

**CURSOS
TÉCNICOS**

**DESENVOLVIMENTO DE
SISTEMAS WEB II**

Eixo Informática para Internet

Unidade 8

SUMÁRIO

UNIDADE 8

1. Recapitulando semana anterior.....	3
2. Middleware.....	4
2.1 Implementando Middleware de autenticação.....	4
2.2 Criando páginas de login e cadastro de usuário.....	7
2.3 Protegendo rotas da aplicação.....	12
2.4 Funcionalidade de logout.....	13
2.5 Permissionamento do projeto exemplo.....	14
2.6 Adaptando o layout com base no tipo de usuário.....	15
3. Publicação de projeto WEB.....	16
4. Referências.....	18

UNIDADE 8

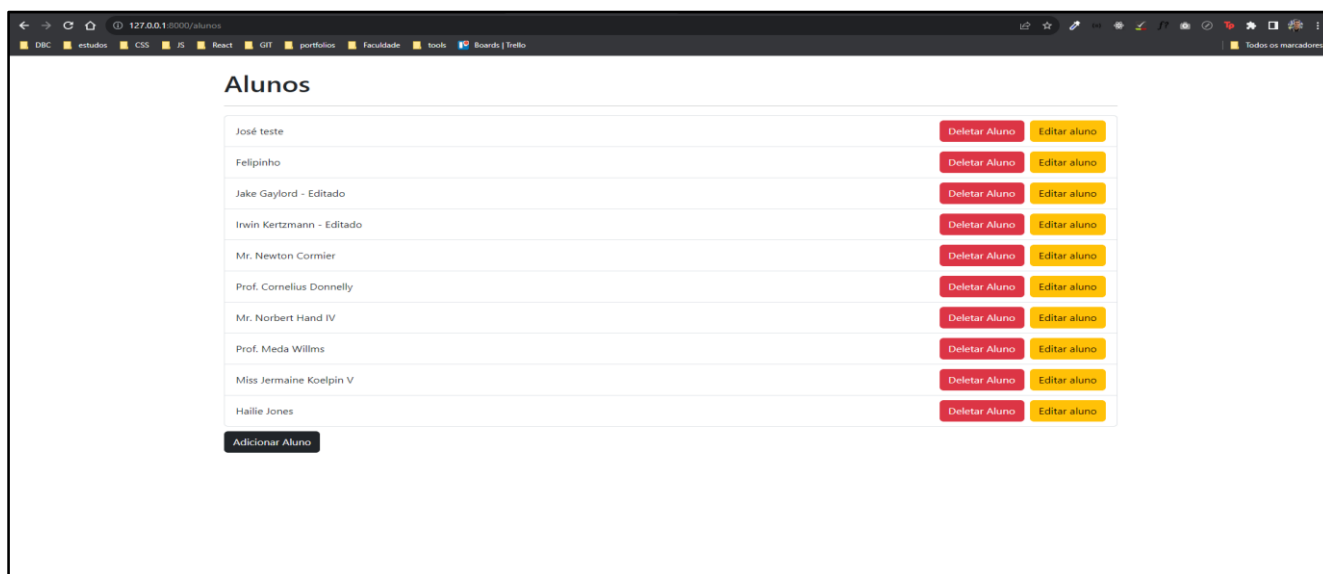
Middleware, autenticação e publicação de projetos

Nesta última unidade, iniciaremos como é de praxe pela recapitulação da unidade anterior, e na sequência vamos focar nos aspectos relacionados ao middlewares, autenticação e publicação de projetos.

1. Recapitulando semana anterior

Na última semana, fizemos um CRUD (Create, Read, Update, Delete) de uma aplicação baseada no controle e listagem de alunos de um determinado sistema fictício.

Projeto desenvolvido durante a unidade 7.



Note que, com esse simples sistema nós podemos **listar os nomes dos alunos** disponíveis na nossa base de dados, **atualizar as suas informações**, **criar um novo aluno** ou até mesmo **deletar o registro de um aluno**. Fizemos tudo isso, utilizando as facilidades do Laravel com o Eloquent.

Porém, por mais que esse projeto seja simples, **temos pontos de melhorias a fazer**. Como por exemplo:

- **Todos os usuários do nosso sistema podem Criar, Deletar, Atualizar ou Consumir informações detalhadas de um aluno?**
- **Quais usuários utilizam a nossa aplicação, há algum controle sobre o acesso de rotas?**
- **A rota padrão da aplicação deve redirecionar para a página padrão do Laravel?**

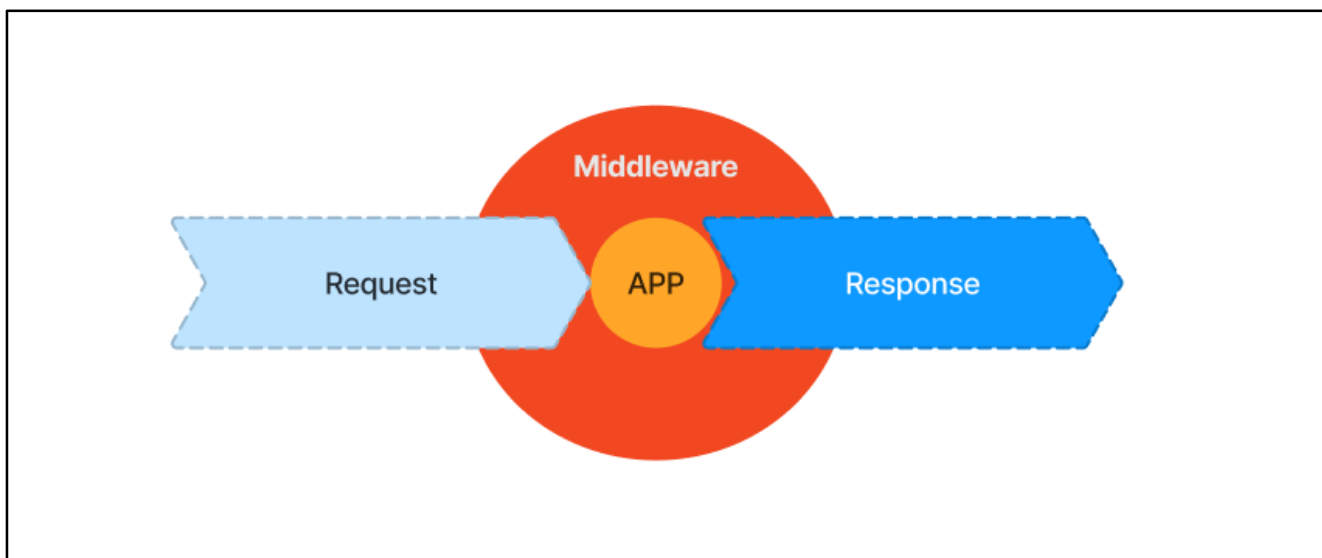
Vamos buscar soluções para os questionamentos acima, implementar, e por fim, dar uma introdução sobre o tema de publicação de projetos ou seja:

> deploy.

2. Middleware

Middleware fornece um mecanismo conveniente para **inspecionar e filtrar solicitações HTTP** que entram na sua aplicação. Por exemplo, **o Laravel inclui um middleware que verifica se o usuário da sua aplicação está autenticado**. Se o usuário não estiver autenticado, o middleware redirecionará o usuário para a tela de login da sua aplicação. No entanto, se o usuário estiver autenticado, o middleware permitirá que a solicitação prossiga para dentro da aplicação.

Exemplificação gráfica do funcionamento de um Middleware.



Como mencionado acima, um Middleware filtra requisições HTTP, neste caso podemos receber um **Request** que seja interceptado pelo Middleware, ou seja, ele pode bloquear esta solicitação dependendo de alguma lógica que possa ter falhado, como por exemplo, a falta de alguma informação importante na requisição.

Podemos utilizar este recurso então para bloquear algumas rotas da nossa aplicação, fazendo com que somente usuários cadastrados possam acessá-las.

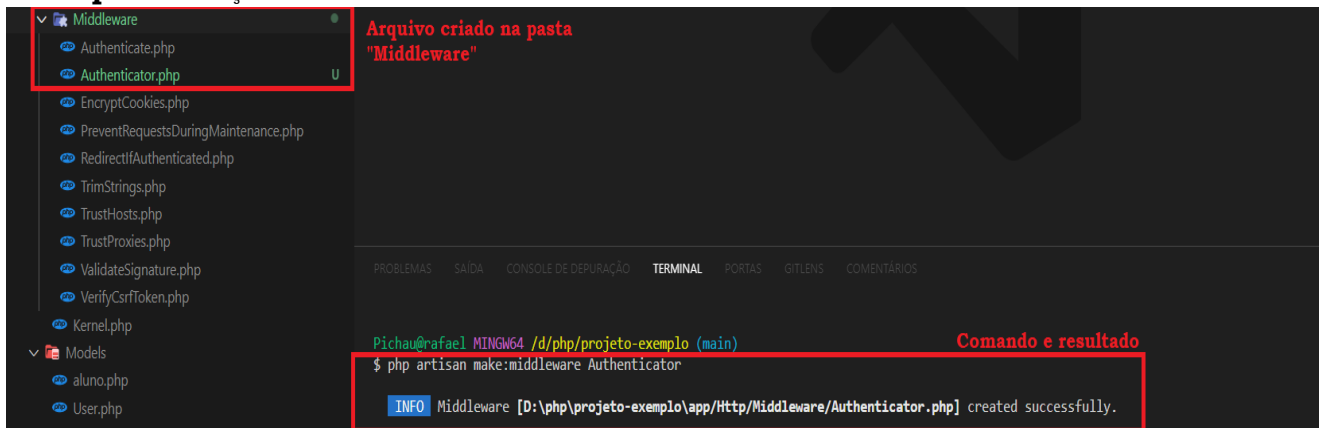
2.1 Implementando Middleware de autenticação

Para implementar nosso middleware precisamos criar um arquivo para atribuir esta lógica.

