**Aula 7 – CRUD-Produto em Laravel**

**Autenticação de Usuário - Login**

**Objetivos:**

* Conhecer os recursos de Autenticação de Usuários do Laravel;
* Implementar a Autenticação Manualmente;
* Implementar a Autenticação via recursos prontos do Laravel;

Muitas aplicações web devem implementar uma maneira para seus usuários autenticarem-se com a aplicação e "fazerem login". Implementar esse recurso em aplicações web pode ser complexo e potencialmente arriscado. O Laravel dispõe de ferramentas prontas para implementar a autenticação de forma rápida, segura e fácil.

Ao final desta Aula devemos ter já em funcionamento o Login de Usuários da Aplicação.

**Autenticação de uma Aplicação Web**

Antes de iniciarmos com o aprendizado dos recursos de autenticação do Laravel, é importante entender como funciona autenticação de usuário em uma Aplicação Web. Ao usar um navegador da web, um usuário fornecerá seu nome de usuário e senha por meio de um formulário de login. Se essas credenciais estiverem corretas, o aplicativo armazenará informações sobre o usuário autenticado na sessão do usuário. Sessão é um recurso do servidor Web que fica associado a somente uma conexão do usuário. Um cookie emitido para o navegador contém o ID da sessão para que solicitações subsequentes ao aplicativo possam associar o usuário à sessão correta. Após o cookie da sessão ser recebido, o aplicativo recuperará os dados da sessão com base no ID da sessão, observe que as informações de autenticação foram armazenadas na sessão e considerará o usuário como "autenticado".

O Laravel inclui serviços de autenticação e sessão integrados que são normalmente acessados por meio das *facades* Auth e Session. Esses recursos fornecem autenticação baseada em cookie para solicitações iniciadas por navegadores da web. Eles fornecem métodos que permitem verificar as credenciais de um usuário e autenticá-lo. Além disso, esses serviços armazenarão automaticamente os dados de autenticação adequados na sessão do usuário e emitirão o cookie de sessão do usuário.

Podemos implementar manualmente a camada de autenticação da Aplicação ou utilizar um dos pacotes de autenticação gratuitos do Laravel, tais como: Laravel Breeze, Laravel Jetstream e Laravel Fortify. Primeiramente vamos implementar a autenticação manualmente e depois faremos via pacote Laravel Breeze.

1. **Implementando Autenticação Manual**

Basicamente para implementarmos a Autenticação sem usar os pacotes de autenticação do Laravel temos que:

* Definir a rota para a tela de Login
* Definir o Controlador de Login, o qual se encarregará de chamar a view com a tela de login e depois receber as credenciais de acesso e fazer a autenticação;
* Definir a view com a tela e formulário de Login;
* Receber as credenciais de acesso no Controlador de Login, validar essas credenciais, colocar as informações de login na Sessão e permitir que o usuário acesse nossa Aplicação.

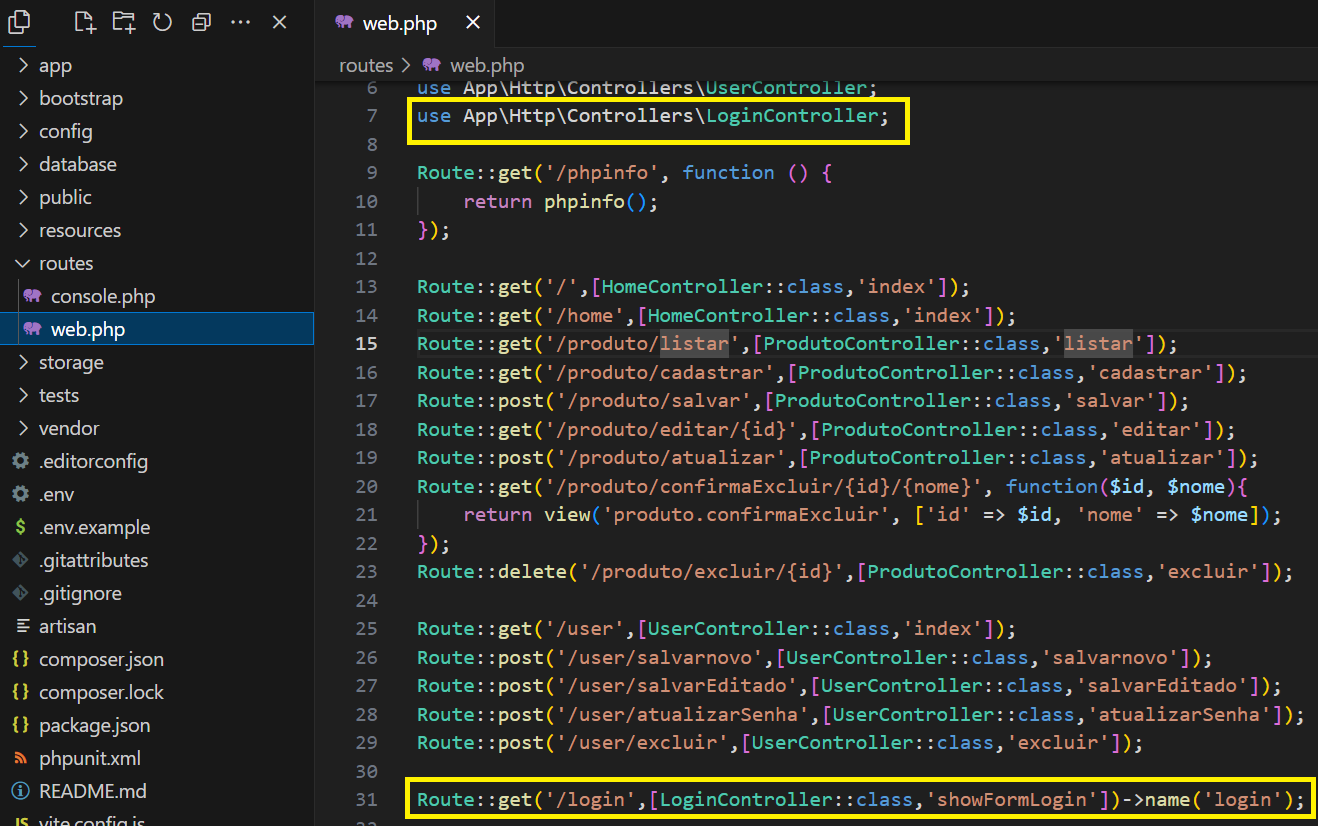
**Importante**: Antes de iniciar, faça uma cópia da pasta da Aplicação para depois implementar a Autenticação com o Laravel Breeze.

**7.1.1. Definindo a Rota para a Tela de Login**

Acrescente as linhas a seguir no arquivo routes\web.php:

use App\Http\Controllers\LoginController;

Route::get('/login',[LoginController::class,'showFormLogin'])->name('login');

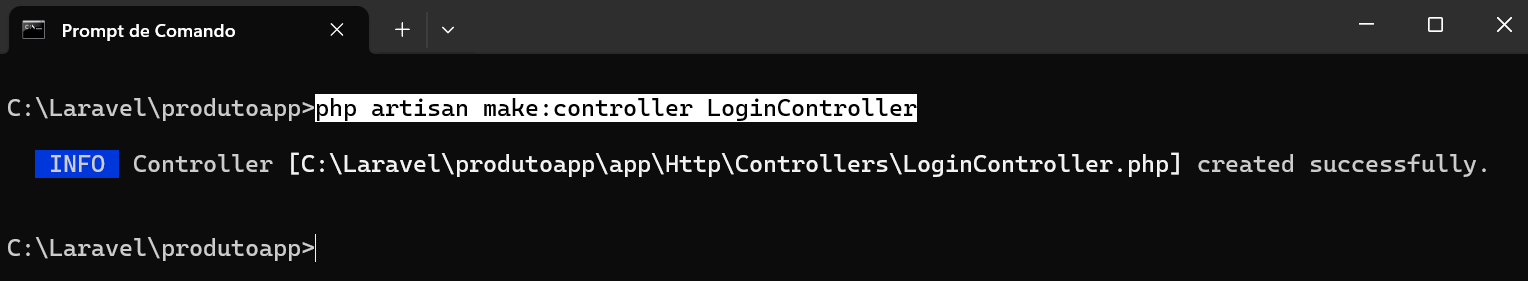


Note na definição da rota a chamada ao método (->name('login')) o qual está nomeando a rota. Este recurso traz alguns benefícios na manutenção da Aplicação relacionada com as URLs de acesso.

**7.1.2. Criando o LoginController**

Digite o seguinte comando no prompt para criar o Controlador:

php artisan make:controller LoginController

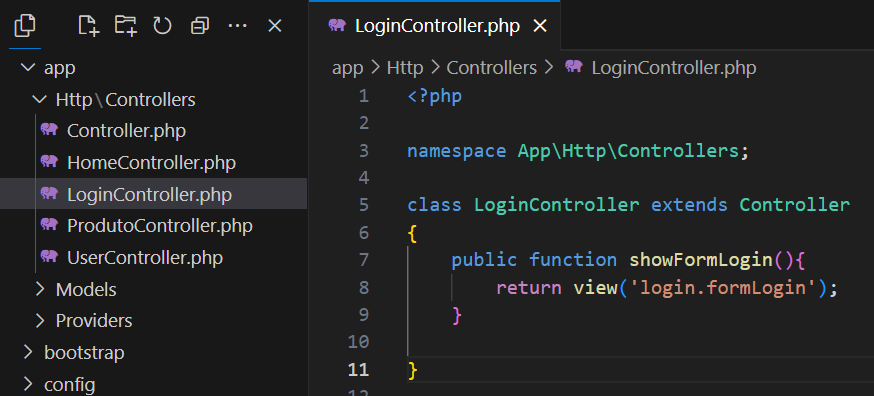


Agora crie o método showFormLogin() no app\Http\Controllers\LoginController.php com o seguinte código:

public function showFormLogin(){

return view('login.formLogin');

}



Vamos criar uma view simples só para testarmos o fluxo da função Login:

* Crie uma pasta chamada login em resources\views;
* Dentro da pasta criada (resources\views\login) crie um arquivo formLogin.blade.php com o seguinte conteúdo:

Chegamos na View formLogin.blade.php

Tela de celular com aplicativo aberto

O conteúdo gerado por IA pode estar incorreto.

Salve tudo e teste. No Navegador acesse: http://localhost:8000/login

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Como visto, chegamos até a view que vai exibir a página de Login.

**7.1.3. Criando a View formLogin**

Troque o conteúdo da view resources\views\login\formLogin.blade.php pelo seguinte código:

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="utf-8">

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

<title>CRUD-Produto Laravel - Login</title>

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

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

<style>

body {

background: #f8f9fa;

height: 100vh;

display: flex;

align-items: center;

justify-content: center;

}

.login-card {

max-width: 400px;

width: 100%;

padding: 2rem;

#border-radius: 1rem;

background-color: #fff;

box-shadow: 10px 16px 24px -2px rgba(0,0,0,0.4);

}

.form-icon {

position: absolute;

top: 50%;

transform: translateY(-50%);

left: 0.75rem;

color: #6c757d;

}

.form-logo {

width: 50%;

height: 50%;

margin-left: 25%;

margin-top: -20%;

margin-bottom:20px;

border-radius: 3%;

padding: 0px;

box-shadow: 6px 6px 10px -2px rgba(0, 0, 0, 0.2);

}

.input-with-icon {

padding-left: 2.5rem;

}

</style>

</head>

<body>

<div class="login-card">

<img src="imagens/CrudLogo.png" class="form-logo" alt="CRUD\_logo">

<h4 class="mb-4 text-center">Identifique-se</h4>

{{-- Mensagens de Erro --}}

@if(session('erro'))

<div class="alert alert-danger d-flex align-items-center" role="alert">

<i class="bi bi-exclamation-triangle-fill me-2"></i>

{{ session('erro') }}

</div>

@endif

{{-- Validação de Campos da Entrada pelo validate do Laravel --}}

@if ($errors->any())

<div class="alert alert-danger">

<ul class="mb-0">

@foreach ($errors->all() as $erro)

<li>{{ $erro }}</li>

@endforeach

</ul>

</div>

@endif

<form action="{{-- route('validarLogin') --}}" method="post">

@csrf

<div class="mb-3 position-relative">

<label for="email" class="form-label">Endereço de e-mail</label>

<i class="bi bi-envelope-fill form-icon"></i>

<input type="email" class="form-control input-with-icon" id="email" name="email" value="{{ old('email')}}" autofocus>

