

Instituto Nacional de Telecomunicações – INATEL

**Pós-graduação em Desenvolvimento de Aplicações para
Dispositivos Móveis e Cloud Computing**

**DM107 - Desenvolvimento de Web Services com segurança sob
plataforma Java e PHP**

Parte 2

Sobre a apostila

Este é um material de apoio para o curso de Pós-graduação em Desenvolvimento de Aplicações para Dispositivos Móveis e *Cloud Computing* do Instituto Nacional de Telecomunicações – INATEL, referente a disciplina DM107 - Desenvolvimento de Web Services com segurança sob plataforma Java e PHP.

SUMÁRIO

1.	INTRODUÇÃO	7
2.	JERSEY	7
2.1.	CLASSES DE RECURSOS	7
2.1.1.	@Path	7
2.1.1.1.	Regras de Precedência	9
2.2.	<i>ENCODING</i>	10
2.3.	@GET, @PUT, @POST, @DELETE, ... (HTTP METHODS)	11
2.4.	@PRODUCES.....	11
2.5.	@CONSUMES.....	13
2.6.	INJEÇÕES.....	13
2.6.1.	@PathParam.....	13
2.6.2.	@QueryParam.....	14
2.6.3.	@MatrixParam.....	14
2.6.4.	@HeaderParam	15
2.6.5.	@FormParam	15
2.6.6.	@CookieParam.....	15
2.6.7.	@BeanParam	16
3.	INTRODUÇÃO A LINGUAGEM PHP	16
3.1.	POR QUE ESCOLHER PHP	17
3.2.	INSTALAÇÃO.....	17
3.2.1.	Instalando PHP no Linux (Sem interface gráfica)	18
3.3.	SINTAXE BÁSICA	20
3.3.1.	Variáveis.....	21
3.3.2.	Comentários	21

3.3.2.1. Comentário de uma linha.....	21
3.3.2.2. Comentários de bloco	21
3.3.3. Operadores	21
3.3.3.1. Aritméticos	21
3.3.3.2. Comparação	22
3.3.3.3. Lógicos	22
3.3.3.4. Para Strings.....	22
3.3.3.5. Atribuição	22
3.3.3.6. Incremento e decremento.....	22
3.3.4. Estruturas de Controle.....	22
3.3.5. Estruturas de Repetição.....	23
3.3.5.1. <i>While</i>	23
3.3.5.2. <i>do-while</i>	23
3.3.5.3. <i>for</i>	24
3.3.5.4. <i>foreach</i>	24
3.3.6. Quebra de fluxo	24
3.3.6.1. Break.....	24
3.3.6.2. Continue	24
3.3.7. Array.....	25
3.3.8. Funções	26
3.4. RECEBENDO DADOS DO FORMULÁRIO	26
3.5. SESSÕES	30
3.6. HEADERS E VARIÁVEIS DE SISTEMAS NO PHP	33
3.7. INTEGRAÇÃO ENTRE PHP E BANCO DE DADOS	36
3.8. INTRODUÇÃO A ORIENTAÇÃO A OBJETOS EM PHP	43
3.8.1. Classe e objetos	43

3.8.2.	Herança	44
3.8.3.	Construtor	45
3.8.4.	Destrutor	46
3.8.5.	Encapsulamento.....	47
3.8.6.	Interface.....	47
3.8.7.	Classe Abstrata	48
3.9.	DESENVOLVIMENTO DE WEB SERVICE COM PHP	51
3.9.1.	Introdução ao <i>framework</i> Slim	51
3.9.2.	Configurando Rotas no Slim	55
3.9.3.	Web service com segurança em PHP	60

LISTA DE FIGURAS

Figura 1 - Download da ferramenta XAMPP	18
Figura 2 - Instalação do XAMPP	19
Figura 3 – Painel de controle do XAMPP	19
Figura 4 - Teste da instalação do XAMPP	20
Figura 9 - Resultado da submissão do formulário com o método GET	29
Figura 10 - Resultado da submissão do formulário com o método POST	30
Figura 11 - Guardando uma informação em um array	31
Figura 12 – Guardando uma informação em um array.....	32
Figura 13 – Guardando múltiplas informações em um array	33
Figura 14 - Variáveis Super Globais PHP	35
Figura 15 - Iniciando o serviço do MySQL.....	36
Figura 16 - Tela inicial do PHPMyAdmin	37
Figura 17 - Criando as tabelas no banco de dados MySQL	38

1. INTRODUÇÃO

O desenvolvimento de *Application Programming Interface* (API) RESTful que suporta de forma perfeita a exposição de seus dados em uma variedade de representação de *media types* e abstraia os detalhes de baixo nível da comunicação cliente-servidor não é uma tarefa fácil sem um bom conjunto de ferramentas.

Para simplificar o desenvolvimento de serviços Web RESTful e seus clientes em Java, uma API JAX-RS padrão e portátil foi projetada.

O *framework* RESTful Jersey possui código aberto e foi criado para fornecer suporte para APIs JAX-RS e servir como implementação de Referência JAX-RS (JSR 311 e JSR 339). Além disso o Jersey fornece sua própria API que estende o conjunto de ferramentas JAX-RS com recursos e utilitários adicionais para simplificar ainda mais o serviço RESTful e o desenvolvimento de clientes. O Jersey também expõe inúmeras extensões para que os desenvolvedores possam expandir o Jersey para atender melhor às suas necessidades.

2. JERSEY

2.1. CLASSES DE RECURSOS

As classes de recursos são *Plain Old Java Object* (POJOs) que são anotadas com `@Path`. Elas têm pelo menos um método anotado com `@Path` ou uma anotação de designação de método de recurso, como `@GET`, `@PUT`, `@POST`, `@DELETE`.

2.1.1. `@Path`

O valor da anotação `@Path` é um caminho *Uniform Resource Identifier* (URI) relativo.

Por exemplo, se um arquivo WAR for implado em um *container* servlet, o WAR terá um URI básico que navegadores e clientes remotos usam para acessá-lo. As expressões `@Path` são relativas a este URI.

Os modelos de caminho URI são URIs com variáveis incorporadas na sintaxe URI. Essas variáveis são substituídas em tempo de execução para que um recurso responda a uma solicitação com base no URI substituído. As variáveis são indicadas por chaves. Por

exemplo, veja a seguinte anotação `@Path`:
`@Path ("/ users/{username}")`

Neste exemplo, será solicitado ao usuário inserir seu nome e, em seguida, um serviço da Web Jersey configurado para responder às solicitações deste modelo de caminho URI responderá. Por exemplo, se o usuário inseriu seu nome de usuário como "Galileo", o serviço da Web responderá ao seguinte *Uniform Resource Locator* (URL):
<http://example.com/users/Galileo>

Um valor `@Path` pode ou não começar com um '/', não faz diferença. Da mesma forma, por padrão, um valor `@Path` pode ou não terminar em '/', não faz diferença e, portanto, as URLs de solicitação que terminam ou não terminam em '/' serão ambas correspondentes.

Também é possível aplicar o `@Path` para um método Java. Se isso for feito, o padrão de correspondência URI é um concatenação da expressão `@Path` da classe e da do método. Por exemplo:

```
@Path("/orders")
public class OrderResource {
    @GET
    @Path("unpaid")
    public String getUnpaidOrders() {
        ...
    }
}
```

Assim, o padrão URI para `getUnpaidOrders()` seria o URI relativo `/orders/unpaid/`

As expressões `@Path` não se limitam a expressões de correspondência de curingas simples. Pode-se alterar o valor `@Path` para combinar apenas dígitos:

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id : \\d+}")
    public String getCustomer(@PathParam("id") int id) {
        ...
    }
}
```

As expressões regulares não são limitadas ao combinar um segmento de URI. Por exemplo:

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id : .+}")
    public String getCustomer(@PathParam("id") String id) {
```



```

    ...
}

@GET
@Path("/{id : .+}/address")
public String getAddress(@PathParam("id") String id) {
    ...
}
}

```

2.1.1.1. Regras de Precedência

Juntas, as expressões `@Path` para `getCustomer()` e `getAddress()` são ambíguas. Por exemplo uma solicitação `GET /customers/bill/burke/address` pode corresponder ao método `getCustomer()` ou ao método `getAddress()`, dependendo de qual expressão foi correspondida primeiro pelo provedor JAX-RS.

A especificação JAX-RS definiu regras de classificação e precedência rigorosas para combinar expressões URI e é baseada em um algoritmo mais específico. O provedor JAX-RS agrupa o conjunto de expressões URI e classifica-as com base na seguinte lógica:

1. A chave primária do tipo é o número de caracteres literais na URI completa. O tipo está em ordem decrescente. Neste exemplo ambíguo, o padrão do método `getCustomer()` tem 11 caracteres literais: `/clientes/`. O padrão do método `getAddress()` possui 18 caracteres literais: `/clientes/` mais endereço. Portanto, o provedor JAX-RS tentará combinar o padrão de `getAddress()` antes `getCustomer()`.

2. A chave secundária do tipo é o número de expressões de modelo incorporadas no padrão - isto é, `{id}` ou `{id: .+}`. Esse tipo está em ordem decrescente.

3. A chave terciária do tipo é o número de expressões de modelo não padrão. Uma expressão de modelo padrão é aquela que não define uma expressão regular - isto é, `{id}`.

```

1 /customers/{id}/{name}/address
2 /customers/{id : .+}/address
3 /customers/{id}/address
4 /customers/{id : .+}

```

As expressões 1-3 vêm primeiro porque todas têm mais caracteres literais do que a expressão 4. Embora as expressões 1-3 tenham o mesmo número de caracteres literais, a expressão 1 vem primeiro porque a regra de classificação nº 3 é acionada. Ela tem mais expressões de modelo do que o padrão 2 ou 3. As expressões 2 e 3 possuem o mesmo número de caracteres literais e o mesmo número de expressões de modelo. A expressão 2 é

classificada antes da 3 porque desencadeia a regra de classificação nº 3; Tem um padrão de modelo que é uma expressão regular.

Essas regras de classificação não são perfeitas. Ainda é possível ter ambigüidades, mas as regras cobrem 90% de uso casos. Se a aplicação tiver ambigüidades de correspondência URI, o design da aplicação provavelmente também será complicado e será necessário revisar e refatorar o esquema URI.

2.2. ENCODING

A especificação URI só permite certos caracteres dentro de uma string URI. Ele também reserva certos caracteres para seu próprio uso. Em outras palavras, não é possível usar esses caracteres como parte de seus segmentos URI. Este é o conjunto de caracteres permitidos e reservados:

- Os caracteres alfabéticos US-ASCII a-z e A-Z são permitidos.
- Os caracteres de dígito decimal 0-9 são permitidos.
- Todos esses outros caracteres são permitidos: _- !. ~ '() *.
- Esses caracteres são permitidos, mas são reservados para sintaxe URI: ;, \$ & + =/?\ [] @.

Todos os outros caracteres devem ser codificados usando o caractere "%" seguido de um número hexadecimal de dois dígitos. Este número hexadecimal corresponde ao caractere hexadecimal equivalente na tabela ASCII.

Ao criar expressões @Path, é possível codificar sua string, mas não é preciso. Se um caracter no padrão @Path for um caracter ilegal, o provedor JAX-RS codificará automaticamente o padrão antes de tentar combinar e enviar pedidos *Hypertext Transfer Protocol* (HTTP). Por exemplo:

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("roy&fielding")
    public String getOurBestCustomer() {
        ...
    }
}
```

A expressão @Path para getOurBestCustomer () combinaria pedidos recebidos como GET/customers/roy%26fielding.

2.3. @GET, @PUT, @POST, @DELETE, ... (HTTP METHODS)

@GET, @PUT, @POST, @DELETE e @HEAD são anotações de *design* do método de recursos definidas pelo JAX-RS e que correspondem aos métodos HTTP denominados de forma semelhante. O comportamento de um recurso é determinado por qual dos métodos HTTP em que o recurso está respondendo.