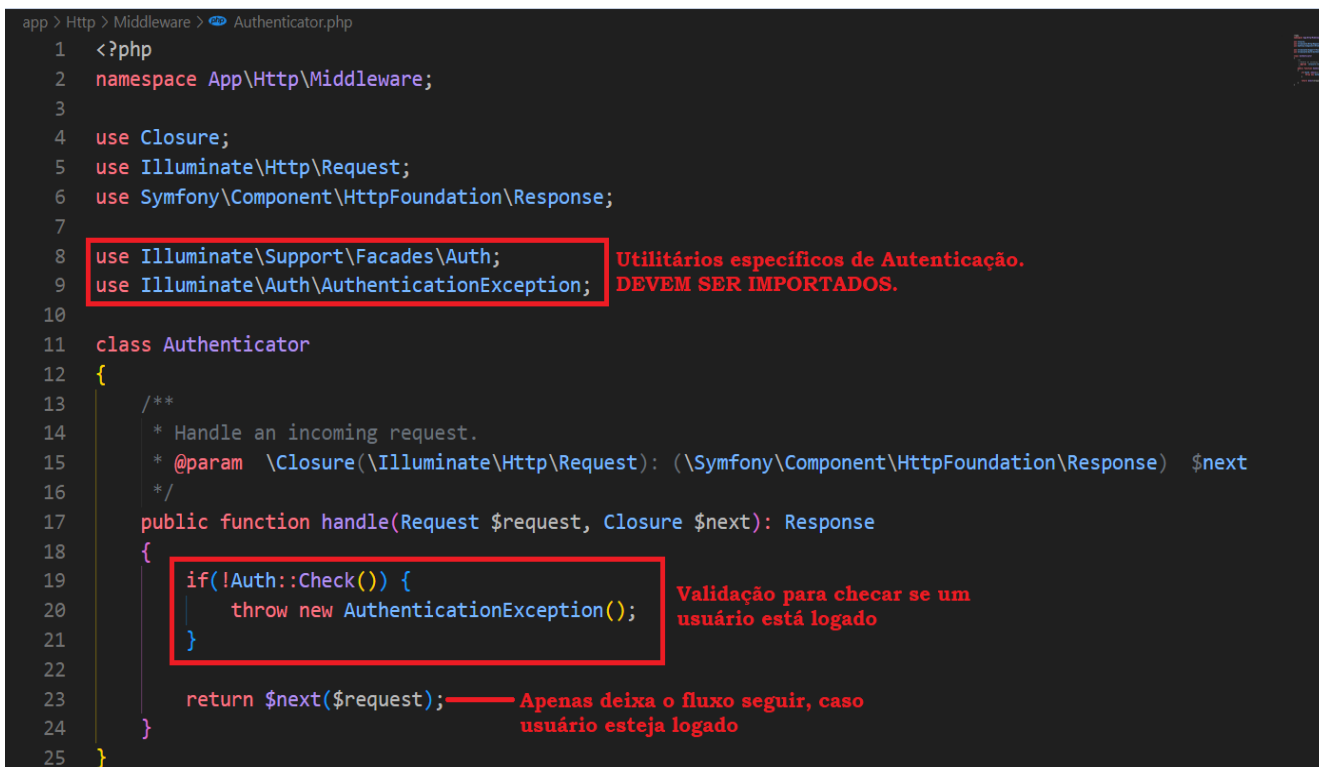
1. Crie o arquivo de middleware “Authenticator”

php artisan make:middleware Authenticator

Exemplo - Criação de middleware



2. Acesse o arquivo criado e atribua uma lógica de Autenticação que gere uma exceção caso o “Auth” não identifique um usuário logado no projeto.



Perceba que utilizamos facades do próprio Laravel para nos auxiliar nesta verificação. Caso um usuário não esteja logado, então é gerada uma exceção de autenticação.

3. Configure seu Middleware recém criado em **app > http > Kernel.php**. Navegue até **\$middlewareAliases** e adicione um **nome** para seu Middleware e o **caminho até o arquivo**.

Exemplo - Inserção de novo Middleware no arquivo Kernel.php.

```
app > Http > Kernel.php
46 ];
47
48 /**
49  * The application's middleware aliases.
50  *
51  * Aliases may be used instead of class names to conveniently assign middleware to routes and groups.
52  *
53  * @var array<string, class-string|string>
54  */
55 protected $middlewareAliases = [
56     'auth' => \App\Http\Middleware\Authenticate::class,
57     'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
58     'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
59     'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
60     'can' => \Illuminate\Auth\Middleware\Authorize::class,
61     'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
62     'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
63     'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,
64     'signed' => \App\Http\Middleware\ValidateSignature::class,
65     'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
66     'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
67     'authenticator' => \App\Http\Middleware\Authenticator::class
68 ];
69 }
```

nome que será utilizado caminho até o arquivo

Após isso, tudo está configurado no projeto para ser utilizado o Middleware criado.

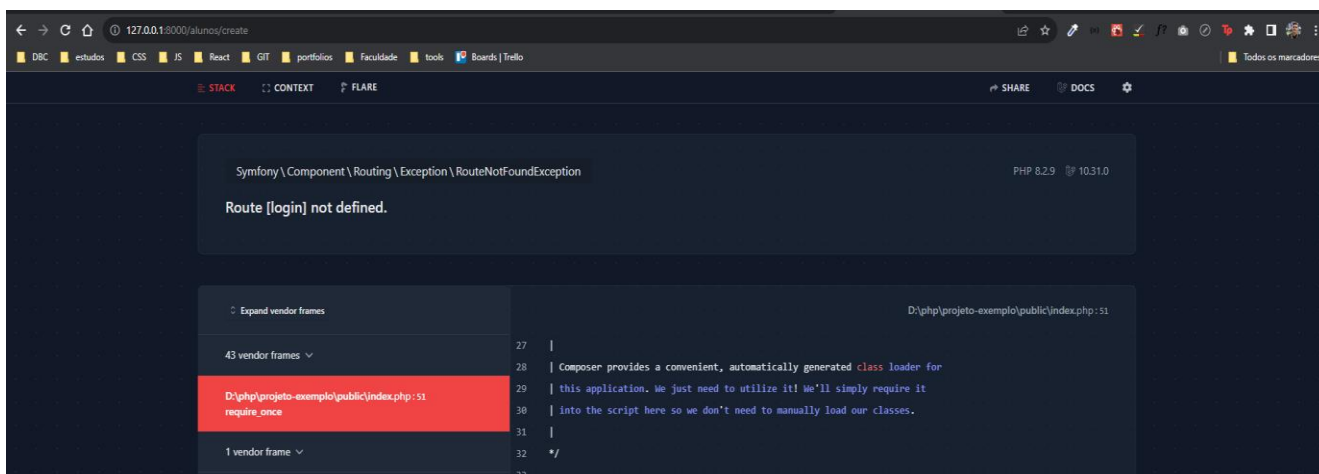
4. No arquivo de rotas web (routes > web.php), configure a rota '/alunos/create' para receber o middleware "Authenticator".

```
routes > web.php
You, há 30 segundos | 1 autor (You)
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 use App\Http\Controllers\AlunosController;
6
7 Route::get('/', function () {
8     return view('welcome');
9 });
10
11 Route::get('/alunos', [AlunosController::class, 'index']);
12 Route::get('/alunos/create', [AlunosController::class, 'create'])->middleware('authenticator');
13 Route::get('/alunos/edit/{id_aluno}', [AlunosController::class, 'edit']);
14
15 Route::post('/alunos/create', [AlunosController::class, 'insert']);
16
17 Route::delete('alunos/delete/{id_aluno}', [AlunosController::class, 'destroy']);
18
19 Route::put('alunos/update/{id_aluno}', [AlunosController::class, 'update']);
```

nome do middleware

Perceba que podemos vincular um Middleware a uma determinada rota. Neste caso, inserimos na rota 'alunos/create'.

