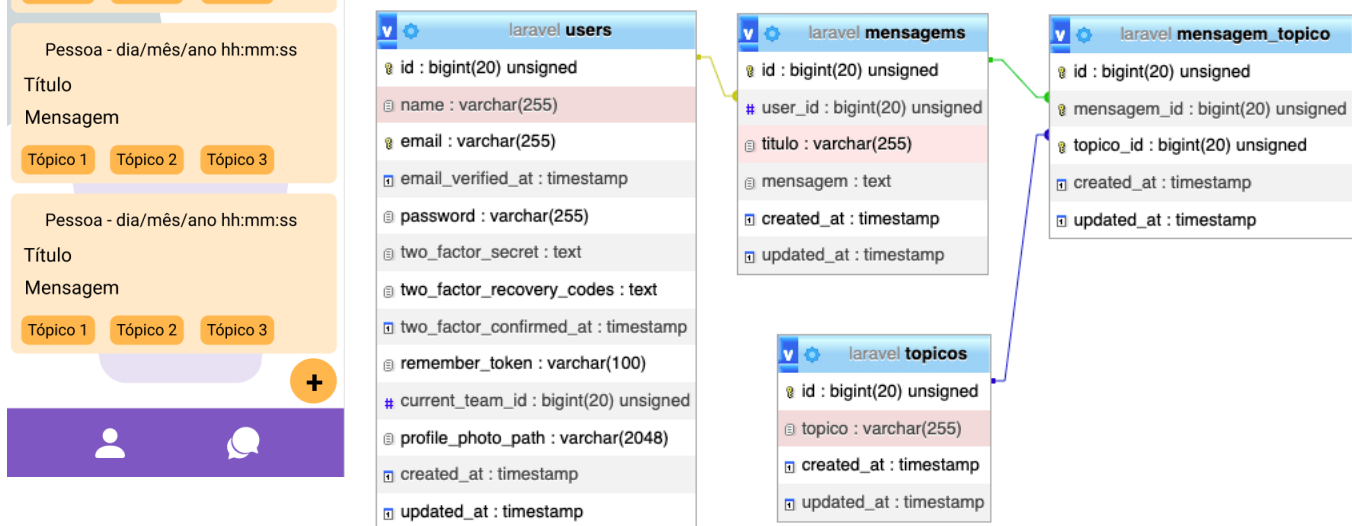
	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">CRUD com Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

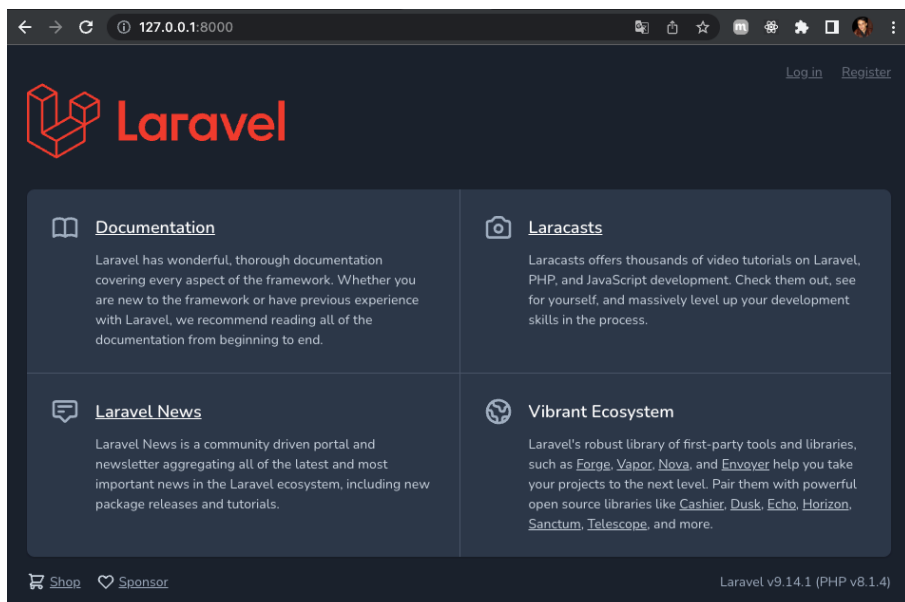
Vamos continuar nosso projeto, no qual, na área administrativa, criaremos um CRUD para envio de mensagens e seleção de tópicos. Na disciplina de mobile implementamos uma tela com título, mensagem e tópicos. Uma mensagem tem vários tópicos e um tópico pode ser selecionado em várias mensagens. Este tipo de relacionamento é chamado muito para muitos (many to many), portanto, precisaremos de uma tabela de relacionamento para interligar a tabela de mensagens com a tabela de tópicos. A entidade relacionamento dessa implementação deve ficar como a imagem abaixo:



A tabela de users nós já temos da instalação do Laravel Jetstream, portanto precisamos criar as tabelas mensagens, tópicos e mensagem_topico. Nesta sequência para possibilitar o relacionamento.

Para darmos início às implementações você deve:

- clonar o seu projeto do seu github;
- abrir o projeto na pasta clonada, abrir o terminal e executar o comando `composer install`;
- após finalizar o comando anterior, deve-se executar o comando `yarn install`;
- após finalizar o comando anterior, deve-se executar o comando `yarn dev`;
- verifique se o arquivo `.env` está presente na pasta do projeto, caso não esteja, faça uma cópia do arquivo `.env.example`, renomeie a cópia para `.env`, faça a configuração do database e gere a key, para isso, é necessário executar o comando no terminal `php artisan key:generate`; Outro comando necessário após essa configuração é solicitar que as tabelas sejam inseridas no seu banco de dados executando o comando `php`




`artisan migrate`;

- após finalizar o comando anterior é possível testar o projeto executando o comando `php artisan serve`;

Deve aparecer uma tela como a ao lado:

Feito isso podemos começar as implementações.

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">CRUD com Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Todo arquivo de implementação que não existir, sempre que possível, deve ser criado com o comando php artisan. Vamos criar os arquivos do Model, da Migration e do Controller com os métodos definidos executando os comandos abaixo, um após o outro:

```
php artisan make:model Mensagem -m -c -r
```

```
php artisan make:model Topico -m -c -r
```

```
php artisan make:model MensagemTopico -m
```

O comando cria o arquivo do Model, a opção -m cria o arquivo da Migration, a opção -c cria o arquivo do controller, a opção -r coloca os métodos do CRUD no arquivo do controller. Poderíamos fazer estas configurações de forma independente, mas optamos por fazermos de forma conjunta, pois os arquivos criados já ficarão relacionados.

Vamos começar o código pela Migration. A primeira é da tabela mensagens.

```

1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('mensagens', function (Blueprint $table) {
17             $table->id();
18             $table->foreignId('user_id')->constrained()
19                 ->onDelete('cascade')->onUpdate('cascade');
20             $table->string('titulo');
21             $table->text('mensagem');
22             $table->timestamps();
23         });
24     }
25
26     /**
27      * Reverse the migrations.
28      *
29      * @return void
30      */
31     public function down()
32     {
33         Schema::dropIfExists('mensagens');
34     }
35 };

```

Observe que parte do código já está no seu arquivo. Acrescente somente o que você não tem.

A segunda migration é a da tabela tópicos.

```
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14      public function up()
15      {
16          Schema::create('topicos', function (Blueprint $table) {
17              $table->id();
18              $table->string('topico');
19              $table->timestamps();
20          });
21      }
22
23      /**
24       * Reverse the migrations.
25       *
26       * @return void
27       */
28      public function down()
29      {
30          Schema::dropIfExists('topicos');
31      }
32  };
```

Assim como na anterior, acrescente somente o código que você não possui.

A terceira migration é a da tabela mensagem_topico.

```

1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('mensagem_topico', function (Blueprint $table) {
17             $table->id();
18             $table->foreignId('mensagem_id')->constrained()
19                 ->onUpdate('cascade')->onDelete('cascade');
20             $table->foreignId('topico_id')->constrained()
21                 ->onUpdate('cascade')->onDelete('cascade');
22             $table->unique(['mensagem_id', 'topico_id']);
23             $table->timestamps();
24         });
25     }
26
27     /**
28      * Reverse the migrations.
29      *
30      * @return void
31      */
32     public function down()
33     {
34         Schema::dropIfExists('mensagem_topico');
35     }
36 };

```

Observe que o comando colocou o nome da tabela como mensagem_topicos, altere para mensagem_topico como o da imagem para ficar mais fácil de realizar o relacionamento de n x n entre as tabelas mensagem e tópico.

