**Aula 2 – CRUD-Produto em Laravel**

**Construindo a Interface da Aplicação**

**Objetivos:**

* Conhecer aspectos da programação front-end da construção de uma aplicação web em Laravel;
* Estruturar a aplicação para receber os códigos de acesso ao banco de dados (programação back-end).

Ao final desta Aula devemos ter a interface da Aplicação estruturada e funcionando com a seguinte aparência:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Antes de programar o acesso ao banco de dados (programação back-end) é necessário preparar a interface com o usuário, a qual permitirá acesso às funcionalidades da Aplicação. Em geral, este trabalho é desempenhado por um profissional especializado em programação front-end.

Como sabemos, uma página web pode ser escrita puramente em HTML. Porém, seu resultado visual pode não ser muito bonito. É onde entra o CSS (Cascading Style Sheets), que dá mais vida e elegância a uma página web.

Programar puramente em HTML e CSS é possível, mas muitas vezes improdutivo. Isso se agrava ainda mais quando consideramos a diversidade de dispositivos que usuários podem usar para acesso à Aplicação web. Assim, existem alguns frameworks que tornam mais fácil e produtiva a construção de interfaces web, possibilitando uma melhor experiência ao usuário. Exemplos desses frameworks são: Bootstrap, Materialize, W3.CSS, Foundation, Skeleton. Aqui utilizaremos o Bootstrap (<https://getbootstrap.com/>).

1. **Criando o Controller da Página Principal (Home) da Aplicação**

Antes de iniciarmos a confecção da interface principal da nossa Aplicação, vamos criar o Controller que vai nos levar até a página principal (Home). No Laravel isso é feito pelo console do Artisan. Então, vá até o console e execute o comando:

php artisan make:controller HomeController

Texto

Descrição gerada automaticamente

Note no Código da Aplicação que foi criado o arquivo app\Http\Controllers\HomeController.php

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Agora acrescente o método index() no HomeController.php

public function index(){

return view('home.index');

}

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

O comando vai chamar a view que deve estar em resources\views\home\index.blade.php, onde:

resources\views – é a pasta padrão do Laravel para as views;

home – é a pasta que vamos criar para colocar as páginas da nossa página principal;

index.blade.php – é o nome do arquivo que vai conter o código da página home.

1. **Criando a View da página principal (Home)**

As views da Aplicação devem ser colocadas na pasta resources\views. Então, crie uma pasta home dentro da pasta resources\views. Em seguida, crie o arquivo index.blade.php com o seguinte conteúdo:

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CRUD-Produto Laravel</title>

</head>

<body>

<h2>CRUD-Produto em Laravel - Home</h2>

</body>

</html>

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

1. **Ajustando a Rota para o Controller da página Home**

O próximo passo é ajustar a rota para chamar o HomeController, o qual queremos que seja acionado quando o usuário tentar acessar a raiz da nossa aplicação (‘/’) ou /home. Acrescente o seguinte código no arquivo de rotas routes\web.php

use App\Http\Controllers\HomeController;

Route::get('/',[HomeController::class,'index']);

Route::get('/home',[HomeController::class,'index']);

Tela de computador com fundo preto

Descrição gerada automaticamente

É importante notar a necessidade de acrescentar o caminho para a classe HomeController (use App\Http\Controllers\HomeController;), onde:

use App\Http\Controllers\HomeController - é o caminho para a classe;

Route:: - é o comando do Laravel para definir uma rota;

get – é o comando HTML de requisição (podendo ser também: post, put, delete, patch, Options);

'/' ou '/home' – é a URI como vai aparecer na URL;

[HomeController::class – nome da classe do Controlador a ser chamado;

'index' – nome do método dentro do Controlador a ser chamado (action).