5. Teste o funcionamento do Middleware. Com sua aplicação rodando, tente entrar na rota ‘alunos/create’, seja digitando na URL ou clicando no botão “Adicionar aluno”.



Ao acessar esta rota, deve retornar o erro **“Route [login]” not defined**. Mas por que?

Por padrão uma exceção de autenticação redireciona o usuário para uma rota de login, para que ele possa preencher os campos e se logar no sistema.

2.2 Criando páginas de login e cadastro de usuário.

Para que possamos de fato autenticar ou criar um novo usuário, vamos criar os templates HTML do Blade para isso.

Página de login

A página de login será o primeiro contato do usuário, então passará a ocupar a rota “/” da aplicação. Esta página terá um formulário que recebe os campos **email** e **senha** e um botão para validar a autenticação. Também será possível navegar para a página de **Registro** através do login para a criação de uma nova conta.

→ Passo-a-passo

1. Crie o controller **“AuthController”** (**php artisan make:controller AuthController**).
2. Defina as rotas. Por enquanto serão 4 (view login e post login, view cadastro e post cadastro).

Exemplo - Criação de rotas agrupadas e nomeadas com **group** e **name**.

```

22 Route::controller(AuthController::class)->group(function () {
23     Route::get('/login', 'login')->name('auth.login');
24     Route::post('/login', 'authenticate')->name('auth.authenticate');
25
26     Route::get('/register', 'cadastro')->name('auth.cadastro');
27     Route::post('/register', 'create')->name('auth.create');
28 });
29

```

Perceba que podemos agrupar rotas que pertencem a um mesmo Controller. Isso faz com que não seja necessário declarar em cada rota o Controller. Fica subentendido que todas estas rotas fazem parte de “AuthController”. **Não se esqueça de importar o seu Controller no topo do arquivo de rotas web. Ex: use App\Http\Controllers\AuthController;**

Rotas nomeadas

Boa prática: Perceba que demos um *name* para as nossas rotas. Isto faz com que o Laravel crie um identificador para esta rota e isto agrega, pois quando declararmos a rota na View utilizamos o seu Nome. **Isto faz com que caso tenha alguma manutenção nas URL's das rotas, não seja necessário alterar a URL em todas as views.**

3. Definição de retorno das views nas funções do AuthController.

```

3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class AuthController extends Controller
8 {
9     public function login()
10     {
11         return view('auth.login');
12     }
13
14     public function cadastro()
15     {
16         return view('auth.cadastro');
17     }
18
19     public function authenticate(Request $request)
20     {
21         dd($request->except('_token'));
22     }
23
24     public function create(Request $request)
25     {
26         dd($request->except('_token'));
27     }
28 }

```

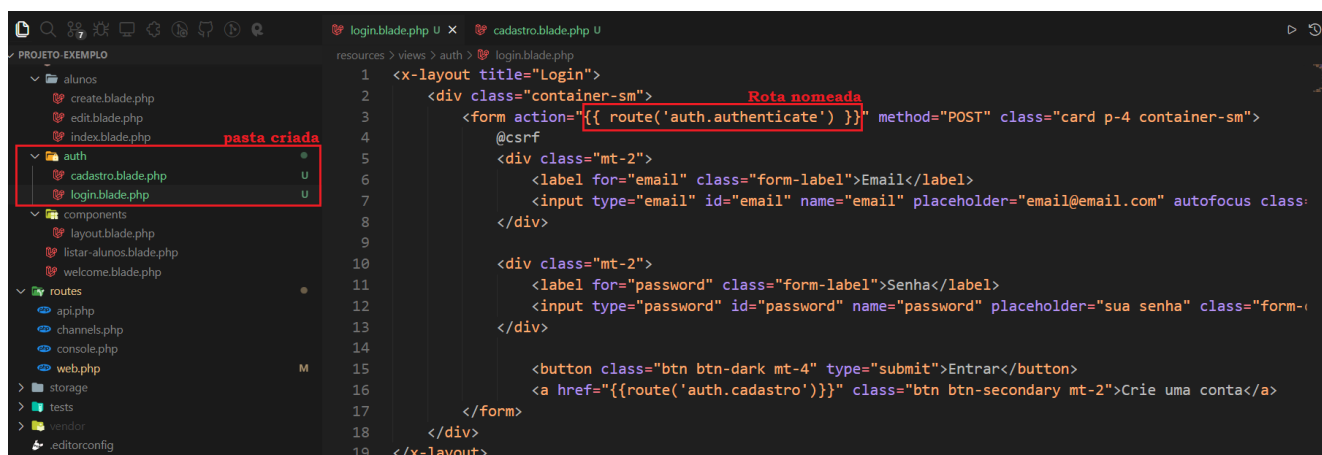
**Retorno das Views.
Rotas GET**

**Funções que lidam com envio do form
Rotas POST**

pegue todos os campos, exceto '_token'

No momento, nos controllers, vamos apenas printar os dados que vieram dos campos do formulário para validar se não há nenhum erro na construção dos forms. Perceba que pegamos todos os campos da **\$request**, a não ser, o **'token'**, isto por que este token é inserido apenas como um recurso de segurança para confirmar que a nossa rota recebeu uma requisição que veio do formulário que criamos. **Para a lógica de criação e autenticação do usuário não será necessário este token.**