</div>

<div class="mb-3 position-relative">

<label for="senha" class="form-label">Senha</label>

<i class="bi bi-lock-fill form-icon"></i>

<input type="password" class="form-control input-with-icon" id="senha" name="senha" required>

</div>

<button type="submit" class="btn btn-warning w-100">

<i class="bi bi-box-arrow-in-right me-1"></i> Entrar

</button>

<div class="mt-3 text-center">

<a href="#">Esqueceu sua senha?</a>

</div>

<div class="mt-3 text-center text-muted">

<small><em>&copy; jlieira-2025</em><small>

</div>

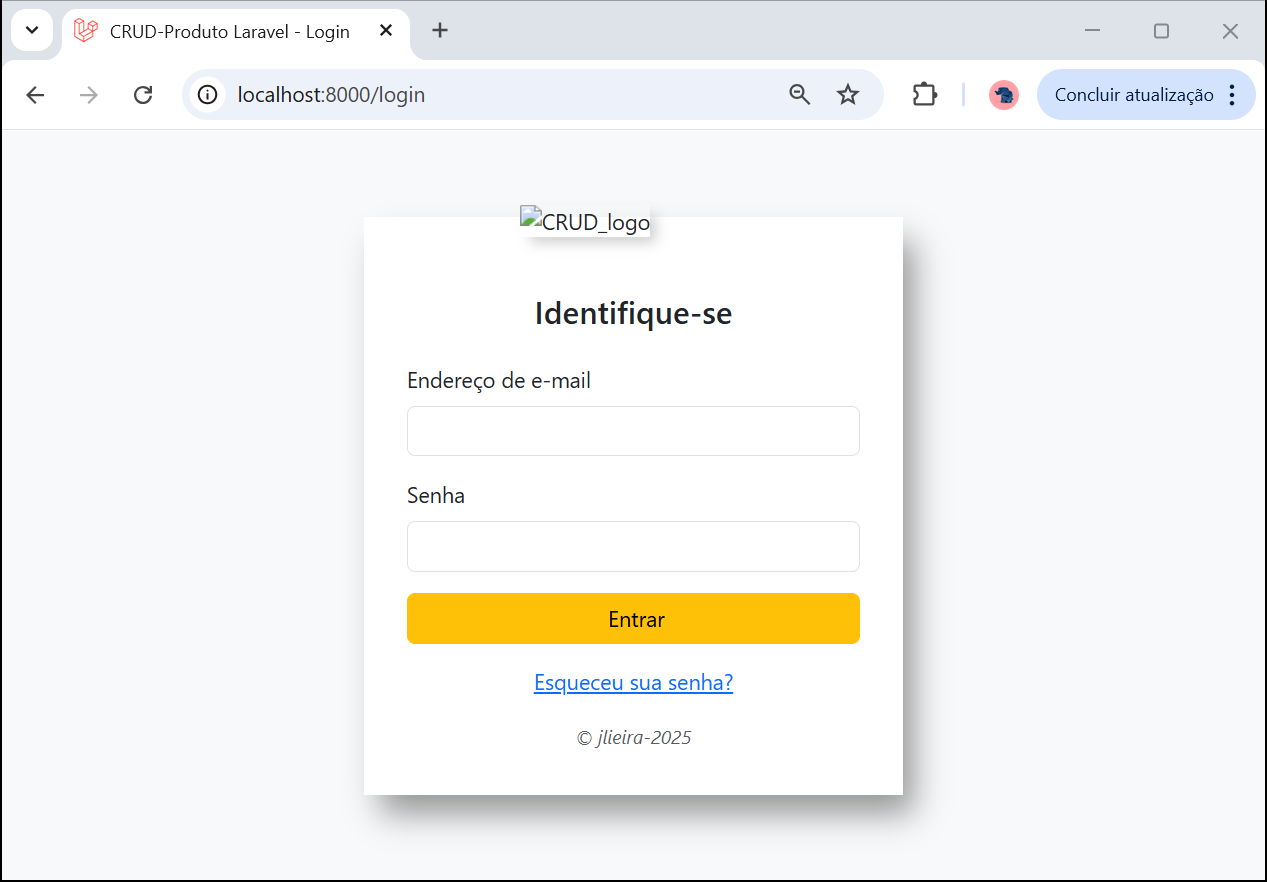
</form>

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

</body>

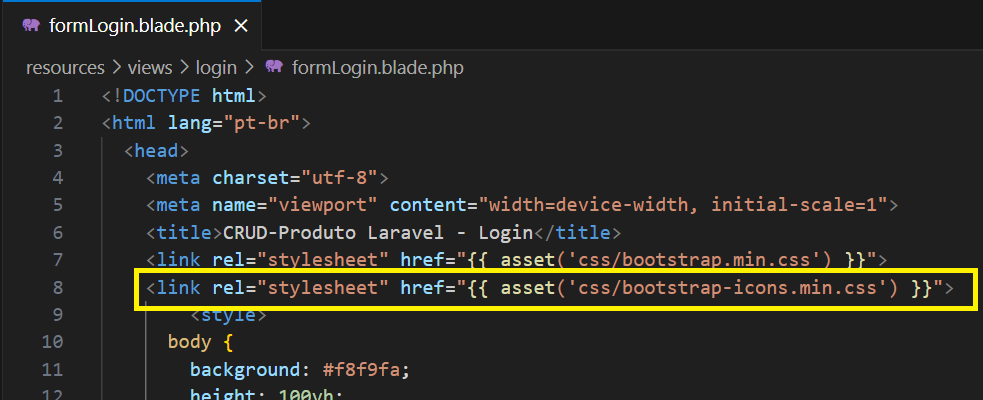
</html>

Salve e teste novamente. Note que está faltando alguns recursos da página que iremos acrescentar a seguir.



**Carregando a Biblioteca de ícones do Bootstrap**

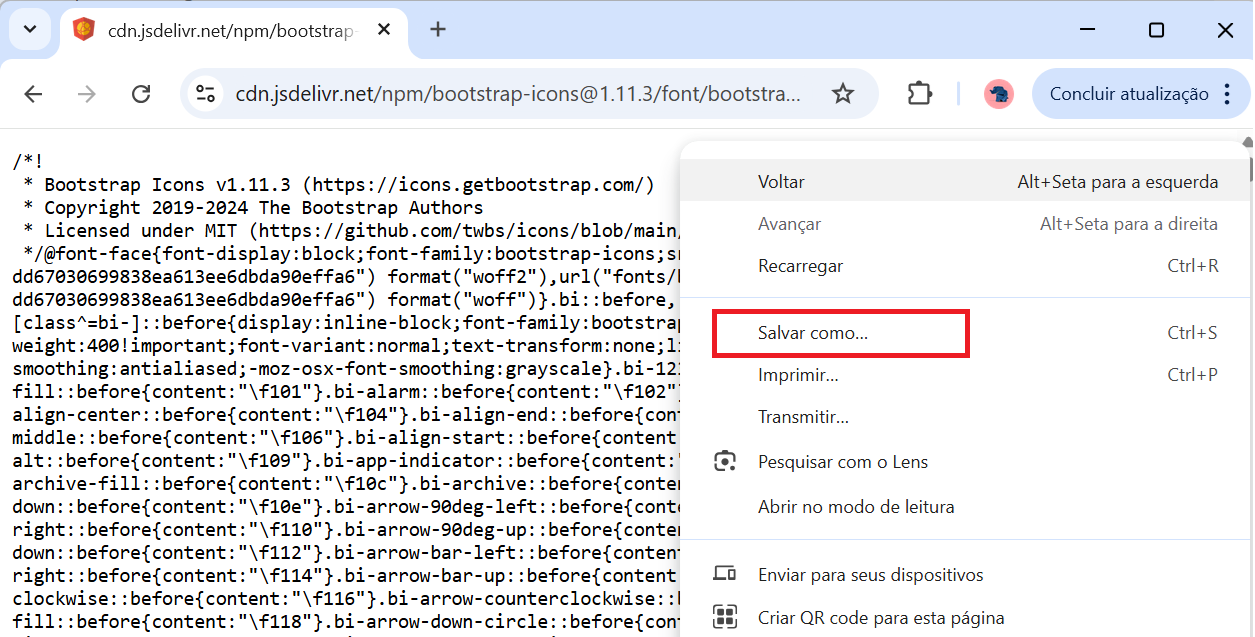
Note no código a utilização da Biblioteca de ícones do Bootstrap:



Faça o download da Biblioteca acessando o seguinte endereço:

https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css

Clique com o botão direito na página e selecione Salvar como... Salve o arquivo na pasta public\css com o nome bootstrap-icons.css

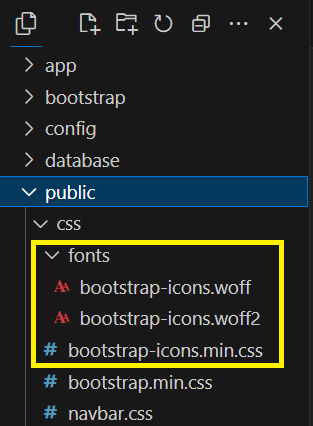


Faça o download do arquivo bootstrap-icons.woff2 através do link a seguir e salve o arquivo dentro da pasta public\css\fonts (crie a pasta fonts se não existir):

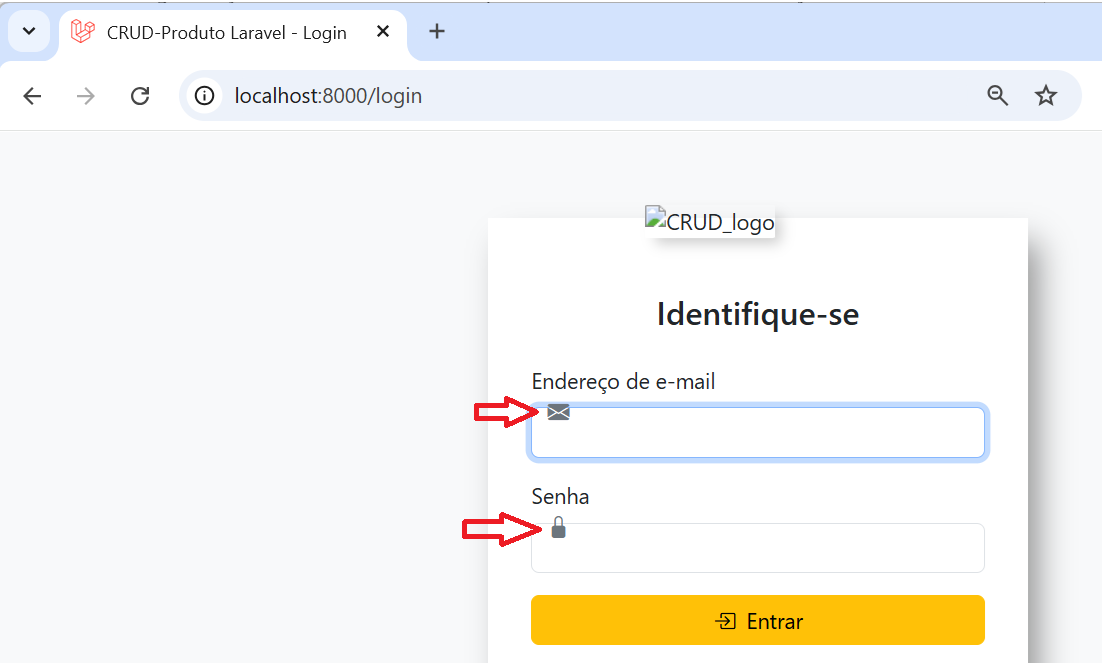
https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/fonts/bootstrap-icons.woff2

Faça o mesmo para o arquivo bootstrap-icons.woff salvando-o na mesma pasta public\css\fonts

https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/fonts/bootstrap-icons.woff



Acesse novamente a página de Login da Aplicação. Já deve aparecer os ícones.

