**Aula 3 – CRUD-Produto em Laravel**

**Implementando Acesso ao Banco de Dados**

**Objetivos:**

* Conhecer aspectos da programação back-end da construção de uma aplicação web com Laravel;
* Implementar o acesso ao banco de dados MySQL com Laravel.

Ao final desta Aula devemos ter já em funcionamento o acesso ao Banco de Dados MySQL para a operação de Consulta (Read-Select) para a nossa Aplicação.

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

Antes de iniciarmos o acesso aos dados da Aplicação, 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.

Como estamos utilizando Laravel precisamos somente criar o Banco de Dados pelas ferramentas anteriores. As tabelas podem ser criadas com recursos do próprio Laravel.

A seguir os passos para criar o Banco de Dados da Aplicação via console do MySQL Client:

* Primeiro, iniciar o Banco de Dados. Aqui vamos utilizar o MySQL que vem no Ambiente de Desenvolvimento Xamp. Então execute o Xamp Control Pannel e inicialize o MySQL:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* Agora abra um prompt do sistema e execute o console do MySQL em:

C:\xampp\mysql\bin\mysql.exe -u root

Aqui estou supondo que o MySQL ainda está no padrão de instalação do Xamp, onde o usuário root está sem senha.

Texto

Descrição gerada automaticamente

* Em seguida crie o banco de dados da Aplicação com o comando:

create database dbproduto\_laravel;

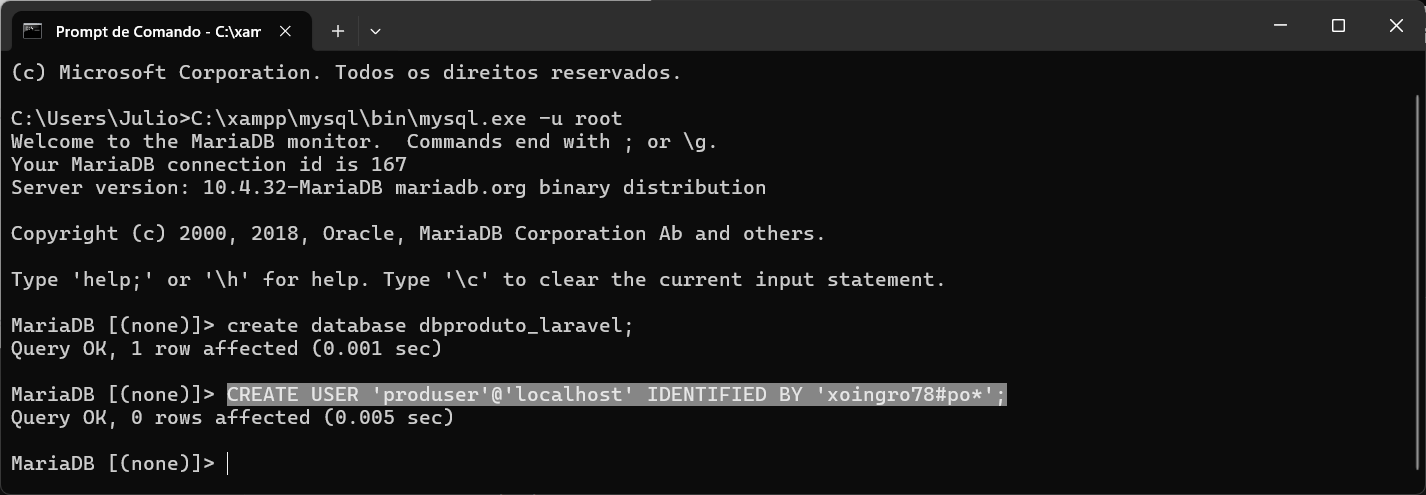
Texto

Descrição gerada automaticamente

* Para ver se o banco foi criado, execute o comando show databases;
* Para sair do console de comandos do MySQL, execute o comando exit;

Agora crie um usuário com privilégios para manipular o banco de dados criado. Este usuário que será utilizado em nossa Aplicação do Laravel para acesso às tabelas desse banco.