4. Criação das views (login e cadastro).

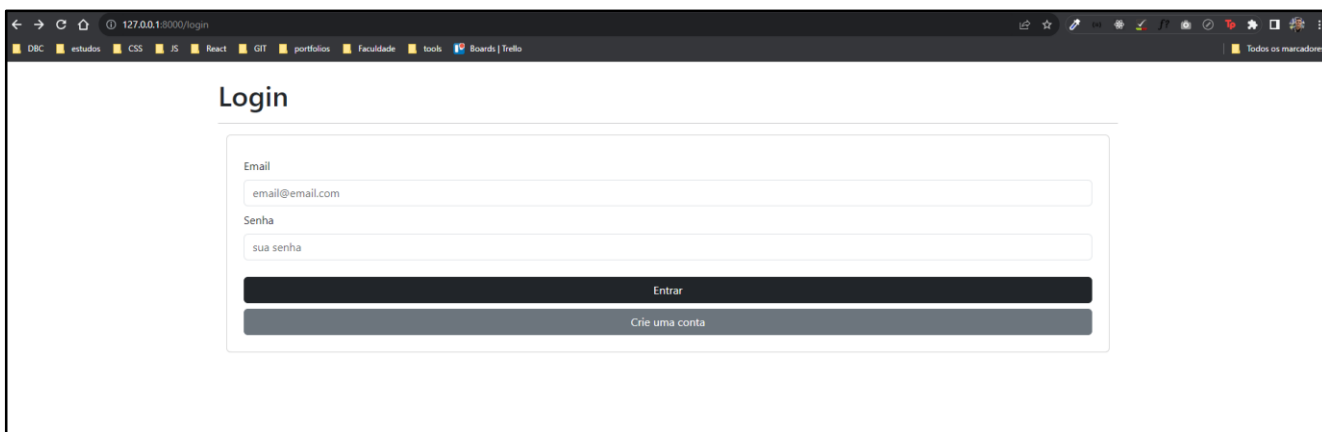


```

1 <x-layout title="Login">
2   <div class="container-sm">
3     <form action="{{ route('auth.authenticate') }}" method="POST" class="card p-4 container-sm">
4       @csrf
5       <div class="mt-2">
6         <label for="email" class="form-label">Email</label>
7         <input type="email" id="email" name="email" placeholder="email@email.com" autofocus class="
8       </div>
9
10      <div class="mt-2">
11        <label for="password" class="form-label">Senha</label>
12        <input type="password" id="password" name="password" placeholder="sua senha" class="form-
13      </div>
14
15      <button class="btn btn-dark mt-4" type="submit">Entrar</button>
16      <a href="{{ route('auth.cadastro') }}" class="btn btn-secondary mt-2">Crie uma conta</a>
17    </form>
18  </div>
19 </x-layout>
  
```

Para a view de login, utilizamos o layout, criamos o formulário de método post e com a action apontando para a **rota nomeada de autenticação**, e também, criamos os campos, labels e botões tanto para autenticar, quanto para navegar para a página de registro.

Exemplo - Resultado visual view login.



Utilizamos o bootstrap para fazer a estilização básica do formulário, campos e botões.

Uso da rota nomeada: Para utilizar a rota nomeada na view basta utilizar o comando **route** juntamente **do nome da rota** dentro de chaves ({{ }})

{{ route('rota.nome-rota') }}

Exemplo - View cadastro

```
resources > views > auth > cadastro.blade.php
1 <x-layout title="Criar conta">
2   <div class="container-sm">
3     <form action="{{ route('auth.create') }}" method="POST" class="card p-4 container-sm">
4       @csrf
5       <div class="mt-2">
6         <label for="email" class="form-label">Email</label>
7         <input type="email" id="email" name="email" placeholder="email@email.com" class="form-control" autofocus required>
8       </div>
9
10      <div class="mt-2">
11        <label for="name" class="form-label">Nome</label>
12        <input type="text" id="name" name="name" placeholder="seu nome" class="form-control" required>
13      </div>
14
15      <div class="mt-2">
16        <label for="password" class="form-label">Senha</label>
17        <input type="password" id="password" name="password" placeholder="sua senha" class="form-control" required>
18      </div>
19
20      <button class="btn btn-dark mt-4" type="submit">Criar conta</button>
21      <a href="{{ route('auth.login') }}" class="link text-center mt-2">Já possuo uma conta</a>
22    </form>
23  </div>
24 </x-layout>
```

Na **view de cadastro**, utilizamos a mesma estrutura do form, mudamos títulos e a action para apontar para a rota nomeada **auth.create**, e também, adicionamos o campo Nome.

Resultado visual:

Visualmente a view seguiu o mesmo padrão, porém modificamos o design do link da página de login para se parecer mais com um link convencional.

Ao preencher os dados e pressionar em “criar conta” o resultado esperado pelo Controller em ambas as páginas é listar exatamente o que foi digitado no formulário.

Exemplo - dd das informações dos campos na função do AuthController.

```
array:3 (3)
  "email" => "rafael@email.com"
  "name" => "rafael teste"
  "password" => "123456"
```

5. Implementação da lógica de criação de usuário no Controller.

```

app > Http > Controllers > AuthController.php
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 use Illuminate\Support\Facades\Hash;
8 use Illuminate\Support\Facades\Auth;
9 use App\Models\User;
10
11 class AuthController extends Controller
12 {
13     public function login()
14     {
15         return view('auth.login');
16     }
17
18     public function cadastro()
19     {
20         return view('auth.cadastro');
21     }
22
23     public function authenticate(Request $request)
24     {
25         dd($request->except('_token'));
26     }
27
28     public function create(Request $request)
29     {
30         $userData = $request->only(['email', 'password', 'name']);
31         $userData['password'] = Hash::make($userData['password']);
32
33         $user = User::create($userData);
34         Auth::login($user);
35         return redirect()->route('alunos.index');
36     }
37 }

```

Importações necessárias para a autenticação

Recupera campos definidos

Transforma senha em formato Hash para salvar no BD

Cria model de usuário com os dados

loga o usuário criado

Redireciona para a página de listagem de alunos

Para criar um usuário, primeiro recuperamos os valores do formulário e inserimos em **\$userData**. Logo após, acessamos o campo **password** salvo em **\$userData** para atribuir o valor da senha recebido no formulário, porém em um formato **Hash**.