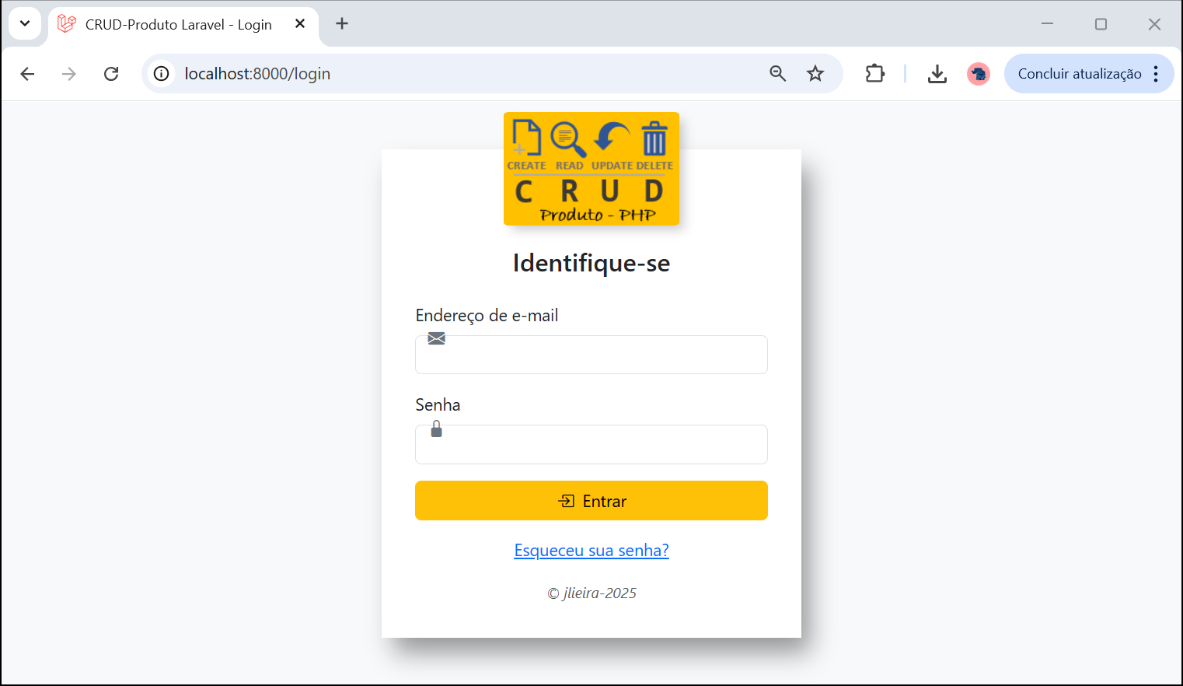


Só está faltando a figura da Logo da Aplicação, cujo arquivo com o nome CrudLogo.png será fornecido com esta documentação. Crie a pasta imagens dentro da pasta public da Aplicação e copie o arquivo de imagem para dentro dela:

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Teste o acesso à página de Login novamente. Veja que agora a página está completa:



**7.1.4. Validando as Credenciais de Login**

Primeiramente habilite o atributo action do formulário de login. No arquivo resources\view\login\formLogin.blade.php alterando a seguinte linha:

<form action="{{-- route('validarLogin') --}}" method="post">

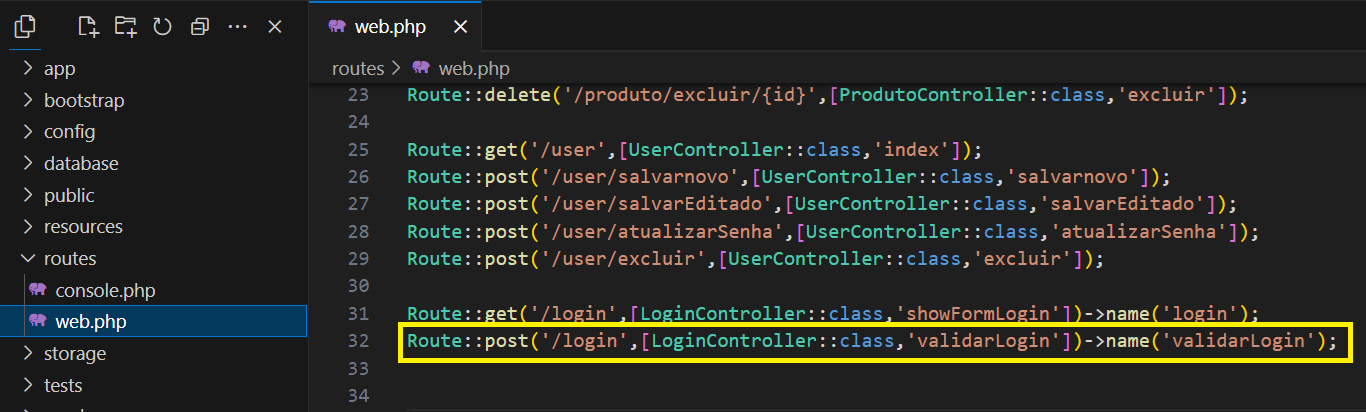
Retire os hifens dentro das chaves. Estes hifens é a forma de comentário do Laravel. Então a linha deverá ficar assim:

<form action="{{ route('validarLogin') }}" method="post">

Note que esta forma de indicar uma URL está usando o esquema de nomeação de rotas. Portanto, quando criarmos a rota para envio das credenciais, temos que nomeá-la de validarLogin. Isso será feito na sequência.

No arquivo routes\web.php acrescente a rota para a validação das credenciais de acesso:

Route::post('/login',[LoginController::class,'validarLogin'])->name('validarLogin');



Note que definimos a Rota com a mesma URL /login, entretanto o verbo HTML é post. E deve ser post porque foi o método de envio que definimos no formulário das credenciais.

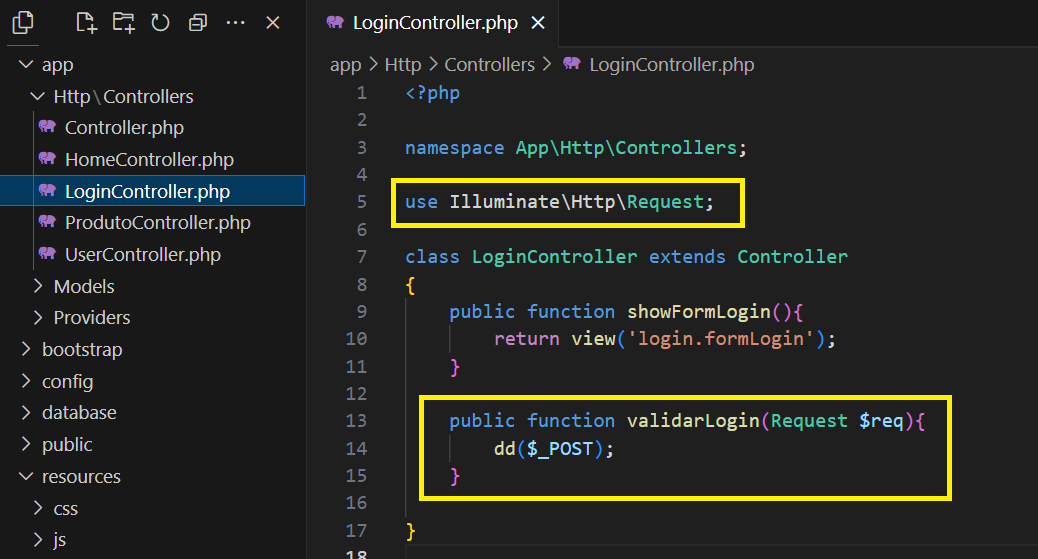
Para fins de teste, acrescente o método validarLogin() no LoginController.php, com o seguinte código:

use Illuminate\Http\Request;

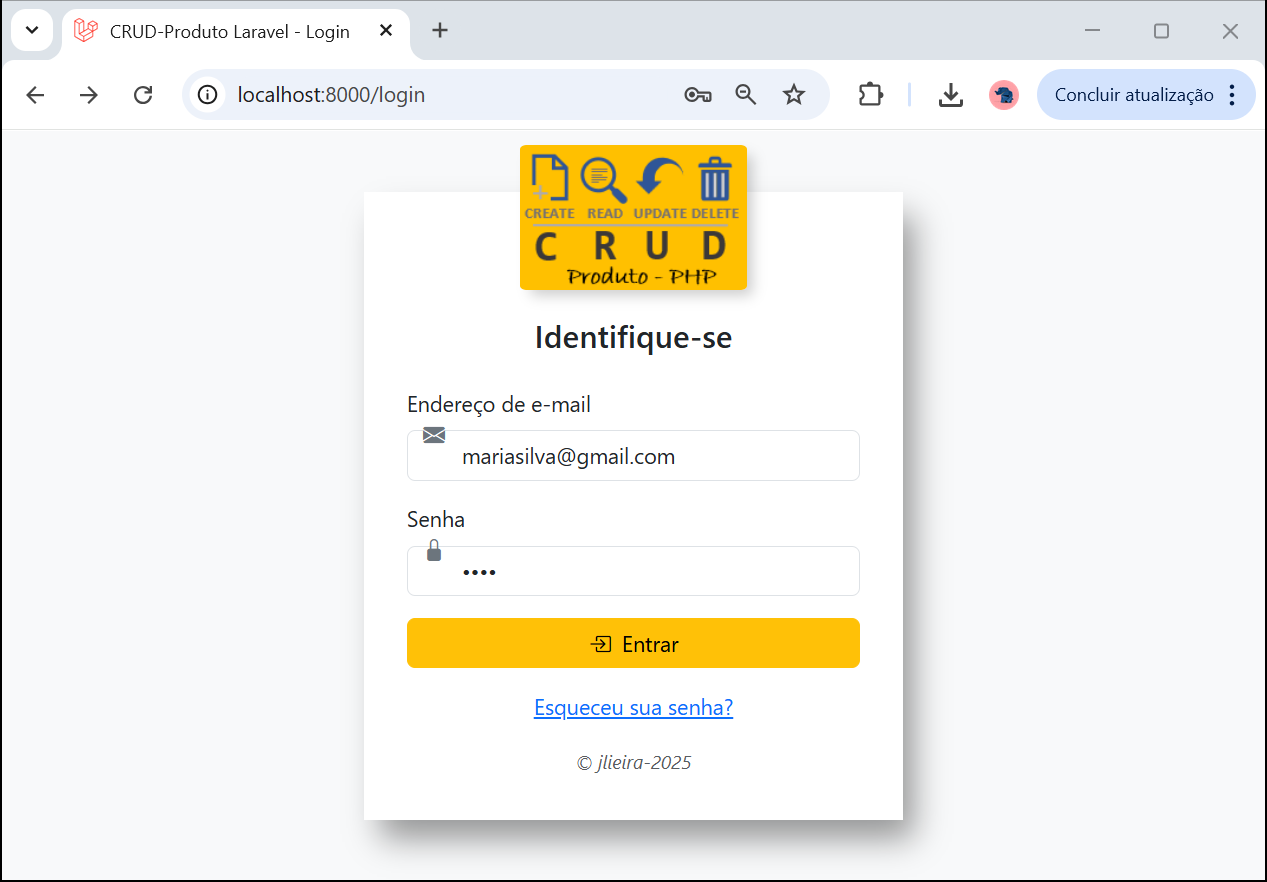
public function validarLogin(Request $req){

dd($\_POST);

}



Salve tudo e teste novamente. Não esqueça de atualizar o Navegador (CTRL F5). Digite algumas credenciais (mesmo que ainda não existam no Banco) e clique em Entrar.



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

O conteúdo gerado por IA pode estar incorreto.

