

## Introdução

Para o trabalho de Programação Back-end, me foi pedido a construção de um CRUD de receitas. Foi uma ótima seleção, já que tenho algumas coisas em mente que podem me auxiliar na execução do trabalho. Então, vamos começar.



Primeiramente, um CRUD é um acrônimo para Create, Read, Update e Delete (criar, ler, atualizar e apagar), que são termos que descrevem as quatro operações básicas para criar e gerenciar elementos de dados persistentes, especialmente em bancos de dados relacionais. Basicamente é uma forma simples de interagir com os dados em um servidor, envolvendo um banco de dados do MySQL e um conjunto de comandos em PHP a serem interpretados pelo servidor Apache, onde terá como retorno uma página em HTML.

Consulte o código fonte completo do projeto [aqui](#).


## Capítulo 1: A primeira conexão

### Parte 1: O banco de dados

A primeira coisa a fazer é arquitetar um banco de dados, para que possamos ter o nosso CRUD. Para isso foi usado duas ferramentas: o XAMPP, que transforma o computador em um servidor para testar o nosso banco de dados, o HeidiSQL (que iremos continuar a usar conforme for o progresso do trabalho), e o MySQL Workbench, que é uma forma gráfica de criar tabelas de banco de dados. Primeiro ligamos o XAMPP, e ligamos o servidor MySQL no momento, para trabalhar no modelo. Depois, abrimos o Workbench e pedimos para criar um “New Model”. Duplo clique em “Add Diagram” e já temos o nosso espaço para trabalhar no modelo. Na janela em que está escrito “Catalog Tree”, dei um duplo clique em “mydb” e dei um nome para aquele banco de dados, e depois construí as tabelas que preciso para o trabalho.

	#	Nome	Tipo de d...	Tama...	Permitir...
	1	idReceita	INT	11	<input type="checkbox"/>
	2	nome	TEXT		<input type="checkbox"/>
	3	autor	INT	11	<input type="checkbox"/>
	4	ingredientes	TEXT		<input type="checkbox"/>
	5	modoPreparo	TEXT		<input type="checkbox"/>
	6	tempoPreparo	INT	11	<input type="checkbox"/>
	7	foto	VARCHAR	255	<input checked="" type="checkbox"/>

Para as receitas precisamos de: identificação, nome, autor, foto, ingredientes, como preparar e tempo estimado de preparo. Na outra tabela temos os dados do usuário.

#	Nome	Tipo de d...	Tama...	Permitir...	Padrão
 1	idUsuario	INT	11	<input type="checkbox"/>	AUTO_INCREME...
2	usuario	VARCHAR	60	<input type="checkbox"/>	Nenhum padrão
3	email	VARCHAR	200	<input type="checkbox"/>	Nenhum padrão
4	senha	VARCHAR	200	<input type="checkbox"/>	Nenhum padrão
5	formacao	TEXT		<input checked="" type="checkbox"/>	NULL
6	foto	VARCHAR	255	<input checked="" type="checkbox"/>	'vazio.png'

Isso porque todos podem ler as receitas, mas apenas os usuários ESPECÍFICOS podem adicionar, editar ou apagar receitas. Note que na primeira tabela existe uma coluna chamada autor, com uma chave verde. Isto é o que chamamos de CHAVE ESTRANGEIRA, onde se refere a relações de tabelas distintas. O **autor** é referente ao **idusuario**, onde podemos fazer a ligação de uma tabela com a outra. Eu explicarei melhor sobre isso no futuro. Depois de pronto, eu clico no menu “Database”, e clico em “Synchronize Model”. Então dei “Next” 3 vezes, selecionei a database, “next” 2 vezes, checo se tem as setas verdes no “Update”, “next” e “execute”. Feito isso eu tenho o banco de dados salvo no servidor. Agora sim podemos seguir em frente.

## Parte 2: Tela de login

Se eu tenho uma tabela de usuário, obviamente eu preciso de uma tela de login. É até bom, pois assim eu posso testar o banco de dados. No meu banco de dados, eu inseri uma conta para teste, chamada 'admin'. Então eu fiz um simples formulário com uma entrada do tipo email e um do tipo senha. O do tipo email não deixa ser enviado se não tiver o '@', e o do tipo senha esconde a entrada com pontinhos. O formulário para login tem o método post, que é o método em que os dados são enviados de forma “escondida”, ao contrário do método get, em que os dados são enviados pela URL. Vou começar com o teste em fazer um arquivo chamado connection.php, que será usado para todas as coisas que envolvem a interação do PHP com o MySQL. Usando o XAMPP, eu já ligo o servidor Apache, para ler os arquivos PHP, para que o servidor interprete o script e então me retorne uma página em HTML.

```
1  <?php
2
3  $host = "localhost";
4  $user = "root";
5  $password = "";
6  $database = "receitasdaora";
7  $port = 3306;
8
9  $connect = new mysqli($host, $user, $password, $database, $port);
10
11  if($connect->connect_error) {
12      die("Erro de conexão: " . $connect->connect_error);
13  }
```

Neste código, o \$connect tenta fazer uma conexão com o banco de dados. e o mysqli pede os seguintes parâmetros: host, que é o endereço IP do servidor em que está instalado o MySQL, e como estamos usando o XAMPP, o servidor é a nossa máquina (localhost), user, que é o usuário que vai rodar os comandos, que na instalação normal do XAMPP, o usuário é o “root”, password, que é a senha para o acesso, que por padrão é vazio (mas se tiver uma senha, coloque-a), database, que é o banco de dados que vamos utilizar, e por fim, port, que é um ponto terminal de comunicação.

```
1 <?php
2 require_once "../database/connection.php";
```

Agora estamos no arquivo que controla o login, e é aqui em que faremos o primeiro teste. Mas antes de continuar, vamos voltar no arquivo que produz o formulário para o usuário.

```
1 <form action="login.php" method="post">
2     <h1 style='text-align: center;'>ENTRAR</h1><br>
3     <label for="email">Endereço de email</label><br>
4     <input type="email" name="email" id="email"><br>
5     <label for="pass">Senha</label><br>
6     <input type="password" name="pass" id="pass"><br>
```

Note que o form action é o mesmo que o nome do arquivo. Então os dados retornam para o arquivo? Como que vai enviar para o script? É aí que mora a mágica: no topo do arquivo, nós incluímos o script do login. E para que isso funcione, o script do login tem que checar uma condição: se certas variáveis existem, porque se cada vez que o usuário quiser fazer login, o script vai rodar, e vai retornar erros porque um dado esperado não existe. Então, como forma de evitar isso, colocamos a condição depois do comando de incluir a conexão. Um lembrete: todas as vezes que tiver um script que vai interagir com o banco de dados, a primeira variável que se usa é o que foi usado para criar a conexão, que no nosso caso é o \$connect.

```
1 if(isset($_POST['email']) && isset($_POST['pass'])) {
2     $mail = $_POST['email'];
3     $pass = $_POST['pass'];
```

A função isset checa se uma variável existe e se tem um valor atribuído. A variável \$\_POST é considerada, no PHP, uma superglobal, ou seja, uma variável que pode ser acessada em qualquer lugar. Essa variável é uma array (que pode armazenar mais de um valor), com os índices que têm os mesmos nomes que têm nos campos de entrada na página do formulário.

```
1 <h1 style='text-align: center;'>ENTRAR</h1><br>
2 <label for="email">Endereço de email</label><br>
3 <input type="email" name="email" id="email" placeholder="exemplo@dominio.com"
4 oninput="checking();"><br>
5 <label for="pass">Senha</label><br>
6 <input type="password" name="pass" id="pass" placeholder="Senha"
7 oninput="checking();"><br>
```

O \$\_POST vai pegar as entradas que tem o mesmo nome especificado na índice. Nesse caso, eu quero pegar o email e o pass. Agora voltando ao script que controla o login. Eu já defini uma variável para definir o email e a senha, e agora irei usar o email para fazer a primeira consulta no banco de dados.

```
1 $command = $connect->prepare("SELECT * FROM `bancousuario` WHERE `email` = ? ;");
2 $command->bind_param("s", $mail);
3 $command->execute();
4 $result = $command->get_result();
5 $user = $result->fetch_object();
```

Lembre-se que o \$connect é aquela variável da connection.php, em que foi iniciada a conexão com o banco de dados, iremos usá-la em todas as consultas que for preciso. Nessa eu estou procurando por algum registro em que a coluna email é a mesma digitada pelo usuário. Eu crio uma variável chamada command, que é a que vai preparar o comando para ser executado. A vantagem em usar o prepare é que pode ser em que algum usuário possa tentar injetar um código em SQL para fazer alguma coisa mal-intencionada, como deletar registro, alterar coisas, etc. Aí eu coloco, nesse mesmo comando, os parâmetros com o

bind\_param. O 's' indica que é uma string, e o 'i' indica que é um número inteiro. Então eu ponho para executar. O \$result vai me retornar o resultado da consulta, e o \$user vai transformar o resultado em objeto. Existem duas formas em receber dados de um resultado de uma consulta: o fetch\_object, e o fetch\_assoc. Com o fetch\_object, o resultado vem em forma de objeto (que para acessar os dados é preciso da variável que está guardando os dados, a setinha (->) e o nome da coluna), e com o fetch\_assoc, o resultado vem em forma de array (que para acessar é só colocar o nome da coluna na índice da variável).

```
1  if($user != NULL) {
2      if(password_verify($pass, $user->senha)) {
3          session_start();
4          $_SESSION['id'] = $user->idUsuario;
5          $_SESSION['user'] = $user->usuario;
6          $_SESSION['photo'] = $user->foto;
7          if($user->formacao != null) {$_SESSION['grade'] = $user->formacao;}
8          $_SESSION['login'] = true;
9          header("Location: ../home/page.php");
10         exit;
11     } else {
12         //Avisa que a senha está errada!
13     }
14 } else {
15     //Avisa que o email não está no sistema!
16 }
```

No mesmo script, ele vai checar se o \$user não está vazio. Se for verdadeiro, ele vai fazer uma outra checagem: se a senha bate com a que é a mesma do resultado da consulta. Esse password\_verify decodifica a senha que está salva no banco de dados e compara com o que veio do formulário (senha codificada eu explicarei no futuro). Se as senhas batem, então ele inicia uma sessão e guarda algumas informações na variável \$\_SESSION, que também é uma superglobal. Guardamos o nome do usuário, o número de identificação, o nome da foto do usuário (também explicarei no futuro). Então, ele é redirecionado à página inicial e termina o script. Senão, no meu caso, ele avisa que o usuário colocou a senha errada. Agora, se a variável \$user é nula, ele avisa que o email não consta no sistema. Mas como que eu aviso, e como ele vai voltar para a tela de login? Primeiro, no meu script, eu criei

uma variável que vai armazenar uma mensagem de erro que será apresentada no formulário.

```
1 <?php echo isset($error_log['mail']) ? $error_log['mail'] : "" ?>  
2 <?php echo isset($error_log['pass']) ? $error_log['pass'] : "" ?>
```

A mensagem é mostrada antes do botão de enviar. Note o comando estranho com a interrogação e dois pontos. Esse é o chamado Operador Ternário. Ele funciona assim: eu quero imprimir o texto que está na variável `error_log` que eu criei, seja na índice `mail` ou `pass`. Mas primeiro ele checa se essa variável com o índice existe, e se sim, ele imprime a mensagem, senão, ele deixa em branco. Se ainda não entendeu, essa linha pode te ajudar:

```
< echo isset(variável) ? verdadeiro : falso; >
```

E segundo, lembra que no form action o arquivo é o mesmo que apresenta o formulário? Então, no clique do botão que vai enviar o formulário, ele vai reler a página, mas com os dados enviados via POST. E no momento que a página carregar, o script do login será carregado primeiro, e assim, as variáveis existem, então os comandos serão executados. Agora que temos um simples sistema de login, precisamos focar em um sistema de cadastro de usuário, pois nas configurações do meu site, eu quero que apenas pessoas logadas tenham acesso às receitas. Como se por exemplo, o site enviará notificações de receitas e novidades no site. Mas acho que isso ficará para a próxima parte, então, vamos à próxima parte! Ah, antes que eu me esqueça, existe um script em Javascript, que checa se tem alguma coisa digitada no email e senha, para que assim possa liberar o botão de enviar o formulário. A única função é essa, então não quis entrar em detalhes, mas se de alguma forma quer ver como é esse script, segue a imagem abaixo.

*Comentário do autor: se você ainda está se perguntando, o que isso tem a ver com o tema? Lembra que eu falei que eu quero que CERTAS PESSOAS tenham o direito de*



editar, inserir e excluir? Por isso a existência desse sistema, calma que chegarei ao ponto.

```
1  function checking() {
2      var email = document.getElementById('email').value;
3      var pass = document.getElementById('pass').value;
4      var submitbtn = document.getElementById('logar');
5
6      const regexmail = /[a-zA-Z0-9]+@[a-zA-Z](\.[a-zA-Z]+)+$/i;
7
8      var mailtest = email.match(regexmail);
9      if(mailtest && pass != null) {
10         submitbtn.classList.add('btnsuccess');
11         submitbtn.disabled = false;
12     } else {
13         submitbtn.classList.remove('btnsuccess');
14         submitbtn.disabled = true;
15     }
16 }
```

### Parte 3: Cadastro de usuário

Mais uma vez lembrando, que eu estou fazendo um site sobre receitas, e por opções pessoais, eu quero que as receitas estejam disponíveis para os usuários logados. E se não tem o login, pode criar uma conta. Vamos lá! O processo é parecido com a do login, mas existem alguns detalhes e algumas diferenças. Irei entrar em detalhes mas de forma breve, porque conforme eu escrevo este relatório, eu fico cada vez mais ansioso para chegar a parte de explicar de como funciona a parte do CRUD no meu site. Começaremos criando um arquivo chamado `signin.php`, em que está o formulário para cadastrar o usuário (também com o método `post`), e também um script que será incluído no formulário, que irá controlar o cadastro do usuário. A estrutura do formulário de cadastro é assim: eu vou pedir o nome, foto de perfil, email, senha, e quero que confirme a senha. Um resumo rápido das entradas deste formulário, é que tem um novo input do tipo `arquivo`, que é o que vai servir para enviar a foto do usuário, mas para que isso seja possível, deve ser inserido o `enctype` na linha do formulário.

```
1 <form action="signin.php" method="post" enctype="multipart/form-data">
```

Esse vai permitir que o arquivo seja enviado para o script. Agora retomando, quase nada muda no formulário, só apenas alguns campos adicionados e um input do tipo `arquivo` para envio de foto.

```
1 <?php
2     require "../database/signin.php";
3     require "../scripts/loginCheck.php";
4     ?>
```

Esse loginCheck faz uma verificação se o usuário está logado. Esse arquivo está incluído também no formulário de login. Basicamente ele checa se o \$\_SESSION['login'] existe e se é verdadeiro (Lembre-se do script de login). Se sim, ele redireciona para a página inicial, até porque o usuário já fez login, então não precisa fazê-lo novamente ou então cadastrar uma conta. Também é um script muito simples, então não vou entrar em detalhes nesse. Antes de voltar ao script que controla o usuário, vamos analisar um outro script, importante tanto para usuário quanto para as receitas que iremos adicionar. Esse script, que vou falar agora, processa as fotos, guarda o nome da foto no banco de dados, e envia a foto para uma pasta no servidor, onde estão os arquivos do site.

```
1  <?php
2      if(!empty($_FILES['photo']['name'])) {
3          $origem = $_FILES['photo']['tmp_name'];
4          $photo = time().$_FILES['photo']['name'];
5          $destino = "../database/imgProfile/$photo";
6          move_uploaded_file($origem, $destino);
7      } else {
8          $photo = "sem_foto.png";
9      }
```

Primeiro, ele checa se existe um arquivo enviado pelo formulário. O 'photo' é o nome do input do tipo foto, e o 'name' guarda o nome do arquivo enviado. Se a condição for verdadeira, o \$origem salva o caminho origem da foto, o \$photo dá um nome para o arquivo, que é a combinação do tempo de agora mais o nome original do arquivo, o \$destino salva o caminho da foto, e a função move\_uploaded\_file() envia o arquivo da origem para a pasta de destino. Na minha configuração o meu destino é assim, dependendo da configuração o destino pode ser diferente; faça alguns testes e veja o caminho certo para mover os arquivos adequadamente. Agora, se nenhum arquivo foi enviado, a variável \$photo guarda o nome da foto em branco, que está salva na mesma pasta para onde vai os arquivos enviados. Agora que expliquei como funciona esse script, vamos voltar ao

que controla o cadastro de usuário. Irei incluir esse script e usar a variável \$photo para as operações de cadastro.

```
1  <?php
2  require_once "../database/connection.php";
3
4  if(isset($_POST['pass'])) {
5      if($_POST['pass'] != $_POST['cpass']) {
6          $error_log['cpass'] = '<label id="formwarning">As senhas não coincidem!</label>';
7          $name = $_POST['name'];
8          $email = $_POST['email'];
9          return;
10     } }

```

Vamos incluir novamente a conexão, e antes de executar, precisamos fazer uma verificação da senha. No cadastro, eu tenho dois campos para senha. O segundo campo pede para inserir a mesma senha novamente como confirmação, na qual o seu nome é o 'cpass'. Se a senha inserida não bater com a senha inserida no campo de confirmação, a primeira variável vai guardar um aviso para informar que as senhas não batem. Então ele guarda o nome e o email, e encerra a execução do script. A mesma lógica se aplica na tela de login: se a variável da linha 5 existe, ele mostra, se não, mostra nada.

```
1  if(isset($_POST['name']) || isset($_POST['email'])) {
2      $testing = $connect->prepare("SELECT * FROM bancousuario WHERE usuario = ? OR email = ?");
3      $testing->bind_param("ss", $_POST['name'], $_POST['email']);
4      $testing->execute();
5      $tests = $testing->get_result();
6      $test = $tests->fetch_assoc();
7
8      if(strcasecmp($test['usuario'], $_POST['name']) == 0) {
9          $error_log['name'] = "<label id='formwarning'>Esse nome já existe!</label>";
10         return;
11     }
12
13     if($test['email']) {
14         $error_log['email'] = "<label id='formwarning'>Esse email já existe!</label>";
15         return;
16     }
17 }

```

Esse, no mesmo script, é um pouco complexo, mas simples de entender. Basicamente esse pedaço verifica se o nome foi enviado por POST OU se o email foi enviado por POST. É esperado que sim, então esse pedaço será executado.

Então ele vai fazer uma consulta para verificar se tem um nome ou email cadastrado no sistema. Se o nome é o mesmo, ele avisa que esse nome já existe no sistema, e que deve escolher outro. Mesma coisa com o email. Se não, o script continua. Agora sim estamos na parte principal do script.

```
1  if(isset($_POST['name']) && isset($_POST['email']) && isset($_POST['pass'])) {
2      require_once "../scripts/processPhoto.php";
3
4      $name = $_POST['name'];
5      $mail = $_POST['email'];
6      $pass = password_hash($_POST['pass'], PASSWORD_BCRYPT);
7
8      $command = $connect->prepare("INSERT INTO bancousuario (`usuario`, `email`, `senha`, `foto`) VALUES ( ?, ?, ?, ?);");
9      $command->bind_param("ssss", $name, $mail, $pass, $photo);
10     $command->execute();
11
12     header("Location: ../home/login.php");
13 }
```

Depois da verificação, é incluído o script que processa a foto enviada. Aquele password\_hash transforma a senha digitada no formulário em uma string criptografada para depois ser mandada pelo banco.

idUsuario	1
usuario	superuser
email	super@user.com
senha	\$2y\$10\$IYcESWF9YLp01/D5rMLHeEbBC1471XcFIR0fD9AGvqXtKbHLY8JC
formacao	super
foto	1686875144sudo.png

Aqui um exemplo do registro. Veja que a senha fica desse jeito com a função. Mas qual é a senha enviada? Simples: “superuser”. Feito isso, inserimos os valores das variáveis no comando da consulta, e executamos. Depois disso, o usuário fica registrado, e então é redirecionado para a tela de login. Depois de logar, já pode acessar as receitas. E para finalizar, finalmente, um script de logout, para encerrar a sessão do usuário. Esses scripts de login, cadastro e logout são chamados por um link. A forma como você quer fazer esse link fica a seu critério. O script de logout é o mais simples de todos, então não vou entrar em meros detalhes.

```
1 <?php
2 session_start();
3 $_SESSION = array();
4 session_destroy();
5
6 header("Location: ../home/page.php");
```

Aqui ele retoma a sessão, destrói os dados da superglobal, e destrói a sessão. E por fim, ele volta na página inicial. Agora que FINALMENTE acabamos de desenvolver esse sistema, e lembrando que CERTAS PESSOAS podem Inserir, Atualizar e Excluir, podemos prosseguir. Vamos, finalmente, para as funções do CRUD.

## Capítulo 2: Crie, Leia, Atualize e Exclua

### Parte 1: Consultar Todos

Na página inicial, tenho um link que vai para uma outra página que irá aparecer os links para ler, atualizar e apagar (os dois últimos apenas para certos usuários). Mas primeiro vamos focar primeiro no conteúdo da página em que estamos.

```
1  <?php
2      session_start();
3      if(!isset($_SESSION['login'])) {
4          header("Location: login.php");
5      }
6
7      require "../database/query/queryAll.php";
8      include "../template/pageHead.php";
9  ?>
```

Como pode ver, ele checa se o usuário está logado para ver a página. Esse page head é só para mostrar os botões fixos acima da página. Isso é mais de design do site, então não vou entrar em detalhes. Vamos focar mais no arquivo da linha 7. Esse que vai ser responsável por fazer uma listagem de arquivos. Bom, obviamente ele não vai produzir os resultados, mas iremos chamar uma função do PHP para isso. Mas primeiro, vamos dar uma olhada no arquivo. Ele que vai ser responsável em trazer todos os dados que constam em uma tabela, que no caso é a que vai ter as receitas.

```

1  <?php
2      require_once "../database/connection.php";
3
4      $sql = "SELECT * FROM `bancoReceita`";
5      $command = $connect->prepare($sql);
6      $command->execute();
7
8      $result = $command->get_result();
9
10     $receitas = [];
11
12     while($receita = $result->fetch_object()) {
13         $receitas[] = $receita;
14     }

```

A linha 2 é familiar. Primeiro executamos um comando para mostrar todas as linhas de dados da tabela “bancoReceita”, que é onde estão as receitas. Depois eu criei uma array para guardar esses resultados. E usando a função while, eu coloco uma linha de dados em cada posição do array. Agora vamos voltar no site que terá a listagem. Nós iremos usar o \$receitas em uma função parecida com o while, mas que para mim é uma combinação do for com o while. Na página em que será mostrada a lista, eu incluí esse script, pois irei usar a variável na função que mencionei.

```

1  <?php foreach($receitas as $recipes): ?>
2  <div class='glass'>
3      <a><h1><?= $recipes->nome; ?></h1></a>
4      <?php if(isset($_SESSION['grade'])): ?>
5          <a href='recipeMan.php?id=<?= $recipes->idReceita ?>'>Editar</a>
6          <a href='../database/query/queryDelete.php?id=<?= $recipes->idReceita ?>'>Apagar</a>
7      <?php endif; ?>
8      <a href='read.php?id=<?= $recipes->idReceita ?>'>Ler</a>
9  </div>
10 <?php endforeach; ?>

```

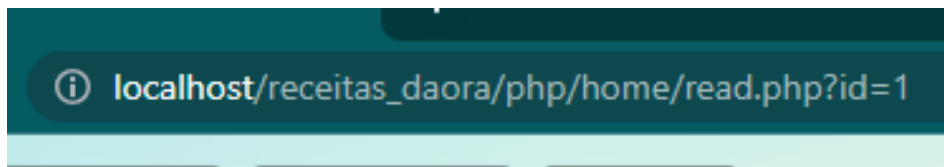
O foreach é uma espécie de função loop que percorre pelos índices de uma array, que no caso é a mesma em que eu coloquei o resultado da consulta. Então eu dei um outro nome, chamado recipes, e depois usei essa variável para mostrar os



dados. Note que não tem uso de chaves, mas sim dois pontos. Isso é a forma mais eficiente do que dar Ctrl C e Ctrl V, porque na teoria, isso é o que ele está fazendo. Aquele ícone no meio checa se o usuário logado é aquele que possui formação, que em suma, é o que tem permissões para atualizar e excluir. Aqueles links no Ler, Editar e Excluir vou explicar mais tarde.

## Parte 2: Consultar por ID e Leitura

Já podemos consultar todas as linhas de uma consulta, mas e se quisermos ler uma publicação, ou seja, ler uma linha da consulta, e uma consulta em específico? Para isso vamos criar um arquivo similar ao que vimos na parte 1, mas a diferença é que só vamos consultar um resultado baseado no id enviado pela URL. E quando precisamos pegar uma variável que veio da URL, não iremos usar o `$_POST`, mas sim o `$_GET`. Esse atua da mesma forma, mas a diferença é que o get pega uma variável digitada na URL.



Na imagem acima, é aquele que está depois da interrogação. URL enviando dados funciona da seguinte forma: primeiro tem-se o endereço da página em exibição, em seguida o ponto de interrogação para separar endereço e variável e depois a variável que será enviada junto com um valor. Esse script vai funcionar quando clicarmos no botão “Ler”.

```
1 <a href='read.php?id=<?= $recipes->idReceita ?>'>Ler</a>
```

O link, do arquivo da listagem, já está estruturado para enviar a variável na URL. Vamos olhar o script disto.

```

1  <?php
2      require_once "../database/connection.php";
3
4      if(isset($_GET['id'])) {
5          $id = $_GET['id'];
6          $sql = "SELECT idReceita, nome, usuario, ingredientes,
7                  tempoPreparo, modoPreparo, bancoreceita.foto
8                  FROM bancoreceita INNER JOIN bancousuario ON
9                  bancoreceita.autor = bancousuario.idUsuario WHERE idReceita = ? ";
10         $command = $connect->prepare($sql);
11         $command->bind_param("i", $id);
12         $command->execute();
13
14         $result = $command->get_result();

```

Simplesmente linha 2. Vamos checar se existe a variável que queremos receber por GET. Então o definimos como \$id, e temos essa extensa linha de consulta. Ele vai querer o id da receita (que será necessária depois), nome da receita, usuário (que é basicamente o autor da postagem, que será o mesmo que fez login, então o processo é automático), os ingredientes (irei explicar o macete por trás disso), tempo de preparo, como preparar e a foto da receita. Eu irei ligar o bancoReceita com o bancoUsuario pelo id do autor da receita com o id do usuário cadastrado. Um exemplo para fixar, eu tenho uma receita de Petit Gateau no bancoReceita, e o autor é uma chave estrangeira de número 3. Chave estrangeira que faz referência ao bancoUsuario. Quem tem o id 3 lá? O usuário “Erick Jacão”. Então, levando em conta esse exemplo, ele vai me retornar a receita de Petit Gateau, postado pelo usuário Erick Jacão. Então preparamos o comando, inserimos o id no comando e colocamos para executar, e peço para retornar o resultado

```

1      $receita = $result->fetch_assoc();
2
3      $ident = $receita['idReceita'];
4
5      $resmodp = $receita['modoPreparo'];
6  }

```

Vamos usar o `fetch_assoc` para diferenciar, esse é aquele que retorna o resultado em forma de array. Para simplificar um pouco, vou pegar o `idReceita` e guardar no `ident`. A coluna `ingredientes` é um pouco diferente, vou explicar como funciona. A coluna `ingredientes` é do tipo texto, em que eu posso guardar uma string longa sem problemas. Mas como `ingredientes` deve ser uma lista, a lógica é a seguinte: a string deve ser uma lista em forma de uma string separada por um delimitador, que no caso é o ponto e vírgula (que mais ou menos é assim: “`exemplo1;exemplo2;exemplo3`”). Eu usarei uma função que logo explicarei em como irei transformar em uma lista.

```
1  <?php
2      session_start();
3      if(!isset($_SESSION['login'])) {
4          header("Location: login.php");
5      }
6
7      require "../database/query/querySingle.php";
8      include "../template/pageHead.php";
9  ?>
```

Já deve saber o que são as linhas 2 ao 4 e 8. Como o usuário vai clicar no botão ler, a página será aberta com a variável na URL, por isso rodamos o script primeiro.

```

1 <h1><?= $receita['nome'] ?></h1>
2 <h3>Por <b><?= $receita['usuario'] ?></b> |
3 Tempo de preparo: <?= $receita['tempoPreparo'] ?> minutos.</h3><br>
4 <br><br>
5 <h3> <b>Ingredientes: </b> </h3>
6 <?php
7     $instrucao = $receita['ingredientes'];
8     $itens = explode(";", $instrucao);
9     echo "<ul>";
10    foreach($itens as $item) {
11        echo "<li>$item</li>";
12    }
13    echo "</ul>";
14    ?><br>
15 <h3><b>Como preparar:</b></h3>
16 <p style='font-size: 18px;'> <?= $receita['modoPreparo'] ?> </p>

```

Com o script executado, nós vamos inserir o resultado na página de leitura. Note que eu não usei o echo. Essa é a forma simplificada do echo, mas ambos operam do mesmo jeito. O que eu queria explicar, sobre a lista, começa na linha 6 e termina na 14. Vou guardar o dado da coluna ingredientes, e depois, usar o explode para quebrar uma string em um array. Essa função requer 2 parâmetros: o delimitador da string, e a string que quero “explodir”. Então eu uso o echo para criar uma lista não ordenada, e uso o foreach para percorrer o array que é resultado do explode. Para cada linha contida no array é imprimida dentro da tag <li>. Depois que terminar o foreach, imprimo o fechamento da lista, e assim encerra esse pedaço do PHP. E, para finalizar, imprimo o texto de modo de preparo da receita.

### Parte 3: Inserir, atualizar e excluir dados

Essa parte envolve algumas mágicas do PHP, e reutilizando o script de consultar por ID. Pois vou usar esse formulário para duas coisas: para atualizar e inserir receitas. E também com o mesmo método de todos os formulários: post. Esse formulário tem o mesmo enctype que vimos no login, pois iremos enviar foto da receita. Primeiro vamos focar no formulário em si. Esse formulário é bem extenso, então vou colocar pequenos pedaços dele.

```
1 <form action='<?php echo isset($receita) ?  
2  "../database/query/queryUpdate.php" :  
3  "../database/query/queryAdd.php"; ?>  
4  method="post" enctype="multipart/form-data">
```

Estamos usando o operador ternário novamente, porque incluímos o script de consulta por ID. A forma de como isso funciona é simples: o usuário logado tem a permissão de adicionar receitas. Então ele vai entrar na página do formulário, e aquele script vai rodar. Já que não vai retornar algum resultado, a variável \$receita está vazia. Ou seja, a condição é falsa, logo o action é o script de inserir dados. Já o usuário logado E com formação, pode adicionar, editar e excluir receitas. E para esse usuário, os botões de editar e excluir irão aparecer. E se clicar em um dos botões de editar, aquele script vai retornar um resultado, logo a condição é verdadeira e o action é o script de atualizar dados.

Os campos de entrada são: o id (se é para atualizar, esse é escondido), nome da receita, autor (que é o mesmo id salvo no \$\_SESSION, então já é automático), tempo de preparo (essa entrada é de números, e só aceita números), uma foto (iremos reutilizar o script de processar foto com um pequeno ajuste), ingredientes (esse não é um input, mas sim um textarea, suporta mais caracteres, e terá de receber uma string e detectar quebras de linha) e o modo de preparo, que também é uma textarea. Eu também coloquei um isset para erro de envio de dados, se caso

um campo não foi enviado, exceto foto, mas eu acredito que isso é muito improvável, pois tenho um Javascript que libera o botão se tem os campos preenchidos. Vou fazer uma cópia do script que processa fotos, porque eu só vou alterar uma coisa: o destino do arquivo, pois eu tenho uma pasta para perfis, e uma pasta para foto das comidas. Essa cópia será incluída tanto no script de atualizar quanto no de inserir. Vamos focar no de inserir dados.

```
1  require_once "../../database/connection.php";
2  require "../../scripts/processRecipe.php";
3
4  if(isset($_POST['nome']) && isset($_POST['tempo']) &&
5      isset($_POST['ingredientes']) && isset($_POST['modpre'])) {
6
7      $nome = $_POST['nome'];
8      $autor = $_SESSION['id'];
9      $tempo = $_POST['tempo'];
10     $ingre = implode(';', explode("\n", $_POST['ingredientes']));
11     $modpre = $_POST['modpre'];
12
13     $sql = "INSERT INTO bancoreceita
14     (`nome`, `autor`, `ingredientes`, `modoPreparo`,
15     `tempoPreparo`, `foto`) VALUES (?, ?, ?, ?, ?, ?);";
16
17     $command = $connect->prepare($sql);
18     $command->bind_param("sissis", $nome, $autor, $ingre,
19         $modpre, $tempo, $photo);
20
21     $command->execute();
22     header('Location: ../../home/recipes.php');
23 }
```

Esse processo você já deve conhecer. Vamos falar da linha 10. Temos duas funções para serem executadas, mas a prioridade é de dentro para fora, ou seja, o implode vai rodar depois do explode. Mas o que essa linha faz? Simples, primeiro ele pega o texto que é a lista de ingredientes, que o usuário deveria ter colocado em forma de lista, quebrando linhas. Isso vira um array e depois uma linha só, com os dados separados por ponto e vírgula para enviar ao banco de dados. Já comentei sobre isso na parte de ler os dados. E o processo é auto explicativo, então vamos seguir em frente com o projeto.

No momento, já vimos sobre inserir dados, agora vamos ver em atualizar dados. O formulário é o mesmo, mas a diferença é que o ID será enviado pela URL, assim ativa o script de consultar por ID. A diferença é que eu vou usar a variável que guarda o resultado e depois colocar como valor em um input escondido.

```
1 <?php if(isset($receita)): ?>
2 <input type="hidden" name='ident' id='ident' value='<?= $ident; ?>'>
3 <?php endif; ?>
```

Como esse é um script para atualizar, a variável receita vai existir, então essa entrada existe. O \$ident é o valor da idReceita da consulta no script de consulta por ID.

```
1 require "../database/connection.php";
2 require "../scripts/processRecipe.php";
3
4 $ident = $_POST['ident'];
5 $nome = $_POST['nome'];
6 $tempo = $_POST['tempo'];
7 $ingre = implode(';', explode("\n", $_POST['ingredientes']));
8 $modpre = $_POST['modpre'];
9
10 $sql = "UPDATE bancoreceita SET nome = ?, ingredientes = ?,
11 modoPreparo = ?, tempoPreparo = ?, foto = ? WHERE idReceita = ? ";
12
13 $command = $connect->prepare($sql);
14 $command->bind_param("sssi", $nome, $ingre, $modpre, $tempo, $photo, $ident);
15 $command->execute();
16
17 header('Location: ../../home/recipes.php');
18 exit();
```

O processo é o mesmo, mas a diferença está na linha 10, em que está o comando da consulta. O campo de identificação no formulário para atualizar tem o nome de ident, que é o que está na linha 4. Esse será essencial para a parte do comando SQL depois do “WHERE”, pois queremos atualizar dados em que o idReceita é igual à variável \$ident. Quando o formulário é para atualizar, os dados recebidos da consulta por ID são colocados como valor em cada uma das entradas, pois o



usuário vai checar o que precisa mudar e enviar os dados, mas tem um detalhe: se o usuário não enviar foto, será considerado como foto vazia. E o script de excluir dados é o mais simples de todos eles, esse se localiza na listagem de receitas.

```
1  require_once "../../database/connection.php";
2
3  if(isset($_GET['id'])) {
4      $sql = "DELETE FROM bancoreceita WHERE idReceita = ? ";
5
6      $command = $connect->prepare($sql);
7      $command->bind_param("i", $_GET['id']);
8      $command->execute();
9
10     header('Location: ../../home/recipes.php');
11 }
```

Na listagem, o botão para chamar pelo script de deletar dados possui a variável ID enviado pela URL. Então rodamos o comando para deletar a linha do banco de dados baseado no id especificado, e depois retorna à página da listagem.

### Capítulo 3: Conclusão e comentários finais

Esse projeto me deixou com a cabeça doendo para raciocinar, até levei dois dias para fazer o sistema de login, e demorei mais porque estava personalizando o site, tanto é que em uma das commits a porcentagem do CSS foi maior do PHP, com uma diferença mínima, e ambos estavam nos 40%. Tem uma porcentagem em Javascript, mas como eu falei, não vou entrar em detalhes, uma vez que ele é mais para a parte do usuário, seja para liberar um botão e checar algumas entradas. Mas no final, eu fiquei feliz com o produto final. Espero que a minha explicação ficou clara, pois eu não sou bom em explicar coisas, mas dei o meu melhor. Obrigado por ler e até mais!