Por fim, criamos um novo modelo deste usuário, realizamos o login dele passando o modelo para **Auth::login**, e em seguida, redirecionamos o usuário para a página '/alunos' através do seu nome de rota atribuído.

Uso do formato Hash para salvar senhas

Em sistemas de autenticação, quando os usuários criam uma nova conta e inserem a senha escolhida, o código do aplicativo submete essa senha a uma função chamada "hashing" (hash) e armazena o resultado no banco de dados. Quando o usuário deseja fazer login mais tarde, o processo é repetido, e o resultado é comparado com o valor armazenado no banco de dados. Se coincidir, significa que o usuário forneceu a senha correta.

Em termos mais simples, é como se a senha fosse transformada em uma espécie de código secreto antes de ser guardada, e quando você tenta fazer login novamente, o sistema verifica se o código gerado a partir da senha que você fornece agora é o mesmo que foi guardado anteriormente. Se forem iguais, o sistema deixa você entrar.

Confira mais informações em: [Hashing explained - CSO](#)

6. Teste o funcionamento. Se você preencher os campos de registro e logo após ser redirecionado para a página de listagem de alunos sem gerar nenhuma exceção, então tudo ocorreu como esperado.

7. Implementação da lógica de autenticar um usuário e redirecionar para a aplicação.

```

23 public function authenticate(Request $request)
24 {
25     $userData = $request->only(['email', 'password']); Recupera os dados do formulário
26
27     if(!Auth::attempt($userData)) { Realiza uma tentativa de login com os dados do form
28         return redirect()->back();
29     }
30     Se não passar, redireciona para a página de login novamente
31     return redirect()->route('alunos.index');
32 } Caso passar, direciona para a página de alunos

```

Para validar a autenticação utilizamos o **Auth:attempt**, que com base em credenciais fornecidas retorna true ou false caso encontre um usuário no banco de dados com a exata senha e e-mail.

8. Teste o login. Insira um e-mail e senha que você tenha cadastrado e tente login. Se tudo der certo, você deve ser redirecionado para a tela de listagem de alunos. [Saiba mais](#).

2.3 Protegendo rotas da aplicação

Agora vamos adicionar uma camada a mais de proteção na nossa aplicação, exigindo que, ações como Criação, edição e deleção só sejam permitidas por usuários que estejam autenticados. Ou seja, um usuário não logado poderá somente ver o nome dos alunos.

1. Primeiramente, vamos agrupar todas as rotas que executam ações dos alunos, ou seja, que consomem o “AlunosController”.

```

routes > web.php
You, há 60 segundos | 1 author (You)
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 use App\Http\Controllers\AlunosController;
6 use App\Http\Controllers\AuthController;
7
8 Route::get('/', function () {
9     return redirect()->route('auth.login');
10 });
11
12 Route::controller(AlunosController::class)->group(function () { Agrupamento das rotas de aluno
13     Route::get('/alunos', 'index')->name('alunos.index');
14     Route::get('/alunos/create', 'create')->name('alunos.create');
15     Route::get('/alunos/edit/{id_aluno}', 'edit')->name('alunos.edit');
16     Route::post('/alunos/create', 'insert')->name('alunos.insert');
17     Route::delete('/alunos/delete/{id_aluno}', 'destroy')->name('alunos.destroy');
18     Route::put('/alunos/update/{id_aluno}', 'update')->name('alunos.update');
19 });
20
21 Route::controller(AuthController::class)->group(function () {
22     Route::get('/login', 'login')->name('login');
23     Route::post('/login', 'authenticate')->name('auth.authenticate');
24
25     Route::get('/register', 'cadastro')->name('auth.cadastro');
26     Route::post('/register', 'create')->name('auth.create');
27 });

```

Note que já nomeamos também todas as rotas de aluno, para depois refatorarmos nas views, fazendo com que nossa aplicação fique mais dinâmica e de fácil manutenção.

2. No arquivo **AlunosController.php**, vamos adicionar uma função construtora para recuperar o middleware e definir quais rotas estarão sendo monitoradas.

```
app > Http > Controllers > AlunosController.php
You, há 4 minutos | 1 author (You)
1 <?php
2 namespace App\Http\Controllers;
3 use Illuminate\Http\Request;
4 use App\Models\Aluno;
5
6 class AlunosController extends Controller
7 {
8     public function __construct()
9     {
10         $this->middleware('authenticator')->except('index');
11     }
12 }
```

Instanciamos o middleware de autenticação para todas as rotas, EXCETO a de listagem de alunos

3. **Faça o teste! Você deve entrar em uma aba anônima do navegador** (para que o Laravel não identifique nenhum login ativo) tente acessar diretamente pela URL a rota **“localhost/alunos/create”**, por exemplo.

→ O resultado esperado é que, o usuário seja redirecionado para a página de login.

2.4 Funcionalidade de logout

Para a funcionalidade de logout vamos inserir uma navbar no header para exibir ‘logout’ ou ‘faça login’ com base no tipo de usuário.

Exemplo - Alteração no **layout.blade.php** adicionando a navbar em todas páginas.

```
resources > views > components > layout.blade.php
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>{{ $title }}</title>
7 <!-- Estilos Bootstrap -->
8 <link
9 href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
10 rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
11 crossorigin="anonymous"
12 >
13
14 </head>
15 <body>
16 <nav class="navbar navbar-light bg-light d-flex justify-content-between align-items-center">
17 <div class="container">
18 <a class="navbar-brand" href="{{route('alunos.index')}}">Controle de alunos</a>
19
20 @auth
21 <form action="{{route('auth.logout')}}" method="POST">
22 @csrf
23 <button type="submit" class="btn btn-dark">Logout</button>
24 </form>
25 @endauth
26
27 @guest
28 <a href="{{route('login')}}" class="btn btn-dark">Faça login</a>
29 @endguest
30 </div>
31 </nav>
```

Variação 1

Variação 2 - Deslogado

Note que usamos a diretiva para exibir o conteúdo adequado ao tipo do usuário. Utilizamos um formulário que chama uma função via POST para deslogar o usuário.