**Validando com validate()**

Agora que já temos os dados no controlador, vamos utilizar o método validate() do Laravel para validar as credenciais. Troque o conteúdo do método validarLogin() para o seguinte:

public function validarLogin(Request $req){

// Validação com mensagens personalizadas

$req->validate([

'email' => 'required|max:255',

'senha' => 'required|min:4|max:255',

], [

'email.\*' => 'O campo Email é obrigatório!',

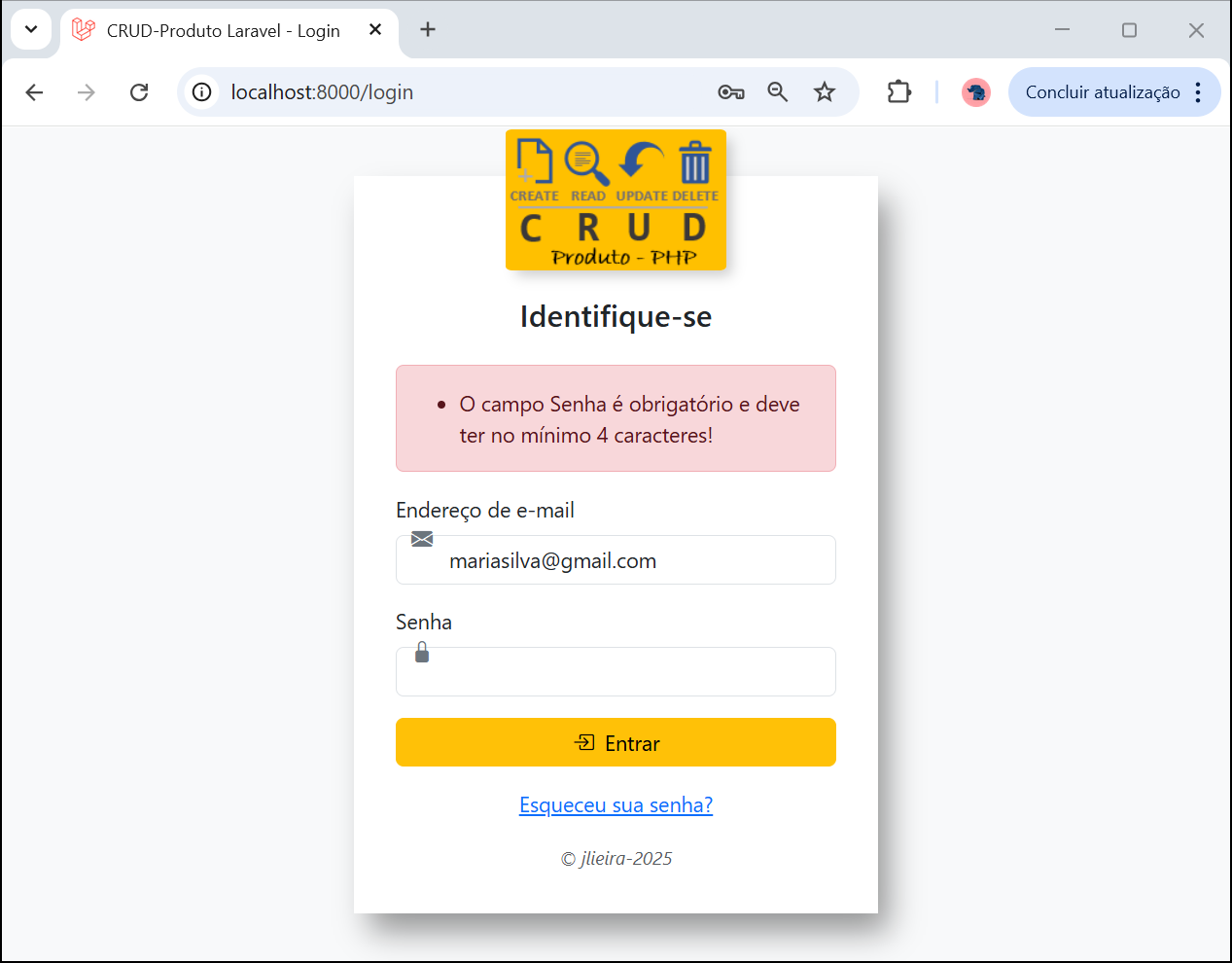
'senha.\*' => 'O campo Senha é obrigatório e deve ter no mínimo 4 caracteres!',

]

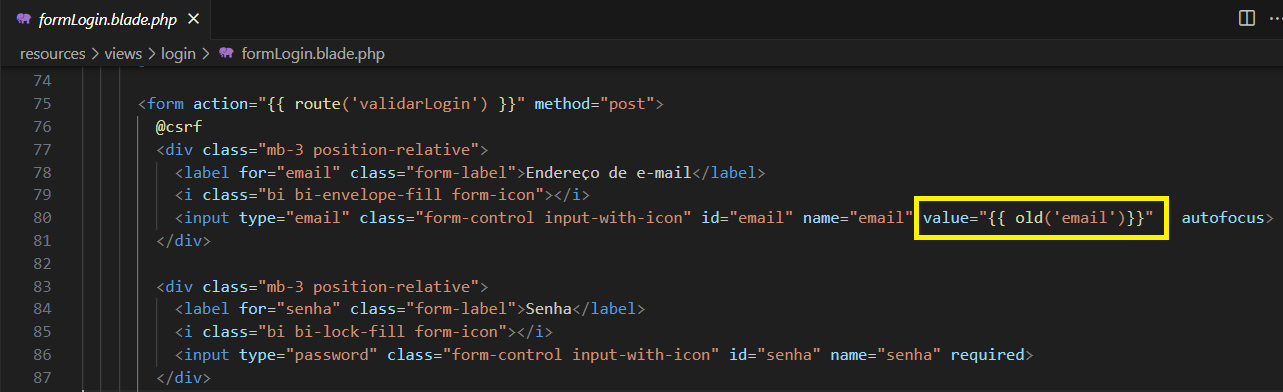
);

die("Passou pelas validações"); }

Salve e teste novamente. Entre com uma senha menor que 4 caracteres. Note que o formulário de Login vai ser reapresentado com um erro.



Note que o e-mail digitado anteriormente foi reapresentado. Isso porque está sendo usado o método old() do Laravel no formulário. Já a senha não.



Caso digite o e-mail no formato de e-mail e a senha com pelo menos 4 caracteres, será exibido a frase “Passou pelas validações”. Isto significa que as informações foram digitadas nos seus formatos corretos, porém, ainda falta verificar se o usuário está cadastrado e se a senha está correta.

**Verificando credenciais com informações do Banco**

No método validarLogin() do LoginController.php, remova o comando die() e acrescente o seguinte código:

//Pega as informações de email/senha recebidas do Formulário

$email = $req->get('email');

$senha = $req->get('senha');

//Busca o usuário no Banco via o email digitado

$usuario = User::where('email',$email)->get()->first();

//Verifica se foi encontrado um usuário com o email e se a senha está correta

if ($usuario && Hash::check($senha, $usuario->password)) {

//Login bem-sucedido - Coloca as informações do usuário na Sessão

session(['usuario.logado'=>true]);

session(['usuario.id'=>$usuario->id]);

session(['usuario.name'=>$usuario->name]);

session(['usuario.email'=>$usuario->email]);

session(['usuario.permissao'=>$usuario->permissao]);

//Direciona o usuário logado para a Home da Aplicação ou

//a página que ele tinha intenção de acessar anteriormente

return redirect()->intended('/');

} else {

//Credenciais inválidas

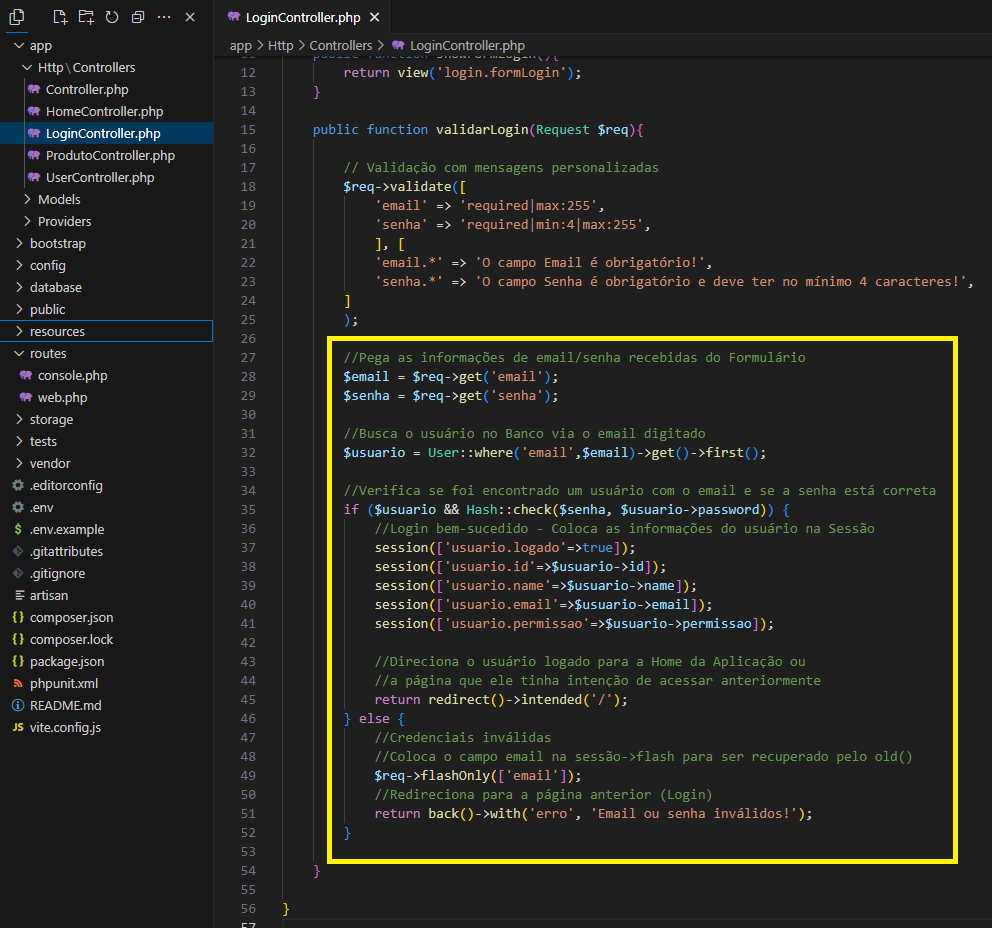
//Coloca o campo email na sessão->flash para ser recuperado pelo old()

$req->flashOnly(['email']);

//Redireciona para a página anterior (Login)

return back()->with('erro', 'Email ou senha inválidos!');

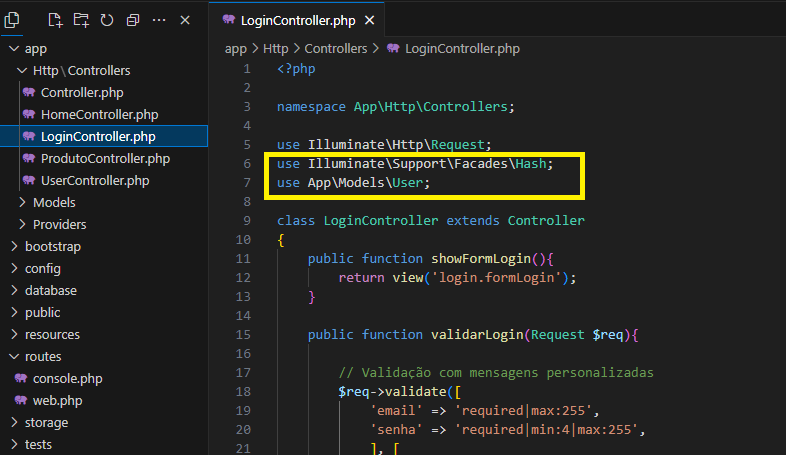
}



Também é necessário acrescentar as seguintes Uses no início do arquivo LoginController.php:

use Illuminate\Support\Facades\Hash;

use App\Models\User;



Salve tudo e teste novamente com usuário cadastrado no Banco. Com as credenciais corretas, após a validação, o usuário será direcionado para a página Home da Aplicação.

**7.1.5. Menu item Perfil do Usuário**

**Objetivo**: Implementar a funcionalidade de Logout, ou seja, saída da Aplicação com a destruição das variáveis de controle de sessão;

Aqui vamos implementar um Menu Perfil do Usuário no qual teremos o item Sair para fazer logout da Aplicação. Então acrescentar um menu item no menu navbar com o nome do usuário que quando clicado vai mostrar um menu dropdown com as opções Perfil, Trocar Senha e Sair. Para tanto, edite o script resources\views\layout\header.blade.php e acrescente o seguinte código logo após o fechamento da tag </ul>. Veja na Figura a seguir a posição correta.