CREATE USER 'produser'@'localhost' IDENTIFIED BY 'xoingro78#po\*';



GRANT CREATE,SELECT,INSERT,UPDATE,DELETE,ALTER,DROP, INDEX ON dbproduto\_laravel.\* TO 'produser'@'localhost';

Texto

Descrição gerada automaticamente

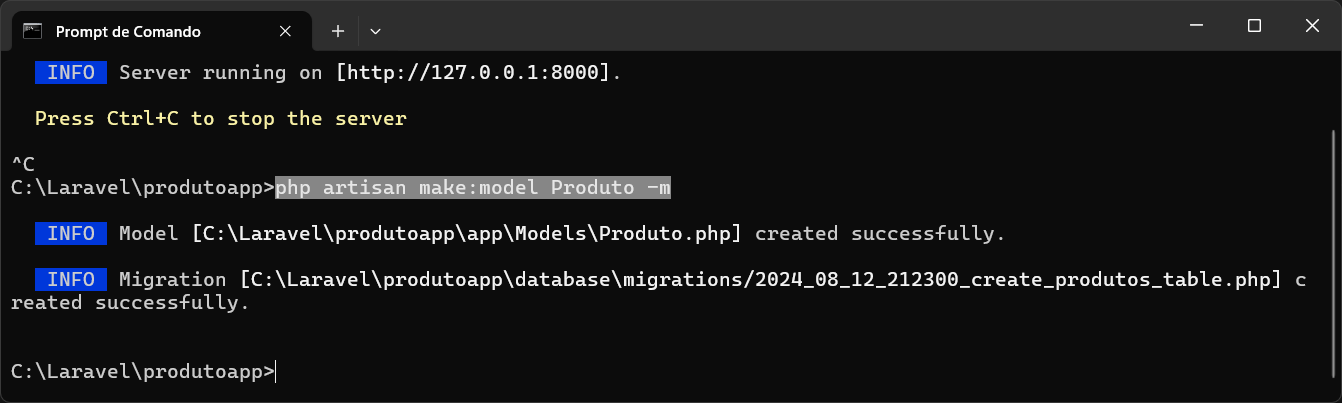
A tabela produtos que armazenará as informações dos produtos será criada via comandos do Laravel.

1. **Criando a Tabela Produtos**

Para criar uma tabela do banco de dados pelo Laravel, primeiro criamos o modelo de dados da tabela e configuramos o Laravel para criar essa tabela no banco de dados. Este processo é denominado de **migration.**

Através do console, execute o seguinte comando do Artisan:

php artisan make:model Produto -m



Este comando cria uma classe chamada Produto na nossa Aplicação e o parâmetro **-m** serve para criar uma migração (migration), a qual servirá para criar a tabela produtos no banco de dados.

Veja em app\Models a classe Produto.php criada:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Veja também em database\migrations a migração 2024\_08\_12\_212300\_create\_produtos\_table.php criada:

Texto

Descrição gerada automaticamente

Note que a migration criada possui dois métodos: up() e down(). O primeiro é para criar a tabela no Banco e o segundo é para desfazer o processo (rollback).

Note também que criamos a classe com a inicial em maiúsculo e no singular (Produto). Já a tabela será criada em minúsculo e no plural (produtos).

Ainda falta determinar os campos (colunas) da tabela. Por default o Laravel já acrescenta dois campos na tabela: id e timestamps.

Vamos criar a tabela produtos com 6 colunas (campos) com a seguinte estrutura:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Agora é necessário colocar essa estrutura na migração que criará a tabela produtos. Então edite-a e dentro do método up() defina os seguintes campos:

public function up(): void

{

Schema::create('produtos', function (Blueprint $table) {

$table->increments('id'); //Unsigned INT, Primary key, autoincremento e não permite Nulo

$table->string('nome', length: 200); //VARCHAR de 200

$table->decimal('preco', total: 10, places:2); //Números com casa decimal, com 8 dígitos antes da vírgula e 2 depois

$table->unsignedInteger('quantidade'); //Inteiro positivo

$table->text('descricao')->nullable(); //TEXT, permite Nulo

$table->timestamps(); //Cria dois campos created\_at e updated\_at do tipo TIMESTAMP que são atualizados automaticamente pelo Eloquent ORM do Laravel

});

}

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

O Laravel tem seus próprios tipos de dados, os quais são mapeados para o tipo de dado específico para o Banco de Dados que será aplicado a migração. Temos que colocar esses tipos na definição das colunas. Uma lista com detalhes de cada tipo pode ser vista na Documentação do Laravel em: <https://laravel.com/docs/11.x/migrations#available-column-types>