Com as migrations criadas e preenchidas execute o comando abaixo para realizar a criação das tabelas no banco de dados conforme descritas nas migrations:

php artisan migrate

Os arquivos do Model, precisam de pequenas alterações para que o relacionamento entre as tabelas ocorra. Você deve alterar o Model Mensagem para que ele fique como o abaixo:

```

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Mensagem extends Model
9  {
10     use HasFactory;
11     public function user()
12     {
13         return $this->belongsTo(User::class);
14     }
15     public function topicos()
16     {
17         return $this->hasMany(Topico::class)->withTimestamps();
18     }
19 }

```

Observe que parte do código você já tem. Você precisa acrescentar os métodos user() e topicos(). Eles são responsáveis por permitir que um objeto da classe Mensagem acesse os dados de usuário e tópico respectivamente.

O Model MensagemTopico não deve ser alterado.

O Model Topico deve ter o conteúdo abaixo.

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Topico extends Model
9  {
10     use HasFactory;
11     public function mensagens()
12     {
13         return $this->hasMany(Mensagem::class);
14     }
15 }
```

Mais uma vez, observe que a maior parte do código você já tem, o que precisa ser acrescentado é o método mensagens().

O Model User também deve ser alterado. Como ele possui um conteúdo mais extenso, vou colocar somente o método que precisa ser acrescentado dentro da classe no final do arquivo:

```
52
53     /**
54      * The accessors to append to the model's array form.
55      *
56      * @var array
57      */
58     protected $appends = [
59         'profile_photo_url',
60     ];
61
62     public function mensagem()
63     {
64         return $this->hasMany(Mensagem::class);
65     }
66 }
```

O conteúdo do arquivo deve ser mantido e o método mensagem() deve ser acrescentado.


Antes de realizarmos a implementação da view, do controller e das rotas, vamos ver um recurso muito útil do Laravel que são o seeder e o factory. O Factory serve para definir um padrão para que se gere um dado aleatório. No nosso exemplo, vamos gerar os tópicos. Primeiro, deve-se executar o comando abaixo para que o arquivo do Factory seja criado:

php artisan make:factory TopicoFactory

```
1  <?php
2
3  namespace Database\Factories;
4
5  use Illuminate\Database\Eloquent\Factories\Factory;
6
7  /**
8   * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Topico>
9   */
10 class TopicoFactory extends Factory
11 {
12     /**
13      * Define the model's default state.
14      *
15      * @return array<string, mixed>
16      */
17     public function definition()
18     {
19         return [
20             'topico' => $this->faker->name()
21         ];
22     }
23 }
```

O conteúdo desse arquivo deve ser o ao lado.

Boa parte do código já foi gerado, você deve acrescentar o return que indica o campo do banco de irá receber os dados fake fabricados.

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">CRUD com Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Feito isso, deve-se gerar o arquivo do Seeder executando o comando abaixo:

```
php artisan make:seeder TopicoSeeder
```

O conteúdo do arquivo deve ser o abaixo:

```

1  <?php
2
3  namespace Database\Seeders;
4
5  use App\Models\Topico;
6  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7  use Illuminate\Database\Seeder;
8
9  class TopicoSeeder extends Seeder
10 {
11     /**
12      * Run the database seeds.
13      *
14      * @return void
15      */
16     public function run()
17     {
18         Topico::factory(10)->create();
19     }
20 }
```

O código que deve ser acrescentado é a chamada do Model que será usado, que é o Topico, chama-se o factory e passa-se o parâmetro 10 que indica que serão gerados 10 registros no banco de dados.

Antes de rodar o comando que insere os registros no banco de dados, deve-se acrescentar no arquivo DatabaseSeeder 3 linhas de código. Ele deve ficar como o abaixo:

```


1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6  use Illuminate\Database\Seeder;
7
8  class DatabaseSeeder extends Seeder
9  {
10     /**
11      * Seed the application's database.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         // \App\Models\User::factory(10)->create();
18
19         // \App\Models\User::factory()->create([
20         //     'name' => 'Test User',
21         //     'email' => 'test@example.com',
22         // ]);
23         $this->call([
24             TopicoSeeder::class
25         ]);
26     }
27 }
```

A chamada do método call permite a passagem de várias classes de seeder que poderão realizar inserções iniciais no seu banco de dados.

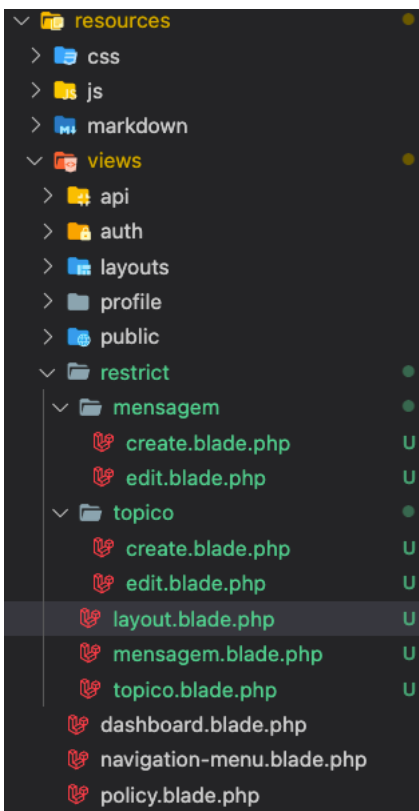
Para fazer a inserção dos registros, deve-se executar o comando abaixo no terminal na pasta do projeto:

```
php artisan db:seed
```

Feito isso, podemos criar nossas Views.

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">CRUD com Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

As views devem ser criadas na pasta resources/views.



Você deve criar uma pasta chamada restrict dentro dela teremos 3 arquivos layout.blade.php, mensagem.blade.php e tópico.blade.php.

Dentro da pasta restrict crie a pasta mensagem, dentro dela deve ter os arquivos create.blade.php e edit.blade.php.

Dentro da pasta restrict crie a pasta topico, dentro dela deve ter os arquivos create.blade.php e edit.blade.php.

Cada arquivo desse terá a função de mostrar a interface para inserir e editar a mensagem e o tópico respectivamente.

A imagem ao lado mostra como deve ficar sua pasta. Os arquivos e pasta em verde, são os que foram criados.

Vou apresentar o código de um a um abaixo.

O arquivo layout.blade.php é o layout principal. Ele é o único arquivo nosso que terá o cabeçalho html e a página completa.

Criei um CSS que deve ser colocado na pasta public/css/restrict/estilo.css.

Este arquivo possui o seguinte conteúdo:

```

1 <!DOCTYPE html>
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>Reuse</title>
9     <link rel="stylesheet" href="{{ asset('css/restrict/estilo.css') }}">
10 </head>
11
12 <body>
13     <header>
14         <picture>
15             
16         </picture>
17         <nav>
18             <ul>
19                 <li>
20                     <a href="{{ url('/mensagem') }}">Mensagens</a>
21                 </li>
22                 <li>
23                     <a href="{{ url('/dashboard') }}">Usuários</a>
24                 </li>
25             </ul>
26         </nav>
27     </header>
28     <main>
29         @yield('content')
30     </main>
31 </body>
32
33 </html>

```

```

* {
    padding: 0;
    margin: 0;
    outline: 0;
    box-sizing: border-box;
}

:root {
    --black: #000000;
    --white: #ffffff;
    --primary: #455a64;
    --primaryLight: #cfd8dc;
    --secondary: #7e57c2;
    --secondaryLight: #e8e2f4;
    --third: #ffb74d;
    --thirdLight: #ffe0b2;
}

header {
    display: flex;
    border-bottom: 0.5rem solid
var(--secondary);
}

header picture img {
    height: 6rem;
}

header nav {
    width: 100%;

```




Centro Federal de Educação Tecnológica de Minas Gerais
Campus VIII – Varginha
Curso Técnico em Informática

Disciplina
Aplicações para Web II

CRUD com Laravel

Professor
Lázaro Eduardo da Silva

```
display: flex;
justify-content: flex-end;
}
header nav ul {
  list-style-type: none;
  display: flex;
  align-items: center;
}
header ul li {
  margin: 0 1rem;
}
header ul li a {
  text-decoration: none;
  color: var(--primary);
}
main {
  min-height: 80vh;
  margin: 0.5rem;
}
/* Botão */
.button {
  display: flex;
  justify-content: center;
  text-decoration: none;
  text-align: center;
  background-color: var(--primary);
  color: var(--white);
  border: 0.2rem solid var(--primary);
  border-radius: 0.5rem;
  margin: 0.5rem 0;
  padding: 0.2rem;
  width: 7rem;
  cursor: pointer;
}
/* Tabela */
main > table {
  width: 100%;
  border-collapse: collapse;
}
main > table tr td, main > table tr th {
  border: 1px solid var(--black);
  padding: 1rem;
}
/* Formulário */
form div {
  display: grid;
  grid-template-columns: 10rem auto;
  margin: 1rem 0;
}
form div div.sub {
  display: grid;
```




Centro Federal de Educação Tecnológica de Minas Gerais
Campus VIII – Varginha
Curso Técnico em Informática

Disciplina
Aplicações para Web II

CRUD com Laravel

Professor
Lázaro Eduardo da Silva

```
grid-template-columns: 2rem auto;
margin: 1rem 0;
}
form div label {
    display: flex;
    flex-direction: column;
    justify-content: center;
}
form div input, form div textarea {
    padding: 0.3rem;
    border: 0.2rem solid var(--primary);
    border-radius: 0.5rem;
    font-size: 1rem;
}
```

A diretiva `@yield('content')` no arquivo `layout.blade.php` indica o local onde o conteúdo dos demais arquivos de view devem ser acrescentados. O código abaixo é do arquivo `mensagem.blade.php`.

```
1  @extends('restrict.layout')
2
3  @section('content')
4      <div>
5          <a href="{{url('mensagem/create')}}" class="button">Adicionar</a>
6      </div>
7      <table>
8          <thead>
9              <tr>
10                 <th>Nome</th>
11                 <th>Titulo</th>
12                 <th>Mensagem</th>
13                 <th>Tópicos</th>
14                 <th>Editar</th>
15                 <th>Remover</th>
16             </tr>
17         </thead>
18         <tbody>
19             @foreach($mensagens as $mensagem)
20                 <tr>
21                     <td>{{ $mensagem->user->name }}</td>
22                     <td>{{ $mensagem->titulo }}</td>
23                     <td>{{ $mensagem->mensagem }}</td>
24                     <td>
25                         @if($mensagem->topicos)
26                             @foreach($mensagem->topicos as $topico)
27                                 <div>{{ $topico->topico }}</div>
28                             @endforeach
29                         @endif
30                     </td>
31                     <td>
32                         <a href="{{route('mensagem.edit',$mensagem->id)}}" class="button">
33                             Editar
34                         </a>
35                     </td>
36                     <td>
37                         <form method="POST" action="{{route('mensagem.destroy',$mensagem->id)}}" onsubmit="return confirm('tem certeza?');">
38                             @csrf
39                             @method('DELETE')
40                             <button type="submit" class="button">
41                                 Remover
42                             </button>
43                         </form>
44                     </td>
45                 </tr>
46             @endforeach
47         </tbody>
48     </table>
49 @endsection
```

O código abaixo é do arquivo `topico.blade.php`.

```

1  @extends('restrict.layout')
2
3  @section('content')
4  <div>
5      <a href="{{url('topico/create')}}" class="button">Adicionar</a>
6  </div>
7  <table>
8      <thead>
9          <tr>
10             <th>Tópico</th>
11             <th>Editar</th>
12             <th>Remover</th>
13          </tr>
14      </thead>
15      <tbody>
16          @foreach($topicos as $topico)
17              <tr>
18                  <td>{{ $topico->topico }}</td>
19                  <td>
20                      <a href="{{route('topico.edit',$topico->id)}}" class="button">
21                          Editar
22                      </a>
23                  </td>
24                  <td>
25                      <form method="POST" action="{{route('topico.destroy',$topico->id)}}" onsubmit="return confirm('tem certeza?');">
26                          @csrf
27                          @method('DELETE')
28                          <button type="submit" class="button">
29                              Remover
30                          </button>
31                      </form>
32                  </td>
33              </tr>
34          @endforeach
35      </tbody>
36  </table>
37  @endsection

```

O código abaixo é do arquivo `restrict/mensagem/create.blade.php`.

```

1  @extends('restrict.layout')
2
3  @section('content')
4  @if(count($errors) > 0)
5  <ul class="validator">
6      @foreach($errors->all() as $error)
7          <li>{{ $error }}</li>
8      @endforeach
9  </ul>
10 @endif
11 <form method="POST" action="{{url('mensagem')}}" enctype="multipart/form-data">
12     @csrf
13     @method('POST')
14     <div>
15         <label for="titulo">Título</label>
16         <input type="text" name="titulo" id="titulo" value="{{ old('titulo') }}" required />
17     </div>
18     <div>
19         <label for="msg">Mensagem</label>
20         <textarea name="mensagem" id="msg" required="{{ old('mensagem') }}"></textarea>
21     </div>
22     <div>
23         <label>
24             Tópicos
25             <a href="{{url('topico/create')}}" class="button">Add Tópico</a>
26         </label>
27         <div class="sub">
28             @foreach($topicos as $topico)
29                 <input type="checkbox" id="top{{ $topico->id }}" value="{{ $topico->id }}" name="topico[]" @if($topico->id==old('topico_id')) checked @endif />
30                 <label for="top{{ $topico->id }}">{{ $topico->topico }}</label>
31             @endforeach
32         </div>
33     </div>
34     <button type="submit" class="button">Salvar</button>
35 </form>
36 @endsection

```



Centro Federal de Educação Tecnológica de Minas Gerais
Campus VIII – Varginha
Curso Técnico em Informática

Disciplina
Aplicações para Web II

CRUD com Laravel

Professor
Lázaro Eduardo da Silva

O código abaixo é do arquivo `restrict/mensagem/edit.blade.php`

```
1 @extends('restrict.layout')
2
3 @section('content')
4 @if(count($errors) > 0)
5 <ul class="validator">
6     @foreach($errors->all() as $error)
7         <li>{{$error}}</li>
8     @endforeach
9 </ul>
10 @endif
11 <form method="POST" action="{{url('mensagem',$mensagem->id)}}">
12     @csrf
13     @method('PUT')
14     <div>
15         <label for="titulo">Título</label>
16         <input type="text" name="titulo" id="titulo" value="{{ $mensagem->titulo }}" required />
17     </div>
18     <div>
19         <label for="msg">Mensagem</label>
20         <textarea name="mensagem" id="msg" required>{{ $mensagem->mensagem }}</textarea>
21     </div>
22     <div>
23         <label>
24             Tópicos
25             <a href="{{url('topico/create')}}" class="button">Add Tópico</a>
26         </label>
27         <div class="sub">
28             @foreach($topicos as $topico)
29                 <input type="checkbox" id="top{{$topico->id}}" value="{{ $topico->id }}" name="topico[]" @foreach($mensagem->topicos as $msgTopico)
30                     @if($topico->id == $msgTopico->id) checked @endif
31             @endforeach
32             />
33             <label for="top{{$topico->id}}">{{$topico->topico}}</label>
34             @endforeach
35         </div>
36     </div>
37     <button type="submit" class="button">Salvar</button>
38 </form>
39 @endsection
```

O código abaixo é do arquivo `restrict/topico/create.blade.php`

```
1 @extends('restrict.layout')
2
3 @section('content')
4 @if(count($errors) > 0)
5 <ul class="validator">
6     @foreach($errors->all() as $error)
7         <li>{{$error}}</li>
8     @endforeach
9 </ul>
10 @endif
11 <form method="POST" action="{{url('topico')}}" enctype="multipart/form-data">
12     @csrf
13     @method('POST')
14     <div>
15         <label for="topico">Tópico</label>
16         <input type="text" name="topico" id="topico" value="{{ old('topico') }}" required />
17     </div>
18     <button type="submit" class="button">Salvar</button>
19 </form>
20 @endsection
```

O código abaixo é do arquivo `restrict/topico/edit.blade.php`

```
1  @extends('restrict.layout')
2
3  @section('content')
4  @if(count($errors) > 0)
5  <ul class="validator">
6      @foreach($errors->all() as $error)
7          <li>{{ $error }}</li>
8      @endforeach
9  </ul>
10 @endif
11 <form method="POST" action="{{url('topico',$topico->id)}}">
12     @csrf
13     @method('PUT')
14     <div>
15         <label for="topico">Tópico</label>
16         <input type="text" name="topico" id="topico" value="{{ $topico->topico }}" required />
17     </div>
18     <button type="submit" class="button">Salvar</button>
19 </form>
20 @endsection
```

Estes são os arquivos das views.

Vamos para os arquivos do controller. Eles já foram criados com os comandos executados anteriormente e estão localizados na pasta `app/Http/Controllers`. São eles o `MensagemController.php` e o `TopicoController.php`. O código abaixo é do `MensagemController`.

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Mensagem;
6  use App\Models\Topico;
7  use Illuminate\Http\Request;
8  use Illuminate\Support\Facades\Auth;
9
10 class MensagemController extends Controller
11 {
12     /**
13      * Display a listing of the resource.
14      *
15      * @return \Illuminate\Http\Response
16      */
17     public function index()
18     {
19         $mensagens = Mensagem::all();
20         return view("restrict/mensagem", compact('mensagens'));
21     }
22
23     /**
24      * Show the form for creating a new resource.
25      *
26      * @return \Illuminate\Http\Response
27      */
28     public function create()
29     {
30         $topicos = Topico::all();
31         return view("restrict/mensagem/create", compact('topicos'));
32     }
33 }
```

```
34  /**
35   * Store a newly created resource in storage.
36   *
37   * @param \Illuminate\Http\Request $request
38   * @return \Illuminate\Http\Response
39   */
40  public function store(Request $request)
41  {
42      $validated = $request->validate([
43          'titulo' => 'required|max:255',
44          'mensagem' => 'required|max:255',
45          'topico' => 'array|exists:App\Models\Topico,id'
46      ]);
47      if ($validated) {
48          // print_r($request->get('topico'));
49          $mensagem = new Mensagem();
50          $mensagem->user_id = Auth::user()->id;
51          $mensagem->titulo = $request->get('titulo');
52          $mensagem->mensagem = $request->get('mensagem');
53          $mensagem->save();
54          $mensagem->topicos()->attach($request->get('topico'));
55          return redirect('mensagem');
56      }
57  }
58
59  /**
60   * Display the specified resource.
61   *
62   * @param \App\Models\Mensagem $mensagem
63   * @return \Illuminate\Http\Response
64   */
65  public function show(Mensagem $mensagem)
66  {
67      //
68  }
69
```

```

70  /**
71   * Show the form for editing the specified resource.
72   *
73   * @param \App\Models\Mensagem $mensagem
74   * @return \Illuminate\Http\Response
75   */
76  public function edit(Mensagem $mensagem)
77  {
78      $topicos = Topico::all();
79      return view("restrict/mensagem/edit", compact('topicos', 'mensagem'));
80  }
81
82  /**
83   * Update the specified resource in storage.
84   *
85   * @param \Illuminate\Http\Request $request
86   * @param \App\Models\Mensagem $mensagem
87   * @return \Illuminate\Http\Response
88   */
89  public function update(Request $request, Mensagem $mensagem)
90  {
91      $validated = $request->validate([
92          'titulo' => 'required|max:255',
93          'mensagem' => 'required|max:255',
94          'topico' => 'array|exists:App\Models\Topico,id'
95      ]);
96      if ($validated) {
97          $mensagem->titulo = $request->get('titulo');
98          $mensagem->mensagem = $request->get('mensagem');
99          $mensagem->save();
100          $mensagem->topicos()->sync($request->get('topico'));
101          return redirect('mensagem');
102      }
103  }
104

```

```

105  /**
106   * Remove the specified resource from storage.
107   *
108   * @param \App\Models\Mensagem $mensagem
109   * @return \Illuminate\Http\Response
110   */
111  public function destroy(Mensagem $mensagem)
112  {
113      $mensagem->delete();
114      return redirect("mensagem");
115  }
116  }

```

O arquivo TopicoController.php deve ter o seguinte conteúdo:

```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Topico;
6  use Illuminate\Http\Request;
7
8  class TopicoController extends Controller
9  {