```
@PUT
public Response putContainer() {
    System.out.println("PUT CONTAINER " + container);

    URI uri = uriInfo.getAbsolutePath();
    Container c = new Container(container, uri.toString());

    Response r;
    if (!MemoryStore.MS.hasContainer(c)) {
        r = Response.created(uri).build();
    } else {
        r = Response.noContent().build();
    }

    MemoryStore.MS.createContainer(c);
    return r;
}
```

Por padrão, o tempo de execução JAX-RS suportará automaticamente os métodos HEAD e OPTIONS, se não for implementado explicitamente. Para HEAD, o tempo de execução irá invocar o método GET implementado (se presente) e ignorar a entidade de resposta (se configurado). Uma resposta retornada para o método OPTIONS depende do tipo de *Media Type* solicitado definido no cabeçalho *'Accept'*. O método OPTIONS pode retornar uma resposta com um conjunto de métodos de recursos suportados no cabeçalho *'Accept'* ou retornar um documento Web Application Description Language (WADL).

2.4. @PRODUCES

A anotação @Produces é usada para especificar os tipos de representação de *Media Type* que um recurso pode produzir e enviar de volta para o cliente. Neste exemplo, o método Java produzirá representações identificadas pelo tipo de *Media Type* "text/plan". @Produces podem ser aplicados nos níveis de classe e método.

```
@Path("/myResource")
@Produces("text/plain")
public class SomeResource {
    @GET
    public String doGetAsPlainText() {
        ...
    }
}
```

```

    @GET
    @Produces("text/html")
    public String doGetAsHtml() {
        ...
    }
}

```

O método `doGetAsPlainText` possui a mesma *Media Type* da anotação `@Produces` presente no nível da classe. A anotação `@Produces` do método `doGetAsHtml` substitui a configuração `@Produces` do nível de classe e especifica que o método pode produzir *HyperText Markup Language* (HTML) ao invés de texto sem formatação.

Se uma classe de recurso for capaz de produzir mais de um tipo de *Media Type*, o método de recurso escolhido corresponderá ao tipo de *Media Type* mais aceitável declarado pelo cliente. Mais especificamente, o cabeçalho ‘Accept’ do pedido HTTP declara o que é mais aceitável. Por exemplo, se o cabeçalho ‘Accept’ for "Accept: text/plain", o método `doGetAsPlainText` será invocado. Alternativamente, se o cabeçalho ‘Accept’ for "Accept: text/plain;q=0.9, text/html", que declara que o cliente pode aceitar tipos de *Media Type* de "text/plain" e "text/html", mas prefere o último, então o método `doGetAsHtml` será invocado.

Mais de um tipo de *Media Type* pode ser declarado na mesma declaração `@Produces`, por exemplo:

```

@GET
@Produces({"application/xml", "application/json"})
public String doGetAsXmlOrJson() {
    ...
}

```

O método `doGetAsXmlOrJson` será invocado se qualquer um dos tipos de *Media Type* "application/xml" e "application/json" forem aceitáveis. Se ambos forem igualmente aceitáveis, o primeiro será escolhido porque ocorre primeiro.

Opcionalmente, o servidor também pode especificar o fator de qualidade para tipos de *Media Type* individuais. Estes são considerados se vários são igualmente aceitáveis pelo cliente. Por exemplo:

```

@GET
@Produces({"application/xml; qs=0.9", "application/json"})
public String doGetAsXmlOrJson() {
    ...
}

```

No exemplo acima, se o cliente aceita "application/xml" e "application/json" (igualmente), um servidor sempre envia "application/json", uma vez que "application/xml" possui um menor fator de qualidade.

2.5. @CONSUMES

A anotação `@Consumes` é usada para especificar os tipos de *Media Type* de representações que podem ser consumidas por um recurso.

```
@POST
@Consumes("text/plain")
public void postClichedMessage(String message) {
    // Store the message
}
```

Neste exemplo, o método Java consumirá representações identificadas pelo tipo de *Media Type* "text/plain". Observe que o método do recurso retorna vazio. Isso significa que nenhuma representação é retornada e a resposta com um código de status de 204 (Sem Conteúdo) será retornada ao cliente.

`@Consumes` podem ser aplicados na classe e nos níveis do método e mais de um tipo de *Media Type* pode ser declarado na mesma declaração `@Consumes`.

2.6. INJEÇÕES

Muitas vezes é necessário pegar informações de uma solicitação HTTP e injetar em um método Java. O JAX-RS permite que você pegue esta informação, através de um conjunto de anotações e APIs de injeção.

2.6.1. @PathParam

Extraí um parâmetro da URL de solicitação que corresponde ao caminho declarado em `@Path`.

```
@Path("/customers")
public class CustomerResource {
    ...
    @Path("{id}")
    @GET
    @Produces("application/xml")
    public StreamingOutput getCustomer(@
        PathParam("id") int id) {
        ...
    }
}
```

É possível fazer referência a mais de um parâmetro de caminho URI em seus métodos Java. Por exemplo:

```
@Path("/customers")
public class CustomerResource {
    ...
    @Path("{first}-{last}")
    @GET
    @Produces("application/xml")
```

```

        public StreamingOutput getCustomer(@PathParam("first") String
        firstName,@PathParam("last") String lastName) {
            ...
        }
    }
}

```

Às vezes, um parâmetro de caminho URI será repetido por diferentes expressões `@Path` que irão compor o padrão de correspondência URI de um método de recurso. O parâmetro do caminho pode ser repetido pela expressão `@Path` da classe ou por um localizador *subresource*. Nesses casos, a anotação `@PathParam` sempre faz referência ao parâmetro do caminho final. Por exemplo:

```

@Path("/customers/{id}")
public class CustomerResource {
    @Path("/address/{id}")
    @Produces("text/plain")
    @GET
    public String getAddress(@PathParam("id") String addressId)
    {...}
}

```

2.6.2. @QueryParam

Permite injetar parâmetros de consulta URI individuais em seus parâmetros Java.

```

@Path("/customers")
public class CustomerResource {
    @GET
    @Produces("application/xml")
    public String getCustomers(@QueryParam("start") int start,
    @QueryParam("size") int size) {
        ...
    }
}

```

2.6.3. @MatrixParam

A idéia dos parâmetros da matriz é que eles são um conjunto arbitrário de pares nome-valor incorporados em um segmento do caminho URI.

A ideia básica dos parâmetros da matriz é que ela representa recursos que são endereçáveis pelos seus atributos, bem como pelo seu identificador bruto.

```

@Path("/{make}")
public class CarResource {
    @GET
    @Path("/{model}/{year}")
    @Produces("image/jpeg")
    public Jpeg getPicture(@PathParam("make") String make,
    @PathParam("model") String model,
    @MatrixParam("color") String color) {

```

```
    ...  
}
```

2.6.4. @HeaderParam

É usada para injetar valores de cabeçalho de solicitação HTTP.

```
@Path("/myservice")  
public class MyService {  
    @GET  
    @Produces("text/html")  
    public String get(@HeaderParam("Referer") String referer) {  
        ...  
    }  
}
```

2.6.5. @FormParam

É usada para acessar os *bodies* da solicitação `application / x-www-form-urlencoded`.

Em outras palavras, é usado para acessar entradas individuais postadas por um documento de formulário HTML.

Por exemplo, se houver formulário para registrar novos clientes:

```
<FORM action="http://example.com/customers" method="post">  
  <P>  
    First name: <INPUT type="text" name="firstname"><BR>  
    Last name: <INPUT type="text" name="lastname"><BR>  
    <INPUT type="submit" value="Send">  
  </P>  
</FORM>
```

Pode-se enviar este formulário diretamente para um serviço *backend* descrito da seguinte forma:

```
@Path("/customers")  
public class CustomerResource {  
    @POST  
    public void createCustomer(@FormParam("firstname") String  
first,  
    @FormParam("lastname") String last) {  
        ...  
    }  
}
```

2.6.6. @CookieParam

Os servidores podem armazenar informações de estado em *cookies* no cliente e podem recuperar essas informações quando o cliente fizer o próximo pedido. Muitas aplicações *web* usam *cookies* para configurar uma sessão entre o cliente e o servidor. Eles também usam *cookies* para lembrar as preferências de identidade e usuário entre os pedidos.

Esses valores de cookie são transmitidos para frente e para trás entre o cliente eo servidor através de *cookies* presentes no cabeçalho.

A anotação `@CookieParam` permite injetar *cookies* enviados por um pedido de cliente para seus métodos de recurso JAX-RS.

```
@Path("/myservice")
public class MyService {
    @GET
    @Produces("text/html")
    public String get(@CookieParam("customerId") int custId) {
        ...
    }
}
```

2.6.7. `@BeanParam`

A anotação `@BeanParam` é algo novo adicionado na especificação JAX-RS 2.0. Permite injetar uma classe específica da aplicação cujos métodos ou campos são anotados com qualquer um dos parâmetros de injeção discutidos acima.

```
public class MyBeanParam {
    @PathParam("p")
    private String pathParam;

    @MatrixParam("m")
    private String matrixParam;

    @HeaderParam("header")
    private String headerParam;

    private String queryParams;

    public MyBeanParam(@QueryParam("q") String queryParams) {
        this.queryParams = queryParams;
    }