Mais detalhes e formas de criar rotas em https://laravel.com/docs/11.x/routing

Para testar, execute o servidor embutido do PHP através do comando artisan do Laravel:

php artisan serve

Texto

Descrição gerada automaticamente

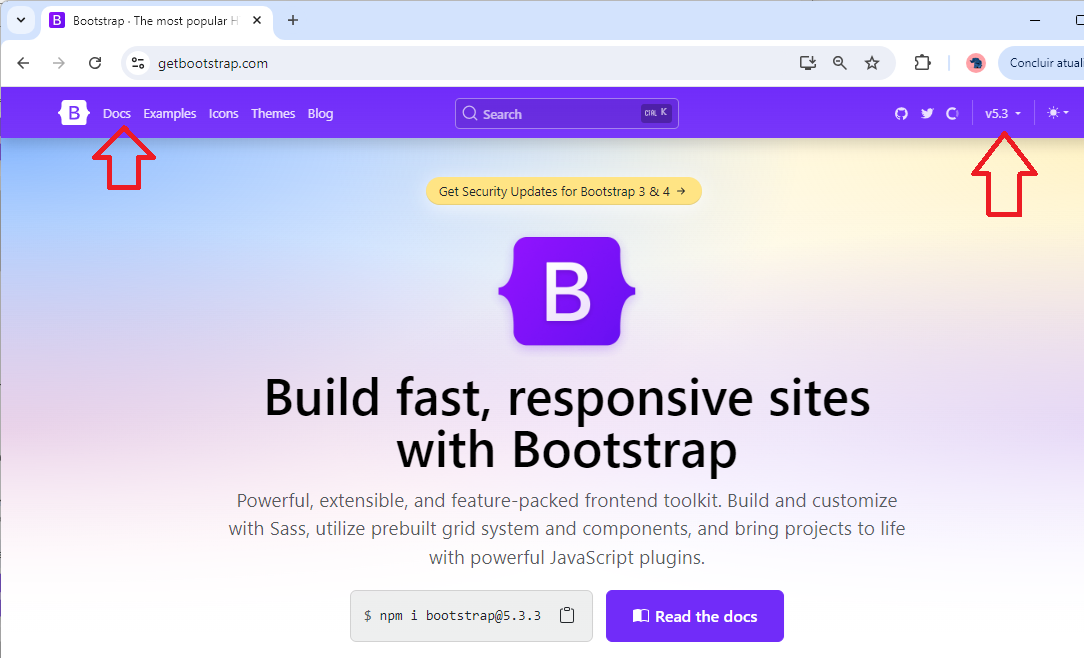
Agora abra um Navegador e aponte para <http://localhost:8000/> ou <http://localhost:8000/home>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

1. **Montando o layout da Aplicação com Bootstrap**

Para construir a interface da Aplicação (ou do site) vamos utilizar o Bootstrap. A maneira mais fácil de aprender a utilizar o Bootstrap é através da documentação e exemplos disponíveis em seu site (<https://getbootstrap.com/>).



Acesse o site e clique em **Docs**. No lado direito selecione a versão 5.3 caso não seja a que esteja aparecendo. Selecione o segundo código clicando em **Copy** para copiar o código apresentado para o Clipboard.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

No Editor, troque todo o conteúdo do arquivo resources\views\home\index.blade.php existente pelo código copiado (remova o código existente e depois pressione CTRL C para colar). Depois troque o conteúdo das tags <html>, <title> e <h1> como abaixo:

<html lang="pt-br">

<title>CRUD-Produto Laravel</title>

<h1>CRUD-Produto em Laravel - Home </h1>

Veja a figura:

Uma imagem contendo Texto

Descrição gerada automaticamente

Este template faz as configurações e carrega os scripts CSS e javascript necessários para seu funcionamento e, portanto, sempre devem fazer parte de uma página web que utilize o bootstrap.

Salve o projeto e abra novamente no Navegador.

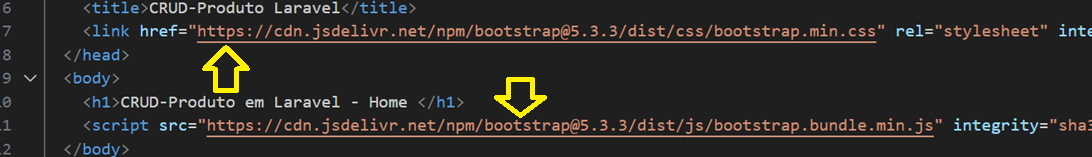
Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Note que agora a mensagem já aparece com algumas formatações de fonte já inseridas pelo Bootstrap.