Uma observação quanto à coluna dataCadastro, cujo objetivo é guardar a data em que o produto foi criado ou alterado. O Laravel trabalha por padrão com o tipo timestamps(), o qual cria dois campos na tabela chamados created\_at e updated\_at do tipo TIMESTAMP, os quais são atualizados automaticamente pelo Modelo Eloquent ORM do Laravel, um mapeador objeto-relacional que utilizaremos para realizar as operações com o Banco de Dados. Portanto, para que essa interação funcione, algumas convenções devem ser seguidas. É importante verificar essas regras na documentação do Laravel em: <https://laravel.com/docs/11.x/eloquent#eloquent-model-conventions>

Também por convenção, o Eloquent assume que a tabela do banco de dados correspondente de cada modelo tem uma coluna de chave primária chamada id. Se necessário, você pode definir uma propriedade protected com nome $primaryKey em seu modelo para especificar uma coluna diferente que serve como chave primária do seu modelo, veja a documentação no link anterior.

* 1. **Aplicando a migration para criar a tabela produtos no Banco**

Agora temos que executar o comando do Artisan para criar a tabela no Banco. Porém, antes temos que configurar o Laravel para acesso ao nosso Banco de Dados. Isso pode ser feito em dois lugares:

* No arquivo .env da raiz da aplicação;
* Ou em config\database.php.

É preciso ficar atento para o fato de que o arquivo .env não sobe para o GitHub, caso esteja usando. Por consequência se o deploy da Aplicação para o servidor de produção for feito do GitHub, este arquivo não será colocado no servidor de produção.

Também notar que as configurações colocadas no arquivo .env prevalecem às configurações colocadas no config\database.php.

Edite o arquivo .env na raiz da Aplicação e adicione ou altera as seguintes configurações:

DB\_CONNECTION=mariadb

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=dbproduto\_laravel

DB\_USERNAME=produser

DB\_PASSWORD="xoingro78#po\*"

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Note que aqui está sendo utilizado o MariaDB.

Aqui também será alterado o config\database.php para refletir as configurações do .env. Localize as configurações para o mariadb e altere as configurações a seguir:

'database' => env('DB\_DATABASE', 'dbproduto\_laravel'),

'username' => env('DB\_USERNAME', 'produser'),

'password' => env('DB\_PASSWORD', 'xoingro78#po\*'),

'collation' => env('DB\_COLLATION', 'utf8mb4\_general\_ci'),

Texto

Descrição gerada automaticamente

Note acima que o Laravel vai utilizar o PDO para acessar o banco. Portanto, é necessário verificar se o PHP tem o PDO e o driver/extensão específico pdo\_mysql (veja pelo phpinfo()). Caso NÃO tenha, habilitar no php.ini, como a seguir:

Texto

Descrição gerada automaticamente

Não esqueça de reiniciar o servidor web (php artisan serve) para recarregar as alterações do php.ini.

Uma imagem contendo Tabela

Descrição gerada automaticamente

Para criar a tabela produtos no Banco, basta executar a migration pelo console do Artisan.

php artisan migrate

Texto

Descrição gerada automaticamente

Acesse o Banco de Dados e verifique se a tabela produtos foi criada e se sua estrutura está correta.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Note que a tabela produtos foi criada e a estrutura de colunas está correta. Porém, note também que foram criadas várias outras tabelas do Banco. Tais tabelas são utilizadas para controles do Laravel.

* 1. **Populando a tabela com o Tinker do Laravel**

O Laravel possui uma ferramenta de console que utiliza os modelos do Eloquent para acesso aos dados no Banco de Dados. Vamos utilizá-la para adicionar dois produtos:

* No console do Laravel, digite:

php artisan tinker

Texto

Descrição gerada automaticamente

* Para buscar todos os registros na tabela produtos, digite:

App\Models\Produto::all();

Texto

Descrição gerada automaticamente

Note que não foi retornado nenhum produto, pois a tabela está vazia.