    public String getPathParam() {
        return pathParam;
    }
    ...
}

@POST
public void post(@BeanParam MyBeanParam beanParam, String entity) {
    final String pathParam = beanParam.getPathParam(); // contains
    injected path parameter "p"
    ...
}
```

3. INTRODUÇÃO A LINGUAGEM PHP

O acrônimo PHP, significa *Hypertext Preprocessor* é uma linguagem de programação interpretada cujo o principal objetivo é permitir a escrita de páginas *web* dinâmicas, no entanto é possível explorar outras possibilidades com PHP.

A linguagem PHP foi concebida durante o outono de 1994 por Rasmus Lerdorf. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua *home-page* apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas. A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como “*Personal Home Page Tools*” (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas *macros* e alguns utilitários que rodavam no *back-end* das *home-pages*: um livro de visitas, um contador e algumas outras coisas.

Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de PHP/FI, o “FI” veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (*Form Interpreter*). Ele combinou os *scripts* do pacote *Personal Home Page Tools* com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.

Estima-se que em 1996 PHP/FI estava sendo usado por cerca de 15.000 *sites* pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP. Ele deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por Zeev Suraski e Andi Gutmans, e esse novo interpretador foi a base para a versão 3.

3.1. POR QUE ESCOLHER PHP

- PHP nasceu para a *web* e sua integração com servidores web é simples.
- PHP tem uma curva de aprendizado suave, comparada a outras linguagens.
- PHP é uma tecnologia livre.
- É fácil encontrar serviços de hospedagem que oferecem PHP.
- Serviços de hospedagem PHP é mais barato que serviços semelhantes para outras tecnologias.

3.2. INSTALAÇÃO

Uma das melhores opções é a utilização do XAMPP que além de gratuito disponibiliza tudo que é preciso para desenvolver em PHP.

O *download* do XAMP pode ser feito [aqui](#).

Será considerado a utilização do sistema operacional Windows, porém, é possível utilizar o XAMPP em outras plataformas.

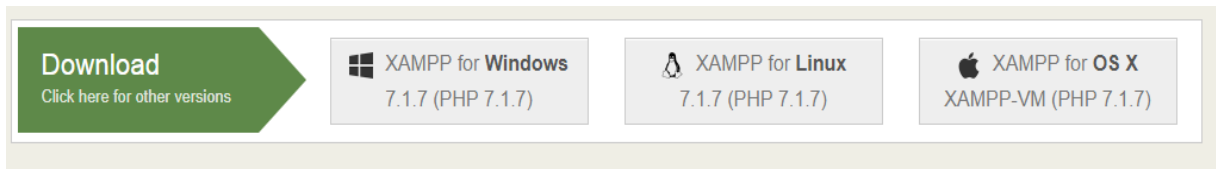


Figura 1 - Download da ferramenta XAMPP

3.2.1. Instalando PHP no Linux (Sem interface gráfica)

Com o comando (*sudo aptitude install php5 php5-mysql apache2 libapache2-mod-php5 mysql-server*) é possível baixar e instalar o PHP e suas dependências, esse comando é referente a distribuição Ubuntu/Debian.

Caso seja utilizado outra distribuição consulte a disponibilidade dos pacotes PHP5, MySQL e Apache para esta distribuição.

Após o *download* do instalador execute-o e siga o processo padrão de instalação sempre pressionando *NEXT*.

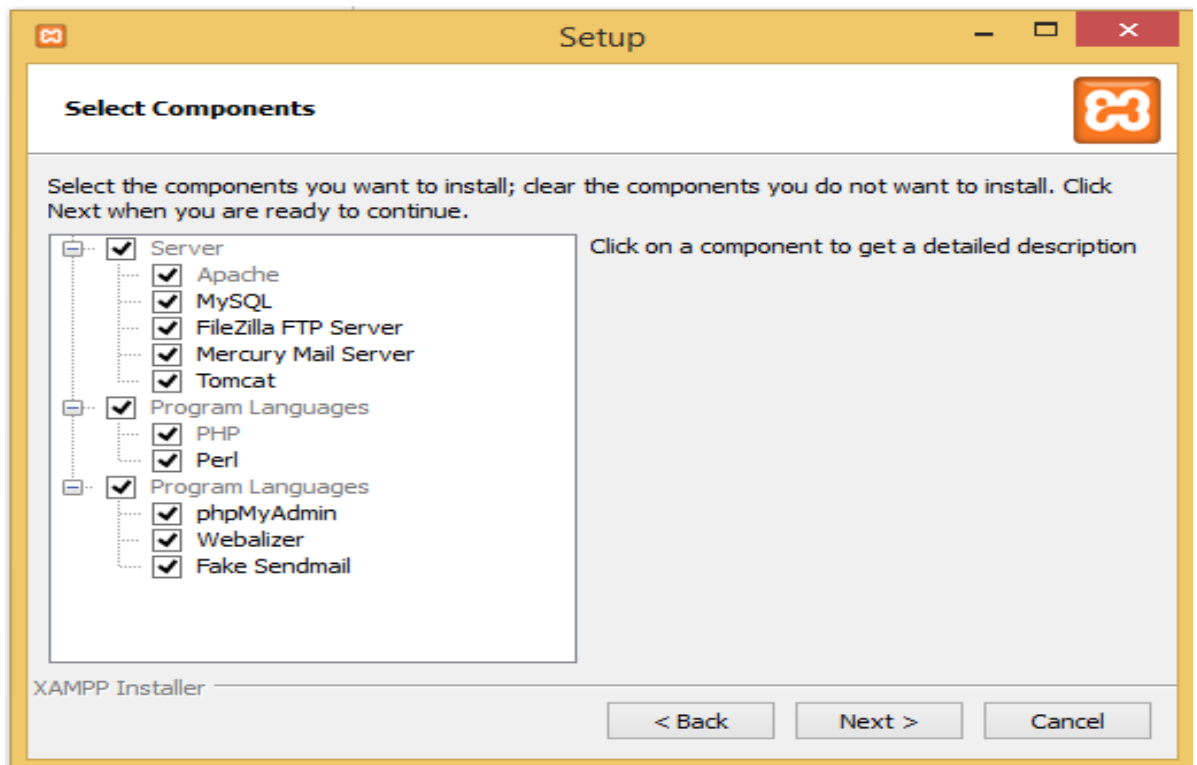


Figura 2 - Instalação do XAMPP

Após a finalização da instalação será apresentado a seguinte caixa de diálogo

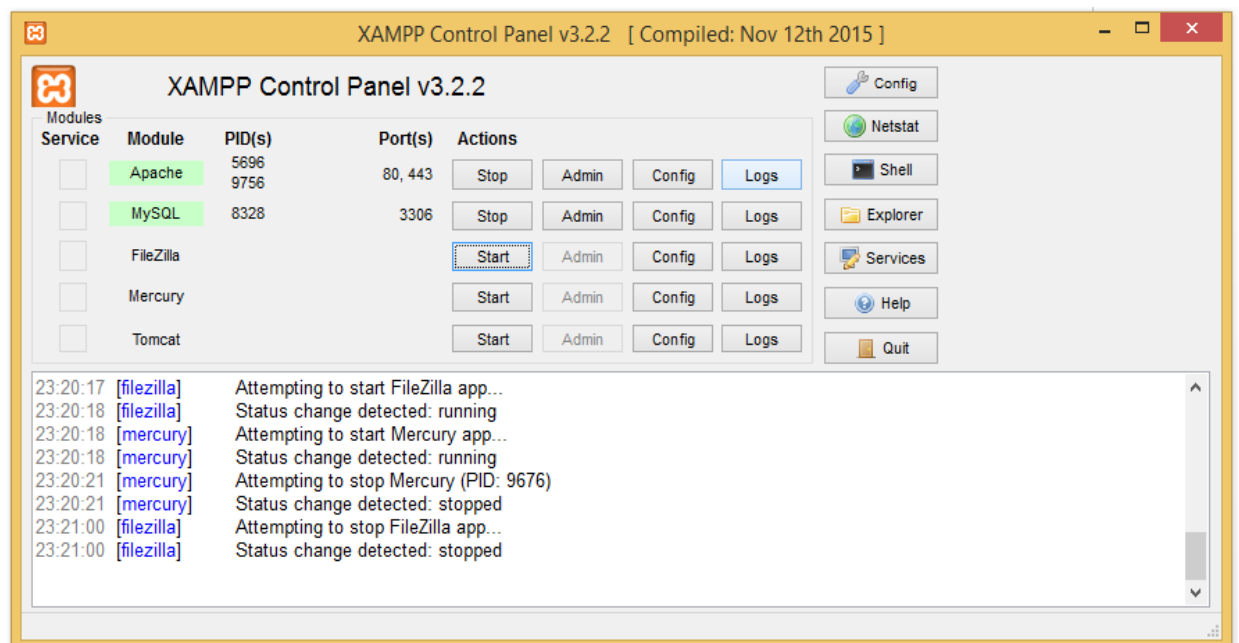


Figura 3 – Painel de controle do XAMPP

Basicamente não será necessário subir todos os serviços, por agora apenas o Apache e MySQL são suficientes.

Para testar se a instalação está correta acesse <http://localhost/xampp/>.

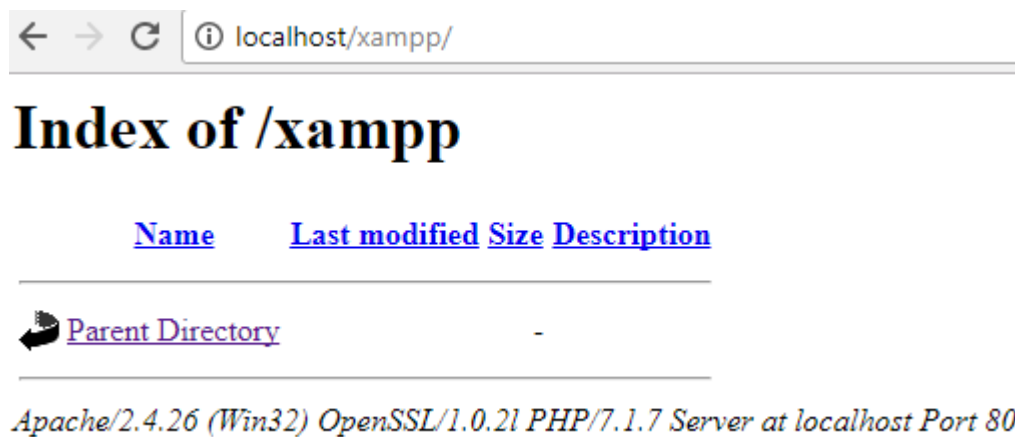


Figura 4 - Teste da instalação do XAMPP

3.3. SINTAXE BÁSICA

É possível delimitar código PHP de quatro formas:

Forma 1:

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

Forma 2:

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

Forma 3:

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

Forma 4:

```
<% Commandos; %>
```

Prática 1 – Criando meu primeiro programa em PHP

Crie um arquivo chamado primeiroPrograma.php com o seguinte conteúdo:

```
<html>
  <head>
    <title>Minha Página </title>
  </head>
  <body>
    <h1>
      <?php echo "PHP"; ?>
    </h1>
```

```
</body>
</html>
```

Adicione o arquivo primeiroPrograma.php dentro do diretório httpd do Xampp, no meu caso o caminho desse diretório é: **C:\xampp\htdocs**

Teste o *script* no navegador acessando <http://localhost/primeiroPrograma.php>.

3.3.1. Variáveis

As variáveis em PHP são declaradas com \$ por exemplo:

```
<?php
    $a = 10;
    $b = "10";
    $c = 10.2;

    echo "Número Inteiro:" . $a . "<br>" . "String:" . $b . "<br>"
    . "Ponto Flutuante:" . " " . $c;
?>
```

3.3.2. Comentários

3.3.2.1. Comentário de uma linha

Pode ser delimitado pelo caracter “#” ou por duas barras (//).Exemplo:

```
<? echo "teste"; #isto é um teste ?>
<? echo "teste"; //este teste é similar ao anterior ?>
```

3.3.2.2. Comentários de bloco

Tem como delimitadores os caracteres “/*” para o início do bloco e “*/” para o final do comentário. Exemplo

```
<?
    echo "teste"; /* Isto é um comentário com mais de uma linha
    que funciona corretamente
    */
?>
```

3.3.3. Operadores

3.3.3.1. Aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão

3.3.3.2. Comparação

==	Igual a
!=	Diferente de
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a

3.3.3.3. Lógicos

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

3.3.3.4. Para Strings

.	Concatenação
---	--------------

3.3.3.5. Atribuição

=	Atribuição simples
-=	Atribuição com subtração
+=	Atribuição com adição
*=	Atribuição com multiplicação
/=	Atribuição com multiplicação
%=	Atribuição com módulo
.=	Atribuição com concatenação

3.3.3.6. Incremento e decremento

++	Incremento
--	Decremento

3.3.4. Estruturas de Controle

3.3.4.1. *if/else*

Exemplo

```
<?php
    if ($a > $b) {
        echo "a is greater than b";
    } else {
        echo "a is NOT greater than b";
    }
?>
```

3.3.4.2. *switch*

Exemplo

```
<?php

    switch ($i) {
        case 0:
            echo "i equals 0";
            break;
        case 1:
            echo "i equals 1";
            break;
        case 2:
            echo "i equals 2";
            break;
    }
?>
```

3.3.5. Estruturas de Repetição

3.3.5.1. *While*

Exemplo

```
<?php
/* example 1 */

    $i = 1;
    while ($i <= 10) {
        echo $i++;
    }
?>
```

3.3.5.2. *do-while*

Exemplo

```
<?php
    $i = 0;
    do {
        echo $i;
```

```

    } while ($i > 0);
?>

```

3.3.5.3. *for*

```

<?php
    for ($i = 1; $i <= 10; $i++) {
        echo $i;
    }
?>

```

3.3.5.4. *foreach*

O construtor *foreach* fornece uma maneira fácil de iterar sobre *arrays*. O *foreach* funciona somente em *arrays* e objetos, e emitirá um erro ao tentar usá-lo em uma variável com um tipo de dado diferente ou em uma variável não inicializada. Exemplo

```

<?php
    $arr = array(1, 2, 3, 4);
    foreach ($arr as &$value) {
        $value = $value * 2;
    }
    // $arr is now array(2, 4, 6, 8)
    unset($value); // break the reference with the last element
?>

```

3.3.6. Quebra de fluxo

3.3.6.1. Break

O comando *break* pode ser utilizado em laços de *do*, *for* e *while*, além do uso já visto no comando *switch*. Ao encontrar um *break* dentro de um desses laços, o interpretador PHP para imediatamente a execução do laço, seguindo normalmente o fluxo do script. Exemplo:

```

while ($x > 0) {
    ...
    if ($x == 20) {
        echo "erro! x = 20";
        break;
    }
    ...
}

```

3.3.6.2. Continue

O comando `continue` também deve ser utilizado no interior de laços, e funciona de maneira semelhante ao `break`, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Exemplo:

```
for ($i = 0; $i < 100; $i++) {  
    if ($i % 2) continue;  
    echo " $i ";  
}
```

3.3.7. Array

Exemplo

```
<?php  
$array[0] = "valor1";  
$array[1] = "valor2";  
$array[2] = "valor3";  
print_r($array);  
echo "<br>";  
echo "Array na posição 1:" . $array[1];  
?>
```

Outras maneiras de inicializar um array.

