurso": "1"}`. The response body is also JSON, containing the message `"Curso desativado corretamente."`.

Refletindo no banco de dados, vejam:

	id_curso	descricao	dtcria	estatus
▶	1	Sistemas para Internet Fatec	2023-06-09 19:12:22	D
*	NULL	NULL	NULL	NULL

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

```

DELETE ▾ http://localhost/estagio/curso/apagarCurso
Send
200 OK 36 ms 111 B Just Now ▾

```

JSON	Auth	Query	Header 1	Docs	Preview	Header 6	Cookie	Timeline
<pre> 1 v { 2 "idCurso": "1", 3 "nomeCurso": "Gastronomia" 4 } 5 </pre>					<pre> 1 v { 2 "codigo": 99, 3 "msg": "Os campos vindos do FrontEnd não representam o método de consulta, verifique." 4 } </pre>			

Atributos diferentes de acordo com solicitado.

```

DELETE ▾ http://localhost/estagio/curso/apagarCurso
Send
200 OK 40.3 ms 64 B Just Now ▾

```

JSON	Auth	Query	Header 1	Docs	Preview	Header 6	Cookie	Timeline
<pre> 1 v { 2 "idCurso": "" 3 } 4 </pre>					<pre> 1 v { 2 "codigo": 3, 3 "msg": "ID do curso não informado ou zerado." 4 } </pre>			

ID não informado

```

DELETE ▾ http://localhost/estagio/curso/apagarCurso
Send
200 OK 35.8 ms 92 B Just Now ▾

```

JSON	Auth	Query	Header 1	Docs	Preview	Header 6	Cookie	Timeline
<pre> 1 v { 2 "idCurso": "3" 3 } 4 </pre>					<pre> 1 v { 2 "codigo": 5, 3 "msg": "O ID do curso passado não está cadastrado na base de dados." 4 } </pre>			

ID do curso a ser “apagado” não se encontra cadastrado no banco de dados

Ao final, ao vermos a estrutura do *Insomnia*, podemos visualizar da seguinte forma:

The screenshot shows the Insomnia REST Client interface. The top bar has tabs for 'Insomnia - Aula_PW_II (staging)' and 'INSERIRCURSO'. The menu bar includes Application, Edit, View, Window, Tools, and Help. The main area shows a 'Dashboard / Aula_PW_II' with a 'staging' dropdown and a 'Cookies' button. Below is a 'Filter' input field with a dropdown arrow and a '+' button. The left sidebar shows a tree structure with 'EstagioFatec' expanded, containing 'PROFESSOR' and 'ALUNO'. Under 'ALUNO', there is a folder named 'CURSO' which is also expanded. Inside 'CURSO', four methods are listed: 'DEL APAGARCURSO', 'PUT ALTERARCURSO', 'GET CONSULTARCURSO', and 'POST INSERIRCURSO'. The 'CURSO' folder and its contents are highlighted with a red rectangular border.

Com isso finalizamos essa etapa do objeto Curso de nosso projeto.

26.9. Objeto Aluno

Vamos programar a classe de Aluno, seguindo as mesmas premissas da classe Curso, vamos fazer um **CRUD** (*Create, Read, Update and Delete*) para os alunos da instituição.

26.9.1. Controller de Aluno (métodos inserir, atualizar, consultar e apagar)

Seguindo os conceitos da Orientação a Objetos, iremos estruturar nossa classe com atributos privados e criação dos *getters* e *setters* que utilizaremos, vejam:

The screenshot shows the VS Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar shows a tree structure for 'ESTAGIO': 'application', 'cache', 'config', and 'controllers'. The 'controllers' folder is highlighted with a red box and has a red arrow pointing to it from the left. Inside 'controllers', there are files: 'Aluno.php' (highlighted with a red box) and 'Curso.php'. The right side shows the code editor for 'Aluno.php'. The code starts with a PHP opening tag and defines a class 'Aluno' with private attributes and methods.

```

<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Aluno extends CI_Controller {
    public function __construct() {
        parent::__construct();
        // Atributos privados
        $this->load->model('Aluno_model');
    }
}

```

A seguir a codificação:

```
Aluno.php X
application > controllers > Aluno.php
1  <?php
2  defined('BASEPATH') OR exit('No direct script access allowed');
3
4  class Aluno extends CI_Controller{
5
6      //Atributos da classe
7      private $json;
8      private $resultado;
9
10     //Atributos privados da classe
11     private $ra;
12     private $idCurso;
13     private $nome;
14     private $estatus;
15
16     //Getters dos atributos
17     public function getRA(){
18         return $this->ra;
19     }
20
21     public function getIdCurso(){
22         return $this->idCurso;
23     }
24
25     public function getName(){
26         return $this->nome;
27     }
28
29     public function getEstatus(){
30         return $this->estatus;
31     }
32
33     //Setters dos atributos
34     public function setRA($raFront){
35         $this->ra = $raFront;
36     }
37
38     public function setIdCurso($idCursoFront){
39         $this->idCurso = $idCursoFront;
40     }
41
42     public function setName($nomeFront){
43         $this->nome = $nomeFront;
44     }
45
46     public function setEstatus($estatusFront){
47         $this->estatus = $estatusFront;
48     }
49 }
```

Na sequência, vamos observar e programar o método inserirAluno(), vejam:

```
50 //Métodos da classe
51 //Cadastrar o Aluno
52 public function inserirAluno(){
53     //Dados vindos do FrontEnd
54     $json = file_get_contents('php://input');
55     $resultado = json_decode($json);
56
57     //Array com os dados que deverão vir do Front
58     $lista = array("ra" => '0',
59                   "nome" => '0',
60                   "idCurso" => '0',
61                   "estatus" => '0');
62
63     //Verificação dos parâmetros passados na Helper
64     if (verificarParam($resultado, $lista) == 1){
65         //Fazendo os setters
66         $this->setRA($resultado->ra);
67         $this->setIdCurso($resultado->idCurso);
68         $this->setName($resultado->nome);
69         $this->setEstatus($resultado->estatus);
70 }
```

```

1  /*
2   ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
3   01 - ALUNO CADASTRADO CORRETAMENTE (MODEL)
4   02 - HOUVE ALGUM PROBLEMA DE SALVAMENTO DOS DADOS (MODEL)
5   03 - RA DO ALUNO NÃO INFORMADO OU ZERADO (CONTROLLER)
6   04 - ID DO CURSO NÃO INFORMADO OU ZERADO (CONTROLLER)
7   05 - NOME NÃO INFORMADO (CONTROLLER)
8   06 - STATUS NÃO CONDIZ COM O PERMITIDO (CONTROLLER)
9   07 - ID DO CURSO PASSADO NÃO ESTÁ CADASTRADO NA BASE (MODEL)
10  08 - ALUNO JÁ SE ENCONTRA CADASTRADO NA BASE (MODEL)
11  99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
12
13  */
14  if (trim($this->getRA()) == "" || $this->getRA() == 0){
15      $retorno = array('codigo' => 3,
16                      'msg'     => 'RA do Aluno não informado ou zerado.');
17  }elseif (trim($this->getIdCurso()) == "" || $this->getIdCurso() == 0){
18      $retorno = array('codigo' => 4,
19                      'msg'     => 'ID do curso não informado ou zerado.');
20  }elseif ($this->getEstatus() != "D" && $this->getEstatus() != ""){
21      $retorno = array('codigo' => 4,
22                      'msg'     => 'Status não condiz com o permitido.');
23  }elseif(strlen($this->getNome()) == 0){
24      $retorno = array('codigo' => 5,
25                      'msg'     => 'Nome do aluno não informado.');
26  }else{
27      //Realizo a instância da Model
28      $this->load->model('M_aluno');
29
30      //Atributo $retorno recebe array com informações da validação do acesso
31      $retorno = $this->M_aluno->inserirAluno($this->getRA(), $this->getIdCurso(), $this->getNome(),
32                                              $this->getEstatus());
33  }
34 }else{
35     $retorno = array('codigo' => 99,
36                     'msg'     => 'Os campos vindos do FrontEnd não representam o método de inserção,
37                               verifique.');
38 }
39
40 echo json_encode($retorno);
41
42 }

```

Na sequência, vamos observar e programar o método consultarAluno(), vejam:

```
113 //Consultar Aluno(s)
114 public function consultarAluno(){
115     //Dados vindos do FrontEnd
116     $json = file_get_contents('php://input');
117     $resultado = json_decode($json);
118
119     //Array com os dados que deverão vir do Front
120     $lista = array("ra"      => '0',
121                   "idCurso" => '0',
122                   "nome"    => '0',
123                   "estatus" => '0');
124
125     //Verificação dos parâmetros passados na Helper
126     if (verificarParam($resultado, $lista) == 1){
127         //Fazendo os setters
128         $this->setRA($resultado->ra);
129         $this->setIdCurso($resultado->idCurso);
130         $this->setNome($resultado->nome);
131         $this->setEstatus($resultado->estatus);
132
133         /*
134             ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
135             01 - ALUNO CONSULTADO CORRETAMENTE (MODEL)
136             02 - NÃO HOUVE RETORNO NA CONSULTA DOS DADOS (MODEL)
137             04 - STATUS NÃO CONDIZ COM O PERMITIDO (CONTROLLER)
138             99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
139         */
140
141         if ($this->getEstatus() != "D" && $this->getEstatus() != ""){
142             $retorno = array('codigo' => 4,
143                             'msg'    => 'Status não condiz com o permitido.');
144         }
145     }
146 }
```

```
144 }else{
145     //Realizo a instânciada Model
146     $this->load->model('M_aluno');
147
148     //Atributo $retorno recebe array com informações da validação do acesso
149     $retorno = $this->M_aluno->consultarAluno($this->getRA(), $this->getIdCurso(), $this->getNome(),
150                                         | $this->getEstatus());
151 }
152 }else{
153     $retorno = array('codigo' => 99,
154                      'msg'    => 'Os campos vindos do FrontEnd não representam o método de consulta,
155                               | verifique.');
156 }
157
158 echo json_encode($retorno);
159
160 }
```

Na sequência, vamos observar e programar o método alterarAluno(), vejamos:

```
161 //Alterar Curso
162 public function alterarAluno(){
163     //Dados vindos do FrontEnd
164     $json = file_get_contents('php://input');
165     $resultado = json_decode($json);
166
167     //Array com os dados que deverão vir do Front
168     $lista = array("ra"      => '0',
169                   "idCurso" => '0',
170                   "nome"    => '0');
171
172     //Verificação dos parâmetros passados na Helper
173     if (verificarParam($resultado, $lista) == 1){
174         //Fazendo os setters
175         $this->setRA($resultado->ra);
176         $this->setIdCurso($resultado->idCurso);
177         $this->setNome($resultado->nome);
178
179         /*
180             ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
181             01 - ALUNO ALTERADO CORRETAMENTE (MODEL)
182             02 - HOUVE ALGUM PROBLEMA DE ALTERAÇÃO DOS DADOS (MODEL)
183             03 - ID DO CURSO NÃO INFORMADO OU ZERADO (CONTROLLER)
184             04 - NOME DO ALUNO NÃO INFORMADO (CONTROLLER)
185             05 - O ID DO CURSO INFORMADO NÃO ESTA CADASTRADO NA BASE DE DADOS (MODEL)
186             06 - ALUNO NÃO CADASTRADO NA BASE DE DADOS (MODEL)
187             99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
188
189         */
190
191         if(strlen($this->getRA()) == 0){
192             $retorno = array('codigo' => 3,
193                             'msg'   => 'RA do aluno não informado.');
194         }elseif ($this->getIdCurso() == "" || $this->getIdCurso() == 0){
195             $retorno = array('codigo' => 3,
196                             'msg'   => 'ID do curso não informado ou zerado.');
197         }elseif(strlen($this->getNome()) == 0){
198             $retorno = array('codigo' => 4,
199                             'msg'   => 'Nome do aluno não informado.');
200         }else{
201             //Realizo a instância da Model
202             $this->load->model('M_aluno');
203
204             //Atributo $retorno recebe array com informações da validação do acesso
205             $retorno = $this->M_aluno->alterarAluno($this->getRA(), $this->getIdCurso(), $this->getNome());
206         }
207     }else{
208         $retorno = array('codigo' => 99,
209                         'msg'   => 'Os campos vindos do FrontEnd não representam o método de consulta, verifique.');
210     }
211
212     echo json_encode($retorno);
213
214 }
```

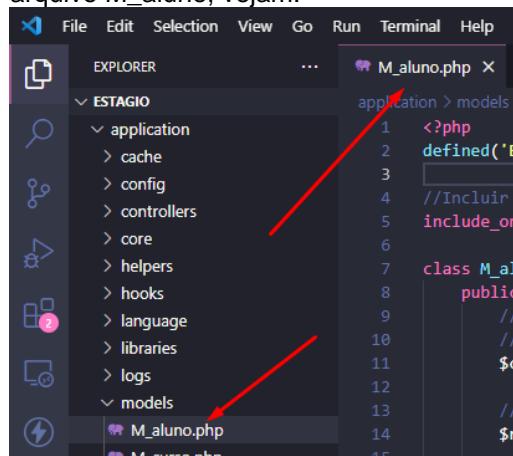
E para finalizar, vamos programar o método apagarAluno(), vejam:

```

217     // "Apagar" Aluno
218     public function apagarAluno(){
219         //Dados vindos do FrontEnd
220         $json = file_get_contents('php://input');
221         $resultado = json_decode($json);
222
223         //Array com os dados que deverão vir do Front
224         $lista = array("ra" => '0');
225
226         //Verificação dos parâmetros passados na Helper
227         if (verificarParam($resultado, $lista) == 1){
228             //Fazendo os setters
229             $this->setRA($resultado->ra);
230
231             /*
232              ** VALIDAÇÕES NECESSÁRIAS RETORNOS POSSÍVEIS **
233              01 - ALUNO "APAGADO" CORRETAMENTE (MODEL)
234              02 - HOUVE ALGUM PROBLEMA NA "EXCLUSÃO" DO ALUNO (MODEL)
235              03 - RA DO ALUNO NÃO INFORMADO OU ZERADO (CONTROLLER)
236              04 - RA DO ALUNO INFORMADO NÃO ESTA CADASTRADO NA BASE DE DADOS (MODEL)
237              99 - OS CAMPOS INFORMADOS PELO FRONTEND NÃO REPRESENTAM O MÉTODO (CONTROLLER)
238             */
239
240             if(strlen($this->getRA()) == 0){
241                 $retorno = array('codigo' => 3,
242                                 'msg'      => 'RA do aluno não informado.');
243             }else{
244                 //Realizo a instância da Model
245                 $this->load->model('M_aluno');
246
247                 //Atributo $retorno recebe array com informações da validação do acesso
248                 $retorno = $this->M_aluno->apagarAluno($this->getRA());
249             }
250         }else{
251             $retorno = array('codigo' => 99,
252                             'msg'      => 'Os campos vindos do FrontEnd não representam o método de consulta, verifique.');
253         }
254
255
256         echo json_encode($retorno);
257
258     }
259 }
```

26.9.2. Model de Aluno (métodos inserir, atualizar, consultar e apagar)

Após a criação da camada *Controller*, com seus devidos métodos, abordando validações de entrada de dados, quantidade de elementos e regras de negócio, vamos fazer a camada de modelo, reforço a utilização de conceitos da Orientação a Objetos, vamos primeiramente criar o arquivo *M_aluno*, vejam:



Nesse momento, percebam na linha 6 que fazemos a inclusão da model *M_curso* nesse arquivo, isso é necessário, pois iremos utilizar métodos dessa classe em nosso desenvolvimento.

A seguir, vamos fazer o primeiro método, o *inserirAluno()*, onde o mesmo receberá os atributos, *\$ra*, *\$idCurso*, *\$nome* e *\$estatus* que serão armazenados no banco de dados.

```

M_aluno.php X
application > models > M_aluno.php

1  <?php
2  defined('BASEPATH') or exit('No direct script access allowed');
3
4  //Incluir a classe que precisaremos instanciar
5  include_once("M_curso.php");
6
7  class M_aluno extends CI_Model {
8      public function inserirAluno($ra, $idCurso, $nome, $estatus){
9          //verificar se o curso está cadastrado
10         //realizar a instância do objeto curso
11         $curso = new M_Curso();
12
13         //chamar o método de verificação
14         $retornoCurso = $curso->consultarSoCurso($idCurso);
15
16         if ($retornoCurso['codigo'] == 1){
17             //verificar se o aluno já se encontra na base de dados
18             $retornoAluno = $this->consultarSoAluno($ra);
19
20             if ($retornoAluno['codigo'] == 2){
21                 //Instrução de inserção dos dados
22                 $sql = "insert into aluno (ra, id_curso, nome, estatus)
23                         values ('$ra', '$idCurso', '$nome', '$estatus')";
24
25                 $this->db->query($sql);
26
27                 //Verificar se a inserção ocorreu com sucesso
28                 if($this->db->affected_rows() > 0){
29                     $dados = array('codigo' => 1,
30                               'msg'     => 'Aluno cadastrado corretamente.');
31                 }else{
32                     $dados = array('codigo' => 2,
33                               'msg'     => 'Houve algum problema na inserção na tabela de aluno.');
34                 }
35             }else{
36                 $dados = array('codigo' => 8,
37                               'msg'     => 'Aluno já se encontra cadastrado na base de dados.');
38             }
39         }else{
40             $dados = array('codigo' => 7,
41                           'msg'     => 'Curso informado não cadastrado na base de dados.');
42         }
43         //Envia o array $dados com as informações tratadas
44         //acima pela estrutura de decisão if
45         return $dados;
46     }
47 }

```

Na sequência, iremos fazer a consulta do aluno, no método consultarAluno(), iremos receber os atributos `$ra`, `$idCurso`, `$nome` e `$estatus`, e faremos uma Query dinâmica, onde retornaremos somente o que for mencionado por esses atributos, vejam:

```

48  public function consultarAluno($ra, $idCurso, $nome, $estatus){
49      //-----
50      //Função que servirá para quatro tipos de consulta:
51      // * Para todos os alunos;
52      // * Para um determinado aluno;
53      // * Para um determinado Status;
54      // * Para nome do aluno;
55      //-----
56
57      //Query para consultar dados de acordo com parâmetros passados
58      $sql = "select * from aluno
59             where estatus = '$estatus' ";
60
61      if(($ra) != ''){
62          $sql = $sql . "and ra = '$ra' ";
63      }
64
65      if(trim($idCurso) != '' && trim($idCurso) != '0') {
66          $sql = $sql . "and id_curso = '$idCurso' ";
67      }
68
69      if(trim($nome) != ''){
70          $sql = $sql . "and nome like '%$nome%' ";
71      }
72
73      $retorno = $this->db->query($sql);

```

```

74
75     //Verificar se a consulta ocorreu com sucesso
76     if($retorno->num_rows() > 0){
77         $dados = array('codigo' => 1,
78                         'msg' => 'Consulta efetuada com sucesso.',
79                         'dados' => $retorno->result());
80
81     }else{
82         $dados = array('codigo' => 2,
83                         'msg' => 'Dados não encontrados.');
84     }
85
86     //Envia o array $dados com as informações tratadas
87     //acima pela estrutura de decisão if
88     return $dados;
89 }
90

```

Como na classe Curso, a seguir temos um método que chamei de consultarSoAluno(), onde é recebido o atributo `$ra`, ele se fez necessário para quando formos fazer uma simples consulta de existência do mesmo na base dados, onde poderemos utilizar em outras partes de nosso projeto, vejam:

```

91 public function consultarSoAluno($ra){
92     //-----
93     //Função que servirá somente para verificar se o curso está na base de dados
94
95     //Query para consultar dados de acordo com parâmetros passados
96     $sql = "select * from aluno
97             where ra = '$ra'
98                 and estatus = ''";
99
100    $retorno = $this->db->query($sql);
101
102    //Verificar se a consulta ocorreu com sucesso
103    if($retorno->num_rows() > 0){
104        $dados = array('codigo' => 1,
105                         'msg' => 'Consulta efetuada com sucesso.');
106
107    }else{
108        $dados = array('codigo' => 2,
109                         'msg' => 'Dados não encontrados.');
110    }
111
112    //Envia o array $dados com as informações tratadas
113    //acima pela estrutura de decisão if
114    return $dados;
115 }

```

A seguir temos o método alterarAluno(), onde são recebidos os atributos `$ra`, `$idCurso` e `$nome`, reparem na linha 120, fazemos a instância do objeto Curso e na linha 123 fazemos a chamada do método consultarSoCurso() da classe Curso, passando o atributo `$idCurso`, isso se faz necessário para validarmos se o curso que está sendo vinculado ao aluno está cadastrado na base de dado, e a chamada do método consultarSoAluno() na linha 127, onde passamos o atributo `$ra`, fazemos isso para validar se o aluno que será alterado, também está efetivamente está cadastrado, vejam:

```

116
117 public function alterarAluno($ra, $idCurso, $nome){
118     //Verificar se o curso está cadastrado na base de dados
119     //realizar a instância do objeto curso
120     $curso = new M_Curso();
121
122     //chamar o método de verificação
123     $retornoCurso = $curso->consultarSoCurso($idCurso);
124
125     if ($retornoCurso['codigo'] == 1){
126         //verificar se o aluno já se encontra na base de dados
127         $retornoAluno = $this->consultarSoAluno($ra);
128
129         if ($retornoAluno['codigo'] == 1{
130
131             //Instrução de inserção dos dados
132             $sql = "update aluno set nome = '$nome', id_curso = $idCurso
133                             where ra = $ra";
134

```

```

135     $this->db->query($sql);
136
137     //Verificar se a atualização ocorreu com sucesso
138     if($this->db->affected_rows() > 0){
139         $dados = array('codigo' => 1,
140                     'msg'      => 'Dados do aluno atualizados corretamente.');
141
142     }else{
143         $dados = array('codigo' => 2,
144                     'msg'      => 'Houve algum problema na atualização do aluno.');
145     }
146     }else{
147         $dados = array('codigo' => 6,
148                     'msg'      => 'O RA informado não está cadastrado na base de dados.');
149     }
150 }else{
151     $dados = array('codigo' => 5,
152                     'msg'      => 'O ID do curso passado não está cadastrado na base de dados.');
153 }
154 //Envia o array $dados com as informações tratadas
155 //acima pela estrutura de decisão if
156 return $dados;
157 }

```

E para finalizar, temos o método apagarAluno(), onde é passado o atributo `$ra`, nesse caso, vamos reparar na linha 161, que novamente fazemos a chamada do método `consultaSoAluno()`, passando o atributo `$ra`, para validarmos que estaremos “apagando” um aluno que se encontra em nosso banco de dados, vejam:

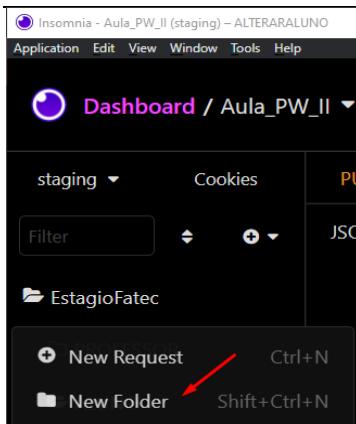
```

159     public function apagarAluno($ra){
160         //Verificar se o curso está cadastrado na base de dados
161         $retornoAluno = $this->consultarSoAluno($ra);
162
163         if ($retornoAluno['codigo'] == 1){
164
165             //Instrução de inserção dos dados
166             $sql = "update aluno set estatus = 'D'
167                   where ra = $ra";
168
169             $this->db->query($sql);
170
171             //Verificar se a atualização ocorreu com sucesso
172             if($this->db->affected_rows() > 0){
173                 $dados = array('codigo' => 1,
174                               'msg'      => 'Aluno desativado corretamente.');
175
176             }else{
177                 $dados = array('codigo' => 2,
178                               'msg'      => 'Houve algum problema na desativação do aluno.');
179             }
180         }else{
181             $dados = array('codigo' => 4,
182                           'msg'      => 'O RA informado não está cadastrado na base de dados.');
183         }
184         //Envia o array $dados com as informações tratadas
185         //acima pela estrutura de decisão if
186         return $dados;
187     }
188 }
189

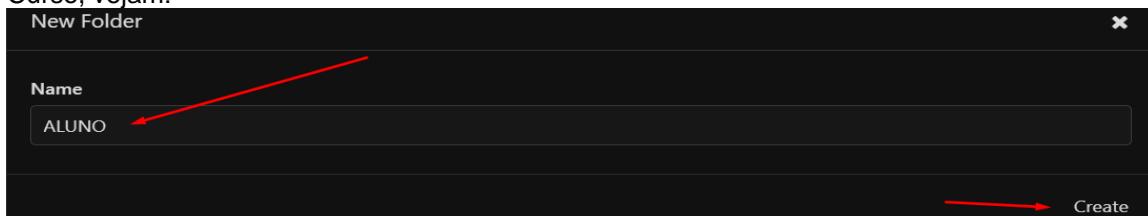
```

26.9.3. Colocando nossa classe no *Insomnia*

Vamos criar uma pasta Aluno, a ideia é armazenarmos os *ENDPOINTS* relacionados a Aluno, vejam:

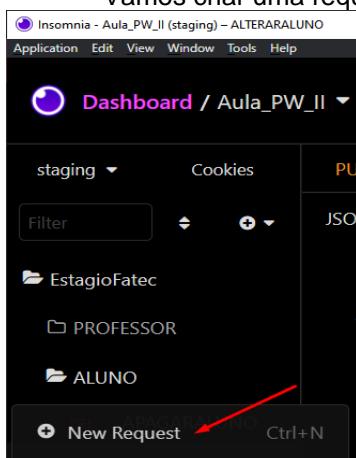


Vamos criar uma pasta Aluno, a ideia é armazenarmos os *ENDPOINTS* relacionados a Curso, vejam:

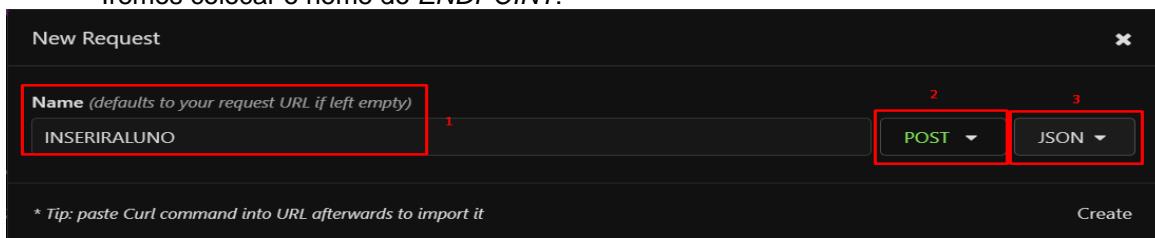


26.9.3.1 Método inserirAluno()

Vamos criar uma requisição, onde vamos invocar nosso método, vejam:



Iremos colocar o nome do *ENDPOINT*:



Sendo: 1 – Nome do *ENDPOINT*; 2 – POST (Criar); 3 – Formato (JSON)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

POST ▾ http://localhost/estagio/aluno/inserirAluno Send

JSON ▾	Auth ▾	Query	Header 1	Docs
<pre>1 ▶ { 2 "ra": "12345", 3 "idCurso": "1", 4 "nome": "Marcos Costa", 5 "estatus": "" 6 }</pre>				

Após isso enviamos a requisição e observamos o retorno com a resposta.

POST ▾ http://localhost/estagio/aluno/inserirAluno Send 200 OK 43 ms 5 3 Minutes Ago ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline	
<pre>1 ▶ { 2 "ra": "12345", 3 "idCurso": "1", 4 "nome": "Marcos Costa", 5 "estatus": "" 6 }</pre>					<pre>1 ▶ { 2 "codigo": 1, 3 "msg": "Aluno cadastrado corretamente." 4 }</pre>				

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

POST ▾ http://localhost/estagio/aluno/inserirAluno Send 200 OK 38.3 ms 121 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline	
<pre>1 ▶ { 2 "ra": "12345", 3 "idCurso": "1", 4 "nome": "Marcos Costa" 5 } 6 }</pre>					<pre>1 ▶ { 2 "codigo": 99, 3 "msg": "Os campos vindos do FrontEnd não representam o método de inserção, verifique." 4 }</pre>				

Atributos diferentes de acordo com solicitado.

POST ▾ http://localhost/estagio/aluno/inserirAluno Send 200 OK 37.3 ms 64 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline	
<pre>1 ▶ { 2 "ra": "", 3 "idCurso": "1", 4 "nome": "Marcos Costa", 5 "estatus": "" 6 }</pre>					<pre>1 ▶ { 2 "codigo": 3, 3 "msg": "RA do Aluno não informado ou zerado." 4 }</pre>				

RA não informado

POST ▾ http://localhost/estagio/aluno/inserirAluno Send 200 OK 26.4 ms 62 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline	
<pre>1 ▶ { 2 "ra": "12345", 3 "idCurso": "1", 4 "nome": "Marcos Costa", 5 "estatus": "A" 6 }</pre>					<pre>1 ▶ { 2 "codigo": 4, 3 "msg": "Status não condiz com o permitido." 4 }</pre>				

Status diferente dos permitidos

POST ▾ http://localhost/estagio/aluno/inserirAluno Send 200 OK 17.4 ms 64 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline	
<pre>1 ▶ { 2 "ra": "12345", 3 "idCurso": "", 4 "nome": "Marcos Costa", 5 "estatus": "" 6 }</pre>					<pre>1 ▶ { 2 "codigo": 4, 3 "msg": "ID do curso não informado ou zerado." 4 }</pre>				

ID do curso não informado

POST ▼ <http://localhost/estagio/aluno/inserirAluno> Send **200 OK** 34.5 ms 76 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 ▶ {
2   "ra": "12345",
3   "idCurso": "4",
4   "nome": "Marcos Costa",
5   "estatus": ""
6 }

```

```

1 ▶ {
2   "codigo": 7,
3   "msg": "Curso informado não cadastrado na base de
dados."
4 }

```

ID do curso não cadastrado na base de dados

POST ▼ <http://localhost/estagio/aluno/inserirAluno> Send **200 OK** 33.2 ms 56 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 ▶ {
2   "ra": "12345",
3   "idCurso": "1",
4   "nome": "",
5   "estatus": ""
6 }

```

```

1 ▶ {
2   "codigo": 5,
3   "msg": "Nome do aluno não informado."
4 }

```

Nome do aluno não informado

Com resultado, podemos consultar no banco de dados e ver se refletimos a ação, vejamos:

```

1 • select * from aluno;
2

```

	ra	id_curso	nome	dtcria	estatus
▶	12345	1	Marcos Costa	2023-06-11 17:55:20	
*	NULL	NULL	NULL	NULL	NULL

26.9.3.2 Método consultarAluno()

Vamos criar uma requisição, onde vamos invocar nosso método, vejamos:

Insomnia - Aula_PW_II (staging) - ALTERARALUNO

Application Edit View Window Tools Help

Dashboard / Aula_PW_II ▾

staging ▾ Cookies PU

Filter JSON

EstagioFatec

PROFESSOR

ALUNO

New Request Ctrl+N

Iremos colocar o nome do *ENDPOINT*:

New Request

Name (defaults to your request URL if left empty)
CONSULTARCURSO 1

Method GET 2

* Tip: paste Curl command into URL afterwards to import it

Create

Sendo: 1 – Nome do *ENDPOINT*; 2 – GET (Consultar)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

GET ▾ http://localhost/estagio/aluno/consultarAluno Send

JSON ▾ Auth ▾ Query Header 1 Docs

```

1 ▶ {
2   "ra":"",
3   "idCurso":"",
4   "nome":"",
5   "estatus":""
6 }
    
```

Após isso enviamos a requisição e observamos o retorno com a resposta.

GET ▾ http://localhost/estagio/aluno/consultarAluno Send 200 OK 19 ms 159 B Just Now ▾

JSON ▾ Auth ▾ Query Header 1 Docs Preview ▾ Header 6 Cookie Timeline

```

1 ▶ {
2   "ra":"",
3   "idCurso":"",
4   "nome":"",
5   "estatus":""
6 }
    
```

1 ▶ {
2 "codigo": 1,
3 "msg": "Consulta efetuada com sucesso.",
4 "dados": [
5 {
6 "ra": "12345",
7 "id_curso": "1",
8 "nome": "Marcos Costa",
9 "dtcria": "2023-06-11 17:55:20",
10 "estatus": ""
11 }
12]
13 }

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

GET ▾ http://localhost/estagio/aluno/consultarAluno Send 200 OK 15.9 ms 111 B Just Now ▾

JSON ▾ Auth ▾ Query Header 1 Docs Preview ▾ Header 6 Cookie Timeline

```

1 ▶ {
2   "ra":"",
3   "idCurso":"",
4   "nome":"",
5   "estatus":"",
6   "nome_mae": ""
7 }
    
```

1 ▶ {
2 "codigo": 99,
3 "msg": "Os campos vindos do FrontEnd não
representam o método de consulta, verifique."
4 }

Atributos diferentes de acordo com solicitado.

GET ▼ http://localhost/estagio/aluno/consultarAluno Send 200 OK 20 ms 62 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
<pre> 1 ▶ { 2 "ra":"", 3 "idCurso":"", 4 "nome":"", 5 "estatus":"Y" 6 }</pre>					<pre> 1 ▶ { 2 "codigo": 4, 3 "msg": "Status não condiz com o permitido." 4 }</pre>			

Status diferente dos permitidos

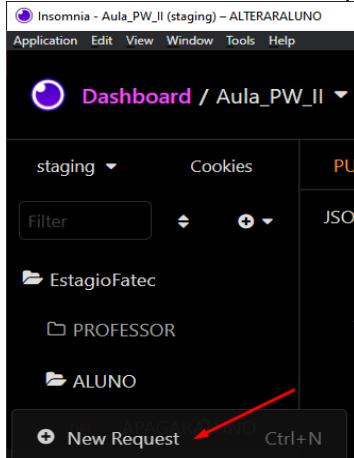
GET ▼ http://localhost/estagio/aluno/consultarAluno Send 200 OK 38.3 ms 50 B Just

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Tim
<pre> 1 ▶ { 2 "ra":"333", 3 "idCurso":"", 4 "nome":"", 5 "estatus":"" 6 }</pre>					<pre> 1 ▶ { 2 "codigo": 2, 3 "msg": "Dados não encontrados." 4 }</pre>			

Dados não encontrados na busca

26.9.3.3 Método alterarAluno()

Vamos criar uma requisição, onde vamos invocar nosso método, vejam:



Iremos colocar o nome do *ENDPOINT*:

New Request

Name (defaults to your request URL if left empty) ALTERARALUNO	2	3
PUT	JSON	

* Tip: paste Curl command into URL afterwards to import it

Create

Sendo: 1 – Nome do *ENDPOINT*; 2 – *PUT* (Atualizar); 3 – Formato (*JSON*)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

PUT ▼ http://localhost/estagio/aluno/alterarAluno Send

JSON ▼ Auth ▼ Query Header 1 Docs

```

1 ▶ {
2     "ra": "12345",
3     "idCurso": "1",
4     "nome": "Marcos Costa de Souza"
5 }

```

Após isso enviamos a requisição e observamos o retorno com a resposta.

PUT ▼ http://localhost/estagio/aluno/alterarAluno Send 200 OK 19.3 ms 63 B Just Now ▼

JSON ▼ Auth ▼ Query Header 1 Docs Preview ▼ Header 6 Cookie Timeline

```

1 ▶ {
2     "ra": "12345",
3     "idCurso": "1",
4     "nome": "Marcos Costa de Sousa"
5 }

```

```

1 ▶ {
2     "codigo": 1,
3     "msg": "Dados do aluno atualizados corretamente."
4 }

```

Refletindo no banco de dados, vejam:

Limit to 50000 rows | ★

```

1 • select * from aluno;
2

```

	ra	id_curso	nome	dtcria	estatus
▶	12345	1	Marcos Costa de Sousa	2023-06-11 17:55:20	
*	NULL	NULL	NULL	NULL	NULL

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

PUT ▼ http://localhost/estagio/aluno/alterarAluno Send 200 OK 24.6 ms 111 B Just Now ▼

JSON ▼ Auth ▼ Query Header 1 Docs Preview ▼ Header 6 Cookie Timeline

```

1 ▶ {
2     "ra": "12345",
3     "idCurso": "1",
4     "nome": "Marcos Costa de Sousa",
5     "nome_mae": "Terezinha Gonçalves"
6 }

```

```

1 ▶ {
2     "codigo": 99,
3     "msg": "Os campos vindos do FrontEnd não
        representam o método de consulta, verifique."
4 }

```

Atributos diferentes de acordo com solicitado.

PUT ▼ http://localhost/estagio/aluno/alterarAluno **Send** **200 OK** 29.5 ms 54 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 ▶ {
2   "ra":"",
3   "idCurso":"1",
4   "nome":"Marcos Costa de Sousa"
5 }

```

RA não informado

PUT ▼ http://localhost/estagio/aluno/alterarAluno **Send** **200 OK** 56.8 ms 64 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 ▶ {
2   "ra":"12345",
3   "idCurso":"",
4   "nome":"Marcos Costa de Sousa"
5 }

```

ID do curso não informado

PUT ▼ http://localhost/estagio/aluno/alterarAluno **Send** **200 OK** 52.3 ms 92 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 ▶ {
2   "ra":"12345",
3   "idCurso":"5",
4   "nome":"Marcos Costa de Sousa"
5 }

```

ID do curso a ser alterado não se encontra cadastrado no banco de dados

PUT ▼ http://localhost/estagio/aluno/alterarAluno **Send** **200 OK** 55.8 ms 56 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 ▶ {
2   "ra":"12345",
3   "idCurso":"1",
4   "nome":""
5 }

```

Nome do aluno não informado

26.9.3.4 Método apagarAluno()

Vamos criar uma requisição, onde vamos invocar nosso método, vejam:

Insomnia - Aula_PW_II (staging) – ALTERARALUNO

Application Edit View Window Tools Help

Dashboard / Aula_PW_II ▾

staging ▾ Cookies PU

Filter JSON

- EstagioFatec
- PROFESSOR
- ALUNO

New Request Ctrl+N

Iremos colocar o nome do ENDPOINT:

New Request

Name (defaults to your request URL if left empty)

APAGARCURSO

DELETE

* Tip: paste Curl command into URL afterwards to import it

Create

Sendo: 1 – Nome do *ENDPOINT*; 2 – *DELETE* (Apagar)

Feito isso precisamos configurar a *url* da *Controller* com o método que vamos trabalhar, e parametrizar os dados de acordo com o especificado:

DELETE ▾ http://localhost/estagio/aluno/apagarAluno

Send

JSON ▾ Auth ▾ Query Header 1 Docs

```

1 ▶ {
2   "ra": "12345"
3 }
4

```

Após isso enviamos a requisição e observamos o retorno com a resposta.

DELETE ▾ http://localhost/estagio/aluno/apagarAluno

Send

200 OK 410 ms 53 B 2 Minutes Ago ▾

JSON ▾ Auth ▾ Query Header 1 Docs Preview ▾ Header 6 Cookie Timeline

```

1 ▶ {
2   "ra": "12345"
3 }
4

```

[1 ▶ {
2 "codigo": 1,
3 "msg": "Aluno desativado corretamente."
4 }]

Refletindo no banco de dados, vejam:

File | New | Open | Save | Print | Filter Rows: | Edit: | Export/Import

Result Grid | Filter Rows: | Edit: | Export/Import

	ra	id_curso	nome	dtria	estatus
▶	12345	1	Marcos Costa de Sousa	2023-06-11 17:55:20	D
*	NULL	NULL	NULL	NULL	NULL

Fizemos o “caminho feliz”, mas também podemos testar as consistências que programamos, vejam:

DELETE ▾ http://localhost/estagio/aluno/apagarAluno

Send **200 OK** 29.7 ms 111 B A Minute Ago ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 var {
2   "ra": "12345",
3   "idCurso": "1"
4 }
5

```

```

1 var {
2   "codigo": 99,
3   "msg": "Os campos vindos do FrontEnd não representam o
método de consulta, verifique."
4 }

```

Atributos diferentes de acordo com solicitado.

DELETE ▾ http://localhost/estagio/aluno/apagarAluno

Send **200 OK** 53.5 ms 54 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 var {
2   "ra": ""
3 }
4

```

```

1 var {
2   "codigo": 3,
3   "msg": "RA do aluno não informado."
4 }

```

RA do aluno não informado

DELETE ▾ http://localhost/estagio/aluno/apagarAluno

Send **200 OK** 39.2 ms 85 B Just Now ▾

JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 6	Cookie	Timeline
--------	--------	-------	----------	------	-----------	----------	--------	----------

```

1 var {
2   "ra": "55"
3 }
4

```

```

1 var {
2   "codigo": 4,
3   "msg": "O RA informado não está cadastrado na base de
dados."
4 }

```

RA do aluno a ser “apagado” não se encontra cadastrado no banco de dados

Ao final, ao vermos a estrutura do *Insomnia*, podemos visualizar da seguinte forma:

The screenshot shows the Insomnia REST Client interface. The title bar says "Insomnia - Aula_PW_II (staging) - APAGARALUNO". The menu bar includes Application, Edit, View, Window, Tools, and Help. The main area has a "Dashboard / Aula_PW" header. Below it, there's a "staging" dropdown and a "Cookies" section. A "Filter" input field is present. Under the "PROFESSOR" section, there's a "ALUNO" folder highlighted with a red box. Inside the ALUNO folder, there are four items: "DEL APAGARALUNO" (highlighted with a red box), "PUT ALTERARALUNO", "GET CONSULTRALUNO", and "POST INSERIRALUNO".