2. Declaração da rota “logout”.

```
Route::controller(AuthController::class)->group(function () {
    Route::get('/login', 'login')->name('login');
    Route::post('/login', 'authenticate')->name('auth.authenticate');

    Route::post('/logout', 'logout')->name('auth.logout');
```

3. Lógica do AuthController para deslogar.

```
public function logout()
{
    Auth::logout(); Uso da facade auth para deslogar

    return redirect()->route('login'); Redireciona usuário para login
}
```

Exemplo - Conteúdo visual da navbar de um usuário não logado.

2.5 Permissionamento do projeto exemplo

Para exemplificar os tipos de usuários e ações que eles podem realizar vamos listar as funcionalidades do nosso projeto exemplo e a relação do que um usuário pode ou não pode fazer. Na tabela, as células que possuem um “X” significa que o usuário **tem permissão para executar essa ação**.

Funcionalidades/Ações	Usuário GUEST (Visitante)	Usuário Autenticado
Conferir Listagem de Alunos	X	X
Criar um novo aluno		X
Editar um Aluno		X
Deletar um aluno		X

Poderíamos expandir esta tabela, caso tivéssemos rotas, por exemplo, que exibem informações detalhadas de um aluno, ou até mesmo cadastram notas, enfim...

2.6 Adaptando o layout com base no tipo de usuário

Atualmente na rota de listagem de alunos um usuário visitante (não logado) consegue visualizar tanto o botão de adicionar um aluno quanto o de editar e deletar. Será que isso faz sentido? Será que não há uma forma de adaptar o layout para exibir um conteúdo diferente ou até mesmo não exibir um elemento em tela?

A resposta é SIM! Podemos utilizar as diretivas do blade **@guest** e **@auth** que já possuem a sua lógica baseada nas funcionalidades nativas do Laravel e conseguem identificar, assim como no Controller, se há um usuário logado.

1. Alterando a página de listagem de aluno para não exibir botões de ação que usuários visitantes não possuem permissão de acessar.

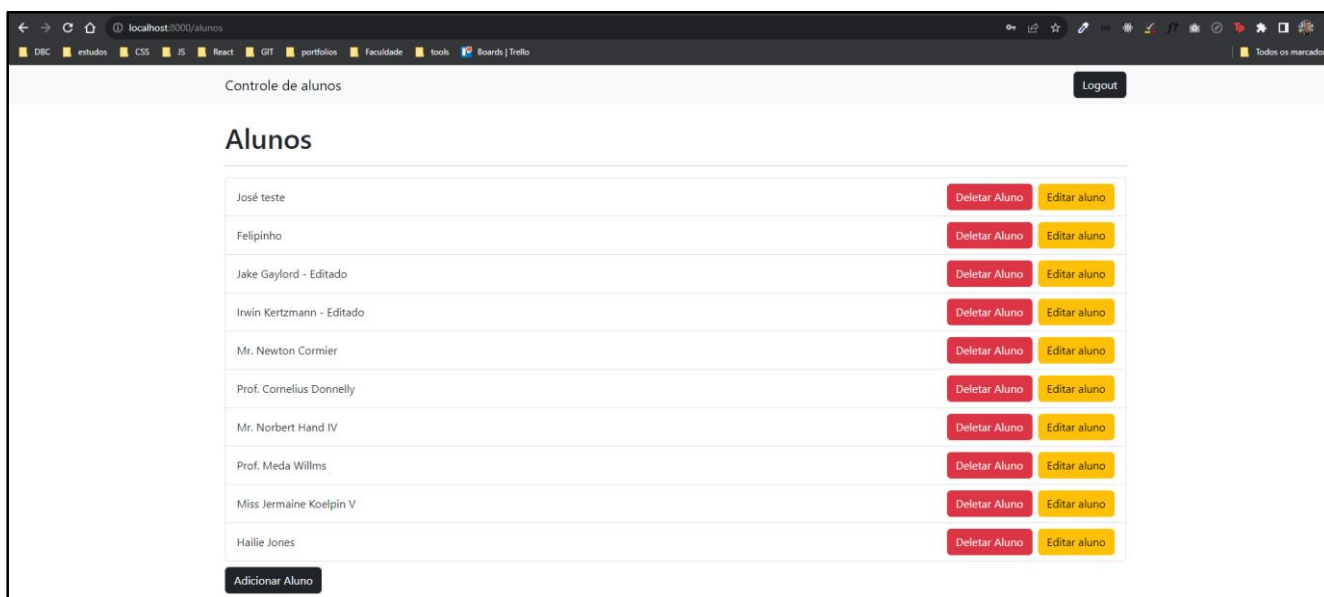
```
resources > views > alunos > index.blade.php
You, há 57 minutos | 1 author (You)
1 <x-layout title="Alunos">
2   <ul class="list-group">
3     @foreach($alunos as $aluno)
4       <li class="list-group-item d-flex justify-content-between align-items-center">
5         {{ $aluno->nm_aluno }}
6
7         @auth
8           <div class="d-flex gap-2">
9             <form action="/alunos/delete/{{ $aluno->cd_aluno }}" method="POST">
10               @csrf
11               @method('DELETE')
12               <button class="btn btn-danger" type="submit">Deletar Aluno</button>
13             </form>
14             <a class="btn btn-warning" href="/alunos/edit/{{ $aluno->cd_aluno }}">Editar aluno</a>
15           </div>
16         @endauth
17       </li>
18     @endforeach
19   </ul>
20
21   @auth
22     <a href="/alunos/create" class="btn btn-dark mt-2">Adicionar Aluno</a>
23   @endauth
24 </x-layout>
```

Esconde botão de editar e deletar para visitantes

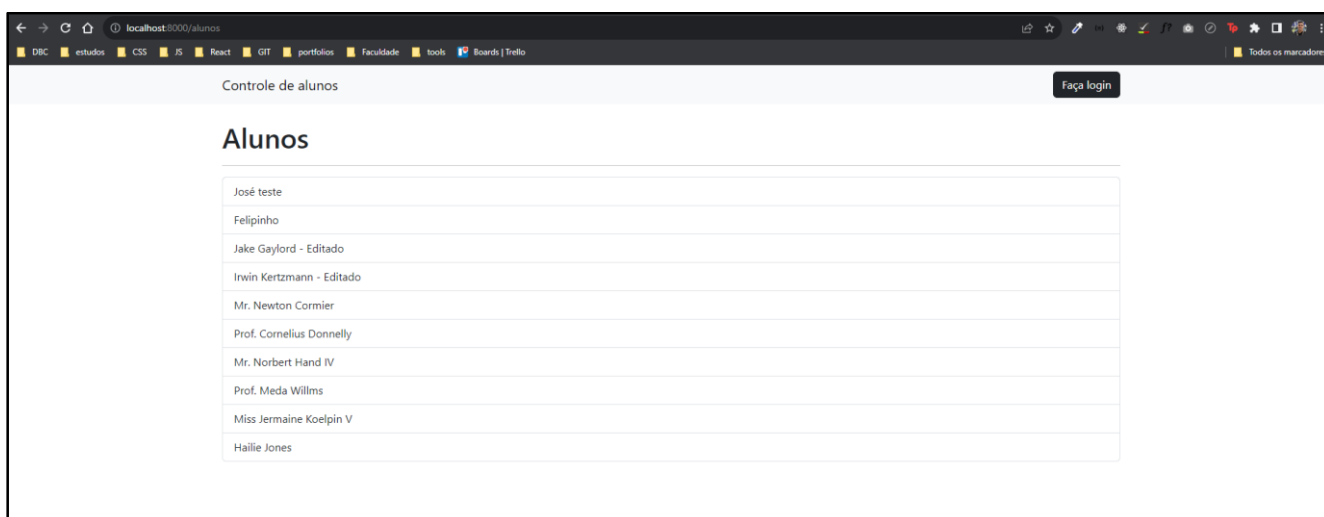
Esconde botão de criar aluno

Exemplo - Visões de usuários da página de listagem de alunos

→ **Autenticado (página alunos.index)**



→ **Visitante (página alunos.index)**



Perceba como nosso projeto está bem mais fluido e dinâmico. Entregando somente o que é permitido para o usuário e sugerindo navegações alternativas, como o login.

3. Publicação de projeto WEB

A hospedagem de sites ou Publicação de projetos WEB refere-se ao serviço de armazenar e disponibilizar um site na internet. Existem vários conceitos fundamentais associados a esse processo. Aqui estão alguns dos principais:

- **Servidor WEB:** um servidor web é um computador que armazena os arquivos do seu site (HTML, imagens, vídeos, etc.) e os disponibiliza para os visitantes da internet.

- **Provedor de Hospedagem:** empresas que oferecem serviços de hospedagem de sites são conhecidas como provedores de hospedagem. Elas possuem servidores e fornecem espaço em disco, largura de banda e outros recursos necessários para manter um site online. Exemplos de empresas que hospedam sites são: Amazon, UOL, Hostinger...
- **Domínio:** o domínio é o endereço único do seu site na internet (por exemplo, www.exemplo.com). Ele é registrado separadamente do serviço de hospedagem e aponta para o servidor onde os arquivos do seu site estão armazenados.

O IP do seu servidor é: **127.0.0.1**. Então, para que as pessoas se lembrem do endereço do seu site você dá um **nome** para esse endereço de ip cadastrando um Domínio que faz o uso do conceito de **DNS** (Domain Name System). Assim o ip aponta para um endereço com um nome mais convencional.

Ex: **127.0.0.1** → **meusite.com.br**

→ Como funcionaria a publicação para um projeto Laravel PHP?

Na documentação do PHP Laravel, são recomendadas duas ferramentas o **Forge** e **Vapor** que auxiliam no processo de publicação de um projeto PHP Laravel. [Saiba mais](#).

NOTA: Fazer o deploy de projetos complexos como estes desenvolvidos com framework e bancos de dados é uma tarefa **COMPLEXA**, que exige bastante atenção e configurações.

Normalmente, o processo de deploy de uma aplicação é realizado por desenvolvedores mais experientes, para evitar que seja publicado algo errado ou quebrado na produção e também para manusear as senhas e acessos de forma mais segura.

→ Repo da semana (app atualizado): [Repo do github](#).

4. Referências

CONSTANTIN, Lucian: **Hashing explained: Why it's your best bet to protect stored passwords**, 2021. Disponível em: <<https://www.csoonline.com/article/570229/hashing-explained-why-its-your-best-bet-to-protect-stored-passwords.html>>. Acesso em: 23 de novembro de 2023.

Múltiplos autores: **Authentication**, não informado. Disponível em: <<https://laravel.com/docs/master/authentication>>. Acesso em: 23 de novembro de 2023.

Múltiplos autores: **Middleware**, não informado. Disponível em: <<https://laravel.com/docs/10.x/middleware>>. Acesso em: 23 de novembro de 2023.

Múltiplos autores: **Easy Deployment With Forge / Vapor**, não informado. Disponível em: <<https://laravel.com/docs/10.x/deployment#deploying-with-forge-or-vapor>>. Acesso em: 23 de novembro de 2023.