```
<?php  
$array = array(0=>"valor1", 1=>"valor2", 2=>"valor3");  
print_r($array);  
echo "<br>";  
echo "Array na posição 1:" . $array[1];  
?>
```

Utilizando a função `array()`

```
<?php  
$array = array();  
print_r($array);  
?>
```

Prática 2 – Exercícios sobre a sintaxe básica PHP

1. Escreva um *script* em PHP que apresente os valores em uma tabela. Use a *tag* `<table>` do HTML para apresentar os valores.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

2. Escreva um *script* em PHP que apresente o número de dias no mês corrente. Use a função `date()` do PHP para auxiliar.
3. Escreva um *script* em PHP que calcule a quantidade de semanas entre duas datas: 07/06/2017 e 28/10/2017.
4. Escreva um *script* em PHP que construa o seguinte padrão. Use a *tag* `
` do HTML se precisar de quebra de linha.

```

*
* *
* * *
* * * *
* * * * *
* * * * *
* * * * *
* * * *
* * *
* *
*

```

5. Escreva um *script* em PHP que apresente todos os números entre 300 e 450 que são divisíveis por 4.

3.3.8. Funções

Uma função em PHP pode ser definida e chamada da seguinte forma:

```

<?php

echo minhaFuncao("argumento1", "argumento2", "argumentoN");

function minhaFuncao($argumento1, $argumento2, $argumentoN){
    return "minha função - ".$argumento1.", ".$argumento2.", ".
$argumentoN;
}

?>

```

3.4. RECEBENDO DADOS DO FORMULÁRIO

Em toda aplicação *web* é muito comum os usuários enviarem dados ao servidor.

A maneira mais comum de receber dados do usuário é utilizando formulários. Abaixo é apresentado um exemplo de um formulário em HTML.

Prática 3 – Recebendo dados do formulário

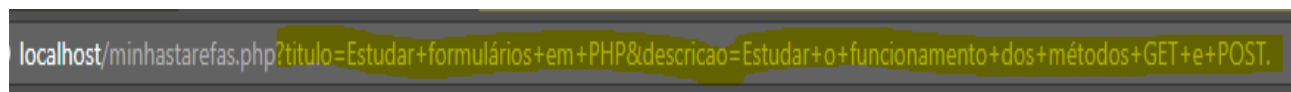
Primeiramente dentro do diretório htdocs da instalação do seu XAMPP crie um arquivo chamado minhastarefas.php com o seguinte conteúdo:

```
<html>
  <head>
    <title>Minhas Tarefas</title>
  </head>
  <body>
    <h1>Minhas Tarefas</h1>
    <form>
      <fieldset><legend>Nova tarefa</legend>
      <label>Qual minha tarefa?<br><input type="text"
name="titulo"/></label><br><br>
      <label>O que devo fazer?<br><textarea name="descricao"
cols="40" rows="10"></textarea></label><br><br>
      <input type="submit" value="Adicionar" /></fieldset>
    </form>
  </body>
</html>
```

Analisando o arquivo minhastarefas.php chegamos à conclusão que apenas criamos um arquivo PHP cujo o conteúdo é um HTML que reflete um formulário *web*. Bom, ainda neste formulário realizaremos o seguinte experimento:

1 – Preencha o formulário

2 – Pressione o botão “Adicionar” e observe a URL.



Observe que a URL mudou e a informação sumiu do formulário, ao pressionarmos o botão “Adicionar”, nós realizamos o *submit* dos dados do formulário para o servidor, neste caso os dados foram enviados para a mesma página minhastarefas.php.

Observa-se que foi adicionado à URL “?titulo=Estudar+formulários+em+PHP&descricao=Estudar+o+funcionamento+dos+métodos+GET+e+POST.”.

É possível perceber que tudo que informamos ao formulário foi submetido. Para esse formato aberto de dados submetidos e adicionados à URL nós dizemos que os dados foram enviados ao servidor usando o método GET.

Baseado na URL podemos tirar as seguintes conclusões:

? – Indica o início do que foi enviado

titulo=Estudar+formulários+em+PHP Indica que o parâmetro título possui o conteúdo Estudar+formulários+em+PHP

& - Indica a separação entre os parâmetros

descricao=Estudar+o+funcionamento+dos+métodos+GET+e+POST. Indica que o parâmetro descrição possui o conteúdo Estudar+formulários+em+PHP.

Como o PHP irá conseguir pegar os dados do formulário?

Simple, já sabemos que o método de envio utilizado foi o GET e que temos que esperar dois parâmetros: título e descrição.

No PHP existem algumas variáveis chamadas de *super globais*, iremos tratar desse assunto mais tarde, por agora, o que precisamos saber que é possível utilizar a variável *super global* \$_GET para pegar os dados da URL corrente. A *super global* \$_GET é um array que armazena os parâmetros da URL.

No arquivo minharefas.php adicione o seguinte conteúdo após a tag </form>

```
<?php
$title = isset($_GET["titulo"]) ? $_GET["titulo"] : "";
$descricao = isset($_GET["descricao"]) ? $_GET["descricao"] : "";
echo "Minha Tarefa é:" . $title . "<br>" . "O que devo fazer é:" .
$descricao;
?>
```

Após submeter o formulário preenchido teremos o seguinte resultado:

Minhas Tarefas



Minha Tarefa é: Estudar formulários em PHP

O que devo fazer é: Estudar o funcionamento dos métodos GET e POST.

Figura 5 - Resultado da submissão do formulário com o método GET

Bom, agora já sabemos como capturar dados da URL em PHP.

Ficou perceptível que o método GET adiciona itens à URL, mas o que devemos fazer em casos onde esses dados não podem ser recebidos via URL? Uma opção é utilizar outra variável *super global* `$_POST`. A *super global* `$_POST` é um *array* que armazena os dados recebidos no corpo de uma requisição.

Para enviar os dados do formulário através do método POST precisamos informar a *tag* `<form>` HTML que esses dados devem ser enviados seguindo este formato.

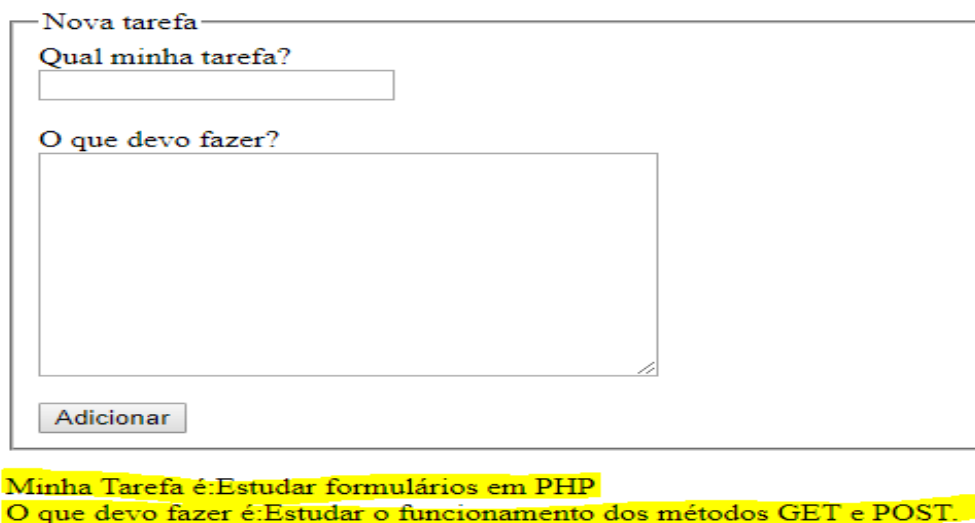
```
<html>
  <head><title>Minhas Tarefas</title></head>
  <body>
    <h1>Minhas Tarefas</h1>
    <form method="POST">
      <fieldset><legend>Nova tarefa</legend>
      <label>Qual minha tarefa?<br><input type="text"
name="titulo"/></label><br><br>
      <label>O que devo fazer?<br><textarea name="descricao"
cols="40" rows="10"></textarea></label><br><br>
      <input type="submit" value="Adicionar" /></fieldset>
    </form>
  </body>
</html>
```

No arquivo `minhastarefas.php` altere o trecho do código para utilizar a super global `$_POST`

```
<?php
$title = isset($_POST["titulo"]) ? $_POST["titulo"] : "";
$descricao = isset($_POST["descricao"]) ? $_POST["descricao"] : "";
echo "Minha Tarefa é:" . $title . "<br>" . "O que devo fazer é:" .
$descricao;
?>
```

Após submeter o formulário preenchido teremos o seguinte resultado:

Minhas Tarefas



Nova tarefa

Qual minha tarefa?

O que devo fazer?

Adicionar

Minha Tarefa é: Estudar formulários em PHP

O que devo fazer é: Estudar o funcionamento dos métodos GET e POST.

Figura 6 - Resultado da submissão do formulário com o método POST

Observe que na URL diferentemente do GET, não foram adicionados os dados do formulário, pois quando utiliza-se o método POST o que é utilizado para enviar os dados para o servidor é a URI (*Uniform Resource Identifier*) e esta não é retornada para o cliente.

Em resumo quando precisamos enviar poucos dados, parâmetros não confidenciais (senhas) o método GET pode ser utilizado e quando deseja-se passar parâmetros confidenciais e uma quantidade maior de parâmetros o método POST é mais recomendado.

3.5. SESSÕES

Antes de entrar efetivamente no assunto sessões, vamos adicionar uma maneira de guardar as tarefas cadastradas pelo usuário. A estrutura que utilizaremos para guardar essas informações será um *array*.

Altere o arquivo `minhastarefas.php` conforme exemplo abaixo:

```

<html>
  <head><title>Minhas Tarefas</title></head>
  <body>
    <h1>Minhas Tarefas</h1>
    <form method="POST">
      <fieldset><legend>Nova tarefa</legend>
      <label>Qual minha tarefa?<br><input type="text"
name="titulo" /></label><br><br>
      <label>O que devo fazer?<br><textarea name="descricao"
cols="40" rows="10"></textarea></label><br><br>
      <input type="submit" value="Adicionar" /></fieldset>
    </form>
    <?php
      $tarefas = array();
      if (isset($_POST["titulo"])) {
        $tarefas[] = $_POST["titulo"];
      }
    ?>
    <table>
      <tr>
        <th>Minhas tarefas</th>
      <tr>
        <?php foreach($tarefas as $tarefa): ?>
      <tr>
        <td><?php echo $tarefa; ?></td>
      </tr>
      <?php endforeach; ?>
    </table>
  </body>
</html>

```

Basicamente alteramos todo o trecho abaixo da *tag* </form>, com esta alteração toda tarefa adicionada terá armazenada seu título em um array.

Minhas Tarefas

Nova tarefa

Qual minha tarefa?

O que devo fazer?

Adicionar

Minhas tarefas

Tarefa 1

Figura 7 - Guardando uma informação em um array

Perceba que se adicionarmos a “Tarefa 2” a “Tarefa 1” irá desaparecer.

Minhas Tarefas

Nova tarefa

Qual minha tarefa?

O que devo fazer?

Adicionar

Minhas tarefas

Tarefa 2

Figura 8 – Guardando uma informação em um array

Isso ocorre pelo fato que o PHP irá interpretar todo o *script* novamente a cada requisição. Uma das alternativas é utilizar outra variável *super global* responsável pelo armazenamento de informações do usuário, essa variável é a `$_SESSION`.

O primeiro passo para utilizar essa variável em nosso *script* é adicionar a seguinte chamada na primeira linha do arquivo `minhastarefas.php`:

```
<?php session_start();?>
<html>
```

O segundo passo será alterar o *script* anterior para que adicione o *array* tarefas na sessão.

```
<?php
$tarefas = array();

if (isset($_POST["titulo"])) {
    $_SESSION["tarefas"][] = $_POST["titulo"];
}

if (isset($_SESSION["tarefas"])) {
    $tarefas = $_SESSION["tarefas"];
}
?>
```

Observe que agora será possível adicionar múltiplas tarefas em nosso array.

Minhas Tarefas

Nova tarefa

Qual minha tarefa?

O que devo fazer?

Adicionar

Minhas tarefas

Tarefa 1

Tarefa 2

Figura 9 – Guardando múltiplas informações em um array

Problema do F5

Porém, perceba que se você atualizar a página com F5 a última tarefa será repetida, como resolver isso? Iremos adicionar uma solução para este problema logo em seguida.

3.6. HEADERS E VARIÁVEIS DE SISTEMAS NO PHP

Em nosso exemplo `minhastarefas.php` é perceptível que o código não está muito organizado, não está no escopo deste material abordarmos detalhes profundos sobre MVC, *frameworks* e boas práticas, no entanto, temos que ter um mínimo de organização em nosso código. Nesse primeiro momento vamos separar algumas responsabilidades.

Vamos criar um diretório dentro do `htdocs` chamado `minhastarefas`. Em nosso exemplo anterior nós implementamos tudo em um único *script*, a fim de melhorar isso separaremos as responsabilidades em quatro arquivos.

Crie dentro da pasta `minhastarefas` os seguintes arquivos: `cadastroTarefas.php`, `listaTarefas.php`, `minhastarefas.php`, `template.php`.

O arquivo `cadastroTarefas.php` será responsável por exibir o formulário, o conteúdo deste arquivo deve ser:

```
<form method="POST">
```

```

        <fieldset>
            <legend>Nova tarefa</legend>
            <label>Qual minha tarefa?<br><input type="text"
name="titulo" /></label><br><br>
            <label>O que devo fazer?<br><textarea name="descricao"
cols="40" rows="10"></textarea></label><br><br>
            <input type="submit" value="Adicionar" />
        </fieldset>
    </form>

```

O arquivo `listaTarefas.php` será responsável por exibir a tabela com as tarefas cadastradas, o conteúdo deste arquivo deve ser:

```

<table>
    <tr>
        <th>Minhas tarefas</th>
    </tr>
    <?php foreach($tarefas as $tarefa): ?>
        <tr>
            <td><?php echo $tarefa; ?></td>
        </tr>
    <?php endforeach; ?>
</table>

```

O arquivo `minhastarefas.php` será responsável por executar toda a lógica de validação e adição do nome da tarefa na sessão, o conteúdo deste arquivo deve ser:

```

<?php
    session_start();

    $tarefas = array();

    if (isset($_POST["titulo"]) && $_POST["titulo"] != "") {
        $_SESSION["tarefas"][] = $_POST["titulo"];
        header("Location: minhastarefas.php");
        die();
    }

    if (isset($_SESSION["tarefas"])) {
        $tarefas = $_SESSION["tarefas"];
    }

    include "template.php";
?>

```

Na seção anterior eu mencionei sobre o problema do F5, basicamente este problema adiciona o último registro na sessão sempre que a página é atualizada, o efeito disso é que o último registro ficava duplicado, para corrigir esse problema foi adicionado as seguintes linhas:

```

header("Location: minhastarefas.php");
die();

```

A função `header("Location: minhastarefas.php")` redirecionará o usuário de volta para “minhastarefas.php” e a função `die()` interrompe a execução imediatamente.

O arquivo `template.php` será responsável pela base da página *web*, ou seja, conterá o HTML base para outras páginas, o conteúdo deste arquivo deve ser:

```
<html>
  <head><title>Minhas Tarefas</title></head>
  <body>
    <h1>Minhas Tarefas</h1>
    <?php include "cadastroTarefas.php"?>
    <?php include "listaTarefas.php"?>
  </body>
  <footer>Minhas Tarefas</footer>
</html>
```

É possível adicionar mais melhorias em nosso projeto, porém, com esta separação nós garantimos um mínimo de organização e simplicidade para adicionar outras funcionalidades.

Em PHP tem-se a oportunidade de utilizar diversas variáveis que possibilitam por exemplo manipular requisições, sessões, entre outras. Abaixo segue a lista de variáveis nativas do PHP, essas variáveis são chamadas de *super globais* e estão disponíveis em todos os escopos.

- [Superglobais](#) — Superglobais são variáveis nativas que estão sempre disponíveis em todos escopos
- [\\$GLOBALS](#) — Referencia todas variáveis disponíveis no escopo global
- [\\$_SERVER](#) — Informação do servidor e ambiente de execução
- [\\$_GET](#) — HTTP GET variáveis
- [\\$_POST](#) — HTTP POST variables
- [\\$_FILES](#) — HTTP File Upload variáveis
- [\\$_REQUEST](#) — Variáveis de requisição HTTP
- [\\$_SESSION](#) — Variáveis de sessão
- [\\$_ENV](#) — Environment variables
- [\\$_COOKIE](#) — HTTP Cookies
- [\\$php_errormsg](#) — A mensagem de erro anterior
- [\\$HTTP_RAW_POST_DATA](#) — Informação não-tratada do POST
- [\\$http_response_header](#) — Cabeçalhos de resposta HTTP
- [\\$argc](#) — O número de argumentos passados para o script
- [\\$argv](#) — Array de argumentos passados para o script

Figura 10 - Variáveis Super Globais PHP

Fonte: http://php.net/manual/pt_BR/reserved.variables.php

Prática 4 – Atividades formulários e sessão

1. No projeto minhas tarefas, existem apenas dois campos (título e descrição), adicione mais um campo ao formulário (campo tarefa concluída), esse campo deve ser um checkbox. O projeto deverá realizar uma validação que previna que o usuário adicione uma nova tarefa já com o *status* de concluída.
2. No projeto minhas tarefas, somente o campo (título) está sendo exibido quando o usuário adiciona uma nova tarefa, adicione na visualização os dois campos (descrição, tarefa concluída) para que estes sejam exibidos na tabela de exibição.

3.7. INTEGRAÇÃO ENTRE PHP E BANCO DE DADOS

Até o momento nós utilizamos apenas a memória do navegador para armazenar os dados de nossa aplicação de tarefas, através do uso de sessões.

Assim como a maioria das linguagens de programação PHP também suporta banco de dados.

O banco de dados a ser utilizado será o MySQL devido ao fato de ser um *software* livre e funcionar muito bem com PHP.

Com XAMPP Control Panel executando, inicie o servidor de banco de dados MySQL conforme imagem abaixo:

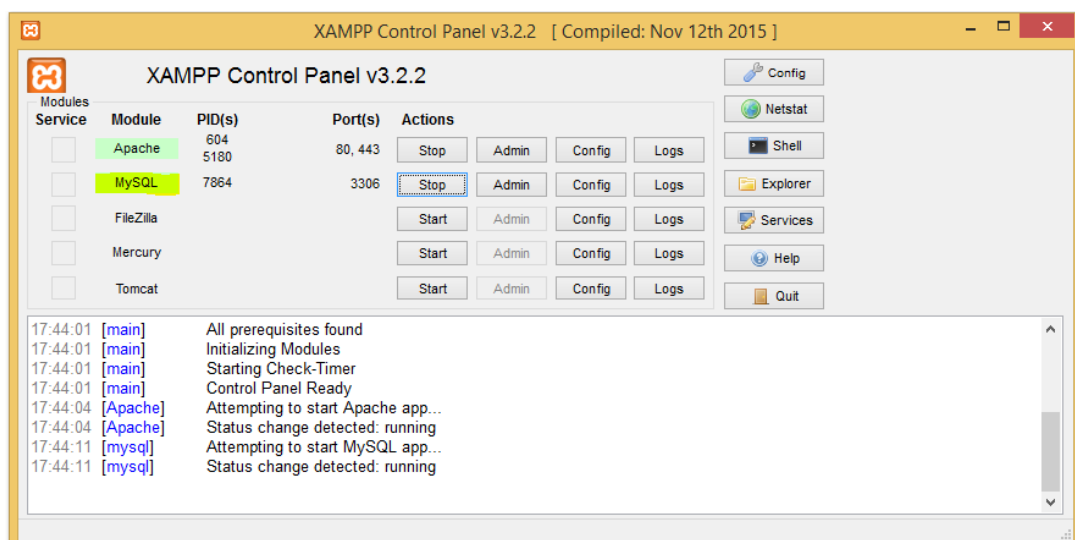


Figura 11 - Iniciando o serviço do MySQL

Para testar se o MySQL está acessível acesse: <http://localhost/phpmyadmin>.

Para esta instalação do XAMPP foi possível entrar diretamente no console de administração do phpMyAdmin. Caso não consiga entrar diretamente, tente o nome de usuário: root e a senha: root.

Abaixo é apresentado a tela principal do phpMyAdmin.



Figura 12 - Tela inicial do PHPMYAdmin

Não está no escopo deste documento detalhar profundamente os conceitos de banco de dados relacional e MySQL, por hora nós iremos criar um banco de dados chamado minhastarefas com uma tabela que irá se chamar tarefas.

Para criar o banco de dados e a tabela, no phpMyAdmin selecione a aba SQL e insira a seguinte DDL:

```
CREATE DATABASE IF NOT EXISTS `minhastarefas`;  
  
USE `minhastarefas`;  
  
CREATE TABLE IF NOT EXISTS `tarefas` (  
  `id` integer NOT NULL auto_increment,  
  `titulo` varchar(50) NOT NULL default '',  
  `descricao` text NOT NULL default '',  
  `concluida` boolean,  
  PRIMARY KEY (`id`)  
);  
  
CREATE USER 'systarefas'@'localhost' IDENTIFIED BY 'systarefas';  
GRANT ALL PRIVILEGES ON minhastarefas.* TO 'systarefas'@'localhost';
```

Pressione o botão Executar.

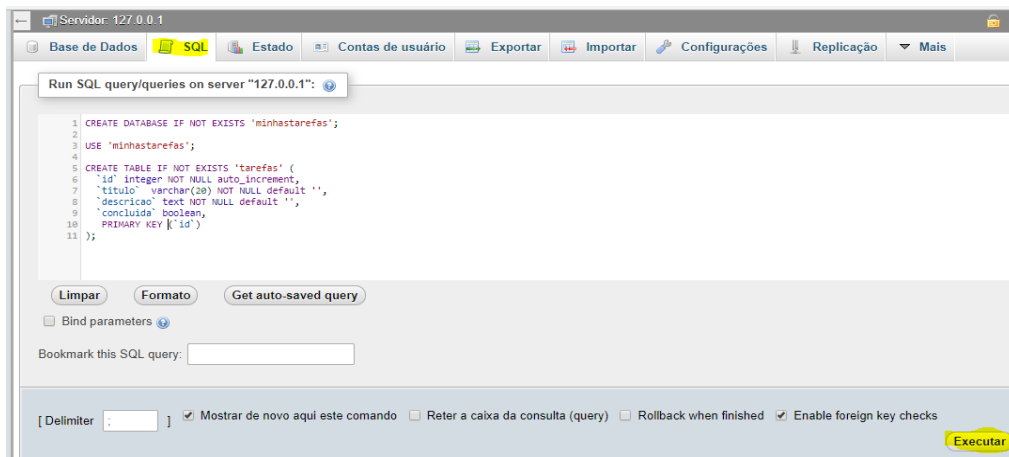


Figura 13 - Criando as tabelas no banco de dados MySQL

Para interação com banco de dados MYSQL a partir do PHP existe uma série de funções que começam com `mysql_` porém, existe uma versão atualizada dessa biblioteca que é a `MySQLi`.

Em nosso projeto iremos utilizar o PHP Data Objects (PDO) que é uma outra opção para trabalhar com banco de dados relacional.

Por que iremos utilizar PDO ao invés de `MySQLi`?

Pelo simples fato que o PDO suporta mais de uma base de dados enquanto que o `MySQLi` suporta apenas o MySQL, ou seja, ao utilizar PDO já estaremos prevenindo problemas de migração de banco de dados.

Ainda utilizando o projeto referência `minhastarefas`, iremos agora adicionar suporte ao banco de dados MySQL.

Primeiro criaremos dentro da pasta do projeto `minhastarefas` um arquivo chamado `conexao.php` e dentro deste arquivo adicionaremos detalhes de conexão com o banco de dados e algumas funções úteis para interagir com banco de dados.

Por padrão o XAMPP já habilita o PDO, caso não consiga utilizá-lo verifique se está instalado e habilitado corretamente, para detalhes de instalação do PDO você pode consultar [aqui](#).

O arquivo `conexao.php` deve possuir o seguinte conteúdo:

```

<?php
//Criando o objeto conexao
$user = "systarefas";
$pass = "systarefas";

```

```

        $db = null;
        try{
            $db = new PDO("mysql:host=localhost;dbname=minhastarefas",
$user, $pass);
            $db->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        } catch(PDOException $e) {
            print "Error ao conectar no banco de dados! " . $e-
>getMessage();
            die();
        }

        function cadastrarTarefa($db, $tarefas){
            $statementHandle = $db->prepare("INSERT INTO tarefas
(titulo,descricao,concluida) VALUES (?, ?, ?)");
            $statementHandle->bindParam(1, $tarefas["titulo"]);
            $statementHandle->bindParam(2, $tarefas["descricao"]);
            $statementHandle->bindParam(3, $tarefas["concluida"]);
            $statementHandle->execute();
        }

        function listarTodasTarefas($db) {
            $statementHandle = $db->prepare("SELECT * FROM tarefas");
            $statementHandle->execute();
            $result = $statementHandle->fetchAll(PDO::FETCH_ASSOC);
            return $result;
        }
?>

```

Inicialmente instanciaremos uma conexão com o banco de dados e posteriormente utilizaremos as funções `cadastrarTarefa` e `listarTodasTarefas`, com essas duas funções será possível persistir as informações no banco de dados e recuperá-las.

No trecho de código abaixo configuramos a URL de conexão com o banco de dados, em nosso caso é um banco local (localhost) e nosso esquema é o `minhastarefas`.

```

$db = new PDO("mysql:host=localhost;dbname=minhastarefas", $user,
$pass);

```

No trecho de código abaixo informamos que queremos ver os erros e exceções caso aconteçam.

```

$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

```

Na função `cadastrarTarefa` temos a seguinte listagem de código:

```

$statementHandle = $db->prepare("INSERT INTO tarefas
(titulo,descricao,concluida) VALUES (?, ?, ?)");
$statementHandle->bindParam(1, $tarefas["titulo"]);
$statementHandle->bindParam(2, $tarefas["descricao"]);
$statementHandle->bindParam(3, $tarefas["concluida"]);
$statementHandle->execute();

```

No trecho de código abaixo preparamos nosso Structured Query Language (SQL) para ser executado, observe que é uma simples clausula INSERT, a diferença é que podemos passar valores dinamicamente para este SQL via *placeholder* (?).

```
$statementHandle = $db->prepare("INSERT INTO tarefas
(titulo,descricao,concluida) VALUES (?, ?, ?)");
```

No trecho de código abaixo fazemos a ligação entre o parâmetros e valores que desejamos inserir no SQL.

```
$statementHandle->bindParam(1, $tarefas["titulo"]);
$statementHandle->bindParam(2, $tarefas["descricao"]);
$statementHandle->bindParam(3, $tarefas["concluida"]);
```

No trecho de código abaixo efetivamente executamos a SQL no banco de dados

```
$statementHandle->execute();
```

Na função `listarTodasTarefas` temos a seguinte listagem de código:

```
$statementHandle = $db->prepare("SELECT * FROM tarefas");
$statementHandle->execute();
$result = $statementHandle->fetchAll(PDO::FETCH_ASSOC);
return $result;
```

No trecho de código abaixo preparamos nosso SQL para ser executado, observe que é uma simples clausula SELECT.

```
$statementHandle = $db->prepare("SELECT * FROM tarefas");
```

No trecho de código abaixo estamos efetivamente executamos a nossa consulta

```
$statementHandle->execute();
```

No trecho de código abaixo informamos o tipo de estratégia para recuperar os dados.

```
$result = $statementHandle->fetchAll(PDO::FETCH_ASSOC);
```

O `fetchAll` retorna uma matriz com todas as linhas do resultado, existem outras alternativas como o `fetch` que obtém a próxima linha de um conjunto de resultados.

Basicamente a configuração `PDO::FETCH_ASSOC` retornará um *array* indexado pelo nome da coluna, quando é escolhida essa opção os resultados da consulta são retornados da seguinte forma:

```
Array (
    [0] => Array
        (
            [id] => 28
            [titulo] => Estudar protocolo HTTP
            [descricao] => Estudar status codes
            [concluida] => 0
        )
    [1] => Array
        (
            [id] => 29
            [titulo] => Estudar PHP
            [descricao] => Estudar PDO
            [concluida] => 0
        )
)
```

Para o nosso contexto essa estratégia funciona bem, porém o PDO permite outros modos de result set:

`PDO::FETCH_COLUMN`

`PDO::FETCH_ASSOC`

`PDO::FETCH_BOTH`

`PDO::FETCH_KEY_PAIR`

`PDO::FETCH_GROUP`

`PDO::FETCH_CLASS`

Você pode encontrar mais detalhes [aqui](#).

Nossa próxima alteração será no script `minhastarefas.php` alteraremos esse *script* para que ao invés de guardar e utilizar os dados da sessão salvar e recuperar os dados do banco de dados. O conteúdo desse arquivo deverá ser assim:

```
<?php
    session_start();
    include "conexao.php";

    $tarefas = array();

    if (isset($_POST["titulo"]) && $_POST["titulo"] != "") {

        $tarefas["titulo"] = $_POST["titulo"];
        $tarefas["descricao"] = $_POST["descricao"];
        $tarefas["concluida"] = "0";

        cadastrarTarefa($db, $tarefas);

        header("Location: minhastarefas.php");
        die();
    }

    $tarefas = listarTodasTarefas($db);

    include "template.php";
?>
```

Basicamente adicionamos o arquivo `include "conexao.php";`

Esse arquivo contém o necessário para que nossa aplicação interaja com banco de dados.

A próxima alteração será no arquivo `listaTarefas.php` que deve possuir o seguinte conteúdo:

```
<table>
    <tr>
        <th>Minhas tarefas</th>
    </tr>
    <?php foreach($tarefas as $tarefa): ?>
        <tr>
            <td><?php echo $tarefa["titulo"]; ?></td>
            <td><?php echo $tarefa["descricao"]; ?></td>
            <td><?php echo $tarefa["concluida"]; ?></td>
        </tr>
    <?php endforeach; ?>
</table>
```

A variável `$tarefas` é uma matriz onde cada posição é um *array* indexado pelo nome da coluna do banco de dados, basicamente nesse *script* existe uma estrutura de repetição que percorre cada posição de um *array* indexado com o nome da coluna.

Acessando a aplicação minhastarefas o resultado deve ser parecido com:

Minhas Tarefas

Nova tarefa

Qual minha tarefa?

O que devo fazer?

Adicionar

Minhas tarefas

Estudar protocolo HTTP Estudar status codes 0

Estudar PHP Estudar PDO 0

Minhas Tarefas

Prática 5 – Atividades banco de dados

1. Adicione uma funcionalidade que permita atualizar uma tarefa existente. Utilize a clausula UPDATE do SQL.

2. Adicione uma funcionalidade que permita apagar uma tarefa existente. Utilize a clausula DELETE do SQL.

3. Adicione uma funcionalidade para exibir as tarefas não concluídas. Na página *web* deve ser adicionado um campo do tipo check box que se selecionado deve buscar apenas as tarefas que não estão concluídas. Lembrando que em nossa base de dados as tarefas concluídas iremos considerar valor 1 e tarefas não concluídas iremos considerar valor 0.

3.8. INTRODUÇÃO A ORIENTAÇÃO A OBJETOS EM PHP

3.8.1. Classe e objetos

Classe é a estrutura mais fundamental para a criação de um objeto. Uma classe nada mais é do que um conjunto de variáveis (propriedades ou atributos) e funções (métodos), que definem o estado e o comportamento do objeto. Quando criamos uma classe, temos como objetivo final a criação de objetos, que nada mais são do que representações dessa

classe em uma variável. Para declararmos uma classe, utilizamos a palavra-chave *class*.

Exemplo:

```
<?php
# noticia.php
class Noticia
{
    public $titulo;
    public $texto;

    function setTitulo($valor)
    {
        $this->titulo = $valor;
    }
    function setTexto($valor)
    {
        $this->texto = $valor;
    }
    function exibeNoticia()
    {
        echo "<center>";
        echo "<b>". $this->titulo . "</b><p>";
        echo $this->texto;
        echo "</center><p>";
    }
}
$not = new Noticia;
$not->titulo = 'DM107';
$not->texto = 'Desenvolvimento de Web Services com segurança sob plataforma
Java e PHP ';
$not->exibeNoticia();
?>
```

Observação: como os atributos são do tipo *public*, podemos atribuir valores diretamente para eles, sem a necessidade de utilizar os métodos. Para manipularmos variáveis na classe, precisamos usar a variável *\$this*, funções e o separador *->*. A classe deve utilizar a variável *\$this* para referenciar seus próprios métodos e atributos.

3.8.2. Herança

Herança é uma forma de reutilização de código em que novas classes são criadas a partir de classes existentes, absorvendo seus atributos e comportamentos, e complementando-os com novas necessidades. Exemplo:

```
<?php
# noticia_heranca.php
include_once('noticia.php');

class NoticiaPrincipal extends Noticia
{
    public $subtitulo;

    function setSubtitulo($valor)
    {
```

```

        $this-> subtítulo = $valor;
    }

    function exibeNotícia()
    {
        echo "<center>";
        echo "<p>". $this->título . "</p>";
        echo $this->texto;
        echo "<p>". $this-> subtítulo . "</p>";
        echo "</center>";
    }
}

$not = new NotíciaPrincipal;
$not->subtítulo = 'Introdução ao PHP';
$not->exibeNotícia();
?>

```

Como mostra o exemplo, a classe `NotíciaPrincipal` herdou todas as características da classe `Notícia`, e ainda foi adicionado o método que dá suporte à exibição de subtítulos nas notícias principais. Nessas sub-classes é possível redefinir métodos, podendo modificá-los da forma que o programador quiser, como foi o caso do método `exibeNotícia()`. Sobrescrever métodos é algo bastante comum no processo de herança, visto que os métodos que foram criados na classe “pai” não precisam ser necessariamente os mesmos que os definidos nas classes “filhas”.

3.8.3. Construtor

É um método que utiliza o nome reservado `__construct()` e que não precisa ser chamado da forma convencional, pois é executado automaticamente quando instanciamos um objeto a partir de uma classe. Sua utilização é indicada para rotinas de inicialização. O método construtor se encarrega de executar as ações de inicialização dos objetos, como por exemplo, atribuir valores a suas propriedades. Exemplo

```

<?php
# noticiaConstrutor.php
class Notícia
{
    public $título;
    public $texto;
    function __construct($valor_tit, $valor_txt)
    {
        $this->título = $valor_tit;
        $this->texto = $valor_txt;
    }
    function exibeNotícia()
    {
        echo "<center>";
        echo "<b>". $this->título . "</b><p>";
        echo $this->texto;
        echo "</center><p>";
    }
}

```

```

    }
    $not = new Noticia('DM107','Introdução ao PHP');
    $not->exibeNoticia();
?>

```

E como ficaria a classe filha `NoticiaPrincipal` com um método construtor?