* 1. **Baixando os scripts do Bootstrap para o servidor local**

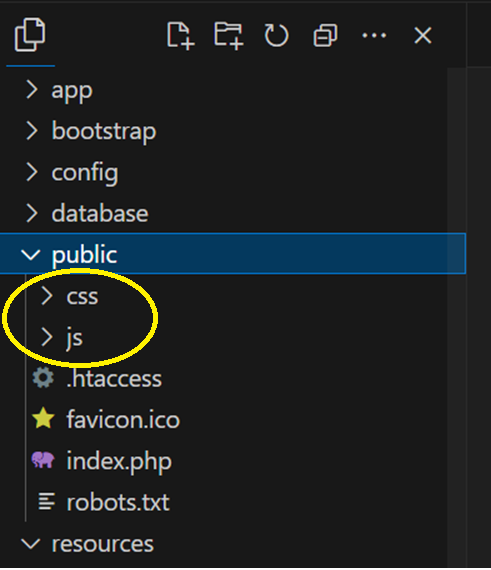
Note que os scripts do Bootstrap são carregados via CDN, que é uma Rede de Distribuição de Conteúdo:



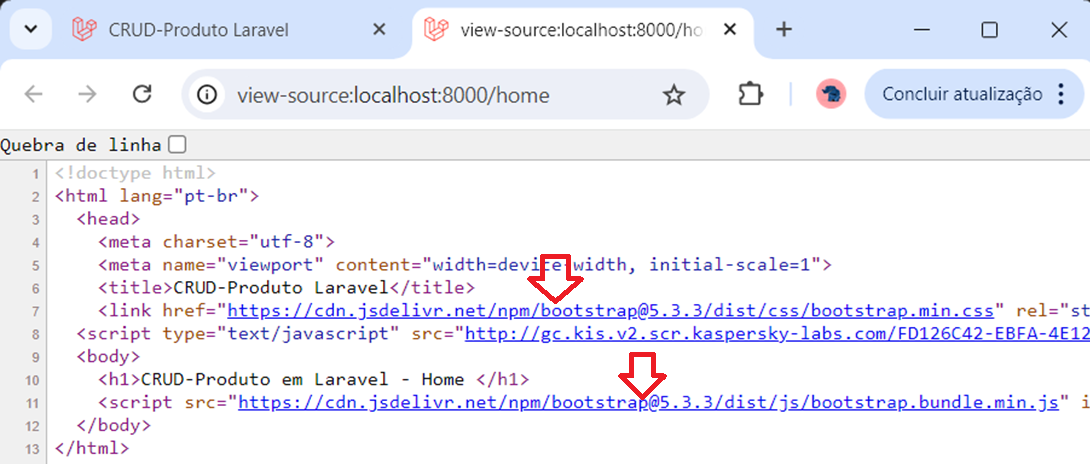
Isto significa que quando alguém acessar nossa página, este conteúdo não será carregado a partir de nosso servidor, e sim, de um servidor remoto. Existem várias vantagens nesse esquema, entretanto, caso nossa aplicação execute somente em uma rede interna da empresa, sem acesso à Internet, não será possível carregar os scripts. Assim como em outras situações, pode ser necessário carregar os scripts do nosso servidor local.

Para baixar os scripts para o nosso servidor local, siga os seguintes passos:

1. Crie as pastas **css** e **js** dentro do public da Aplicação;



1. Estando no Navegador, onde acessou a página da Aplicação, clique com o botão direito e selecione **Exibir código fonte da página**;
2. Na nova aba que se abre, clique no link do bootstrap.min.css