* Para inserir um novo produto, execute os seguintes comandos um de cada vez:

$produto = new App\Models\Produto();

$produto->nome="Geladeira Topstremp";

$produto->preco=2550.60;

$produto->quantidade=5;

$produto->descricao="Geladeira Frost Free com painel touch";

$produto->save();

Texto

Descrição gerada automaticamente

* Busque novamente os registros da tabela:

App\Models\Produto::all();

Texto

Descrição gerada automaticamente

Agora temos 1 produto na tabela

* Para sair do Tinker digite **exit**

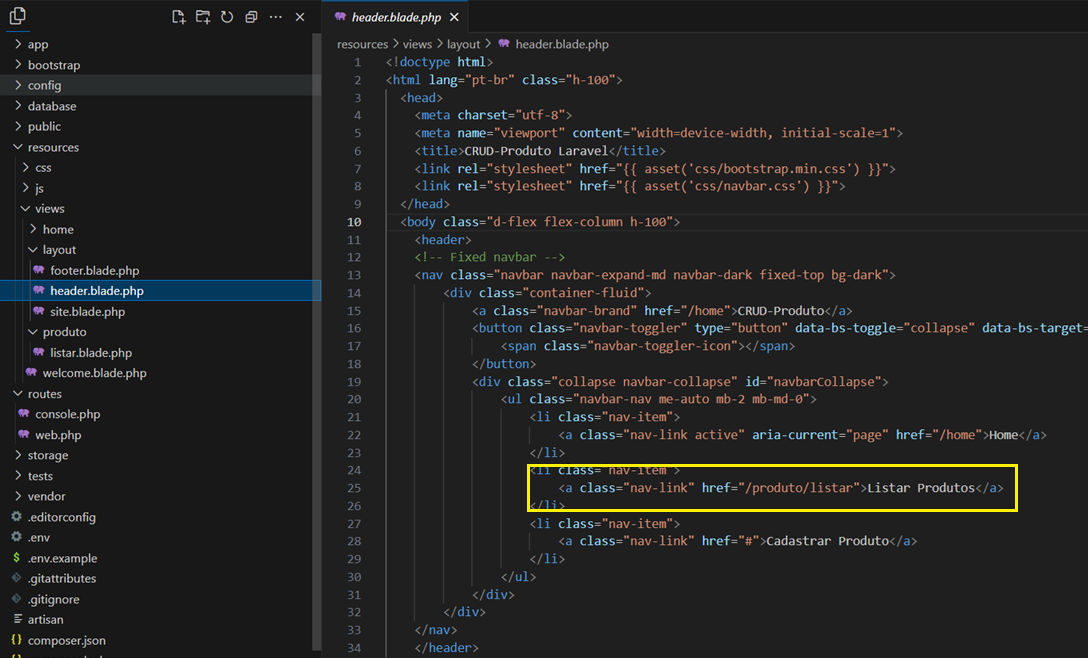
Através do phpMyAdmin veja que foi inserido um registro na tabela produtos.

Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

1. **Implementando a função Listar Produtos**

Primeiramente configurar o item Listar Produtos do menu da Aplicação para chamar o método **listar** do Produto Controller. Isso é feito no arquivo resources\views\layout\header.blade.php:



Agora é preciso definir a rota no arquivo routes\web.php, acrescentando a seguinte linha:

Route::get('/produto/listar',[ProdutoController::class,'listar']);

Texto

Descrição gerada automaticamente

É necessário criar o Controlller do Produto, caso ainda não tenha sido criado. Para tanto, no console da Aplicação, execute:

php artisan make:controller ProdutoController

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Note na saída anterior que o Controller já existe.

Em app\Http\Controllers\ProdutoController.php acrescente o método listar o use do Modelo Produto:

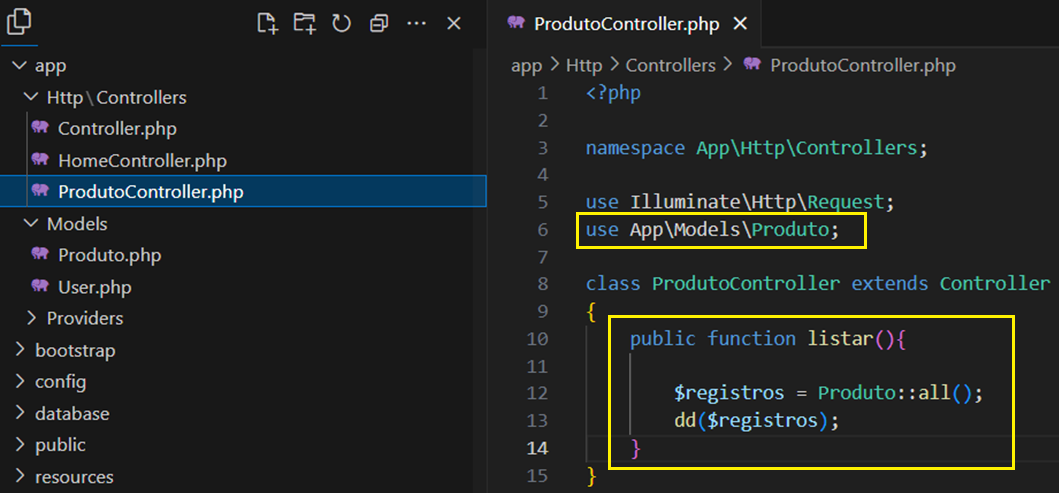
use App\Models\Produto;