<ul class="navbar-nav">

<li class="nav-item dropdown" style="margin-right: 40px">

<a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown">

<i class="bi bi-person-fill"></i> &nbsp;{{ session('usuario.name')}}

</a>

<div class="dropdown-menu dropdown-menu-rend" >

<a class="dropdown-item" href="#">Perfil</a>

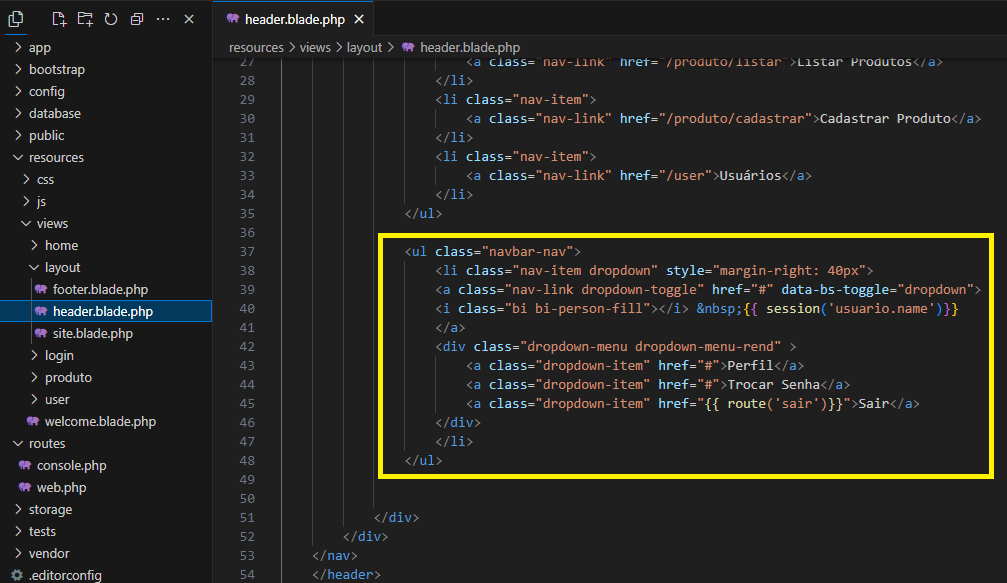
<a class="dropdown-item" href="#">Trocar Senha</a>

<a class="dropdown-item" href="{{ route('sair')}}">Sair</a>

</div>

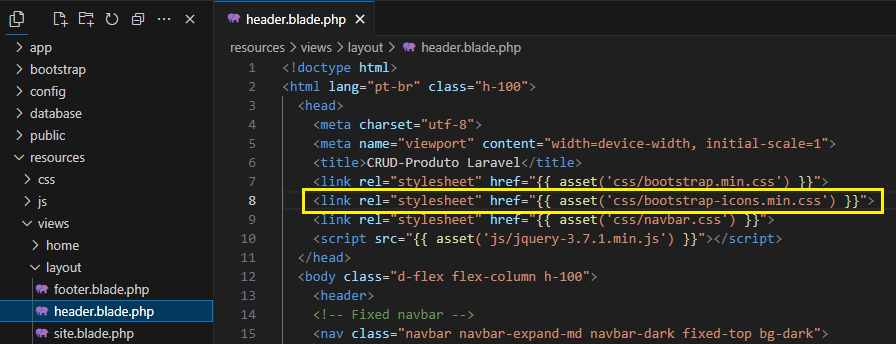
</li>

</ul>



Ainda no arquivo header.blade.php acrescente no início o carregamento da biblioteca de ícones:

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



Acrescente os estilos css (código a seguir) no final do arquivo public\css\navbar.css.

/\* Estilos aplicados ao Menu Dropdown do Perfil do usuário \*/

.dropdown-menu {

background-color: #343a40;

border: 0;

border-radius: 0;

box-shadow: 6px 6px 10px -2px rgba(0, 0, 0, 0.4);

}

/\*Ao passar o mouse sobre o Menu Horizontal, abre o Menu Vertical\*/

.nav-item:hover>.dropdown-menu {

display: block;

}

/\*#menuPrincipal ul li a:hover{\*/

.navbar div ul li a:hover{

background-color: black;

}

/\*Quando clicar no item do menu Vertical, fica marcado\*/

.dropdown-item:focus{

background-color: #343a40; /\*Usei a mesma cor de fundo para não perceber a marcação\*/

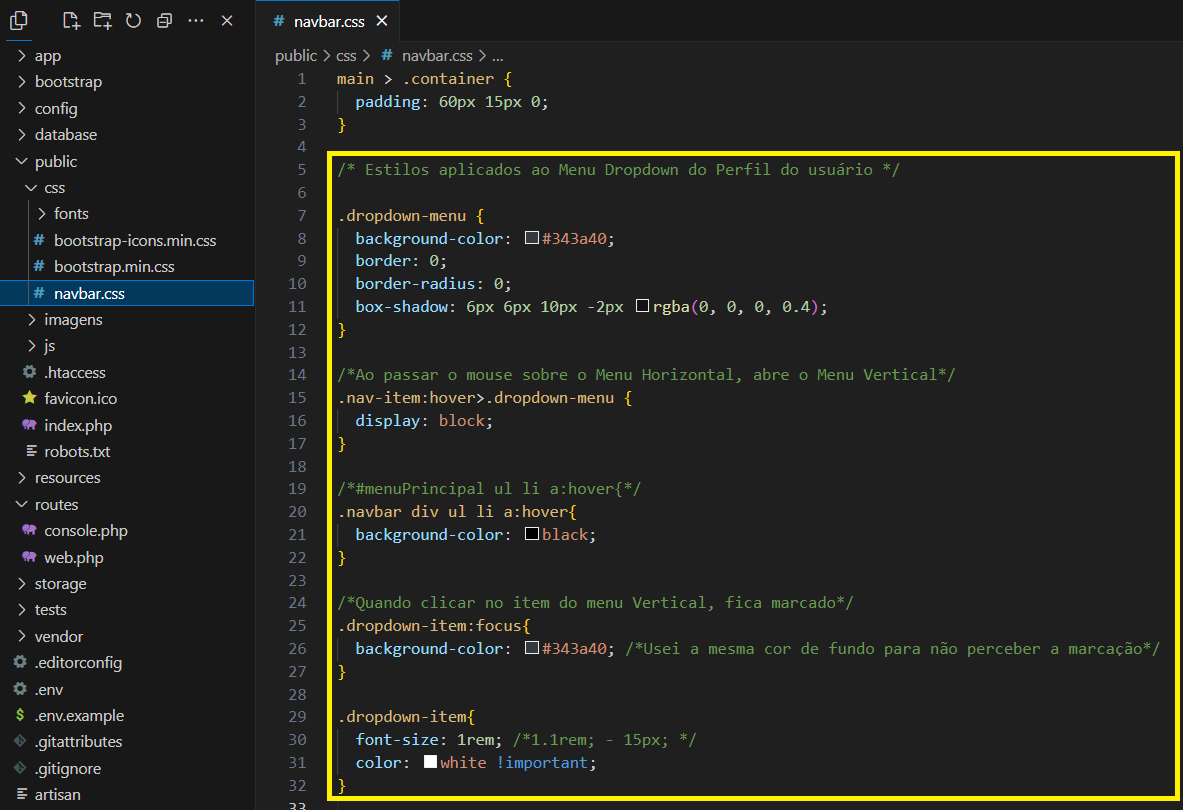
}

.dropdown-item{

font-size: 1rem; /\*1.1rem; - 15px; \*/

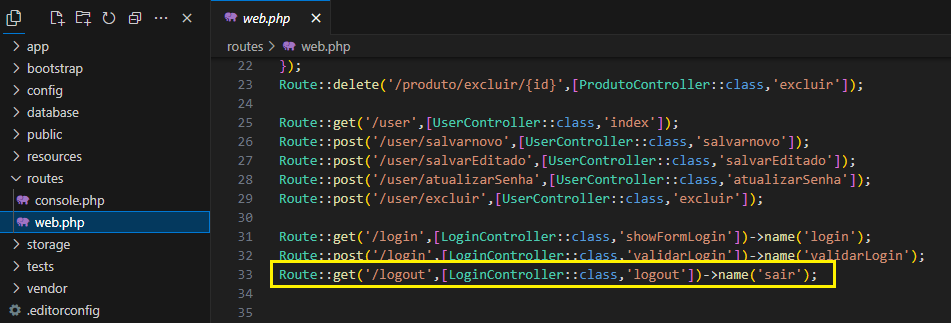
color: white !important;

}



Crie a rota para a função sair acrescentando a linha a seguir no resources\web.php:

Route::get('/logout',[LoginController::class,'logout'])->name('sair');



Por fim, acrescente o método logout na classe LoginController.php com o seguinte conteúdo:

public function logout(Request $req){

//Remove todos os dados da Sessão do usuário

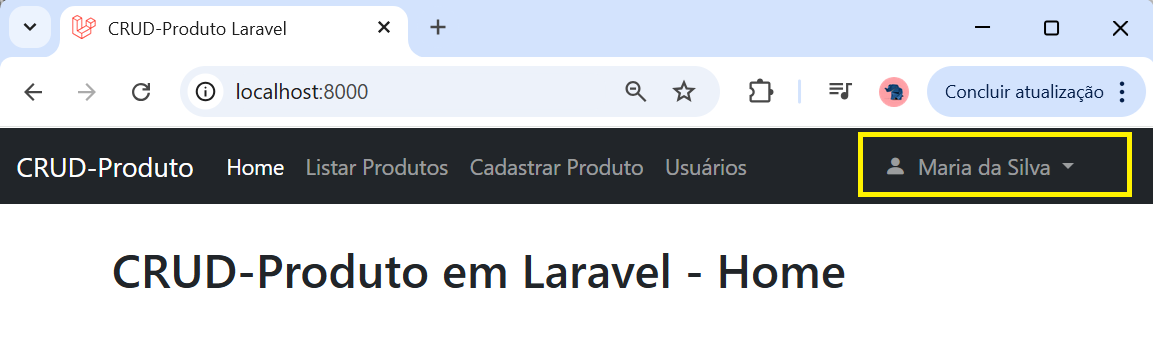
$req->session()->flush();

//Redireciona o usuário para a página de Login

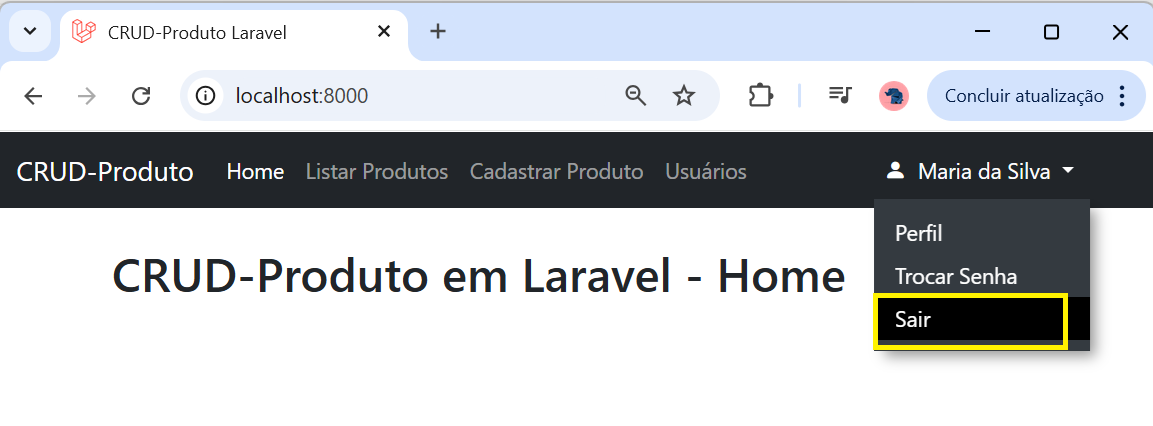
return redirect()->route('login');

}

Salve tudo e teste. Após entrar com um usuário válido, note o item de menu com o nome do usuário mais à direita.



Clique na seta para abrir o menu dropdown e selecione o item Sair.



**7.1.6. Controle de Sessão**

Embora tenhamos implementado a página de Login para Autenticação de Usuários, do jeito que está um usuário não autorizado ainda pode ter acesso ao sistema sem passar pela autenticação, simplesmente digitando o endereço da página home da Aplicação ( <http://localhost:8000/home> ). Faça o teste.

Assim, é preciso fazer o controle de sessão, ou seja, caso o usuário tente acessar uma página da Aplicação sem ter autenticado, será redirecionado para a página de Login. No Laravel esse controle pode ser feito através de um Middleware.

Um Middleware é uma camada de software que fica entre o Usuário e nossa Aplicação. Middlewares fornecem um mecanismo para inspecionar e filtrar requisições HTTP que chegam à aplicação.



Usuário

**Middleware**

Request

Response

Request

Response

Todo pedido (Request) que vem do Usuário é interceptado pelo Middleware antes de chegar na Aplicação. Neste ponto é possível decidir se a requisição será ou não repassada para a Aplicação. Também é possível passar parâmetros do Middleware para a Aplicação. Da mesma forma, o Middleware pode interceptar e alterar a Resposta (Response) da Aplicação antes de enviá-la ao Usuário.

O Laravel já possui vários middlewares, incluindo middlewares para autenticação e proteção contra CSRF. O middleware de Autenticação do Laravel intercepta todos os pedidos do Usuário e já verifica se o usuário está autenticado. Se o usuário não estiver autenticado, o middleware o redirecionará para a tela de login No entanto, se o usuário estiver autenticado, o middleware permitirá que a requisição prossiga para dentro da aplicação.

Middlewares adicionais podem ser criados para executar uma variedade de tarefas além da autenticação. Por exemplo, um middleware de Log pode registrar todas as requisições recebidas em sua aplicação.

Toda a documentação sobre Middleware no Laravel 11 pode ser acessada em: <https://laravel.com/docs/11.x/middleware>

Embora o Laravel já possua um Middleware de Autenticação, nesta sessão vamos criar nosso próprio Middleware de Autenticação com o objetivo de aprender a trabalhar com middleware.

**Criando um Middleware para Autenticação**

**Objetivo**: Criar um Middleware que verifica se o usuário está Autenticado. Se não estiver, direciona para a tela de login.

Primeiro é preciso criar o Middleware via artisan:

php artisan make:middleware Autenticacao

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

O comando acima vai criar o arquivo app\Http\Middleware\Autenticacao.php que deve ter o seguinte conteúdo:

Texto

O conteúdo gerado por IA pode estar incorreto.

É dentro do método handle() que devemos colocar as regras do nosso mddleware para decidir se o pedido deve prosseguir ou não. Em nosso caso, devemos verificar se o usuário está logado. Podemos fazer isso verificando a informação usuário.logado na Sessão.

Troque o conteúdo do método handle() para:

if (session('usuario.logado')){ //Usuário Autenticado

return $next($request);

}else{ //Usuário ainda NÃO Autenticado

//Redireciona para a página anterior (Login)

return redirect()

->route('login')

->with('erro', 'Por favor, Identifique-se!');

}

Para que o Middleware faça seu trabalho, é preciso que seja registrado no mecanismo de Middleware do Laravel. Isso é feito no arquivo bootstrap\app.php:

Texto

O conteúdo gerado por IA pode estar incorreto.

O registro é feito dentro da função ->withMiddleware(function (Middleware $middleware) { e pode ser feito de duas formas:

* **Global**: para isso deve-se utilizar o método append() assim:

$middleware->append([\App\Http\Middleware\Autenticacao::class,]);

Se o middleware for registrado como Global, ele será verificado para todas as requisições à Aplicação. O problema aqui é que nossa aplicação ficará em loop, pois as rotas ‘login’ e ‘validaLogin’ também chamarão o middleware.

* **Por rota**: neste esquema, o middleware deve ser registrado com o método alias() assim:

$middleware->alias([\App\Http\Middleware\Autenticacao::class,]);

Neste caso, também é necessário chamar o middleware em cada rota que desejamos usá-lo.

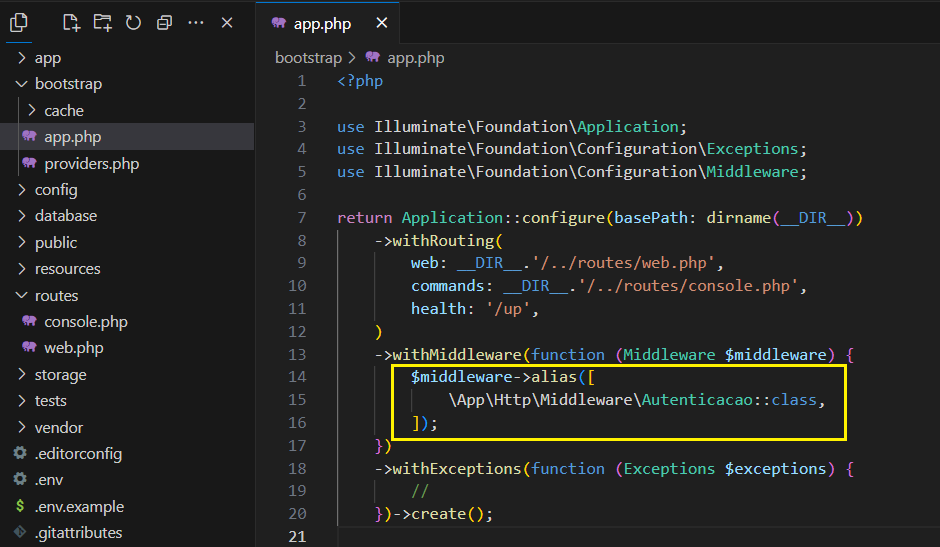
**Registrando o Middleware por Rota**

No arquivo bootstrap\app.php registre o middleware acrescentando o código:

$middleware->alias([

\App\Http\Middleware\Autenticacao::class,

]);



**Atribuindo o Middleware à Rota**

Vamos atribuir o Middleware para a rota /produto/listar da Aplicação. Então, no arquivo routes\web.php altere a rota para:

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

->middleware([Autenticacao::class]);

E também acrescente a linha a seguir no início do arquivo:

use App\Http\Middleware\Autenticacao;



Salve. Vá até o navegador e, caso já esteja logado, saia da Aplicação pelo Sair do menu Perfil do usuário. Agora tente acessar o endereço: <http://localhost:8000/produto/listar> Deverá ser redirecionado para a tela de login. Porém tente acessar <http://localhost:8000/home> O acesso agora será permitido.

Portanto, temos que proteger todas as rotas da Aplicação que necessitam de Autenticação, menos, é claro, as rotas Login e validarLogin. Podemos fazer isso aplicando o middleware à cada rota, o que seria trabalhoso, ou agrupar as rotas que necessitam de autenticação e aplicar o middleware para o grupo.

**Aplicar o Middleware a um Grupo de Rotas**

Troque todo o conteudo do arquivo routes\web.php para o seguinte:

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\HomeController;

use App\Http\Controllers\ProdutoController;

use App\Http\Controllers\UserController;

use App\Http\Controllers\LoginController;

use App\Http\Middleware\Autenticacao;

Route::middleware([Autenticacao::class])->group(function () {

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

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

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

->middleware([Autenticacao::class]);

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

Route::post('/produto/salvar',[ProdutoController::class,'salvar']);

Route::get('/produto/editar/{id}',[ProdutoController::class,'editar']);

Route::post('/produto/atualizar',[ProdutoController::class,'atualizar']);

Route::get('/produto/confirmaExcluir/{id}/{nome}', function($id, $nome){

return view('produto.confirmaExcluir', ['id' => $id, 'nome' => $nome]);

});

Route::delete('/produto/excluir/{id}',[ProdutoController::class,'excluir']);

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

Route::post('/user/salvarnovo',[UserController::class,'salvarnovo']);

Route::post('/user/salvarEditado',[UserController::class,'salvarEditado']);

Route::post('/user/atualizarSenha',[UserController::class,'atualizarSenha']);

Route::post('/user/excluir',[UserController::class,'excluir']);

Route::get('/logout',[LoginController::class,'logout'])->name('sair');

});

Route::get('/login',[LoginController::class,'showFormLogin'])->name('login');

Route::post('/login',[LoginController::class,'validarLogin'])->name('validarLogin');

Texto

O conteúdo gerado por IA pode estar incorreto.

Salve e teste novamente. Se estiver logado, faça logout. Depois tente acessar as rotas diretamente. Agora todas as rotas devem estar protegidas.

**7.2. Autenticação com Laravel Breeze**

**Importante**: Antes de iniciar, renomeie a pasta da Aplicação para produtoappComAutenticacaoManual e restaure a cópia da pasta anterior à implementação da Autenticação Manual para produtoapp. Além disso, faça uma cópia do arquivo routes\web.php em uma pasta fora da Aplicação.

Requisitos recomendados para utilizar o Breeze:

* **Node.js** versão 18.x ou v20.x;
* **NPM** versão 9.x ou superior;
* **PHP 8.2** ou superior;
* **Composer**

Para instalar o node via nvm, execute o comando abaixo no prompt:

nvm install 20

Texto

O conteúdo gerado por IA pode estar incorreto.

Depois execute o comando:

nvm use 20.19.0

Texto

O conteúdo gerado por IA pode estar incorreto.

Caso não tenha o nvm, baixe o nvm, faça o download em <https://github.com/coreybutler/nvm-windows>

**7.2.1. Instalando o Laravel Breeze**

Execute o seguinte comando no terminal dentro da aplicação Laravel:

composer require laravel/breeze --dev

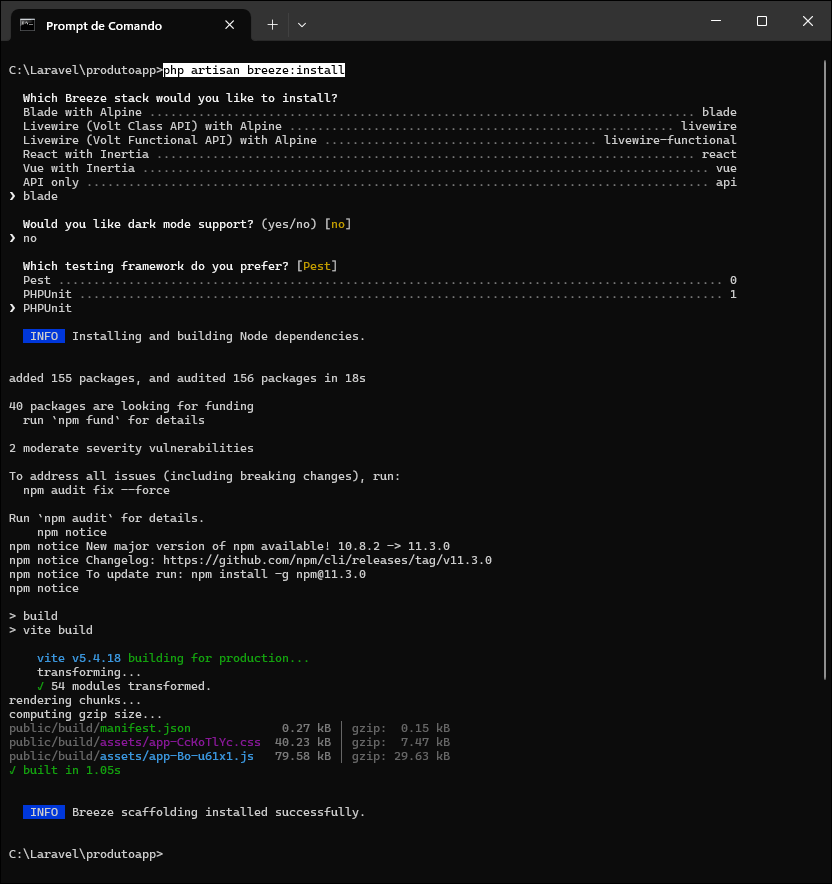
O parâmetro --dev é para instalar o pacote somente em desenvolvimento. Depois que gerar a estrutura, ele não precisa ser executado em produção.

Texto

O conteúdo gerado por IA pode estar incorreto.

Agora instale o frontend que será utilizado com o Breeze. Por padrão, o Breeze usa o Tailwind CSS. Mas é possível escolher entre livewire, react e vue. Selecione **blade**, depois responda **no** para dark mode e depois **PHPUnit** para framework de teste.

php artisan breeze:install



Caso esteja iniciando um projeto ((NÃO É O NOSSO CASO)) e ainda não executou as migrations, execute o seguinte comando:

php artisan migrate

**7.2.2. Instalar as dependências do Frontend e compilar**

Execute o comando a seguir:

npm install && npm run dev

Esse comando é usado para compilar os assets (CSS, JS, etc.) e ficar assistindo mudanças em tempo real para refazer os assets caso sejam alterados. Note que após a execução do comando acima, o terminal será travado e exibe uma tela como a seguir:

Texto

O conteúdo gerado por IA pode estar incorreto.

Isso é normal. Mostra que o Vite (plugin do Laravel 11 responsável pelo frontend) está monitorando as alterações nos arquivos .js, .css, .ts, .vue relacionados ao frontend. Isso é necessário somente em desenvolvimento. Em produção não é necessário executar esse comando. Portanto, temos que deixá-lo executando.

Abra uma nova aba no prompt (ou outro prompt), vá até a pasta da nossa aplicação (produtoapp) e execute o servidor embutido do PHP para servir nossa aplicação php artisan serve

Texto

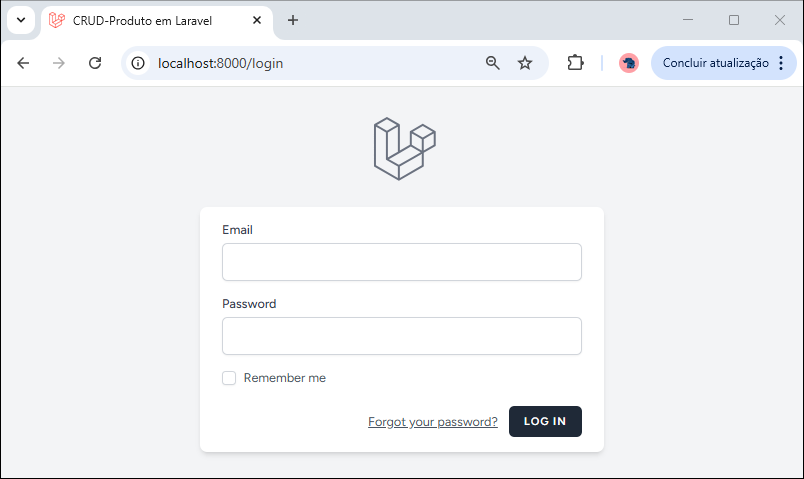
O conteúdo gerado por IA pode estar incorreto.

Acesse a Aplicação no Navegador. Note que será exibida uma página padrão do Laravel e Não a página home da nossa Aplicação.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Note do canto superior direito os links Log in e Register. Clique em Log in. Será exibida a página de Login criada pelo Breeze.



Digite as credenciais (email e senha) de um usuário válido e veja que o acesso será permitido:

Interface gráfica do usuário, Site

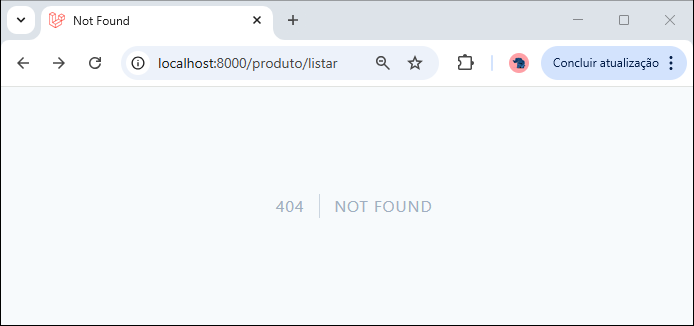
O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

**7.2.3. Integrando o Sistema de Autenticação do Breeze com nossa Aplicação**

Note que quando a autenticação está correta, o acesso é permitido, porém direcionado para uma página de ‘Dashboard’ que não corresponde à página principal da nossa Aplicação. Além disso, se tentar acesso a alguma rota da nossa aplicação, a página não será encontrada. Por exemplo, tente: http://localhost:8000/produto/listar



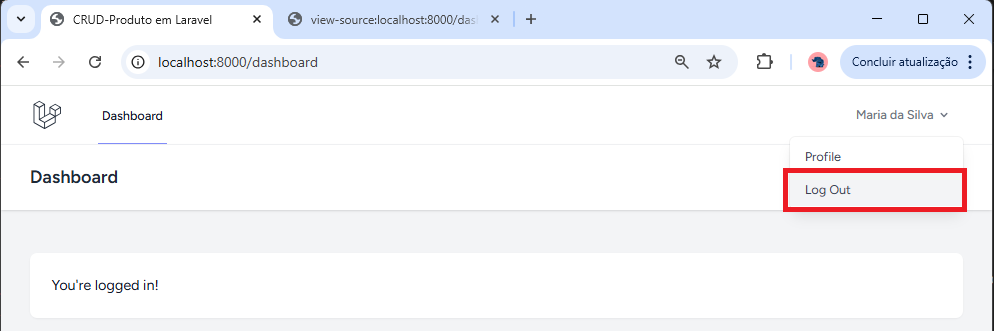
Isso se deve ao fato de o Breeze ter trocado todo o conteúdo do arquivo de rotas (routes\web.php) da nossa Aplicação. Veja:

Tela de computador com texto preto sobre fundo branco

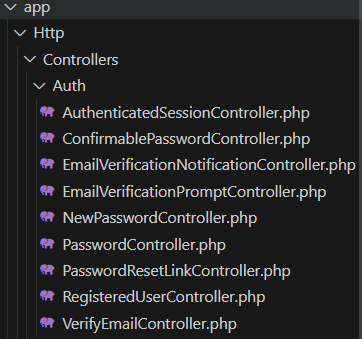
O conteúdo gerado por IA pode estar incorreto.

Assim, sempre que tiver a intenção de utilizar autenticação com Breeze, é recomendado já instalar o Breeze no início do desenvolvimento da Aplicação nomento em que for criar o projeto Laravel.

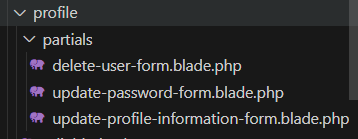
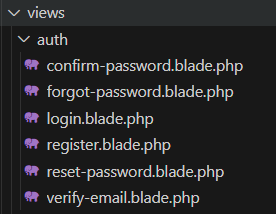
Para integrar o Breeze à nossa Aplicação já existente, primeiramente faça logout da Aplicação se estiver logado:



Além de alterar nosso arquivo de rotas (routes\web), o Breeze acrescentou vários arquivos no código da nossa Aplicação. Alguns são mostrados na figura a seguir.

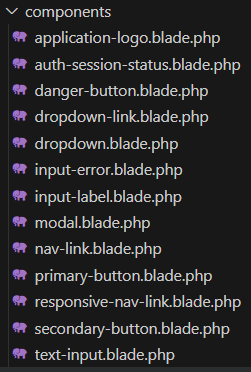


app\Http\Controllers\Auth

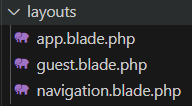


resources\views\auth

resources\views\profile\partials



resources\views\components



resources\views\layouts

**Adaptar o menu Perfil de Usuário**

É preciso colocar o Menu Perfil de Usuário no menu horizontal da nossa Aplicação. Edite o script resources\views\layout\header.blade.php e acrescente o seguinte código logo após o fechamento da tag </ul>. Veja na Figura a seguir a posição correta.

<ul class="navbar-nav">

<li class="nav-item dropdown" style="margin-right: 40px">

<a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown">

<i class="bi bi-person-fill"></i> &nbsp;{{ Auth::user()->name }}

</a>

<div class="dropdown-menu dropdown-menu-rend" >

<a class="dropdown-item" href="{{ route('profile.edit') }}">Perfil</a>

<a class="dropdown-item" href="{{ route('profile.edit') }}">Trocar Senha</a>

<form method="POST" action="{{ route('logout') }}">

@csrf

<a class="dropdown-item" href="route('logout')"

onclick="event.preventDefault();

this.closest('form').submit();">

{{ \_\_('Sair') }}

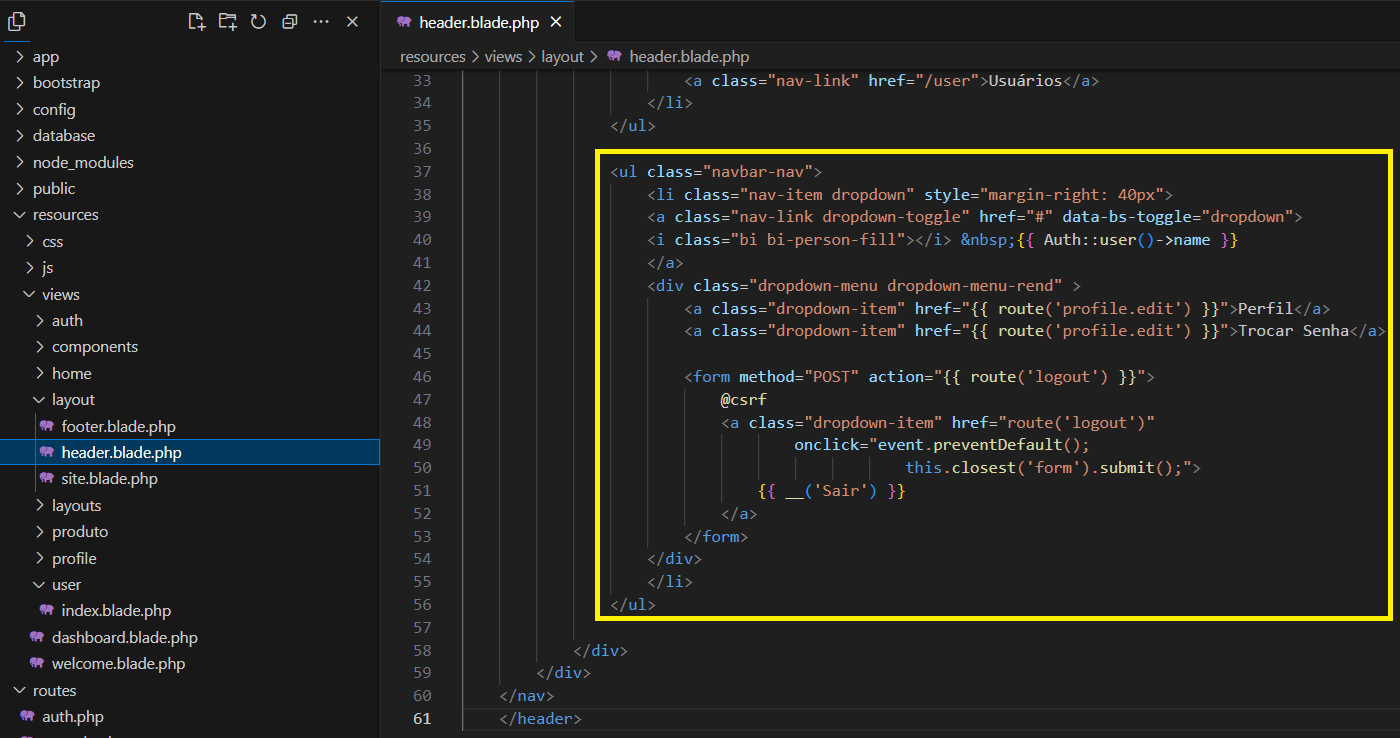
</a>

</form>

</div>

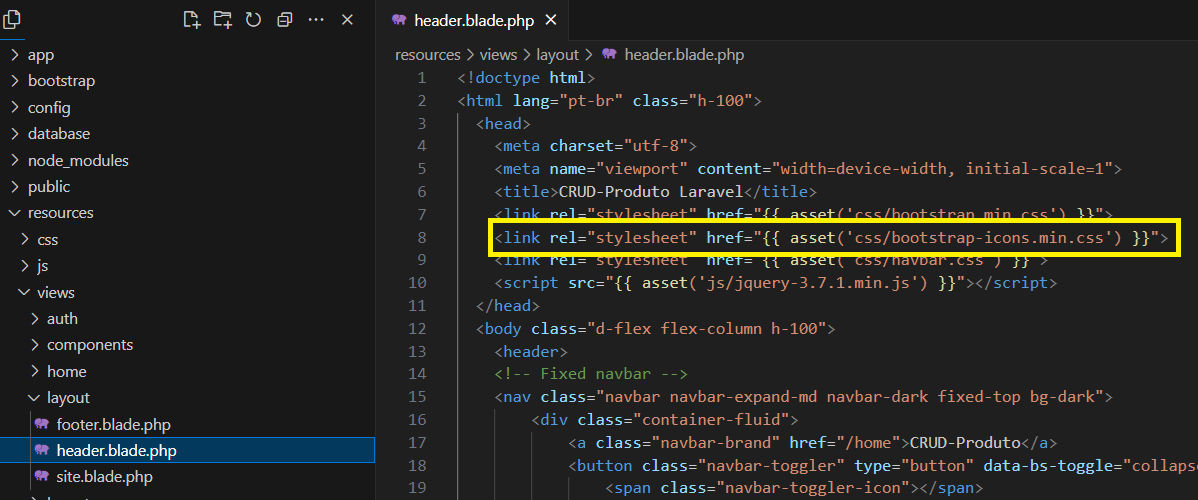
</li>

</ul>



Ainda no arquivo header.blade.php acrescente no início o carregamento da biblioteca de ícones:

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



Acrescente os estilos css (código a seguir) no final do arquivo public\css\navbar.css.

/\* Estilos aplicados ao Menu Dropdown do Perfil do usuário \*/

.dropdown-menu {

background-color: #343a40;

border: 0;

border-radius: 0;

box-shadow: 6px 6px 10px -2px rgba(0, 0, 0, 0.4);

}

/\*Ao passar o mouse sobre o Menu Horizontal, abre o Menu Vertical\*/

.nav-item:hover>.dropdown-menu {

display: block;

}

/\*#menuPrincipal ul li a:hover{\*/

.navbar div ul li a:hover{

background-color: black;

}

/\*Quando clicar no item do menu Vertical, fica marcado\*/

.dropdown-item:focus{

background-color: #343a40; /\*Usei a mesma cor de fundo para não perceber a marcação\*/

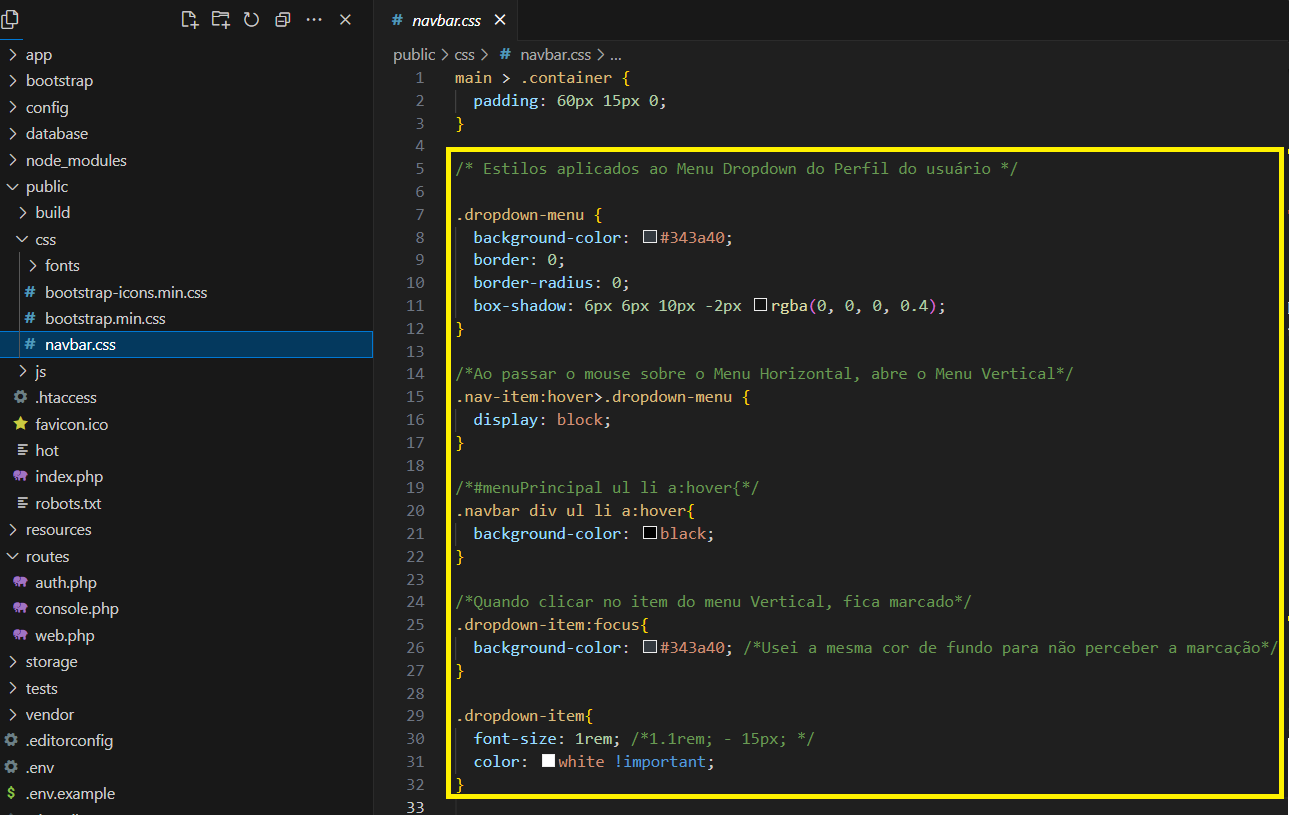
}

.dropdown-item{

font-size: 1rem; /\*1.1rem; - 15px; \*/

color: white !important;

}



Baixe a biblioteca de ícones do Bootstrap conforme procedimentos descritos na página 7 ou copie do projeto anterior, colocando em public\css

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Troque todo o conteúdo do arquivo de rotas routes\web.php pelo seguinte:

<?php

use App\Http\Controllers\ProfileController;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\HomeController;

use App\Http\Controllers\ProdutoController;

use App\Http\Controllers\UserController;

// Route::get('/', function () {

// return view('welcome');

// });

Route::get('/dashboard', function () {

return view('dashboard');

})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {

Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');

Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');

Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');

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

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

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

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

Route::post('/produto/salvar',[ProdutoController::class,'salvar']);

Route::get('/produto/editar/{id}',[ProdutoController::class,'editar']);

Route::post('/produto/atualizar',[ProdutoController::class,'atualizar']);

Route::get('/produto/confirmaExcluir/{id}/{nome}', function($id, $nome){

return view('produto.confirmaExcluir', ['id' => $id, 'nome' => $nome]);

});

Route::delete('/produto/excluir/{id}',[ProdutoController::class,'excluir']);

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

Route::post('/user/salvarnovo',[UserController::class,'salvarnovo']);

Route::post('/user/salvarEditado',[UserController::class,'salvarEditado']);

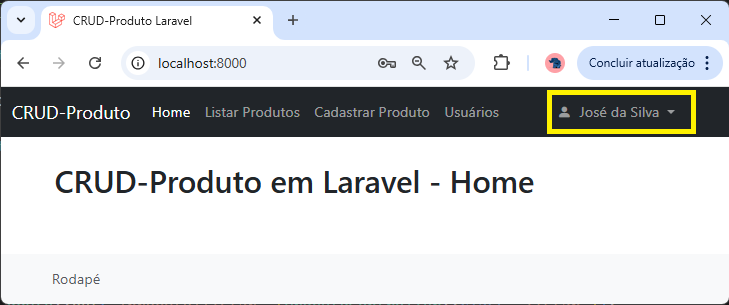
Route::post('/user/atualizarSenha',[UserController::class,'atualizarSenha']);

Route::post('/user/excluir',[UserController::class,'excluir']);

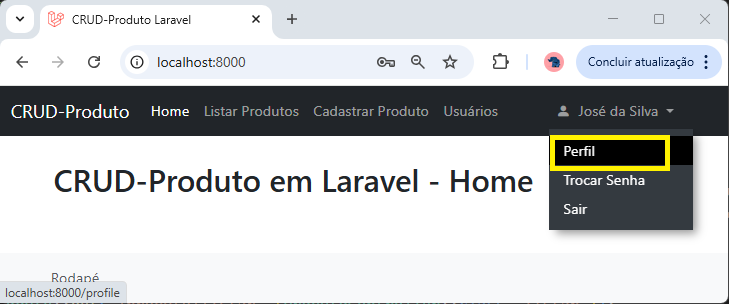
});

require \_\_DIR\_\_.'/auth.php';

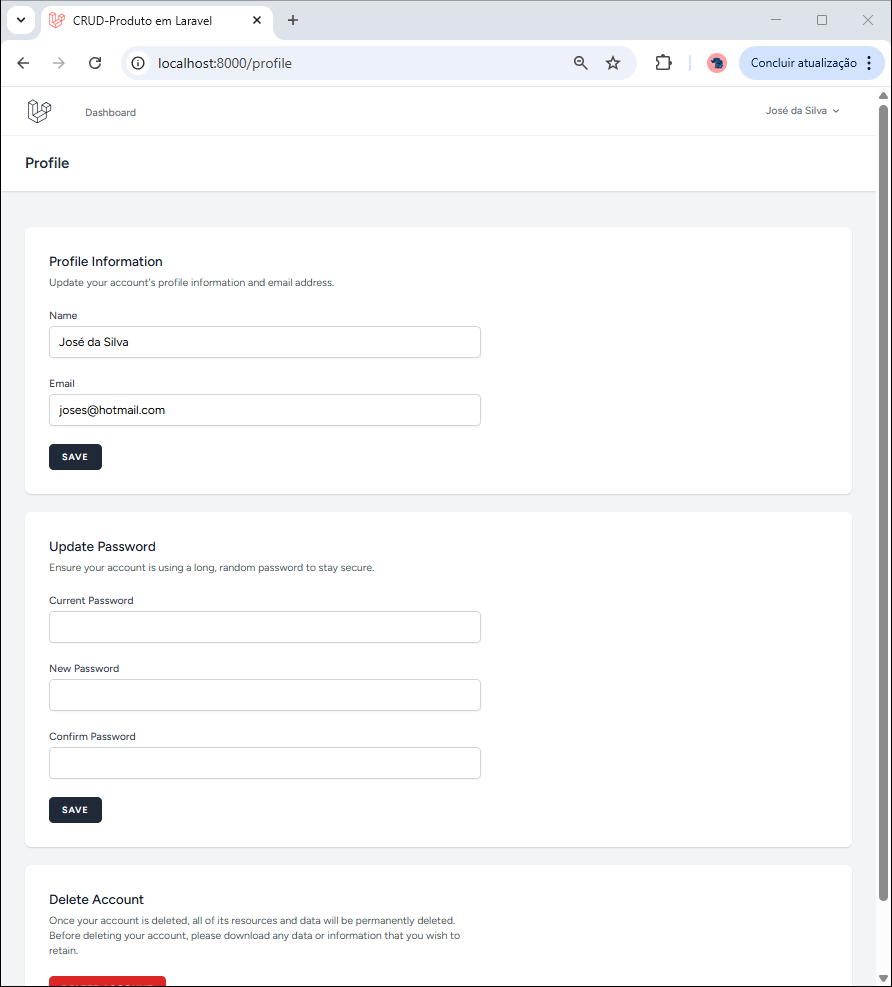
Salve tudo e teste ( <http://localhost:8000/> ) . Após entrar com um usuário válido, note o item de menu com o nome do usuário mais à direita.



Clique na seta para abrir o menu dropdown e selecione o item Perfil.

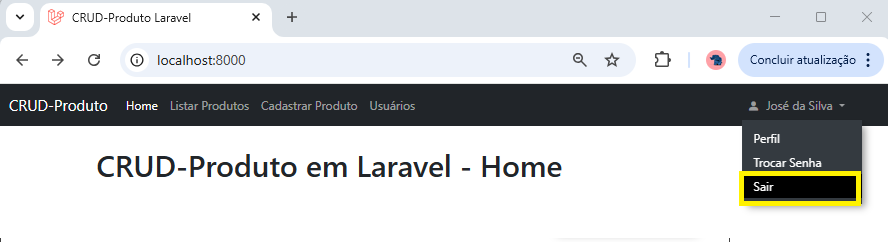


Será exibida a página de edição do Perfil do Usuário do Breeze.

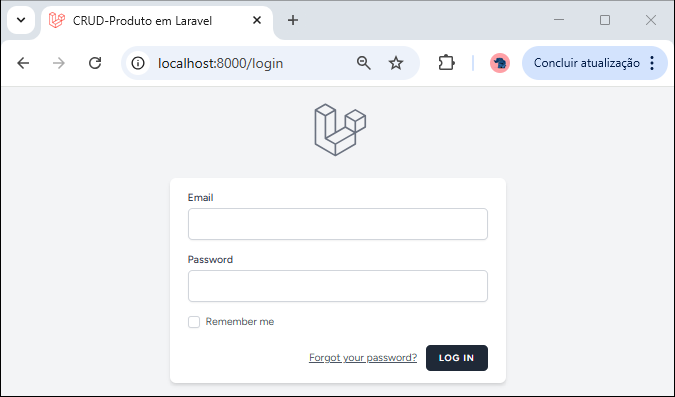


O problema é que esta página Não está no Layout da nossa Aplicação. A mesma tela também é mostrada quando o usuário clicar em Trocar Senha do menu Perfil.

Volte para a tela anterior pelo botão Voltar do Navegador e clique no item Sair do Menu Perfil do Usuário.



Note que será exibida a tela de Login do Breeze.



Tente acessar alguma parte da Aplicação sem se Autenticar, por exemplo <http://localhost:8000/produto/listar> Deve ser redirecionado para a página de Login e, portanto, o Controle de Sessão já está funcionando.