Trabalhando com Múltiplos Arquivos

Todos sabem que não fizemos nossos sistemas em um único arquivo, partir de agora que já foi estudado sobre funções nativas, condicionais, loops, toda essa parte básica do PHP vai partir para sistemas reais. Para isso precisamos aprender trabalhar com múltiplos arquivos, que é aprender não repetir coisas que vai ser utilizadas em todos ou a maioria dos arquivos. Exemplo: Cabeçalhos, área de conteúdos, rodapés etc.

Por exemplo, vamos criar um arquivo específico para o cabeçalho:

Arquivo header.php

```
//sem as tags de abertura do php aqui... certo,  só html por enquanto.

    <h2> Cabecalho </h2>

    <hr>
```

Agora para importar esse arquivo header.php, para dentro da página index.php, vamos abrir esse arquivo e dentro digitamos uma das duas funções do php que são:

require() e **include()**

Dentro do arquivo index.php por exemplo, declaramos uma dessas funções:

Arquivo index.php

```
<?php

    require(' header.php ');

    echo " Conteudo do site... ";

?>
```

Observem ao executar esse arquivo, que juntou o cabeçalho e o conteúdo nesse único arquivo index.php.

Outro exemplo de arquivo com uso do mesmo cabeçalho:

Arquivo sobre.php

```
<?php

  require('header.php');

  echo 'Conteudo sobre...';
```

?>

Ao executar esse arquivo sobre.php observem que juntou o conteúdo do cabeçalho que está no arquivo header.php com o conteúdo sobre.php.

Com uso da função require() percebem que juntamos o conteúdo do cabeçalho em dois arquivos: index.php e sobre.php. Com isso possibilita trabalhar de uma forma mais ágil na manutenção de certas páginas. Um exemplo disso, se precisar mudar o layout do cabeçalho vamos fazer essa manutenção somente do arquivo header.php com efeito em todas que fazem referência a essa página.

Temos outras utilidades também com esses recursos que é ter um arquivo só com os dados de configurações, por exemplo, a configuração com a conexão ao banco de dados. Normalmente pra isso usamos um arquivo chamado config.php que vai ter usuário, senha, etc, ou seja, as permissões para acesso ao banco de dados, e pra isso no index.php, usamos require('config.php'). Funciona como se as linhas de códigos no arquivo config.php por exemplo, estivessem declaradas no próprio index.php. Então em todos os arquivos que precisar dos dados de configuração só declarar dentro do require o nome desse arquivo.

Qual a diferença do require para o include ?

Se fizermos um **require** para um arquivo que não existe ele vai dar um erro, apontando que impede de continuar a leitura do código, encerra, tipo assim:

Warning: require(teste.php): failed to open stream: No such file or directory...

Fatal error: require(): Failed opening required 'teste.php' (include_path='.;C:\php\pear') in...

Com o **include** em um arquivo que não existe, ele vai dar o mesmo erro, porém não impede de continuar a leitura do código:

Warning: require(teste.php): failed to open stream: No such file or directory...

Fatal error: require(): Failed opening required 'teste.php' (include_path='.;C:\php\pear') in...

Cabecalho

_____

Conteudo sobre...

Essa é a diferença básica do include para o require.

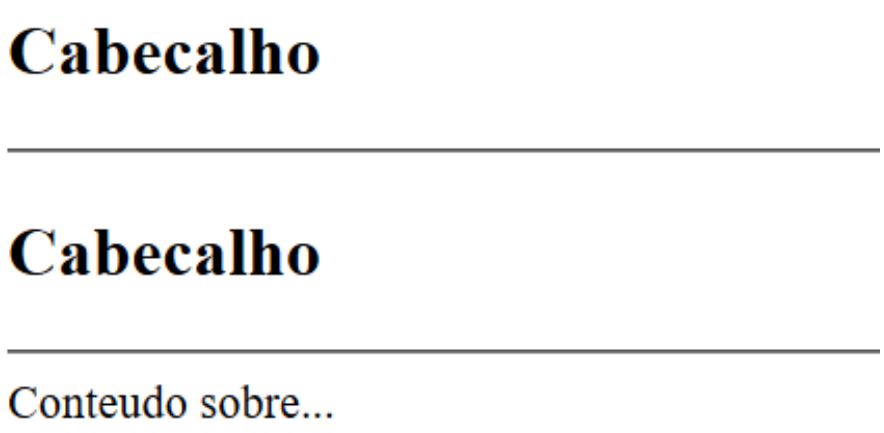
Temos ainda uma variação com o require, por exemplo, quando por qualquer descuido repetir essa função no mesmo arquivo:

Arquivo sobre.php

<?php

```
    require('header.php');
    require('header.php');
    echo 'Conteudo sobre...';
?>
```

Veja ao executar o arquivo sobre.php ele repete duas vezes o conteúdo do cabeçalho, por puxar o header.php duas vezes:



Para isso não ocorrer, principalmente quando estamos lidando com configuração não é bom ocorrer falhas assim, usamos **require_once(),** ou seja, uma única vez. Veja exemplo:

Arquivo sobre.php

```
<?php
    require_once('header.php');
    require_once('header.php');
    echo 'Conteudo sobre...';
?>
```

Veja ao executar o arquivo acima que por mais que ouve a falha de repetir o comando require_once('header.php') não foi executado duas vezes seu conteúdo. Só puxa uma vez o arquivo específicado.

Cabecalho

Conteudo sobre...

Como o requice_once(), não tem perigo de sobrescrever arquivos.

São funções muitos úteis no php, que com certeza iram utiliza-las muitas vezes.

Trabalhando com pastas diferentes

Assunto importante para iniciantes que vão começar a trabalhar com seus projetos. Trata-se de organizar os arquivos e separar em pastas para não ficarem todos os arquivos juntos. Por exemplo, vamos criar uma pasta chamada template, nessa pasta vamos colocar todos os arquivos que fazem parte da estrutura de templates do site. O arquivo header é um exemplo desse tipo de arquivo, então vamos colocar esse arquivo dentro da pasta template. A partir desse procedimento podemos acessar o arquivo header.php da mesma forma que vinhamos fazendo até aqui ?Não, se deixarmos assim:

Arquivo index.php

```
<?php

        require(' header.php ');

        echo " Conteudo do site... ";

?>
```

Vai ocorrer um erro dizendo que não existe o arquivo header.php. Para isso devemos especificar o caminho correto onde está o arquivo, para isso alteramos no require, o caminho completo, ou seja, require('template/header.php');

Arquivo index.php

```
<?php

        require('template/header.php');

        echo " Conteudo do site... ";

?>
```

Conforme alteração no require agora só basta colocar na URL no site assim:

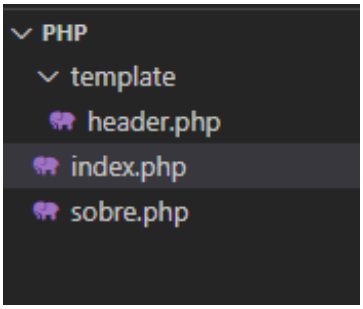
http://localhost/php/index.php que acha o conteúdo do arquivo header.

Ainda podemos fazer dessa forma no require:

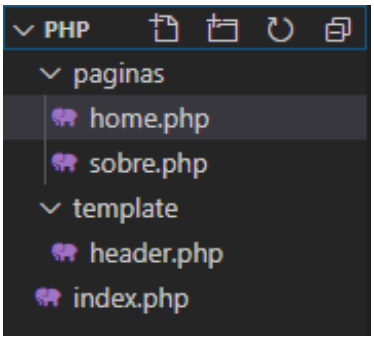
Arquivo index.php

```
<?php
        require('./template/header.php'); // ponto ( . ) seguido de barra ( / ).
        echo " Conteudo do site... ";
?>
```

Isso significa a partir da pasta raiz, que é a partir do php, onde tenho o arquivo index.php, conforme podem ver na figura a seguir que mostra as pastas e arquivos.