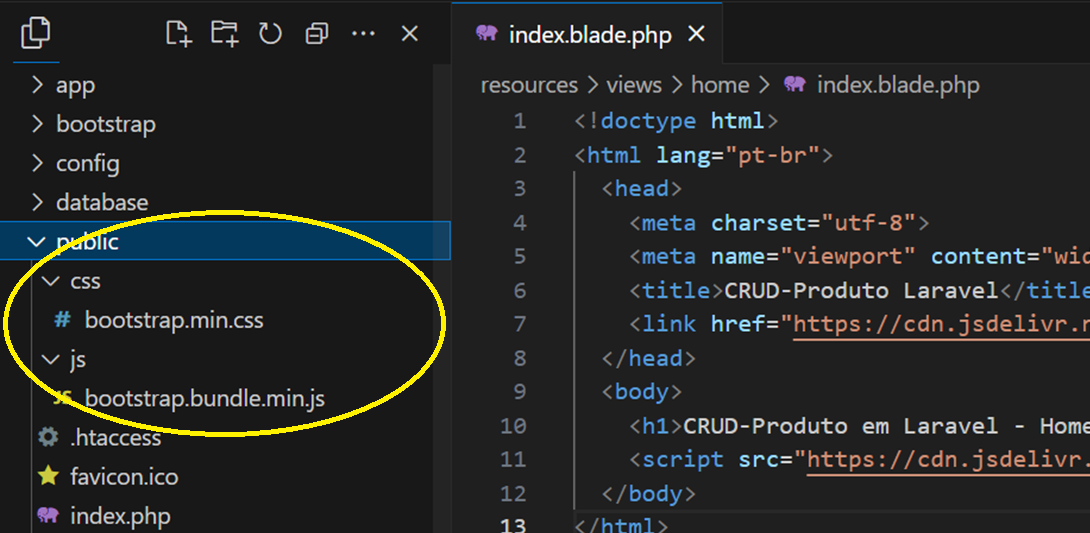
1. O código fonte do script será aberto em outra aba:

Texto

Descrição gerada automaticamente

1. Clique com o botão direito e selecione **Salvar como...** e salve o script na pasta **public\css** da Aplicação, com o nome **bootstrap.min.css**;
2. Faça o mesmo procedimento para o script bootstrap.bundle.min.js, porém este deve ser salvo na pasta **public\js**

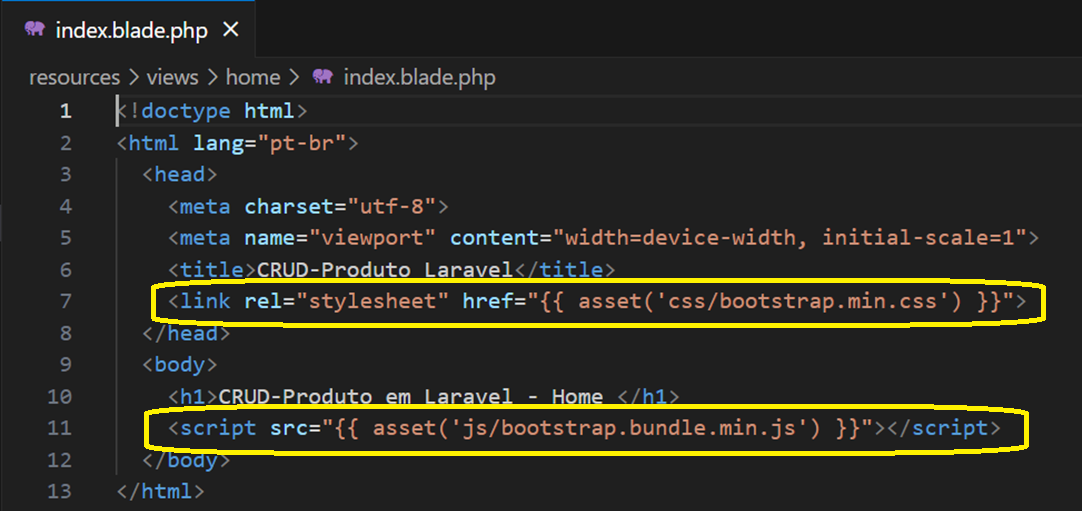
Após estes procedimentos, os scripts já estão locais e devem aparecer no Projeto. Caso não apareça, feche e abra novamente o projeto.



Como agora os scripts estão locais, é necessário mudar a forma que são carregados na página index.blade.php. Os quais devem ficar assim:

<link rel="stylesheet" href="{{ asset('css/bootstrap.min.css') }}">

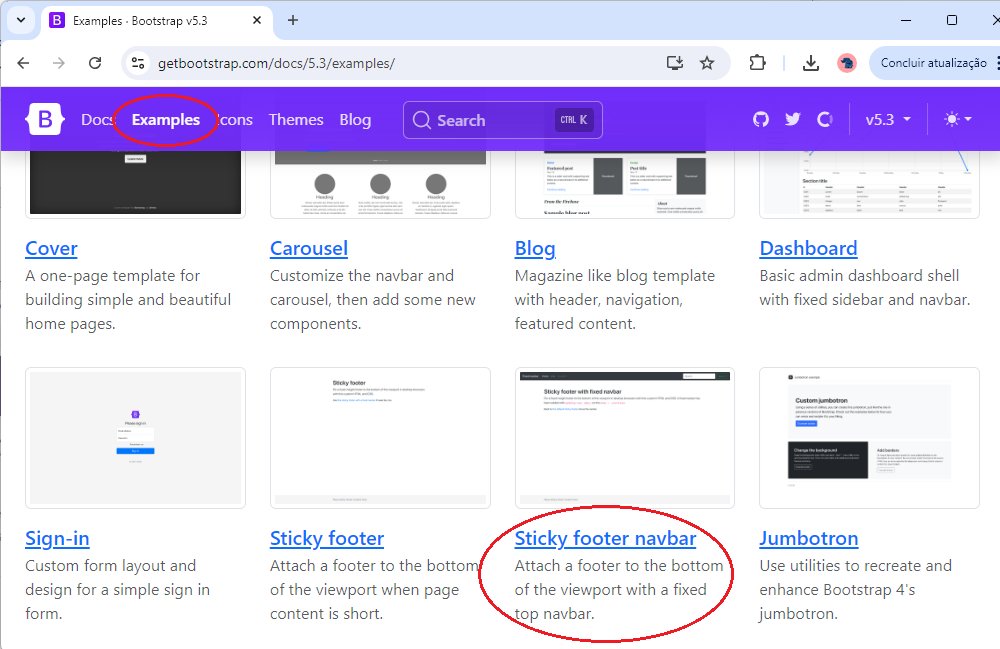
<script src="{{ asset('js/bootstrap.bundle.min.js') }}"></script>



Após fazer as modificações, recarregue a página no Navegador, exiba o código fonte e clique nos links dos scripts para certificar que estão carregando.

1. **Criando o Menu da Aplicação**

Agora que já temos a base para utilização dos recursos do Bootstrap, vamos criar o menu de opções da nossa Aplicação. Para tanto, acesse o site do bootstrap (getbootstrap.com) e clique na opção **Examples**. Localize o exemplo **Stick footer navbar**:



Então clique no exemplo para abri-lo:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Este exemplo apresenta os recursos de interface que queremos, então vamos exibir o código fonte e copiar o que nos interessa. Para tanto, clique com o botão direito na página e selecione **Exibir código fonte da página**.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Na aba que se abre, copie o código fonte e substitua o conteúdo do arquivo resources\views\home\index.blade.php já existente. Depois remova o que não é necessário e substitua os links para os arquivos de css e Javascript do bootstrap. No final o código deverá ficar como a seguir:

<!doctype html>

<html lang="pt-br" class="h-100">

  <head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>CRUD-Produto Laravel</title>

    <link rel="stylesheet" href="{{ asset('css/bootstrap.min.css') }}">

    <link rel="stylesheet" href="{{ asset('css/navbar.css') }}">

  </head>

  <body class="d-flex flex-column h-100">

    <header>

    <!-- Fixed navbar -->

    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">

        <div class="container-fluid">

            <a class="navbar-brand" href="/home">CRUD-Produto</a>

            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">

                <span class="navbar-toggler-icon"></span>

            </button>

            <div class="collapse navbar-collapse" id="navbarCollapse">

                <ul class="navbar-nav me-auto mb-2 mb-md-0">

                    <li class="nav-item">

                        <a class="nav-link active" aria-current="page" href="/home">Home</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">Listar Produtos</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">Cadastrar Produto</a>

                    </li>

                </ul>

            </div>

        </div>

    </nav>

    </header>

    <main class="flex-shrink-0">

    <div class="container">

        <h1 class="mt-5">CRUD-Produto em Laravel - Home</h1>

    </div>

    </main>

    <footer class="footer mt-auto py-3 bg-body-tertiary">

    <div class="container">

        <span class="text-body-secondary">Rodapé</span>

    </div>

    </footer>

    <script src="{{ asset('js/bootstrap.bundle.min.js') }}"></script>

  </body>

</html>

Agora crie o arquivo **navbar.css** em **public\css** com o seguinte conteúdo:

main > .container {

  padding: 60px 15px 0;

}

Tela de celular com fundo preto

Descrição gerada automaticamente

Salve as alterações e abra no Navegador para ver o resultado:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Explicando o código do Navbar:

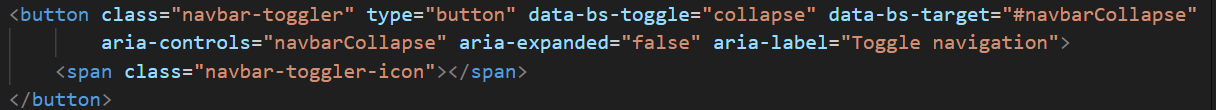


Abre a tag <nav> com algumas classes do Bootstrap. Destaque para:

* navbar-dark e bg-dark – responsáveis por colocar fundo preto e cor de fonte branco;
* fixed-top – responsável por fixar o menu no topo da tela, mesmo quando rolar a página;
* navbar-expand-md – responsável por esconder o menu a partir de dispositivos de tamanho médio.



É usado para colocar uma logo ou, como aqui, um texto fixo que descreve a Aplicação. Quando clicado redireciona para a página home.



Responsável por esconder o menu quando a tela do dispositivo for menor que um dispositivo médio (celular por exemplo), mostrando um ícone para acesso a um menu vertical.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Este trecho de código dentro da tag <div> determina o que será “colapsado”, ou seja, o que ficará dentro do menu vertical quando o dispositivo tiver uma tela pequena. As opções de Menu são itens <li> de uma lista HTML não ordenada <ul>.

1. **Estruturando a Aplicação para receber o código PHP**

Existem partes da Aplicação que sempre serão exibidas, independentemente da função sendo acessada. Por exemplo, os códigos do Menu e do Rodapé sempre deverão compor a página que será exibida para o usuário. O que muda é a parte do meio da página, a qual exibirá os dados da opção desejada.

Para facilitar a implementação das funcionalidades da Aplicação é necessário separar o código do arquivo index.blade.php em diferentes arquivos. Isso evitará ter que repetir o código HTML para cada funcionalidade.

Para tanto, crie em resoures\views uma pasta chamada layout. Dentro desta pasta layout, crie dois arquivos chamados **header.blade.php** e **footer.blade.php**.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Coloque o seguinte conteúdo no arquivo header.blade.php:

Resources\views\layout\header.blade.php

<!doctype html>

<html lang="pt-br" class="h-100">

  <head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>CRUD-Produto Laravel</title>

    <link rel="stylesheet" href="{{ asset('css/bootstrap.min.css') }}">

    <link rel="stylesheet" href="{{ asset('css/navbar.css') }}">

  </head>

  <body class="d-flex flex-column h-100">

    <header>

    <!-- Fixed navbar -->

    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">

        <div class="container-fluid">

            <a class="navbar-brand" href="/home">CRUD-Produto</a>

            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">

                <span class="navbar-toggler-icon"></span>

            </button>

            <div class="collapse navbar-collapse" id="navbarCollapse">

                <ul class="navbar-nav me-auto mb-2 mb-md-0">

                    <li class="nav-item">

                        <a class="nav-link active" aria-current="page" href="/home">Home</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">Listar Produtos</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" href="#">Cadastrar Produto</a>

                    </li>

                </ul>

            </div>

        </div>

    </nav>

    </header>

Note que esse conteúdo nada mais é do que a parte inicial do arquivo index.blade.php.

Da mesma forma, coloque o seguinte código no arquivo footer.blade.php:

resources\views\layout\footer.blade.php

    <footer class="footer mt-auto py-3 bg-body-tertiary">

        <div class="container">

            <span class="text-body-secondary">Rodapé</span>

        </div>

    </footer>

    <script src="{{ asset('js/bootstrap.bundle.min.js') }}"></script>

  </body>

</html>

Note que este conteúdo nada mais é do que a parte final do arquivo index.blade.php.

Agora só falta a parte do meio da página, cujo conteúdo será criado de acordo com a saída de cada opção da Aplicação. Assim, vamos criar uma página template que vai juntar o header, o footer e o conteúdo dinamicamente.

Em resources\views\layout crie um arquivo chamado site.blade.php com o seguinte conteúdo:

resources\views\layout\site.blade.php

@include('layout.header')

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

@yield('main')

</main>

@include('layout.footer')

Texto

Descrição gerada automaticamente

Com isso, as páginas da nossa Aplicação devem fornecer somente a parte do conteúdo da seção <main>.