public function listar(){

$registros = Produto::all();

dd($registros);

}



Note acima o uso do método all() do Eloquent, o qual vai acessar o banco de dados e trazer todos os registros da tabela produtos.

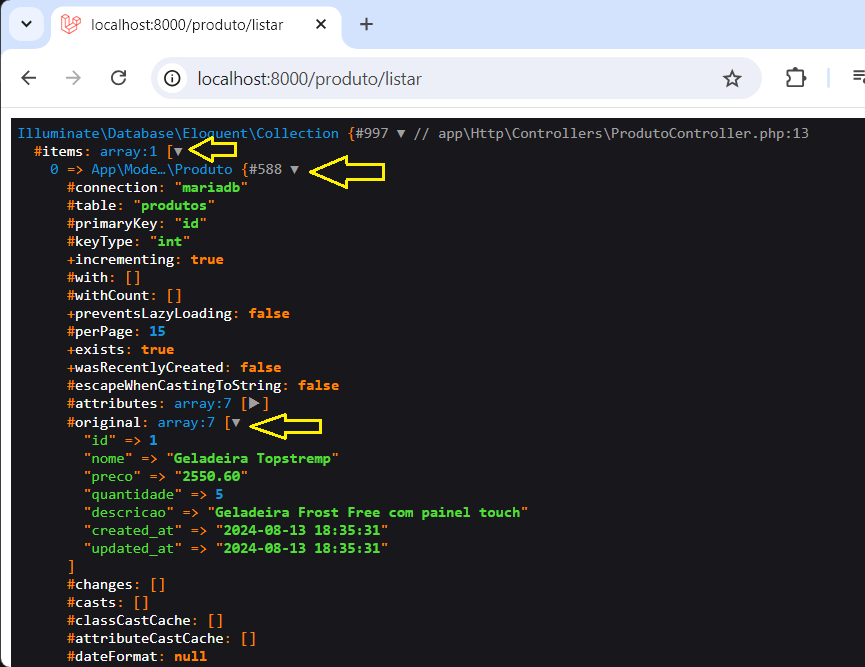
A saída será feita pelo comando dd($registro). Este método é para testes. Depois vamos utilizar a view para fazer a saída correta.

Salve o código, certifique-se que o servidor web embutido do PHP está executando (php artisan serve) e abra um navegador e acesse a opção Listar Produtos da Aplicação:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

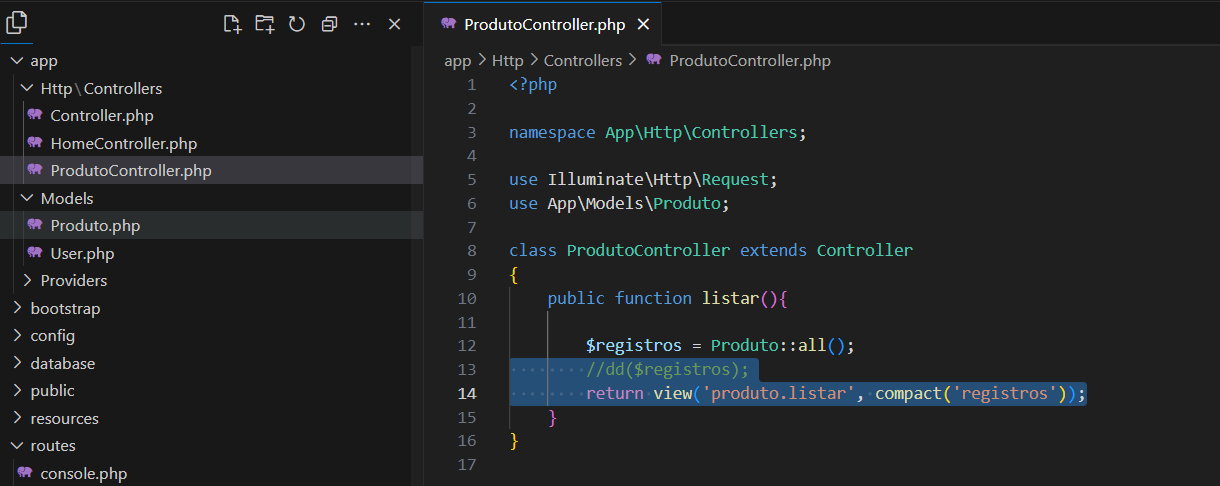
Expanda o item array:1 clicando na seta. Depois expanda novamente clicando na seta da linha do Model e, por último expanda clicando na seta da linha que inicia com #original