Agora vamos fazer outro processo um pouco diferente. Vamos criar outra pasta chamada **paginas** e nessa vamos criar todas as paginas do site. Começamos movendo o arquivo **sobre.php** para a **pasta pagina** e depois criamos um outro arquivo chamado **home.php** dentro dessa pasta pagina, conforme podem ver na figura a seguir:

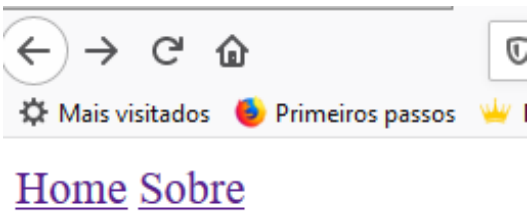


No arquivo index.php vamos fazer as seguintes alterações:

Arquivo index.php

```
<a href="paginas/home.php">Home</a>

<a href="paginas/sobre.php">Sobre</a>
```

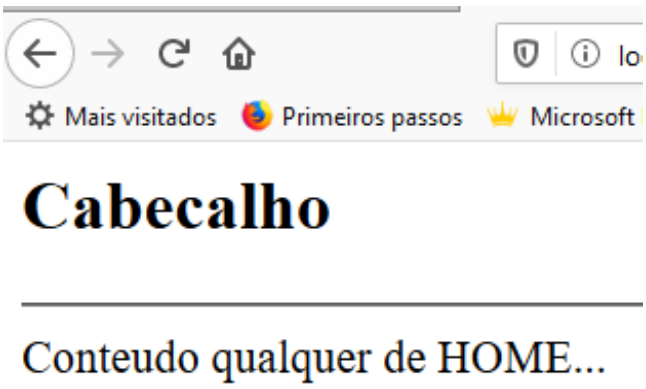
Após executar o index.php aparece os dois link, conforme podem ver na figura a seguir que dá acesso aos arquivos home.php e sobre.php



Agora na página home.php, queremos que ele detenha a pasta template, assim como o arquivo sobre.php, para isso como vamos fazer ? Observem que estamos na pasta paginas no arquivo home.php. Para acessarmos o header.php, no arquivo **home.php** fizemos as seguintes alterações no require():

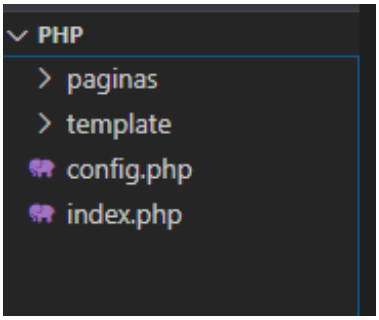
```
<?php

    require('../template/header.php'); // 2 pontos faz voltar uma pasta

?>

Conteudo qualquer de HOME...
```

Para ter acesso ao conteúdo do header.php temos que voltar para a raiz do projeto que é nesse caso, para a pasta php. Por isso fizemos alteração no require('../template/header.php'). Na figura a seguir mostra a pagina home.php.



Agora outro processo muito importante:

Queremos que mostre a versão do sistema no header. E essa versão estará armazenada na raiz, no arquivo que vamos criar chamado config.php. Observem a hierarquia das pastas e arquivos:



E no arquivo chamado config.php fizemos o seguinte:

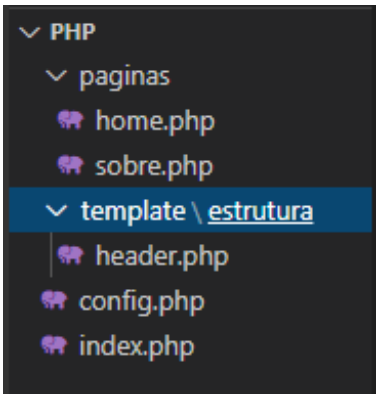
Arquivo config.php

```
<?php
        $versao = '2.2';
?>
```

E queremos exibir no cabeçalho essa versão. Para isso vamos ao arquivo **header.php** que está na **pasta template** e fazemos as seguintes alterações na função require():

```
<?php
        require('../config.php');
?>

<h2> Cabecalho <?php echo $versao ?> </h2>

<hr>
```

Agora vamos criar uma pasta dentro de template chamada estrutura e dentro dessa vamos colocar o agora o arquivo header.php, veja a figura como ficou a hierarquia:



Então dentro da pasta template tem a pasta estrutura e dentro da pasta estrutura está o arquivo header.php.

Nas páginas **home.php** e **sobre.php** temos que alterar no comando require() para:

require('../template/estrutura/header.php');

Por exemplo, esses dois arquivos estão na pasta paginas, e quando acessar elas siginifica que foi puxado da pasta paginas, que volta a pasta raiz e acessa o arquivo config.php.

Assim quando executarmos umas dessas páginas não ocorrerá erro de não visualizar ou encontrar tal arquivo.

O que interessa nesse caso é qual pagina que está sendo acessada, conforme último exemplo foi o sobre.php ou home.php.

Introdução à HTTP Requests

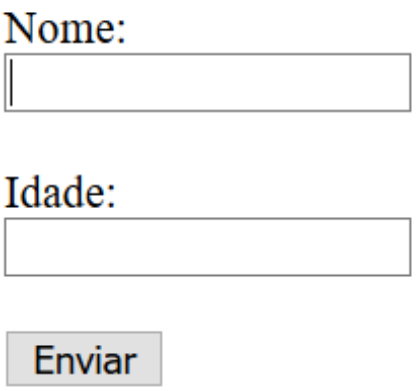
Nessa introdução vamos entender mais sobre requisições http. No arquivo index.php conforme código abaixo:

Arquivo index.php

```
<form method="POST" action="recebe.php">

<label>

        Nome:

        </br>

        <input type="text" name="nome"/>

</label>
```

```
</br>

</br>

<label>

        Idade:

        </br>

        <input type="text" name="idade"/>

</label>

</br>

</br>

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


</form>
```

Ficando assim ao executar o arquivo acima:



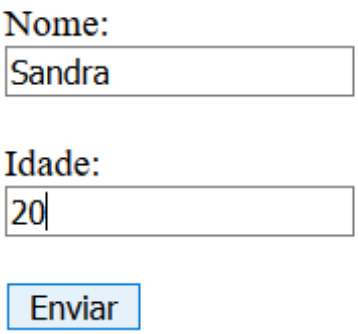
Quando o usuário clicar no botão enviar, é para mandar as informações dos campos do formulário (nome, idade) para o arquivo **recebe.php** que esta especificado na tag form no atributo action. Se não estiver criado ainda o arquivo recebe.php, mostrará como não encontrada essa página, caso contrário, mostrará a página em branco, como podem ver na barra de endereço que ela foi para essa página no momento que o usuário clicou no botão de enviar.

No atributo method = "POST", é a forma que será enviada esses dados do formulário. No PHP tem duas formas de envio de dados: POST e GET.

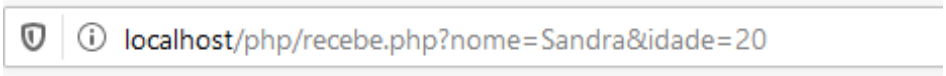
No método POST o envio dos dados se dá de forma internamente. No método GET o envio os dados se dá pela URL, ou seja, o usuário final consegue ver as informações que está sendo enviada, então vamos trocar no atributo method o POST pelo método GET:

<form method="GET" action="recebe.php">

E observem na barra de endereço ao preencher os campos e clicar no botão:



Observem como fica na barra de endereço com o uso do metodo GET:



As variáveis ficam vistas pelo usuário, por isso esse método precisa ter cuidado e saber quando é seguro de se usar no sistema.

Essa é a diferença em relação ao método POST, que envia internamente, não mostrando as informações que foram enviadas.

Então em certos formulários não é aconselhável usar o método GET, por exemplo, um formulário de login em um sistema não pode usar esse método para envio de informações, não é seguro.

Uma observação importante, que vocês precisam saber, é quando não usamos o atributo action no formulário, por exemplo, assim:

<form method="POST">

Isso significa que estamos enviando as informações para o nosso próprio arquivo, nesse caso o arquivo index.php, conforme nosso exemplo, onde está o formulário.

Então, quando especificamos um **action** estamos mandando os dados para um local específico, caso contrário, manda para a própria página onde está o formulário.

Quando não especificamos nada na tag form, conforme a seguir:

<form>

Por padrão o PHP vai enviar pelo método GET.

É importante vocês entenderem essas diferenças, pois a partir de agora vamos começar a trabalhar com formulários.

Pegando Informações do Formulário

Para entender como se dá o recebimento dessas informações do envio dos dados através do formulário.

Vamos usar o mesmo código conforme arquivo index.php, e usando o método GET será enviado para o arquivo recebe.php o nome e idade quando clicar em Enviar.

Arquivo index.php

```
<form method="GET" action="recebe.php">

<label>

        Nome:

        </br>

        <input type="text" name="nome"/>

</label>

</br>

</br>

<label>

        Idade:

        </br>

        <input type="text" name="idade"/>

</label>

</br>

</br>

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


</form>
```

Para saber a melhor forma de receber essas informações agora no arquivo recebe.php, vamos fazer o código necessário para isso:

1ª primeiro, verificar se alguma informação foi enviada;

2ª segundo, pegar os dados, fazer a filtragem, validar, etc.

Vamos começar pegando os dados, itens. Para isso existe uma função chamada **filter_input(),** ela faz duas coisas, pega o campo, e também verifica se está preenchido. Essa função tem basicamente 2 parâmetros, mas pode ter até 4.

No primeiro parâmetro vamos colocar o tipo de método que foi usado para enviar.

O segundo parâmetro é o nome do campo.

Vamos começar pelo campo nome, observem como fica:

```
$nome = filter_input(INPUT_GET,'nome');
```

Só isso já é suficiente para pegar o campo nome no GET e já vai fazer a verificação se está o campo preenchido ou não. Vamos testar no arquivo recebe.php:

Arquivo recebe.php

```
<?php

$nome = filter_input(INPUT_GET,'nome');

$idade = filter_inut(INPUT_GET,'idade');

if($nome){

   echo "Nome: ".$nome;

}else{

   echo "Nao enviou!";

}

?>
```

Seguindo com alterações no arquivo:

Arquivo recebe.php

```
<?php

$nome = filter_input(INPUT_GET,'nome');
```

```
$idade = filter_input(INPUT_GET,'idade');

if($nome && $idade){

   echo "Nome: ".$nome." <br> Idade: ".$idade;

}else{

   echo "Nao enviou!";

}

?>
```

Se quiser que faça a verificação somente de um campo do formulário também pode ser feito sem problemas. Por exemplo, verificar só o nome:

Arquivo recebe.php

```
<?php

$nome = filter_input(INPUT_GET,'nome');

$idade = filter_input(INPUT_GET,'idade');

if($nome){ //Só o nome

   echo "Nome: ".$nome." <br> Idade: ".$idade;

}else{

   echo "Nao enviou!";

}

?>
```

Se enviarmos o formulário via POST, temos que alterar também na função para:

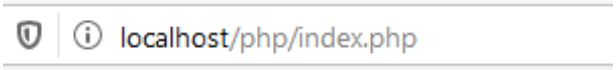
```
$nome = filter_input(INPUT_POST,'nome');

$idade = filter_input(INPUT_POST,'idade');
```

Se quisermos que volte o foco para o formulário no momento que não enviar os dados como podemos fazer? Ao invés de dizer Não enviou! Vamos fazer um redirecionamento com a função **header().** A função header() vai trocar o cabeçalho da requisição. Dentro dela como string colocamos Location, ou seja, trocamos a informação e colocamos para onde queremos enviar. Observem as alterações no arquivo index.php:

Arquivo recebe.php

```
<?php

$nome = filter_input(INPUT_GET,'nome');

$idade = filter_input(INPUT_GET,'idade');

if($nome && $idade){ //Só o nome

    echo "Nome: ".$nome." <br> Idade: ".$idade;

}else{

    header("Location: index.php");

   exit;

}
```

Ou seja, vai fazer um redirecionamento para o index.php quando não enviar informações para o recebe.php. Observem que fica na página index.php, tipo que só atualiza a página, não enviando.



ATENÇÃO: Sempre depois do header colocar um exit. Significa que vai cancelar o restante de qualquer outra coisa a seguir no código. Informação muito importante a respeito do redirecionamento, só podemos usar o a função header() quando não enviamos nenhuma informação para o navegador ainda.

Validando Informações do Formulário

Vamos validar as informações e também verificar se mandamos as informações corretas.

Exemplo:

Arquivo index.php

```
<form method="POST" action="recebe.php">

<label>

        Nome:

        </br>
```

```
        <input type="text" name="nome"/>
</label>
</br>
</br>
<label>
        E-mail:
        </br>
        <input type="text" name="email"/>
</label>
</br>
</br>
<label>
        Idade:
        </br>
        <input type="text" name="idade"/>
</label>
</br>
</br>
<input type="submit" value="Enviar"/>

</form>
```

Ao executar o arquivo index.php fica um formulário igual a da figura a seguir:

Nome:

E-mail:

Idade:

Enviar

E no arquivo recebe.php:

Arquivo recebe.php

```
<?php

$nome = filter_input(INPUT_POST,'nome');

$email = filter_input(INPUT_POST,'email',FILTER_VALIDATE_EMAIL);

$idade = filter_input(INPUT_POST,'idade');

if($nome){

    echo "Nome: ".$nome."</br>";

    echo "E-mail: ".$email."<br>";

    echo "Idade: ".$idade;

}else{

    header("Location: index.php");

   exit;

}
```

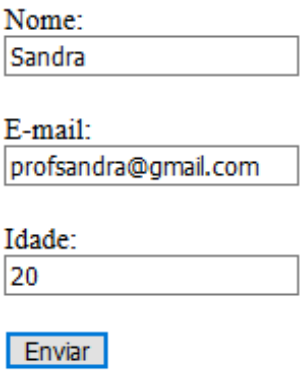
Observem no filter_input() do campo email, foi colocado um terceiro parâmetro nessa função. Com esse processo FILTER_VALIDATE_EMAIL, basta para validar o campo de email. Façam um teste digitando o e-mail sem o " @ " para vocês entenderem como funciona.

Por exemplo, ao preencher o formulário conforme mostrado na figura a seguir:

Observem que o campo do email está preenchido de forma incorreta, não válida, faltando a arroba (@). Isso faz que mostre assim ao enviar:

Nome: Sandra
E-mail:
Idade: 20

Percebem que não mostrou o e-mail porque não foi validado. No momento que for preenchido de forma correta envia:



Ao enviar os dados do formulário conforme exemplificado na figura acima, faz mostrar o valor desse campo, pois está preenchido de uma forma válida:

Nome: Sandra
E-mail: profsandra@gmail.com
Idade: 20

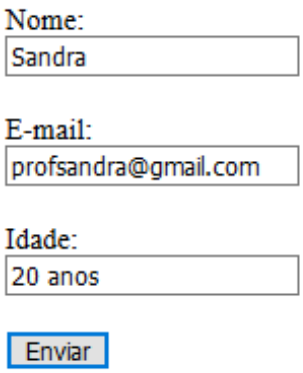
Outro exemplo, vamos validar o campo idade, ou seja, que esse campo só aceite números inteiros. Existe um processo que não é uma validação propriamente dita, que se chama sanitize. Ele vai alterar os dados que foram enviados para ficar em conformidade com o que se espera, que é o **FILTER_SANITIZE_NUMBER_INT**.

Vamos colocar no código do recebe.php o seguinte:

Arquivo recebe.php

```
<?php
$nome = filter_input(INPUT_POST,'nome');
$email = filter_input(INPUT_POST,'email',FILTER_VALIDATE_EMAIL);
$idade = filter_input(INPUT_POST,'idade',FILTER_SANITIZE_NUMBER_INT);
if($nome && $email && $idade){
    echo "Nome: ".$nome."</br>";
    echo "E-mail: ".$email."<br>";
    echo "Idade: ".$idade;
}else{
    header("Location: index.php");
   exit;
}
```

Agora no index.php preenchi o formulário conforme figura a seguir:



Observem como apresenta ao enviar os dados do campo idade, para recebe.php:

Nome: Sandra
E-mail: profsandra@gmail.com
Idade: 20

O FILTER_SANITIZE_NUMBER_INT fez um filtro no campo idade pegando somente números inteiros.

Agora, se não quer que faça um sanitize, quer que mande a informação correta, supondo o preenchimento correto de um valor inteiro, ai faz assim:

$idade = filter_input(INPUT_POST,'idade',FILTER_VALIDATE_INT);

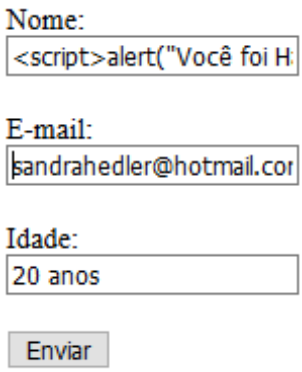
Porém se fazendo assim FILTER_VALIDADE_INT, por exemplo, se preenchermos o campo idade no formulário dessa forma: 20 anos, não irá mostrar nada, não manda o dado preenchido.

Existem outros tipos de filtro que podemos evitar códigos maliciosos. Por exemplo, no campo nome coloco algo que vai ser interpretado como código malicioso, que vai ser interpretada como uma string.

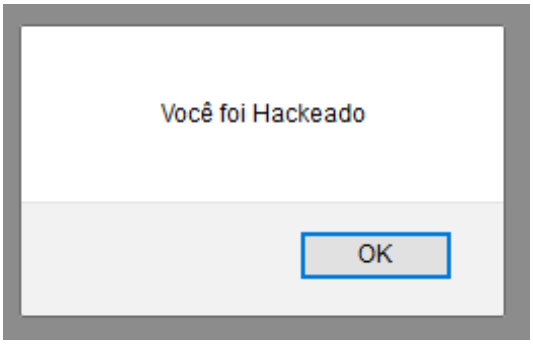
Dentro do campo nome coloco o seguinte código dentro:

<script>alert("Você foi Hackeado") </script>

Como mostrado na figura a seguir:



Isso faz que apareça assim:



Ou seja, executou um código dentro do site.

Então, como nos prevenimos contra esse tipo de coisa ?

Para isso não acontecer usamos outro tipo de filtro, que se chama FILTER_SANITIZE_SPECIAL_CHARS. Observem no código do recebe.php essa aplicação:

<?php

$nome = filter_input(INPUT_POST,'nome',FILTER_SANITIZE_SPECIAL_CHARS);

```
$email = filter_input(INPUT_POST,'email',FILTER_VALIDATE_EMAIL);

$idade = filter_input(INPUT_POST,'idade',FILTER_SANITIZE_NUMBER_INT);

if($nome){

    echo "Nome: ".$nome."</br>";

    echo "E-mail: ".$email."<br>";

    echo "Idade: ".$idade;

}else{

    header("Location: index.php");

   exit;

}

?>
```

Com uso desse filtro FILTER_SANITIZE_SPECIAL_CHARS, podemos verificar inserindo o código <script>alert("Você foi Hackeado") </script> dentro do campo nome no formulário conforme mostrado antes o preenchimento no campo nome. Com uso desse filtro, transforma todo esse código em string, ou seja, não vai mais interpretar como um código.

Nome: <script>alert("Você foi Hackeado") </script>
E-mail: profsandra@gmail.com
Idade: 20

Observem que o código foi todo interpretado como um texto, fazendo com isso que não ocorra um ataque malicioso no sistema.

Além desses, o PHP tem:

FILTER_VALIDATE_IP

FILTER_VALIDATE_URL

FILTER_VALIDATE_FLOAT

FILTER_SANITIZE_EMAIL

FILTER_SANITIZE_SPECIAL_CHARS

FILTER_SANITIZE_URL

FILTER_SANITIZE_NUMBER_FLOAT