```

<?php
# noticia_heranca.php
include_once('noticia.php');

class NoticiaPrincipal extends Noticia
{
    public $subtitulo;

    function __construct($valor_tit, $valor_txt, $valor_sub)
    {
        parent::__construct($valor_tit, $valor_txt);
        $this->subtitulo = $valor_sub;
    }

    function setSubtitulo($valor)
    {
        $this-> subtitulo = $valor;
    }

    function exibeNoticia()
    {
        echo "<center>";
        echo "<p>". $this->titulo . "</p>";
        echo $this->texto;
        echo "<p>". $this-> subtitulo . "</p>";
        echo "</center>";
    }
}

$not = new NoticiaPrincipal('DM107','Introdução ao PHP',
'Desenvolvimento' );
$not->exibeNoticia();
?>

```

O método construtor da classe `Noticia` é herdado e executado automaticamente na subclasse `NoticiaPrincipal`. Porém, as características específicas de `NoticiaPrincipal` não serão inicializadas pelo método construtor da classe pai. Outro detalhe importante: caso a subclasse `NoticiaPrincipal` tenha declarado um método construtor em sua estrutura, este mesmo método da classe `Noticia` não será herdado. Nesse caso podemos chamar o método construtor da Classe `Noticia`, através de uma chamada específica: `parent::__construct()`.

3.8.4. Destruitor

O método destrutor será chamado assim que todas as referências a um objeto em particular forem removidas ou quando o objeto for explicitamente destruído. O método `__destruct` é executado toda vez que um objeto da classe é destruído pelo fim do *script* ou

através da função *unset*. Sua função é basicamente zerar variáveis, limpar dados de sessões, fechar conexões com banco de dados, etc.

Como no método construtor, o método destrutor possui um nome reservado, o `__destruct()`.

3.8.5. Encapsulamento

Este recurso possibilita ao programador restringir ou liberar o acesso às propriedades e métodos das classes. A utilização deste recurso só se tornou possível a partir do PHP 5. Aplica-se este conceito através dos operadores:

Public: Quando definimos uma propriedade ou método como *public*, estamos dizendo que suas informações podem ser acessadas diretamente por qualquer *script*, a qualquer momento. Até este momento, todas as propriedades e métodos das classes que vimos foram definidas dessa forma.

Protected: Quando definimos em uma classe uma propriedade ou método do tipo *protected*, estamos definindo que ambos só poderão ser acessados pela própria classe ou por seus herdeiros, sendo impossível realizar o acesso externo.

Private: Quando definimos propriedades e métodos do tipo *private*, só a própria classe pode realizar o acesso, sendo ambos invisíveis para herdeiros ou para classes e programas externos.

3.8.6. Interface

Interfaces permitem a criação de código que especifica quais métodos uma classe deve implementar, sem ter que definir como esses métodos serão tratados. Interfaces são definidas utilizando a palavra-chave *interface*, e devem ter definições para todos os métodos listados na interface. Classes podem implementar mais de uma interface se desejarem, listando cada interface separada por um espaço.

Dizer que uma classe implementa uma *interface* e não definir todos os métodos na interface resultará em um erro fatal exibindo quais métodos não foram implementados. Vamos criar uma interface que servirá de base para as nossas classes de notícias utilizadas até agora:

```
<?php
# noticiaInterface.php
interface iNoticia
```

```

{
    public function setTitulo($valor);
    public function setTexto($valor);
    public function exibeNoticia();
}
?>

```

A implementação desta interface é simples. Vejamos o exemplo:

```

<?php
# noticiaImpl.php
include_once('noticiaInterface.php');
class Noticia implements iNoticia
{
    protected $titulo;
    protected $texto;
    public function setTitulo($valor)
    {
        $this->titulo = $valor;
    }
    public function setTexto($valor)
    {
        $this->texto = $valor;
    }
    public function exibeNoticia()
    {
        echo "<center>";
        echo "<b>". $this->titulo . "</b><p>";
        echo $this->texto;
        echo "</center><p>";
    }
}
$titulo = 'DM107';
$texto = 'Introdução ao PHP';
$not = new Noticia;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->exibeNoticia();
?>

```

3.8.7. Classe Abstrata

Classes abstratas são classes que não podem ser instanciadas diretamente, sendo necessária a criação de uma sub-classe para conseguir utilizar suas características. Isso não quer dizer que os métodos destas classes também precisem ser abstratos, isso é opcional. Já as propriedades não podem ser definidas como abstratas.

Aqui vemos o conceito de polimorfismo, ou seja, a possibilidade de dois ou mais objetos executarem a mesma ação. Um exemplo prático seria uma moto e um carro, os dois tem a ação em comum de Frear e Acelerar; em orientação a objetos usamos classes abstratas para dar funcionalidades iguais a objetos diferentes.


```

<?php
# noticiaAbstrata.php
abstract class Noticia
{
    protected $titulo;
    protected $texto;
    public function setTitulo($valor)
    {
        $this->titulo = $valor;
    }
    abstract public function setTexto($valor);
    abstract public function exhibeNoticia();
}
?>

```

O exemplo acima nos mostra a utilização tanto de métodos abstratos quanto de métodos comuns. Os métodos abstratos não devem conter código, apenas definição. Quando criamos um método abstrato, fazemos com que ele seja implementado em todas as classes que herdarem dessa classe abstrata.

No exemplo a seguir, os métodos abstratos serão definidos na sub-classe que herdará NoticiaAbstrata:

```

<?php
# noticiaAbstrata.php
include_once('noticiaAbstrata.php');
class NoticiaPrincipal2 extends Noticia
{
    private $subtitulo;

    public function setTexto($valor)
    {
        $this->texto = $valor;
    }
    function setsubtitulo($valor)
    {
        $this->subtitulo = $valor;
    }
    function exhibeNoticia()
    {
        echo "<center>";
        echo "<p>". $this->titulo . "</p>";
        echo $this->texto;
        echo "<p>". $this-> subtitulo . "</p>";
        echo "</center>";
    }
}
$titulo = 'DM107';
$texto = 'Introdução ao PHP';
$subtitulo = 'Desenvolvimento';
$not = new NoticiaPrincipal2;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->setsubtitulo($subtitulo);
$not->exibNoticia();
?>

```

Prática 6 – Atividades orientação a objetos

1. Implemente:

a) Uma classe Equipamento (equipamento.php) com o atributo ligado (tipo boolean) e com os métodos liga e desliga. O método liga torna o atributo ligado true e o método desliga torna o atributo ligado false.

b) Uma sub-classe EquipamentoSonoro (equipamento_sonoro.php) que herda as características de Equipamento e que possui os atributos volume que varia de 0 a 10 e stereo (do tipo boolean). A classe ainda deve possuir métodos para ler e alterar o volume (getter e setter), além dos métodos mono e stereo. O método mono torna o atributo stereo falso e o método stereo torna o atributo stereo verdadeiro. Ao ligar o EquipamentoSonoro através do método liga, seu volume é automaticamente ajustado para 5.

c) Um script PHP (testa_equipamento.php) que instancia 2 objetos da classe Equipamento e 2 objetos da classe EquipamentoSonoro. O *script* ainda deve inserir esses objetos em um array e, depois, listar todos os elementos do array.

2. Escreva:

a) Um *script* PHP (funcionario.php) contendo uma classe Funcionario com atributos privados nome e salario. A classe deverá ter um construtor que receba os atributos como parâmetros. Além disso, deve ter métodos para obter e alterar os atributos nome e salário (getNome, setNome, getSalario, setSalario). O salário jamais poderá ser negativo. Crie também um método __toString() para retornar uma string contendo todos os atributos.

b) Um *script* PHP (testa_funcionario.php) que instancia 3 objetos da classe Funcionario e insere-os em um array. A seguir, o script lista todos os objetos presentes no array.

c) Um *script* PHP (processa.php) que recebe dados (nome e salário) de um formulário (formulario.php), instancia um objeto Funcionario (funcionario.php), preenche o objeto com os dados vindos do formulário e, depois, exibe esse objeto em um outro *script* (mostra.php).

3.9. DESENVOLVIMENTO DE WEB SERVICE COM PHP

Nessa altura já temos um conhecimento básico em PHP. Com este conhecimento foi possível com PHP criar páginas *web* dinâmicas, ver alguns itens sobre o modelo cliente/servidor e consultar um banco de dados. Agora, imagine disponibilizar parte da aplicação *minhastarefas* como um serviço? Por exemplo, um aplicativo móvel interagindo com *minhastarefas* ou um *front-end* qualquer interagindo com a aplicação *minhastarefas*.

Em nossa aplicação utilizamos formulários HTML para envio de dados ao servidor. Bom, nesse cenário quando o servidor receber esses dados irá nos responder com um HTML, porém, se o intuito for disponibilizar a aplicação para outras plataformas, por exemplo Android, IOS esse cenário é inválido, pois retornar um HTML não é desejável nessas plataformas e é aqui que entra o Web Service.

3.9.1. Introdução ao *framework* Slim

Em PHP um dos *frameworks* para *Web Service* mais conhecidos é o [Slim](#).

Para adicionar suporte ao Slim *framework* em nosso ambiente de desenvolvimento, primeiramente vamos instalar o [Composer](#) que é um gerenciador de dependências para PHP.

- Criando a aplicação *minhastarefasapi*

Crie um diretório chamado *minhastarefasapi* dentro do *htdocs* na instalação do XAMPP

Crie um diretório chamado *src* dentro de *minhastarefasapi*

- Adicionando suporte ao Composer

Faça o *download* do seguinte arquivo <https://getcomposer.org/download/1.5.1/composer.phar> e copie para dentro da pasta *src*.

- Adicionando as dependências Slim e NotORM

Com *git bash* aberto dentro da pasta *src*, execute o seguinte comando para fazer o *download* do Slim.

```
"C:\xampp\php\php" composer.phar require slim/slim
```

Após o *download* do Slim, execute o comando para baixar as dependências do [NotORM](#), que é uma biblioteca para manipular dados do banco de dados.

```
"C:\xampp\php\php" composer.phar require vrana/notorm:dev-master
```

Observação: Se você instalou o XAMPP em outra plataforma (Linux ou IOS) o caminho do PHP CLI ("C:\xampp\php\php") pode mudar, desta forma você deve informar o caminho do PHP CLI correspondente a sua instalação.

Neste exemplo foi utilizado PHP via instalação do XAMPP em plataforma Windows, então temos que utilizar o PHP CLI incluso na instalação que neste caso está em: "C:\xampp\php\php"

- Criando e configurando o arquivo htaccess

Crie e configure um arquivo chamado .htaccess que deve ser adicionado dentro do diretório src/public e possuir o seguinte conteúdo:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f

RewriteRule ^ index.php [QSA,L]
```

- Criando e configurando o arquivo index.php

O arquivo index.php será onde adicionararemos o código da nossa API, nesse arquivo iremos efetivamente ver o Slim *framework* em ação.

Crie um arquivo chamado index.php que deverá ser adicionado dentro do diretório src/public e possuir o seguinte conteúdo:

```
<?php
    use \Psr\Http\Message\ServerRequestInterface as Request;
    use \Psr\Http\Message\ResponseInterface as Response;

    require '../vendor/autoload.php';

    $app = new \Slim\App;
    $app->get('/api/{nome}', function (Request $request, Response
$response) {
        $nome = $request->getAttribute('nome');
        $response->getBody()->write("Bem vindo a API, $nome");

        return $response;
    });

    $app->run();
?>
```

O trecho de código abaixo apenas traz as classes padrões de Request e Response que utilizaremos ao longo de nossa API, essas classes já abstraem muitas características do protocolo HTTP, o Slim *Framework* suporta [PSR-7](#) que é o padrão PHP para troca de mensagens baseadas no protocolo HTTP.

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;
```

O trecho de código abaixo inclui o *autoload* que permite consultar as dependências adicionadas via Composer, neste caso temos acesso as classes do Slim e também a outras dependências caso necessário.

```
require '../vendor/autoload.php';
```

O trecho de código abaixo inicializa o Slim.

```
$app = new \Slim\App;
```

O trecho de código abaixo configura uma rota para: */api/qualquercoisa*

```
$app->get('/api/{nome}', function (Request $request, Response
$response) {

    $nome = $request->getAttribute('nome');

    $response->getBody()->write("Bem vindo a API, $nome");

    return $response;

});
```

Basicamente ao receber uma requisição GET `$app->get('/api/{nome}')` por exemplo: */api/teste* o parâmetro “nome” cujo valor será “teste” é extraído do request

`$nome = $request->getAttribute('nome');` e após isso uma resposta será construída `$response->getBody()->write("Bem vindo a API, $nome");` e consequentemente utilizada como resposta a essa requisição.

O trecho de código abaixo informa ao Slim que a configuração está pronta e que a execução pode continuar.

```
$app->run();
```

Acessando o endereço <http://localhost/minhastarefasapi/src/public/api/teste> o resultado deve ser:

Bem vindo a API, teste

3.9.1.1. Adicionando suporte ao banco de dados via Slim

Em nosso projeto de API é possível configurar uma série de itens na inicialização do Slim.

No arquivo `index.php` adicione o seguinte trecho de código após a instrução `require`:

```

$config['displayErrorDetails'] = true;
$config['addContentLengthHeader'] = false;

$config['db']['host'] = "localhost";
$config['db']['user'] = "systarefas";
$config['db']['pass'] = "systarefas";
$config['db']['dbname'] = "minhastarefas";

```

Altere também o trecho de código responsável pela inicialização do Slim e adicione a dependência responsável pela conexão com banco de dados:

```

$app = new \Slim\App(["config" => $config]);

$container = $app->getContainer();

```

Neste passo adicione a dependência do banco de dados ao nosso projeto:

```

$container['db'] = function ($c) {
    $dbConfig = $c['config']['db'];
    $pdo = new PDO("mysql:host=" . $dbConfig['host'] . ";dbname=" .
$dbConfig['dbname'],
        $dbConfig['user'], $dbConfig['pass']);
    $pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,
PDO::FETCH_ASSOC);
    $db = new NotORM($pdo);
    return $db;
};

```

Utilizamos uma função anônima que configurará os parâmetros necessários para se conectar ao banco de dados.

Outra informação importante é que utilizamos o PDO novamente como nossa biblioteca padrão para a comunicação com banco de dados, porém, a novidade é que utilizaremos o PDO como *driver* para uma outra biblioteca, o [NotORM](#) que facilitará a manipulação dos dados no banco de dados.

Após essa configuração já será possível utilizar a dependência db apenas utilizando `$this->db`

3.9.2. Configurando Rotas no Slim

Conforme visto anteriormente neste documento na seção **Introdução ao framework Slim** iremos utilizar o Slim para criarmos nossa API RESTFull.

Uma característica importante é que o Slim trata de maneira diferente o “/” no final, por exemplo: **/produtos** é diferente de **/produtos/**

GET

É possível criar rotas para suportar o método GET, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->get('/produtos/{id}', function(Request $request, Response $response) {
    //Retorna o produto identificado pelo id
});
```

Neste trecho de código nós temos um código que toda vez que for recebido um padrão de URI do tipo **/produtos/1** irá chamar uma call-back que irá tratar essa requisição.

POST

É possível criar rotas para suportar o método POST, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->post('/produtos', function(Request $request, Response $response) {
    //Cria um novo produto
});
```

PUT

É possível criar rotas para suportar o método PUT, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
```

```
$app->put('/produtos/{id}', function(Request $request, Response $response) {
    //Atualiza o produto identificado pelo id
});
```

DELETE

É possível criar rotas para suportar o método DELETE, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->delete('/produtos/{id}', function(Request $request, Response $response) {
    //Remove o recurso identificado pelo id
});
```

OPTIONS

É possível criar rotas para suportar o método OPTIONS, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->options('/produtos/{id}', function(Request $request, Response $response) {
    //Retorna cabeçalhos de resposta para essa rota, por exemplo
    // os métodos que são permitidos para este recurso
});
```

PATCH

É possível criar rotas para suportar o método PATCH, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->patch('/produtos/{id}', function(Request $request, Response $response) {
    //Altera alguma coisa no produto identificado pelo id.
});
```


ANY

É possível criar rotas para suportar o método PATCH, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->any('/produtos/{id}', function(Request $request, Response $response) {
    //Faz alguma ação para os produtos ou produto identificado pelo
    //id. Nesse tipo de rota o método não é informado
    //explicitamente, porém é possível verificar através do objeto
    //$request qual método foi utilizada para a requisição
    //($request->getMethod()).
});
```

CUSTOM

É possível criar rotas customizadas, para suportar o método CUSTOM, esse tipo de rota pode ser construído da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App();
$app->map(['GET', 'POST'], '/produtos', function(Request $request, Response $response) {
    //Cria um novo produto ou lista todos os produtos.
});
```

Agrupando rotas

Em um cenário onde temos vários serviços é comum que dentre esses vários serviços existam lógicas comuns entre eles, ou seja, as rotas podem ser agrupadas. Esse tipo de agrupamento de rotas pode ser configurado da seguinte maneira:

```
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

$app->group('/produtos/{id}', function () {

    $this->map(['GET', 'PUT'], '', function(Request $request, Response $response) {

        //Recupera, apaga, atualiza um produto identificado pelo id

    })->setName('produto');
```

```

        $this->delete('/delete-produto', function Request $request,
        Response $response) {

            //Rota para /produtos/{id}/delete-produto

            //Apaga o produto identificado pelo id

        })->setName('produto-delete-produto');

    });

```

Manipulando respostas

No Slim é possível responder as requisições de duas maneiras:

1 – Apenas utilizando a função **echo()**. Neste caso o conteúdo do **echo()** será adicionada ao objeto **\Psr\Http\Message\ResponseInterface**.

2 – Retornar o objeto **\Psr\Http\Message\ResponseInterface**

A fim de permitir a customização das respostas, considere utilizar **\Psr\Http\Message\ResponseInterface** para construir suas respostas.

Introdução ao NotORM

[NotORM](#), é uma biblioteca PHP que simplifica o trabalho com banco de dados.

Configurando a conexão

```

$pdo = new PDO("mysql:host=host;dbname=dbname",
'usuario', 'senha');

$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);

$db = new NotORM($pdo);

```

Observando o código acima é possível observar que o NotORM utiliza o PDO como driver padrão de conexão com o banco de dados.

Pesquisando em uma tabela

O trecho de código abaixo busca todos os dados da tabela “tarefas”

```
foreach($db->tarefas() as $tarefa) {  
    echo "$tarefa[titulo]\n"; //imprime o dado da coluna título  
}
```

Inserindo dados em uma tabela

O trecho de código abaixo insere dados na tabela “tarefas” e retorna o dado inserido.

```
$tarefa = $db->tarefas()->insert(array(  
    "id" => 1, //Você pode omitir esse campo ou null caso auto incremento  
    "titulo" => "Estudar NotORM",  
    "descricao" => "Estudar operações SQL",  
    "concluida" => 0,  
));
```

Apagando dados em uma tabela

O trecho de código abaixo apaga uma tarefa cujo o id é igual a 1.

```
$tarefa = $db->tarefa()->where('id', 1);  
if ($tarefa->fetch()) {  
    $deleted = $tarefa->delete();  
}
```

Atualizando dados em uma tabela

```
$tarefa = $db->tarefa()->where('id', 2);  
$updated = null;  
if ($tarefa->fetch()) {  
    $tarefaUpdate=(array); //um array com os dados da tarefa  
    $updated = $tarefa->update($tarefaUpdate);  
}
```

3.9.3. Web service com segurança em PHP

O Slim framework nos oferece um mecanismo chamada de [Middleware](#). Esse mecanismo permite a execução de código depois do request recebido e antes do response, ou seja, é possível trabalhar como mecanismo de autenticação, cache entre outros.

Prática 7 – Adicionando middleware de autenticação ao Slim

1 – Crie um diretório no **htdocs** chamado **basicAuth**

2 - Dentro do diretório **src** do projeto, abra o gitBash e execute o seguinte comando:

```
"C:\xampp\php\php" composer.phar require tuupola/slim-basic-auth:3.0.0-rc.2
```

3 – Dentro do diretório **src**, crie um diretório chamado **public** e dentro deste diretório adicione o arquivo **index.php** com o seguinte conteúdo:

```
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require "../vendor/autoload.php";

$app = new \Slim\App;

$app->add(new Tuupola\Middleware\HttpBasicAuthentication([
    "users" => [
        "eu" => "eu"
    ]
]));

$app->get("/api/{nome}", function (Request $request, Response
$response) {
    $nome = $request->getAttribute("nome");
    $response->getBody()->write("Bem vindo a API Basic Auth,
$nome");
    return $response;
});

$app->run();
?>
```

4 - Crie e configure o arquivo **.htaccess**

Crie e configure um arquivo **.htaccess** que deve ser adicionado dentro do diretório **src/public** e possuir o seguinte conteúdo:

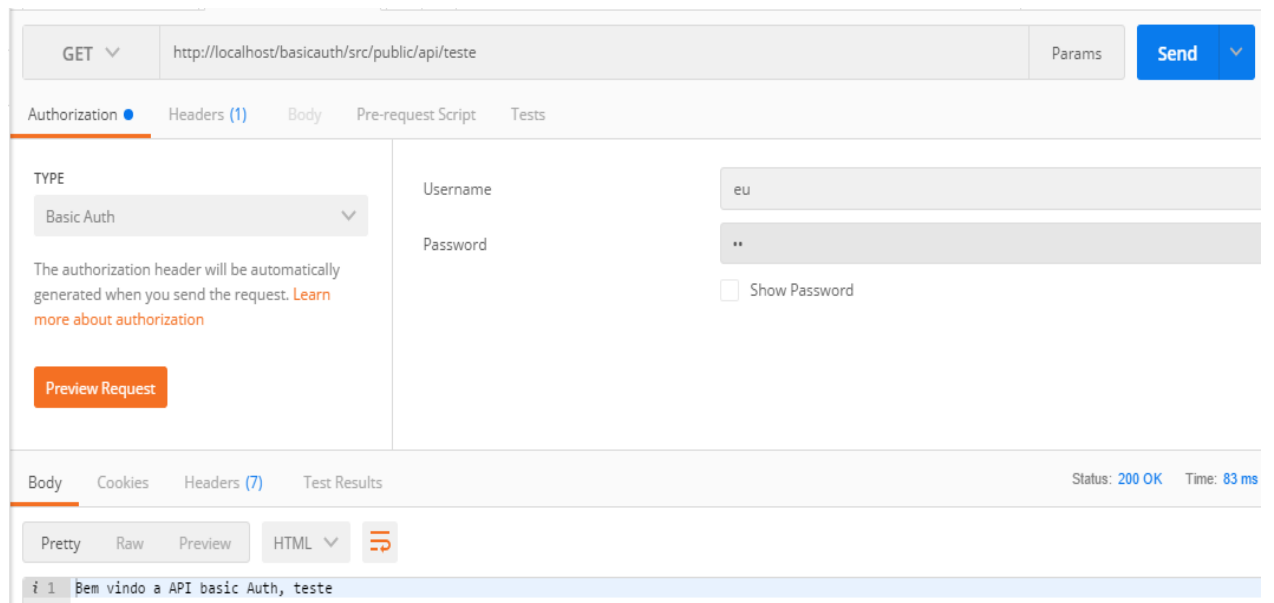
```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f

RewriteRule ^ index.php [QSA,L]
```

5 – A partir do Postman faça uma requisição para o seguinte endereço:

http://localhost/basicauth/src/public/api/teste



REFERÊNCIAS

- Lopez, Antonio. Learning PHP 7. Birmingham: Packt Publishing, 2016.
- NotORM. Disponível em < <http://www.notorm.com/>> Acesso em 01 Outubro 2017.
- PHP Manual. Disponível em < <http://php.net/manual/en/index.php>> Acesso em 17 Setembro 2017.
- Slim Framework. Disponível em < <https://www.slimframework.com/>>. Acesso em 30 Setembro 2017.
- PHP Hypertext Preprocessor. Disponível em <<https://secure.php.net/>>. Acesso em 15 Ago. 2017
- Souza, Jaime Freire. Avaliação do Protocolo HTTP 2.0.
- Disponível em < <http://www2.uesb.br/computacao/wp-content/uploads/2014/09/MONOGRAFIA-JAIME-FINAL.pdf>>. Acesso em 15 Ago. 2017.