Como modelo, vamos alterar a página Home da Aplicação, que simplesmente mostra a mensagem CRUD – Produto em Laravel – Home, para utilizar este esquema.

Altere o arquivo resources\views\home\index.blade.php para o seguinte conteúdo:

resources\views\home\index.blade.php

@extends('layout.site')

@section('main')

<div class="container mt-4">

<h1>CRUD-Produto em Laravel - Home</h2>

</div>

@endsection

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

No código acima, tudo o que estiver dentro de @section(‘main’) ... @endsection será colocado no arquivo resources\views\layout\site.blade.php no local onde aparece a notação @yield('main').

Assim, quando acessamos a Aplicação em /home, a rota Route::get('/home',[HomeController::class,'index']) inicializará o Controlador app\Controllers\HomeController.php e chamará o método index(), o qual chama a View resources\views\home\index.blade.php que monta a página de saída utilizando o layout padrão da nossa Aplicação.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

1. **Listar Produtos**

Da mesma forma, podemos criar a página inicial da opção Listar Produtos, a qual, por enquanto, será praticamente igual à página Home, exceto pelo texto que será exibido.

Crie uma pasta chamada produto em resources\views. Depois crie um arquivo dentro desta pasta chamado listar.blade.php, com o seguinte conteúdo:

@extends('layout.site')

@section('main')

<div class="container mt-4">

<h1>Listagem de Produtos</h2>

</div>

@endsection

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Agora, programe o link do menu Listar Produtos da Aplicação. Edite o arquivo resources\views\layout\header.blade.php e altere a linha abaixo:

<li class="nav-item">

<a class="nav-link" href="/produto/listar">Listar Produtos</a>

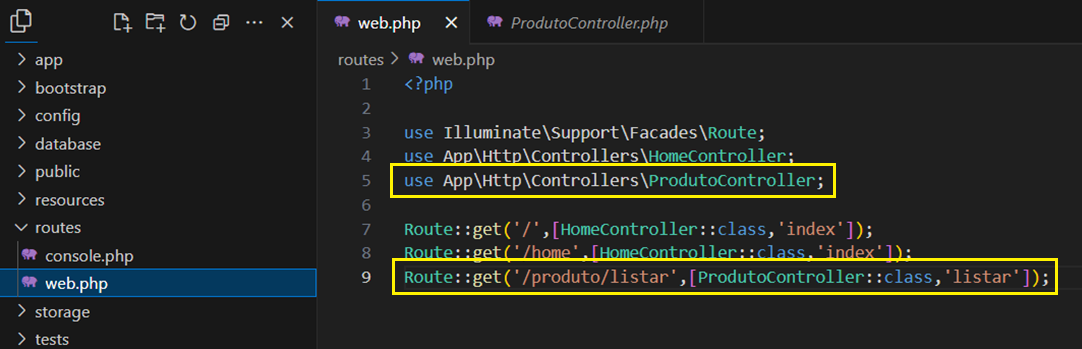
</li>

Agora é necessário criar a rota para esta funcionalidade da Aplicação. Edite o arquivo routes\web.php e acrescente a seguinte linha:

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

E, no início desse mesmo arquivo, acrescente a linha:

use App\Http\Controllers\ProdutoController;



Por último, é preciso criar o Controlador e o método dentro do Controlador. Para tanto, siga os procedimentos:

* Vá até o console (prompt de comando) que executa o php artisan serve e dê <CTRL> <C>;
* No prompt digite: php artisan make:controller ProdutoController

Texto

Descrição gerada automaticamente

* No Editor, edite o arquivo app\Http\Controllers\ProdutoController.php, e acrescente o seguinte método:

public function listar(){

return view('produto.listar');

}

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

* Inicialize novamente o servidor web do php no console com o comando php artisan serve

Texto

Descrição gerada automaticamente

* No Navegador, acesse http://localhost:8000/produto/listar

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Note que a dinâmica de funcionamento da Aplicação para esta funcionalidade (solicitação de página, rota, controlador, view) está funcionando. Só falta agora implementar o acesso ao banco de dados para buscar os produtos e listá-los. Isso será feito na próxima aula.

**Atividades da Aula**

1. Faça a página inicial da opção Cadastrar Produtos do menu, semelhante ao que foi feito para a opção Listar Produtos;