O único produto cadastrado foi mostrado e, portanto, os dados foram buscados no Banco.

O próximo passo é enviar os dados para a view que irá formatar a saída e enviar para o Navegador. Para isso comente o código da linha dd($registros) e acrescente a linha:

return view('produto.listar', compact('registros'));



A view a ser chamada ficará dentro de resources\views\produto com o nome listar.blade.php. Caso não exista, crie a pasta produto. O conteúdo inicial do arquivo listar.blade.php será:

@extends('layout.site')

@section('main')

<div class="container mt-4">

<h1>Listagem de Produtos</h2>

@foreach($registros as $registro)

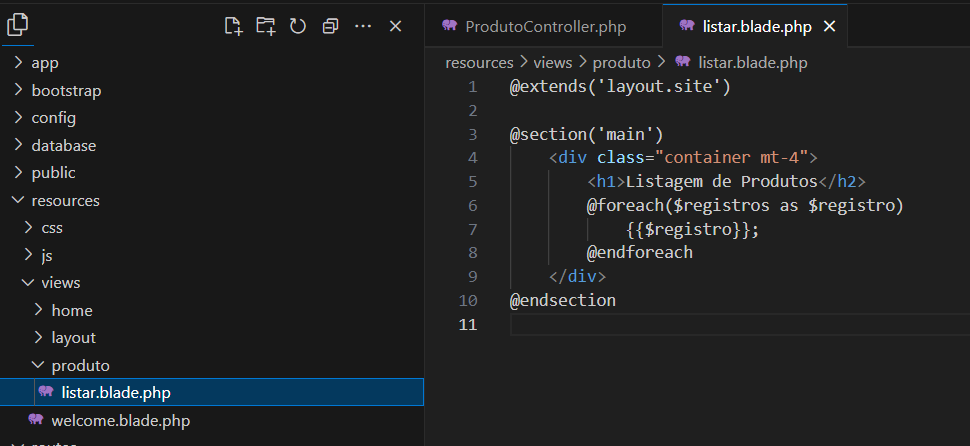
{{$registro}};

@endforeach

</div>

@endsection

Note que o código @foreach .. @endforeach é do Laravel.



A saída deverá ficar assim:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Agora que já temos os dados na view, vamos inserir o código HTML/Bootstrap para melhorar a saída. Troque todo o conteúdo da view pelo seguinte:

resources\views\produto\listar.blade.php

@extends('layout.site')

@section('main')

    <div class="container mt-4">

        <h1>Listagem de Produtos</h2>

        <div class="table-responsive">

            <table class="table table-bordered table-hover table-sm">

                <thead>

                    <tr class="table-info">

                        <th class="info">Id</th>

                        <th class="info">Nome</th>

                        <th class="info">Descrição</th>

                        <th class="info">Preço</th>

                        <th class="info">Qtde.</th>

                        <th class="info">Cadastro</th>

                    </tr>

                </thead>

                <tbody>

                    @foreach($registros as $registro)

                        <tr>

                            <td>{{ $registro->id }}</td>

                            <td>{{ $registro->nome }}</td>

                            <td>{{ $registro->descricao }}</td>

                            <td>{{ $registro->preco }}</td>

                            <td>{{ $registro->quantidade }}</td>

                            <td>{{ $registro->updated\_at }}</td>

                        </tr>

                    @endforeach

                </tbody>

            </table>

        </div>

    </div>

@endsection

Salve e teste. A saída deverá ser:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente