**Aula 11 – PHP CRUD-Produto**

**Manutenção de Produto com MVC e OO**

**Objetivos:**

* Implementar a manutenção de Produtos na estrutura MVC com Orientação a Objetos;
* Implementar o acesso ao banco de dados MySQL através de PDO.

Observação: Vamos utilizar o mesmo Banco de Dados criado na Aula 02 deste curso. Portanto, se já possui o banco crudproduto com a tabela produto, pode pular a próxima seção e ir direto para a seção 11.2.

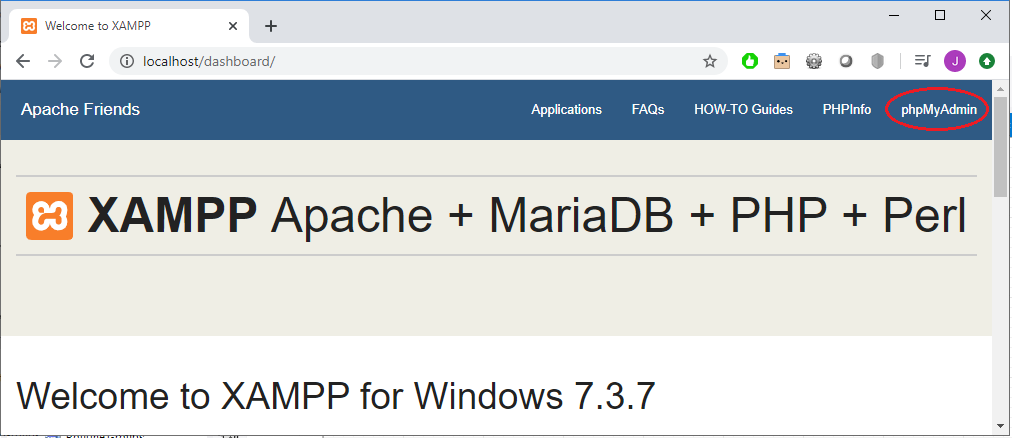
* 1. **Criando o Banco de Dados da Aplicação no MySQL**

Antes de iniciarmos o acesso aos dados da Aplicação via PHP, é preciso criar o Banco e as Tabelas que armazenarão os dados da Aplicação. Estas ações também podem ser feitas via PHP, porém é comum realizá-las diretamente no Banco de Dados com o auxílio de uma ferramenta de administração do Banco. Para o MySQL existem várias formas, dentre elas se destacam:

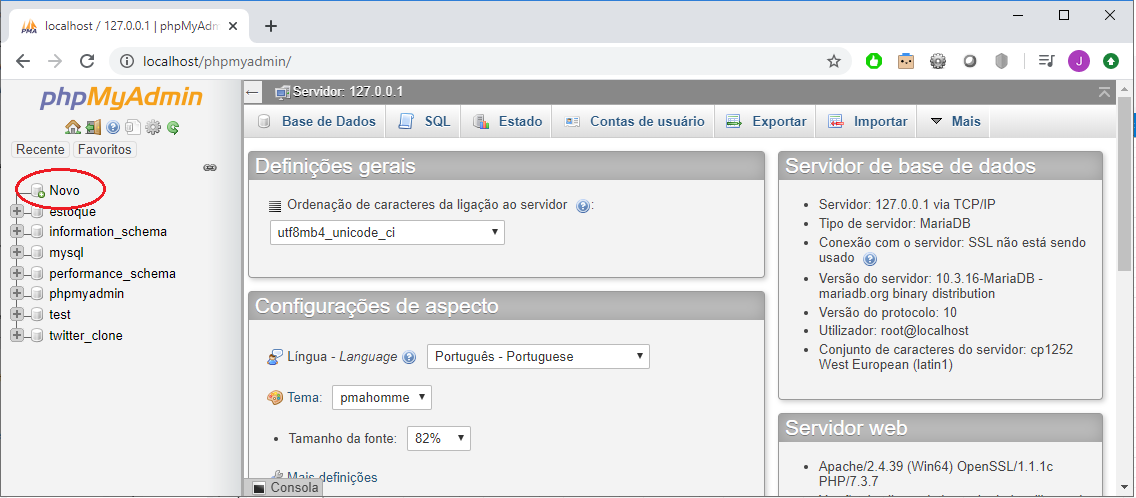
1. MySQL Client que é uma ferramenta de linha de comando, geralmente instalada junto com o MySQL;
2. phpMyAdmin que é um gerenciador via web e, portanto, exige a instalação de um Servidor Web para seu funcionamento;
3. MySQL Workbench que é um cliente com interface gráfica que além de permitir a administração do Banco, permite a modelagem do Banco.

A maioria dos Ambientes de Desenvolvimento Web PHP para Windows tais como o Xampp e o WampServer já vêm com o phpMyAdmin instalado. O MySQL Client é instalado junto com o MySQL e o MySQL Workbench possui uma versão gratuita (Community) do site <https://www.mysql.com/products/workbench/>

Aqui será exemplificado como criar o Banco de Dados e as Tabelas da Aplicação pelo phpMyAdmin no ambiente Xampp. Para tanto, abra um Navegador e digite <http://localhost>. Clique em phpMyAdmin no canto superior direito.

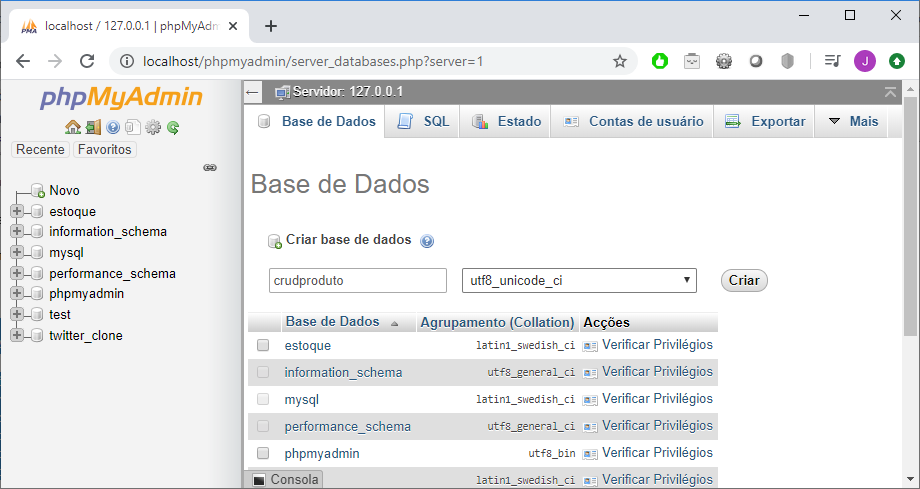


O phpMyAdmin vai conectar ao MySQL local usando o usuário root sem senha, o qual é o padrão da instalação do Xampp. Caso não consiga conectar, verifique se o MySQL foi iniciado. Caso tenha colocado uma senha para o usuário root, utilize-a.

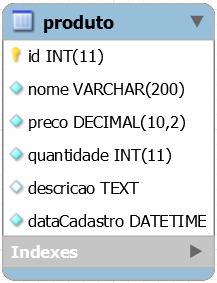


Primeiramente, crie o Banco de Dados, clicando em **Novo** do lado esquerdo na página, onde são mostrados os Bancos já existentes.

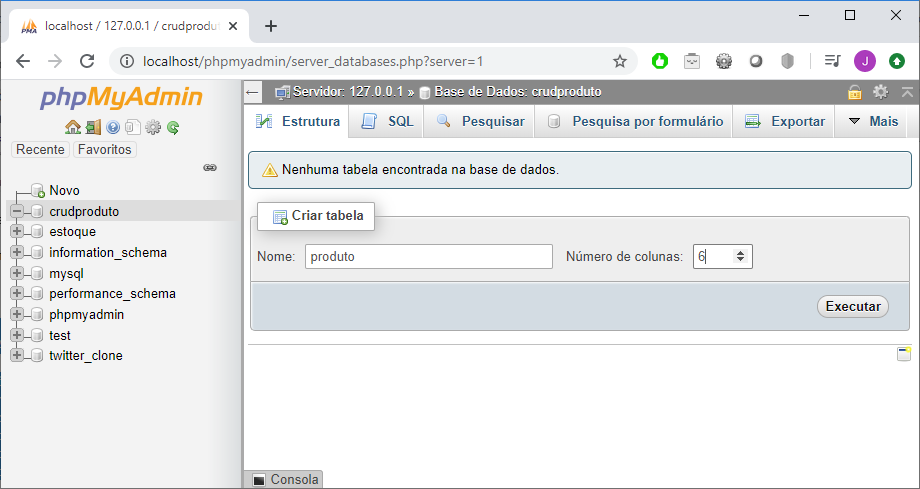
Na página que se abre, na caixa de texto Nome da base de dados, coloque o nome **crudproduto** que será o nome do Banco de Dados da nossa Aplicação. Na caixa de seleção da frente selecione **utf8\_unicode\_ci** para tipo de collation (codificação dos caracteres). Clique no botão criar.



Após esse passo, o Banco já aparece na relação do lado esquerdo e do lado direito já é mostrada a interface para criar uma Tabela. Vamos criar uma tabela chamada produto com 6 colunas (campos) com a seguinte estrutura:

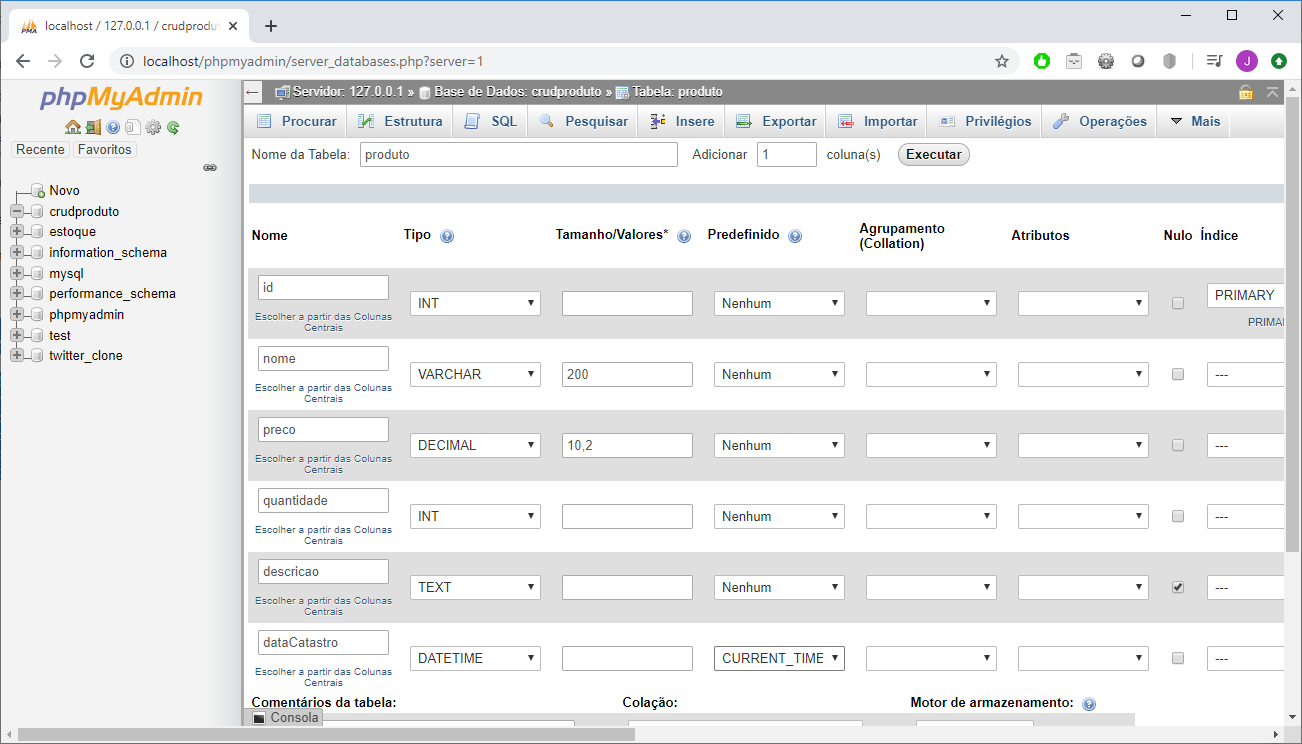


Coloque o Nome produto e em Número de colunas coloque 6. Depois clique no botão Executar.



Abre-se uma nova página para especificar o nome e demais características de cada coluna. Note que são muitos campos de informações sobre cada coluna: Nome, Tipo, Tamanho/Valores, Predefinido, Agrupamento(Collation), Atributos, Nulo, Índice, A\_I Comentários, Validade, Mover coluna(s), Tipo MIME, Transformação de visualização do navegador, Opções de transformação de visualização do navegador, Transformação de entrada e Opções de transformação de dado. Entretanto, nem todos precisam ser preenchidos. Siga a tabela abaixo para criar os campos. Após preencher os campos, clique no botão Guarda.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nome | Tipo | Tamanho/Valores | Predefinido | Nulo | Índice | A\_I |
| id | INT |  |  |  | PRIMARY | SIM |
| nome | VARCHAR | 200 |  |  |  |  |
| preco | DECIMAL | 10,2 |  |  |  |  |
| quantidade | INT |  |  |  |  |  |
| descricao | TEXT |  |  | SIM |  |  |
| dataCadastro | DATETIME |  | CURRENT\_TIMESTAMP |  |  |  |



Outra forma é utilizar comandos SQL para criar a tabela. Para isso, marque o Banco crudproduto do lado esquerdo. Depois clique na aba SQL do lado direito da interface do phpMyAdmin. Digite (ou cole) os seguintes comandos:

USE crudproduto;

CREATE TABLE produto

(

id INT PRIMARY KEY AUTO\_INCREMENT,

nome VARCHAR(200) NOT NULL,

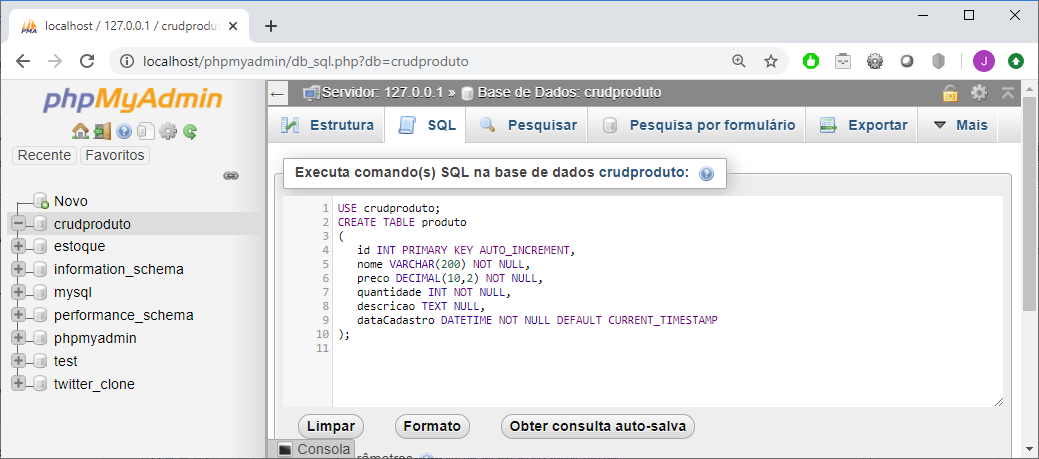
preco DECIMAL(10,2) NOT NULL,

quantidade INT NOT NULL,

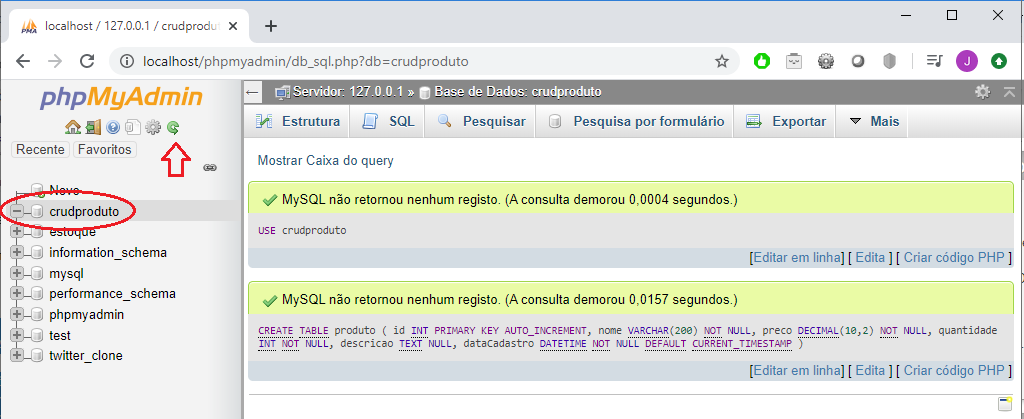
descricao TEXT NULL,

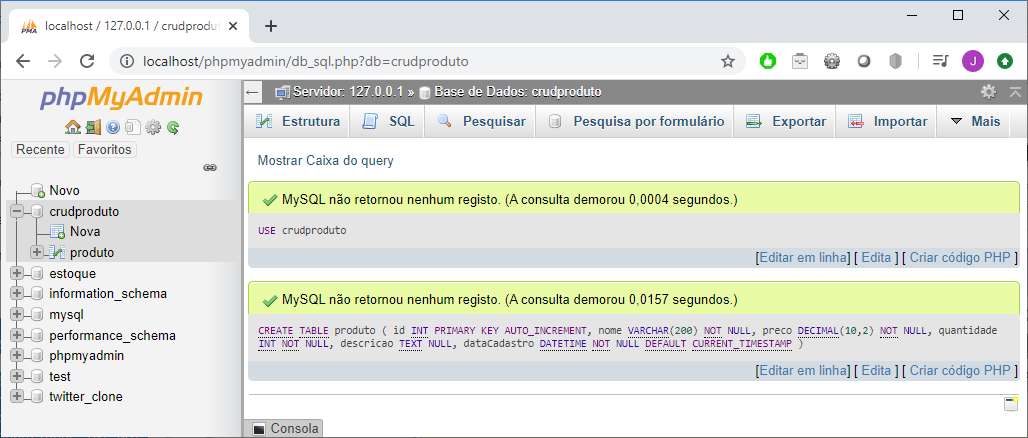
dataCadastro DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP

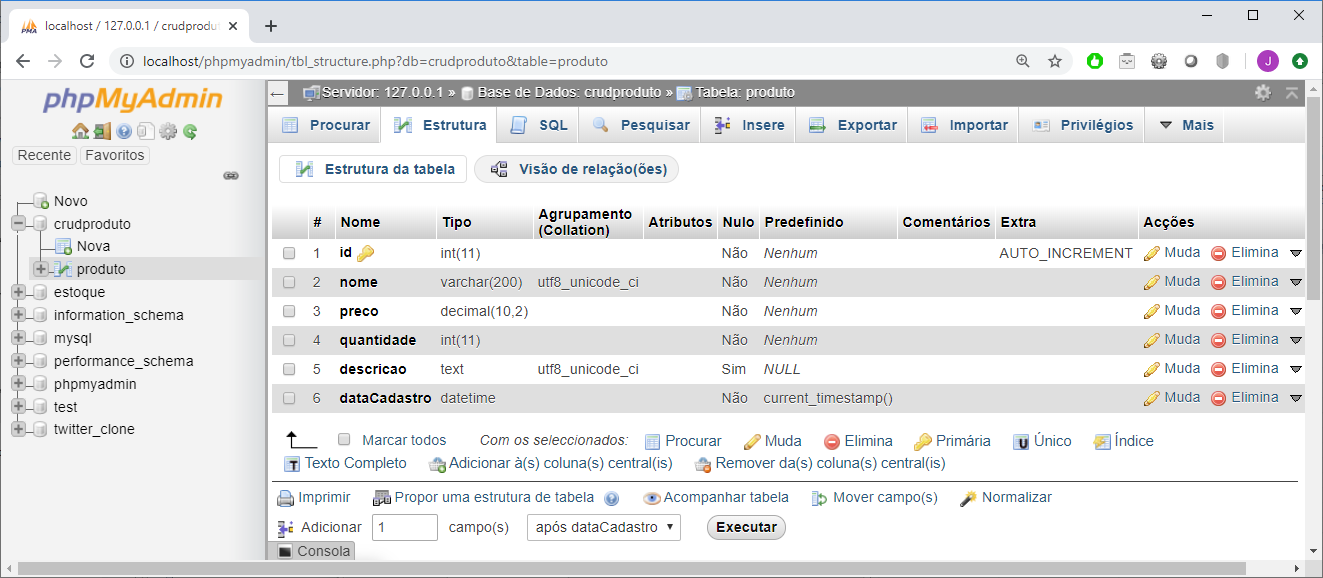
);



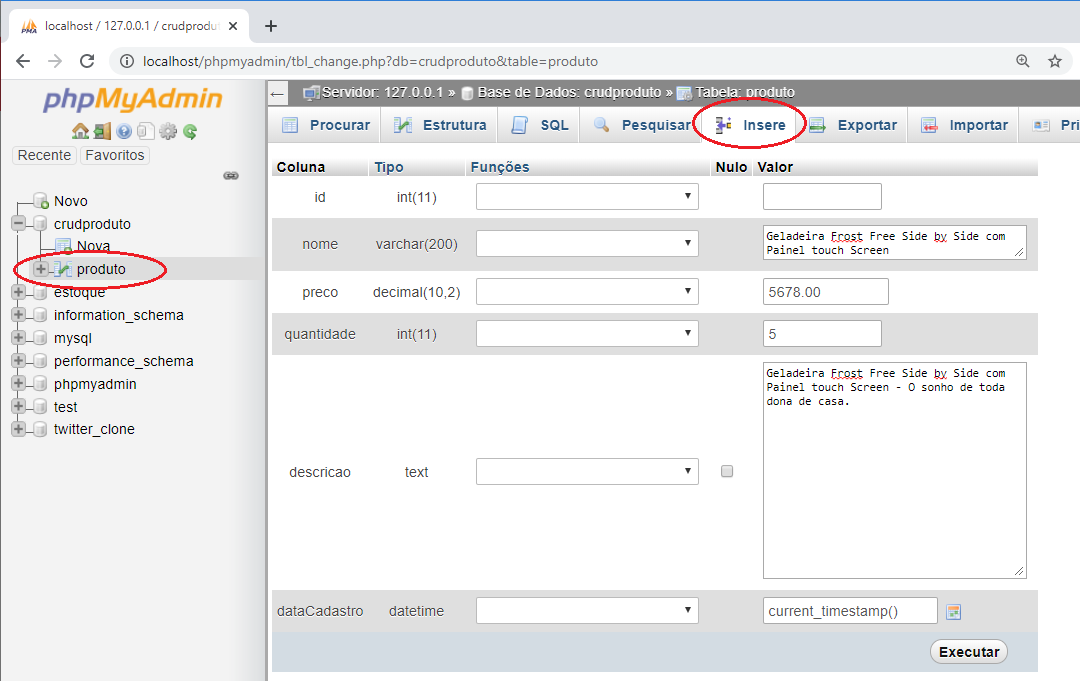
Após criar a tabela, para que apareça no lado esquerdo abaixo do banco, clicar no ícone recarregar para atualizar.



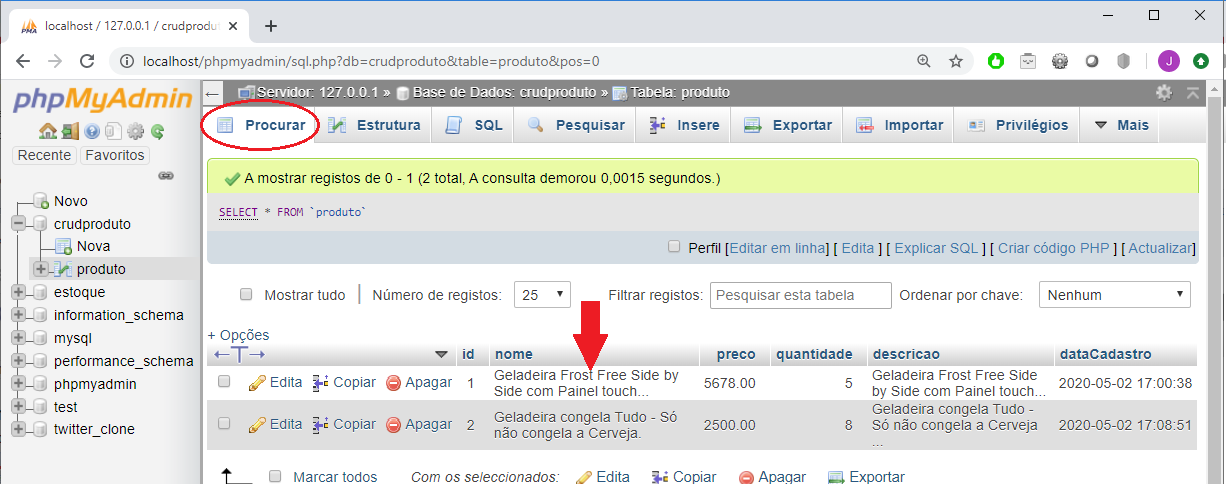




Através do phpMyAdmin é possível popular as tabelas do Banco de Dados. Isso é feito marcando a tabela a qual quer inserir dados (produto) e clicando na aba Insere do lado direito da interface. Preencha com os dados do produto e clique no botão Executar no canto inferior direito. Digite os dados e clique em Executar. Não é necessário digitar o id nem a dataCadastro, pois o id é Auto Incremento e a data será preenchida automaticamente com a data e hora do sistema no momento do cadastro.



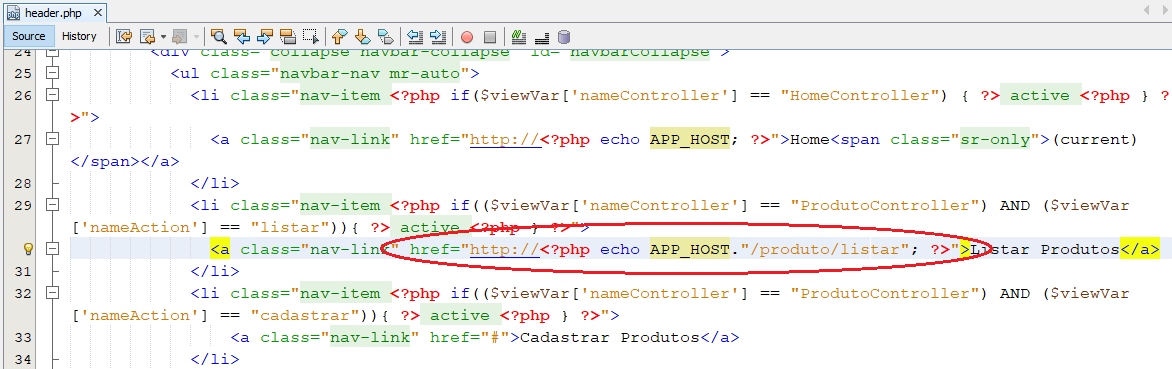
Após o cadastro, é possível ver os produtos cadastrados clicando na aba Procurar.



* 1. **Implementando a função Listar Produtos**

Primeiramente é necessário ativar o link para acionar a função Listar Produtos no menu navbar. Seguindo nossa estrutura MVC, edite o arquivo header.php e coloque no href da tag <a> do item Listar Produto do navbar o seguinte:

href="http://<?php echo APP\_HOST."/produto/listar"; ?>"



Note que a URL completa será: <http://localhost/CRUD-ProdutoMVC/produto/listar>, pois APP\_HOST é uma constante definida na classe App.php e de acordo com a estrutura MVC produto identifica o Controller e listar identifica a Action.

Para entender a dinâmica de funcionamento da estrutura MVC de nossa Aplicação, vamos implementar uma versão para teste do Controlller e da Action. Então, crie um arquivo tipo classe PHP dentro da pasta App/Controllers com o nome ProdutoController.php e coloque o seguinte conteúdo.

\App\Controllers\ProdutoController.php

<?php

namespace App\Controllers;

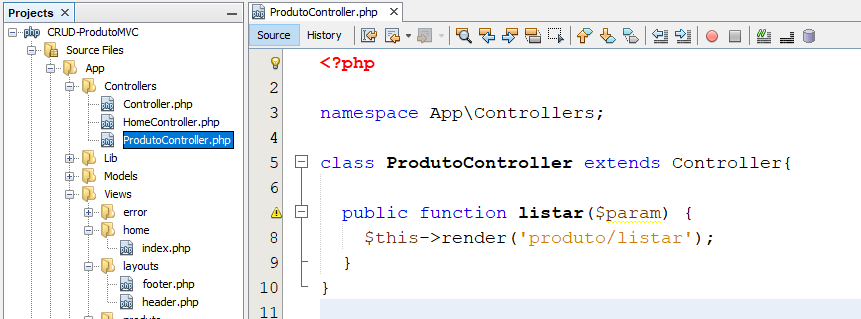
class ProdutoController extends Controller{

public function listar($param) {

$this->render('produto/listar');

}

}



Agora crie uma pasta chamada produto dentro da pasta App/Views. Dentro desta pasta produto, crie um arquivo tipo PHP web page com o nome listar.php com o seguinte conteúdo.

\App\Views\produto\listar.php

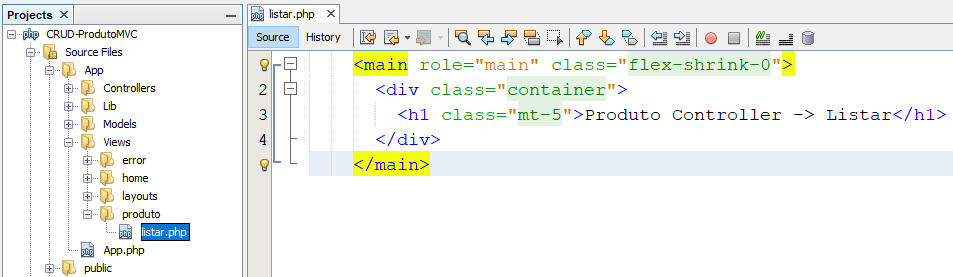
<main role="main" class="flex-shrink-0">

<div class="container">

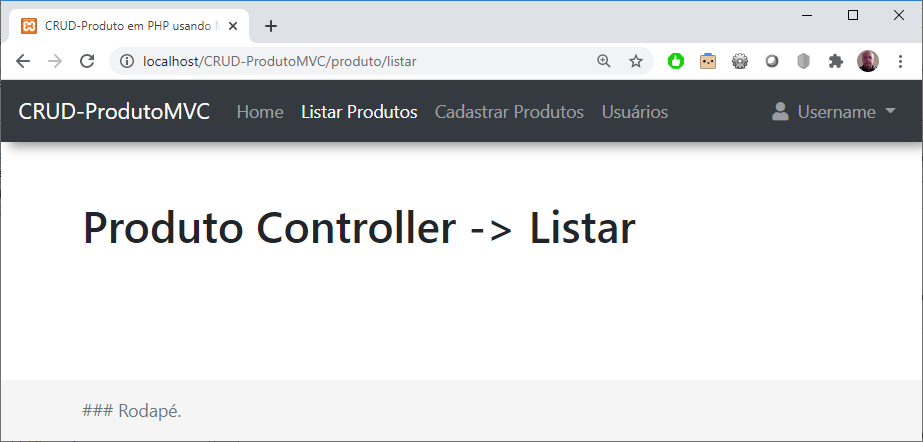
<h1 class="mt-5">Produto Controller -> Listar</h1>

</div>

</main>



Salve os arquivos e teste. Clique na opção Listar Produtos do menu.



Este teste mostra que a dinâmica da Aplicação no modelo MVC está fluindo corretamente. Note o padrão da URL para chamar a função Listar Produtos (localhost/CRUD-ProdutoMVC/**produto/listar**). Este será também o padrão para as demais funcionalidades relacionadas à manutenção de Produtos (produto/cadastrar para o Cadastrar, produto/editar e produto/excluir).

Agora vamos implementar de fato a Listagem de Produtos. Os passos para isso podem ser resumidos em:

1. Conectar com o Banco;
2. Selecionar/buscar as informações (produtos);
3. Montar a página com as informações que serão exibidas para o usuário.

**11.2.1 A Classe Conexão**

Para conectar com o Banco, vamos criar uma classe. Assim, na pasta APP/Lib crie um arquivo tipo classe PHP com o nome Conexao.php com o seguinte código.

\App\Lib\Conexao.php

<?php

namespace App\Lib;

use PDO;

use PDOException;

use Exception;

class Conexao

{

private static $connection;

private function \_\_construct(){}

public static function getConnection() {

$pdoConfig = DB\_DRIVER . ":". "host=" . DB\_HOST . ";";

$pdoConfig .= "dbname=".DB\_NAME.";";

try {

if(!isset(self::$connection)){

self::$connection = new PDO($pdoConfig, DB\_USER, DB\_PASSWORD);

self::$connection->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

}

return self::$connection;

} catch (PDOException $e) {

throw new Exception("Erro de conexão com o banco de dados",500);

}

}

}

Note que o acesso ao Banco será feito via PDO (Php Data Objects). Todas as constantes (DB\_DRIVER, DB\_HOST, DB\_NAME, DB\_USER e DB\_PASSWORD) foram declaradas na classe App\App.php.

Note também que o atributo $connection é do tipo static. Isso significa que o atributo pertence à classe e não a uma determinada instância dela, podendo ser acessado sem necessidade de instanciar um objeto. Para acesso ao atributo static devemos utilizar o operador self:: e não o $this->. O mesmo vale para o método getConnection() que também é static.

Por último, veja que foi criado um método construtor vazio do tipo private. Isso é para NÃO permitir que a classe seja instanciada. Ou seja, a classe conexão NÃO poderá ser instanciada. Ela será utilizada pela classe BaseDAO que veremos logo adiante. Isso garante um ponto único de acesso à conexão. Conceito este chamado classe singleton, o que garante maior segurança de acesso ao banco de dados, pois será mantido somente uma conexão, evitando a abertura de vários conexões o que poderia acarretar em estouro de pilha (*stack overflow*).

**11.2.2 A Classe BaseDAO**

É nesta classe que tratamos a persistência ao Banco de Dados via métodos do PDO. Crie uma pasta chamada DAO dentro de APP/Models/DAO. Dentro desta pasta, crie um arquivo tipo classe PHP com o nome BaseDAO.php com o seguinte conteúdo.

\App\Models\DAO\BaseDAO

<?php

namespace App\Models\DAO;

use App\Lib\Conexao;

abstract class BaseDAO

{

private $conexao;

public function \_\_construct()

{

$this->conexao = Conexao::getConnection();

}

public function select($sql)

{

if(!empty($sql))

{

return $this->conexao->query($sql);

}

}

public function insert($table, $cols, $values)

{

if(!empty($table) && !empty($cols) && !empty($values))

{

$parametros = $cols;

$colunas = str\_replace(":", "", $cols);

// $table $colunas $cols

//INSERT INTO usuario (nome,email) VALUES (:nome,:email);

$stmt = $this->conexao->prepare("INSERT INTO $table ($colunas) VALUES ($parametros)");

$stmt->execute($values);

return $stmt->rowCount();

}else{

return false;

}

}

public function update($table, $cols, $values, $where=null)

{

if(!empty($table) && !empty($cols) && !empty($values))

{

if($where)

{

$where = " WHERE $where ";

}

$stmt = $this->conexao->prepare("UPDATE $table SET $cols $where");

$stmt->execute($values);

return $stmt->rowCount();

}else{

return false;

}

}

public function delete($table, $where=null)

{

if(!empty($table))

{

/\*

DELETE produto WHERE id = 1

\*/

if($where)

{

$where = " WHERE $where ";

}

$stmt = $this->conexao->prepare("DELETE FROM $table $where");

$stmt->execute();

return $stmt->rowCount();

}else{

return false;

}

}

}

Note que é uma classe abstrata (abstract), ou seja, não pode ser instanciada. Será usada pelas classes filhas (ProdutoDAO, UsuarioDAO etc.) que criaremos mais adiante. O construtor da classe cria uma conexão com o Banco de Dados através do método getConnection() da classe Conexao que criamos anteriormente (por isso que demos um use App\Lib\Conexao). Na sequência temos os métodos select(), insert(), update() e delete(), os quais são responsáveis pelas quatro operações feitas em um banco de dados. Estes métodos foram criados de maneira que possam ser chamados para acesso a qualquer tabela do banco, seja para acesso à tabela de produtos, ou para a tabela usuários etc.

O PDO utiliza os métodos prepare() e execute(), os quais se baseiam no conteúdo das variáveis $table, $cols e $values para formarem as strings SQL. Por exemplo, considere que o método insert() tenha sido chamado para inserir um novo produto, veja abaixo como ficaria a string SQL montada pelo método prepare() utilizando as variáveis.

// $table $colunas $cols

//INSERT INTO produto (nome,descricao,preco,quantidade) VALUES (:nome,:descricao, :preco, :quantidade);

$stmt = $this->conexao->prepare("INSERT INTO $table ($colunas) VALUES ($parametros)");

Ou seja:

Nome da Tabela: $table = produto

Nome das colunas da Tabela: $colunas = nome,descricao,preco,quantidade

Rótulos (nome das colunas com sinal :): $cols = :nome,:descricao, :preco, :quantidade

$values Valores a serem inseridos no lugar dos rótulos no formato:

:nome=>”Geladeira Gela Bem”,

:descrição=>”Geladeira da marca Gela bem mas não congela a cerva”,

:preco=>“3500,00”,

:quantidade=>”3”

**11.2.3 A Classe ProdutoDAO**

A classe ProdutoDAO será Filha da classe BaseDAO e responsável por implementar o acesso à tabela de produtos do banco de dados. Crie um arquivo tipo classe PHP com o nome ProdutoDAO.php na pasta App/Models/DAO, com o seguinte conteúdo.

\App\Models\DAO\ProdutoDAO.php

<?php

namespace App\Models\DAO;

use App\Models\Entidades\Produto;

class ProdutoDAO extends BaseDAO

{

public function listar($id = null)

{

if($id) {

$resultado = $this->select(

"SELECT \* FROM produto WHERE id = {$id}"

);

return $resultado->fetchObject(Produto::class);

}else{

$resultado = $this->select(

'SELECT \* FROM produto'

);

return $resultado->fetchAll(\PDO::FETCH\_CLASS, Produto::class);

}

return false;

}

public function salvar(Produto $produto)

{

try {

$nome = $produto->getNome();

$preco = $produto->getPreco();

$quantidade = $produto->getQuantidade();

$descricao = $produto->getDescricao();

return $this->insert(

'produto',

":nome,:preco,:quantidade,:descricao",

[

':nome'=>$nome,

':preco'=>$preco,

':quantidade'=>$quantidade,

':descricao'=>$descricao

]

);

}catch (\Exception $e){

throw new \Exception("Erro na gravação de dados.", 500);

}

}

public function atualizar(Produto $produto)

{

try {

$id = $produto->getId();

$nome = $produto->getNome();

$preco = $produto->getPreco();

$quantidade = $produto->getQuantidade();

$descricao = $produto->getDescricao();

return $this->update(

'produto',

"nome = :nome, preco = :preco, quantidade = :quantidade, descricao = :descricao",

[

':id'=>$id,

':nome'=>$nome,

':preco'=>$preco,

':quantidade'=>$quantidade,

':descricao'=>$descricao,

],

"id = :id"

);

}catch (\Exception $e){

throw new \Exception("Erro na gravação de dados.", 500);

}

}

public function excluir(Produto $produto)

{

try {

$id = $produto->getId();

return $this->delete('produto',"id = $id");

}catch (Exception $e){

throw new \Exception("Erro ao deletar", 500);

}

}

}

**11.2.4 A Classe Produto**

Esta classe modela a entidade Produto. Crie uma pasta chamada Entidades dentro da pasta App/Models. Depois, dentro da pasta Entidades, crie um arquivo tipo classe PHP de nome Produto.php com o seguinte conteúdo.

\App\Models\Entidades\Produto.php

<?php

namespace App\Models\Entidades;

use DateTime;

class Produto

{

private $id;

private $nome;

private $preco;

private $quantidade;

private $descricao;

private $dataCadastro;

public function getId()

{

return $this->id;

}

public function setId($id)

{

$this->id = $id;

}

public function getNome()

{

return $this->nome;

}

public function setNome($nome)

{

$this->nome = $nome;

}

public function getPreco()

{

return $this->preco;

}

public function setPreco($preco)

{

$this->preco = $preco;

}

public function getQuantidade()

{

return $this->quantidade;

}

public function setQuantidade($quantidade)

{

$this->quantidade = $quantidade;

}

public function getDescricao()

{

return $this->descricao;

}

public function setDescricao($descricao)

{

$this->descricao = $descricao;

}

public function getDataCadastro()

{

return new DateTime($this->dataCadastro);

}

public function setDataCadastro($dataCadastro)

{

$this->dataCadastro = $dataCadastro;

}

}

**11.2.5 Refazendo a Classe ProdutoControlller**

O objetivo agora é implementar a Listagem de Produtos. Para tanto, é preciso implementar o acesso ao Banco para buscar os dados no método listar() dentro do ProdutoController. Depois chamamos a view produto/listar e passamos os dados dos produtos para ela exibir.

Vamos implementar nossa action listar. Troque o conteúdo da classe ProdutoController.php pelo seguinte.

\App\Controllers\ProdutoController.php

<?php

namespace App\Controllers;

use App\Lib\Sessao;

use App\Models\DAO\ProdutoDAO;

use App\Models\Entidades\Produto;

class ProdutoController extends Controller

{

public function listar()

{

$produtoDAO = new ProdutoDAO(); //Conecta ao Banco

self::setViewParam('listaProdutos',$produtoDAO->listar());//busca os dados

$this->render('/produto/listar'); //Passa os dados p/ a view listar

Sessao::limpaMensagem();

}

}

Note na classe acima que é instanciada a classe ProdutoDAO, a qual conecta no Banco, depois é chamado o método listar() da classe ProdutoDAO ($produtoDAO->listar()), o qual vai buscar todos os produtos no banco. Esse resultado será colocado no atributo $viewVar da classe mãe Controller.php pelo método setViewParam(). Por último, é chamado o método render(), também da classe mãe Controller.php, o qual vai montar a página de saída a qual exibirá os produtos ao usuário.

**11.2.6 Refazendo a view listar produto**

Para testarmos o funcionamento da funcionalidade Listar Produtos, vamos implementar uma versão simples da view listar. Troque o conteúdo do arquivo listar.php pelo seguinte:

\App\Views\produto\listar.php

<main role="main" class="flex-shrink-0">

<div class="container">

<h1 class="mt-5">Produto Controller -> Listar</h1>

<?php

echo '<PRE>';

print\_r($viewVar);

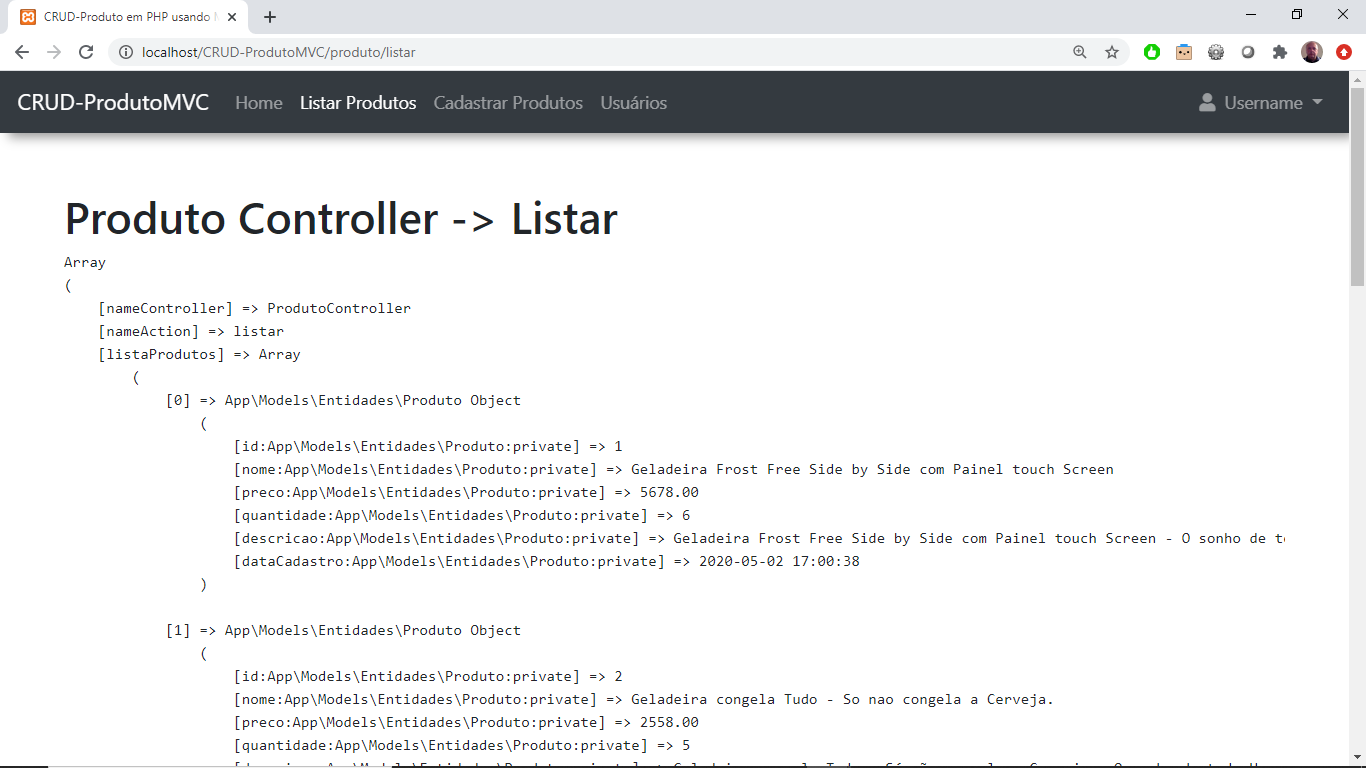
echo '</PRE>';

?>

</div>

</main>

Salve todos os arquivos e teste a Aplicação. Ao clicar no item Listar Produtos do menu, serão listados todos os produtos cadastrados, como mostrado na Figura a seguir.



Note pelo código anterior que a view listar mostrou os produtos através do array associativo $viewVar. Esta é a forma de troca de dados entre o Controlador e a View.

Agora que vimos que a comunicação Controlador-View está funcionando, vamos exibir os dados adequadamente. Troque o conteúdo do arquivo listar.php pelo seguinte:

\App\Views\produto\listar.php

<main role="main" class="flex-shrink-0">

<div class="container">

<h1 class="mt-5">Listagem de Produtos</h1>

<?php

//Mensagens de Erro ou Sucesso na execução das funções

echo $Sessao::retornaMensagem();

$Sessao::limpaMensagem();

if (count($viewVar['listaProdutos'])>0){

echo '<div class="table-responsive">';

echo ' <table class="table table-bordered table-hover table-sm">';

echo ' <thead >';

echo ' <tr style="background-color: #bee5eb;">';

echo ' <th class="info">Id</th>';

echo ' <th class="info">Nome</th>';

echo ' <th class="info">Descrição</th>';

echo ' <th class="info">Preço</th>';

echo ' <th class="info">Qtde.</th>';

echo ' <th class="info">Cadastro</th>';

echo ' <th class="info"></th>';

echo ' </tr>';

echo ' </thead>';

echo ' <tbody>';

foreach ($viewVar['listaProdutos'] as $objProduto) {

$id = $objProduto->getId();

$nome = $objProduto->getNome();

$preco = $objProduto->getPreco();

$qtde = $objProduto->getQuantidade();

$descricao = $objProduto->getDescricao();

$dataCadastro = ($objProduto->getDataCadastro())->format('d/m/Y');

echo '<tr>';

echo ' <td>'.$id.'</td>';

echo ' <td>'.$nome.'</td>';

echo ' <td>'.$descricao.'</td>';

echo ' <td>'.$preco.'</td>';

echo ' <td>'.$qtde.'</td>';

echo ' <td>'.$dataCadastro.'</td>';

echo ' <td> <a href="http://'.APP\_HOST.'/produto/editar/'.$id.'" class="btn btn-info btn-sm">Editar</a>

<a href="http://'.APP\_HOST.'/produto/excluirConfirma/'.$id.'/'.$nome.'" class="btn btn-danger btn-sm mt-1">Excluir</a>';

echo '</tr>';

}

echo ' </tbody>';

echo ' </table>';

echo '</div>';

}else {

echo "Nenhum Produto Encontrado.";

}

?>

</div>

</main>

Note que o array $viewVar['listaProdutos'] é um array de objetos da classe Produto.php. Assim, devemos utilizar os métodos da classe para acesso a cada um de seus atributos, os quais são private.

Note também a programação do link href="http://'.APP\_HOST.'/produto/editar/'.$id.'" no botão Editar. Aqui estamos utilizando o esquema de URL amigável. Neste caso, produto indica o Controlador, editar o Action (método) e $id contém o id do produto a ser editado.

* 1. **Implementando a função Editar Produto**

Esta funcionalidade da Aplicação será ativada a partir do botão Editar na Listagem de Produto implementada anteriormente. Como visto na seção anterior, ao clicar no botão Editar, será enviada uma URL contendo /produto/editar/id, onde:

* **produto** indica o Controlador que deve ser acionado;
* **editar** indica o método (action) dentro do Controlador que deve ser executado;
* **id** será o identificador do produto que deve ser editado.

Seguindo nossa estrutura MVC, devemos:

1. Implementar o método editar dentro da classe ProdutoController.php, o qual vai buscar no banco os dados do produto a ser editado e repassá-los para a view editar;
2. Implementar a view editar.php a qual deverá mostrar o formulário com os dados do produto, aguardar o usuário fazer as alterações e, ao salvar, enviar um pedido para salvar os dados no banco;

Edite o arquivo ProdutoController.php e acrescente o método editar() com o conteúdo a seguir:

\App\Controllers\ProdutoController.php

public function editar($params)

{

$id = $params[0]; //Pega o id do produto a ser editado

$produtoDAO = new ProdutoDAO();

$objProduto = $produtoDAO->listar($id);

if ($objProduto==null) //Se NÃO achou produto

{

Sessao::gravaMensagem('<div class="alert alert-danger" role="alert">Falha ao recuperar dados do produto id='.$id.'</div>');

$this->redirect('/produto/listar');

}

self::setViewParam('produto',$objProduto);

$this->render('/produto/editar');

Sessao::limpaMensagem();

}

No código acima, primeiro recuperamos o id do produto a ser editado que vem no array indexado $params. Depois acessamos o Banco e buscamos os dados do produto a ser editado (via método listar() da classe ProdutoDAO). Note que será retornado um objeto da classe Produto(). Se por algum motivo a busca falhar, montamos um mensagem de erro e redirecionamos para a View listar(). Os dados do produto são passados para o $viewVar da classe mãe Controller(), via método setViewParam(). Por último executamos o método render() da classe mãe passando para ele a view que deve ser chamada.

Agora vamos criar a view que mostrará o formulário com os dados do produto a ser editado. Crie um arquivo tipo PHP Web Page com o nome editar.php em App/Views/produto com o seguinte código:

\App\Views\produto\editar.php

<main role="main" class="flex-shrink-0">

<div class="container">

<div class="row">

<div class="col-md-3"></div>

<div class="col-md-6">

<h1 class="mt-2">Editar dados do Produto</h1>

<?php

//Mensagens de Erro ou Sucesso na execução das funções

echo $Sessao::retornaMensagem();

$Sessao::limpaMensagem();

?>

<form action="<?php echo 'http://'.APP\_HOST.'/produto/salvar/editar';?>" method="post" id="formEditarProduto">

<div class="form-group">

<input type="hidden" name="id" value="<?php echo $viewVar['produto']->getId();?>">

</div>

<div class="form-group">

<label for="nome">Nome do Produto</label>

<input type="text" class="form-control" name="nome" value="<?php echo $viewVar['produto']->getNome(); ?>" required>

</div>

<div class="form-group">

<label for="descricao">Descrição</label>

<textarea class="form-control" name="descricao"><?php echo $viewVar['produto']->getDescricao(); ?>

</textarea>

</div>

<div class="form-group">

<label for="preco">Preço</label>

R$ <input type="text" class="form-control <?php if ($Sessao::retornaErro('erropreco')!="") echo "is-invalid"; ?>" name="preco" value="<?php echo $viewVar['produto']->getPreco();?>">

<div class="invalid-feedback">

<?php echo $Sessao::retornaErro('erropreco'); $Sessao::limpaErro(); ?>

</div>

</div>

<div class="form-group">

<label for="quantidade">Quantidade</label>

<input type="number" class="form-control" name="quantidade" value="<?php echo $viewVar['produto']->getQuantidade();?>" required>

</div>

<button type="submit" class="btn btn-success btn-sm">Salvar</button>

</form>

</div>

<div class=" col-md-3"></div>

</div>

</div>

</main>

**11.3.1. Salvando os dados do produto editado**

Ao clicar no botão Salvar do formulário de edição do Produto, programamos (form action="<?php echo 'http://'.APP\_HOST.'/produto/salvar/editar';?>") para executar o método salvar() do Controlador ProdutoController.php . O método salvar() receberá como parâmetro a palavra editar. Isso porque o método salvar() será utilizado em duas situações: para salvar os dados de um produto editado e também para salvar os dados de um novo produto. Assim, o parâmetro editar indicará ao método qual situação deverá ser executada.

Portanto, acrescente no final do ProdutoController.php o método salvar() com o seguinte conteúdo:

\App\Controllers\ProdutoController.php

public function salvar($param) {

$cmd = $param[0]; //Pega o comando: editar ou novo

//Sanitização dos dados do Formulário

$dadosform = Util::sanitizar($\_POST);

$objproduto = new Produto();

//Transfere os dados do Produto do Formulário para o Objeto

$objproduto->setProduto($dadosform);

$errovalidacao = false;

//Aplicar a Validação dos Dados segundo as regras de negócio

//Aqui pode-se criar uma classe separada de Validação

if (empty($dadosform['preco'])) {

Sessao::gravaMensagem('<div class="alert alert-danger" role="alert">Verifique os Campos em Vermelho.</div>');

//Habilita o 'is-invalide' do feedback do campo preço do Form

Sessao::gravaErro('erropreco','Este campo deve ser preemchido');

$errovalidacao = true;

}

if ($errovalidacao) { //Houve erro na validacao

//Guarda os dados do POST na viewVar para reapresentar os dados

self::setViewParam('produto',$objproduto);

if ($cmd=='editar'){ //O produto está sendo editado

$this->render('/produto/editar');//Retorna ao Formulário de edição

}elseif ($cmd == 'novo'){ //O produto está sendo cadastrado

$this->render('/produto/cadastrar');//Retorna ao Formulário de cadastro de novo produto

}

die(); //Isso é necessário senão ele vai continuar e cadastrar o produto!!!

}

//Se passou pela validação sem erros, continua aqui

$produtoDAO = new ProdutoDAO(); //Conecta no Banco

if ($cmd=='editar'){ //Salvar produto editado

$produtoDAO->atualizar($objproduto);

Sessao::gravaMensagem('<div class="alert alert-success" role="alert">Produto atualizado com sucesso.</div>');

}elseif ($cmd == 'novo'){ //Salvar novo produto

$produtoDAO->salvar($objproduto);

Sessao::gravaMensagem('<div class="alert alert-success" role="alert">Novo Produto gravado com sucesso.</div>');

}

//Limpa Tudo

Sessao::limpaErro();

//Redireciona para o listar que vai exibir msg de feedback

$this->redirect('/produto/listar');

}

Ainda no ProdutoController.php, acrescente no seu início o seguinte código:

use App\Lib\Util;

Na classe App/Models/Entidades/Produto.php acrescente no final o método setProduto() com o seguinte conteúdo:

public function setProduto($dados) {

$this->id = $dados['id'];

$this->nome = $dados['nome'];

$this->preco = $dados['preco'];

$this->quantidade = $dados['quantidade'];

$this->descricao = $dados['descricao'];

$this->dataCadastro = $dados['dataCadastro'];

}

Na classe App/Lib/Sessao.php troque os métodos gravaErro() e retornaErro() pelos seguintes:

public static function gravaErro($key,$msg) {

$\_SESSION['erro'][$key] = $msg;

}

public static function retornaErro($key) {

return (isset($\_SESSION['erro'][$key])) ? $\_SESSION['erro'][$key] : "";

}

Antes de testar, implemente a classe App/Lib/Util do próximo item, a qual contém o método sanitizar(). Depois salve tudo e teste a funcionalidade Editar um Produto.

**11.3.2. A classe App/Lib/Util**

Crie um arquivo tipo classe PHP em App/Lib com o nome Util.php e coloque o seguinte código:

\App\Lib\Util.php

<?php

/\*\*

\* Description of Util

\* Esta classe contém os métodos de utilizade Geral utilizados

\* Em várias outras classes

\*/

namespace App\Lib;

class Util {

/\* sanitizar()

\* ----------

\* Faz uma limpeza nos dados do array ou string para evitar ataques

\* de injection e XSS(Cross-Site Scripting)

\* - O parâmetro de entrada pode ser um array ou uma string

\* - Retorna um array ou uma string, dependendo da entrada

\*/

public static function sanitizar($dados) {

if (is\_array($dados)) {

foreach ($dados as $chave => $valor) {

$valor = filter\_var($valor, FILTER\_SANITIZE\_STRING); //Remove HTML TAGS

$valor = str\_replace("&#x", '', $valor); //Remove início hexadecimal

$valor = stripslashes($valor); //Remove \

$dados[$chave] = trim($valor);

}

} else {

$dados = filter\_var($dados, FILTER\_SANITIZE\_STRING);

$dados = str\_replace("&#x", '', $dados); //replace o prefixo Hexadecimal com nada

$dados = stripslashes($dados);

$dados = trim($dados);

}

return $dados;

}

}

* 1. **Implementando a função Excluir Produto**

Esta funcionalidade da Aplicação será ativada a partir do botão Excluir na Listagem de Produto. Como visto na seção 11.2.6, ao clicar no botão Excluir, será enviada uma URL contendo /produto/excluirConfirma/id/nome, onde:

* **produto** indica o Controlador que deve ser acionado;
* **excluirConfirma** indica o método (action) dentro do Controlador que deve ser executado;
* **id** será o identificador do produto que deve ser excluído;
* **nome** indica o nome do produto que deve ser excluído.

Seguindo nossa estrutura MVC, devemos:

1. Implementar o método excluirConfirma() dentro da classe ProdutoController.php, o qual vai chamar a view App/View/produto/excluirConfirma.php;
2. Implementar a view excluirConfirma.php a qual deverá mostrar o formulário de confirmação da exclusão com o Nome do produto. Ao confirmar, chamar o método excluir() dentro da classe ProdutoController.php;
3. Implementar o método excluir() dentro do ProdutoController.php o qual vai enviar um pedido para excluir o produto no banco;

Edite o arquivo ProdutoController.php e acrescente no final os métodos excluirConfirma()e excluir() com o conteúdo a seguir:

\App\Controllers\ProdutoController.php

public function excluirConfirma($param) //Confirma Exclusão do produto

{

$dados = Util::sanitizar($param); //Pega o id do produto a ser excluído e sanitiza

$objproduto = new Produto();

$objproduto->setId($dados[0]); //Pega o id do produto a ser excluído

$objproduto->setNome($dados[1]); //Pega o nome do produto a ser excluído

if (!is\_numeric($objproduto->getId())){ //Validação

die("Id do produto não é numérico!");

}

self::setViewParam('produto',$objproduto);

$this->render('/produto/excluirConfirma');//Retorna ao Formulário

}

public function excluir($param)

{

$objproduto = new Produto();

//Pega o id do produto a ser excluído

$objproduto->setId(Util::sanitizar($\_POST['id']));

$produtoDAO = new ProdutoDAO();

if(!$produtoDAO->excluir($objproduto)){

Sessao::gravaMensagem('<div class="alert alert-danger" role="alert">Produto Não Encontrado.</div>');

}else{

Sessao::gravaMensagem('<div class="alert alert-success" role="alert">Produto excluído com sucesso!.</div>');

}

$this->redirect('/produto/listar');

}

Crie um arquivo PHP web page excluirConfirma.php em App/Views/produto com o seguinte código:

<main role="main" class="flex-shrink-0">

<div class="container">

<div class="row">

<div class="col-md-3"></div>

<div class="col-md-6">

<h1 class="mt-2">Exclusão de Produto</h1>

<form action="<?php echo 'http://'.APP\_HOST.'/produto/excluir/';?>" method="post" id="formExcluirProduto">

<input type="hidden" name="id" value="<?php echo $viewVar['produto']->getId();?>">

<div class="card text-white bg-danger mb-3" style="max-width: 22 rem;">

<div class="card-header">Confirmação da Exclusão do Produto</div>

<div class="card-body">

<h5 class="card-title">Excluir?</h5>

Confirma exclusão do produto: <?php echo $viewVar['produto']->getNome();?> ?

<button type="submit" class="btn btn-primary btn-sm mt2">Confirmar</button>

<a href="<?php echo 'http://'.APP\_HOST.'/produto/listar/';?>" class="btn btn-info btn-sm mt2">Cancelar</a>

</div>

</div>

</form>

</div>

<div class=" col-md-3"></div>

</div>

</div>

</main>

* 1. **Implementando a função Cadastrar Produtos**

Esta funcionalidade da Aplicação será ativada a partir do item Cadastrar Produtos do menu horizontal.

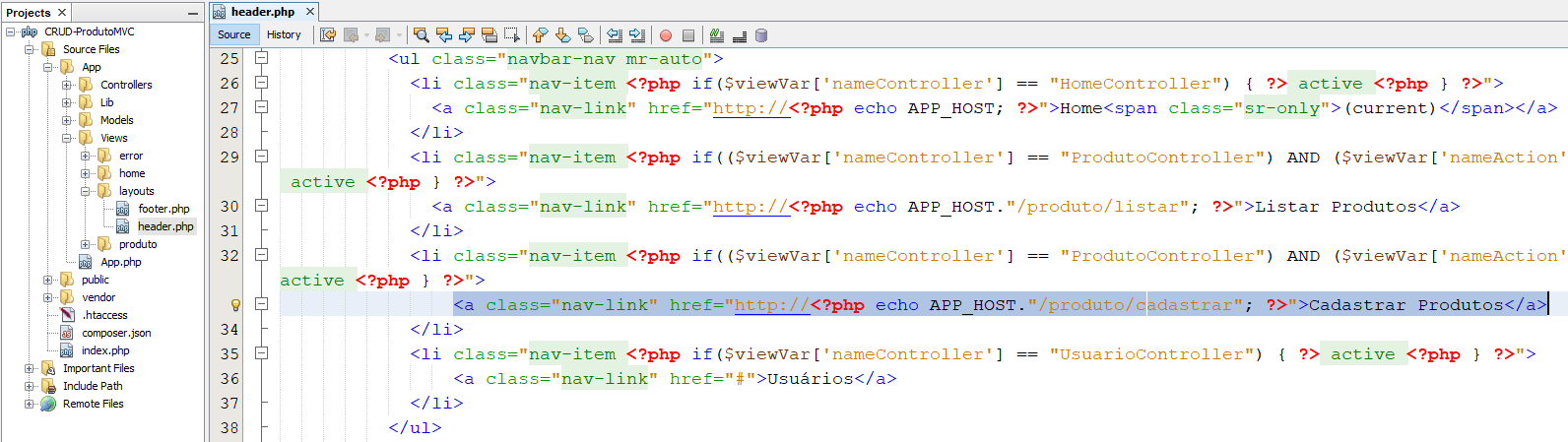
Seguindo nossa estrutura MVC, devemos:

1. No arquivo App/Views/layouts/header.php colocar o link no item do menu horizontal Cadastrar Produtos para chamar o método cadastrar() do ProdutoController (/produto/cadastrar);
2. Implementar o método cadastrar()dentro da classe ProdutoController.php, o qual vai chamar a view App/Views/produto/cadastrar.php;
3. Implementar a view App/Views/produto/cadastrar.php a qual deverá mostrar o formulário de cadastro de um novo produto. Ao clicar no botão Salvar, deverá chamar o método salvar() dentro da classe ProdutoController.php enviando como parâmetro a palavra novo (produto/salvar/novo) para indicar que vai salvar um novo produto;
4. O método salvar() dentro do ProdutoController.php já foi implementado na seção 11.3.1 e sua função é salvar os dados do produto no banco;

Edite o arquivo App/Views/layouts/header.php e troque o item do menu Cadastrar Produto pelo seguinte:

App/Views/layouts/header.php

<a class="nav-link" href="http://<?php echo APP\_HOST."/produto/cadastrar"; ?>">Cadastrar Produtos</a>



Agora acrescente o método cadastrar() no ProdutoController.php com o seguinte código:

public function cadastrar()

{

$this->render('/produto/cadastrar');

Sessao::limpaMensagem();

Sessao::limpaErro();

}

Crie um arquivo PHP Web page em App/Views/produto/cadastrar.php com o seguinte código:

App/Views/produto/cadastrar.php

<main role="main" class="flex-shrink-0">

<div class="container">

<div class="row">

<div class="col-md-3"></div>

<div class="col-md-6">

<h1 class="mt-2">Cadastro de Produto</h1>

<?php

//Mensagens de Erro ou Sucesso na execução das funções

echo $Sessao::retornaMensagem();

$Sessao::limpaMensagem();

?>

<form action="<?php echo 'http://'.APP\_HOST.'/produto/salvar/novo';?>" method="post" id="formCadastro">

<div class="form-group">

<label for="nome">Nome do Produto</label>

<input type="text" class="form-control" name="nome" value="<?php if ($viewVar['produto']) echo $viewVar['produto']->getNome();?>" required>

</div>

<div class="form-group">

<label for="descricao">Descrição</label>

<textarea class="form-control" name="descricao"><?php if ($viewVar['produto']) echo $viewVar['produto']->getDescricao();?></textarea>

</div>

<div class="form-group">

<label for="preco">Preço</label>

R$ <input type="text" class="form-control <?php if ($Sessao::retornaErro('erropreco')!="") echo "is-invalid"; ?>" name="preco" value="<?php if ($viewVar['produto']) echo $viewVar['produto']->getPreco();?>">

<div class="invalid-feedback">

<?php echo $Sessao::retornaErro('erropreco'); $Sessao::limpaErro(); ?>

</div>

</div>

<div class="form-group">

<label for="quantidade">Quantidade</label>

<input type="number" class="form-control" name="quantidade" value="<?php if ($viewVar['produto']) echo $viewVar['produto']->getQuantidade();?>" required>

</div>

<button type="submit" class="btn btn-success btn-sm">Salvar</button>

</form>

</div>

<div class=" col-md-3"></div>

</div>

</div>

</main>