```

```
10  /**
11   * Display a listing of the resource.
12   *
13   * @return \Illuminate\Http\Response
14   */
15  public function index()
16  {
17      $topicos = Topico::all();
18      return view("restrict/topico", compact('topicos'));
19  }
20
21  /**
22   * Show the form for creating a new resource.
23   *
24   * @return \Illuminate\Http\Response
25   */
26  public function create()
27  {
28      return view("restrict/topico/create");
29  }
30
31  /**
32   * Store a newly created resource in storage.
33   *
34   * @param \Illuminate\Http\Request $request
35   * @return \Illuminate\Http\Response
36   */
37  public function store(Request $request)
38  {
39      $validated = $request->validate([
40          'topico' => 'required|max:255',
41      ]);
42      if ($validated) {
43          $topico = new Topico();
44          $topico->topico = $request->get('topico');
45          $topico->save();
46          return redirect('topico');
47      }
48  }
```

```
49
50  /**
51   * Display the specified resource.
52   *
53   * @param \App\Models\Topico $topico
54   * @return \Illuminate\Http\Response
55   */
56  public function show(Topico $topico)
57  {
58      //
59  }
60
```



```
61  /**
62   * Show the form for editing the specified resource.
63   *
64   * @param  \App\Models\Topico $topico
65   * @return \Illuminate\Http\Response
66   */
67  public function edit(Topico $topico)
68  {
69      return view("restrict/topico/edit", compact('topico'));
70  }
71
72  /**
73   * Update the specified resource in storage.
74   *
75   * @param  \Illuminate\Http\Request $request
76   * @param  \App\Models\Topico $topico
77   * @return \Illuminate\Http\Response
78   */
79  public function update(Request $request, Topico $topico)
80  {
81      $validated = $request->validate([
82          'topico' => 'required|max:255',
83      ]);
84      if ($validated) {
85          $topico->topico = $request->get('topico');
86          $topico->save();
87          return redirect('topico');
88      }
89  }
90
91  /**
92   * Remove the specified resource from storage.
93   *
94   * @param  \App\Models\Topico $topico
95   * @return \Illuminate\Http\Response
96   */
```

```
97  public function destroy(Topico $topico)
98  {
99      $topico->delete();
100     return redirect("topico");
101 }
102 }
103
```

A última modificação que precisa ser realizada é no arquivo routes/web.php para que ao chegar uma requisição em um determinado endereço, ela seja encaminhada para o controller correspondente. O código do desse arquivo está abaixo.

```

1  <?php
2
3  use App\Http\Controllers\MensagemController;
4  use App\Http\Controllers\TopicoController;
5  use Illuminate\Support\Facades\Route;
6
7  /*
8  |-----
9  | Web Routes
10 |-----
11 |
12 | Here is where you can register web routes for your application. These
13 | routes are loaded by the RouteServiceProvider within a group which
14 | contains the "web" middleware group. Now create something great!
15 |
16 */
17
18 Route::get('/', function () {
19     return view('welcome');
20 });
21
22 Route::middleware([
23     'auth:sanctum',
24     config('jetstream.auth_session'),
25     'verified'
26 ])->group(function () {
27     Route::get('/dashboard', function () {
28         return view('dashboard');
29     }->name('dashboard'));
30
31     Route::resource("mensagem", MensagemController::class);
32     Route::resource("topico", TopicoController::class);
33 });
34

```

Esta rota resource cria as rotas do CRUD e já indica o método que deve ser acessado. Para visualizar as rotas criadas você pode executar o comando abaixo no terminal

php artisan route:list

Coloco abaixo o destaque para as rotas mensagem

```

GET|HEAD      mensagem ..... mensagem.index > MensagemController@index
POST          mensagem ..... mensagem.store > MensagemController@store
GET|HEAD      mensagem/create ..... mensagem.create > MensagemController@create
GET|HEAD      mensagem/{mensagem} ..... mensagem.show > MensagemController@show
PUT|PATCH    mensagem/{mensagem} ..... mensagem.update > MensagemController@update
DELETE        mensagem/{mensagem} ..... mensagem.destroy > MensagemController@destroy
GET|HEAD      mensagem/{mensagem}/edit ..... mensagem.edit > MensagemController@edit

```

e topico

```

GET|HEAD      topico ..... topico.index > TopicoController@index
POST          topico ..... topico.store > TopicoController@store
GET|HEAD      topico/create ..... topico.create > TopicoController@create
GET|HEAD      topico/{topico} ..... topico.show > TopicoController@show
PUT|PATCH    topico/{topico} ..... topico.update > TopicoController@update
DELETE        topico/{topico} ..... topico.destroy > TopicoController@destroy
GET|HEAD      topico/{topico}/edit ..... topico.edit > TopicoController@edit


```

Coloque a aplicação para rodar executando o comando abaixo:

php artisan serve

E teste o CRUD criado. Para testá-la é necessário fazer o login no seu site e depois trocar a url para:

http://127.0.0.1:8000/mensagem

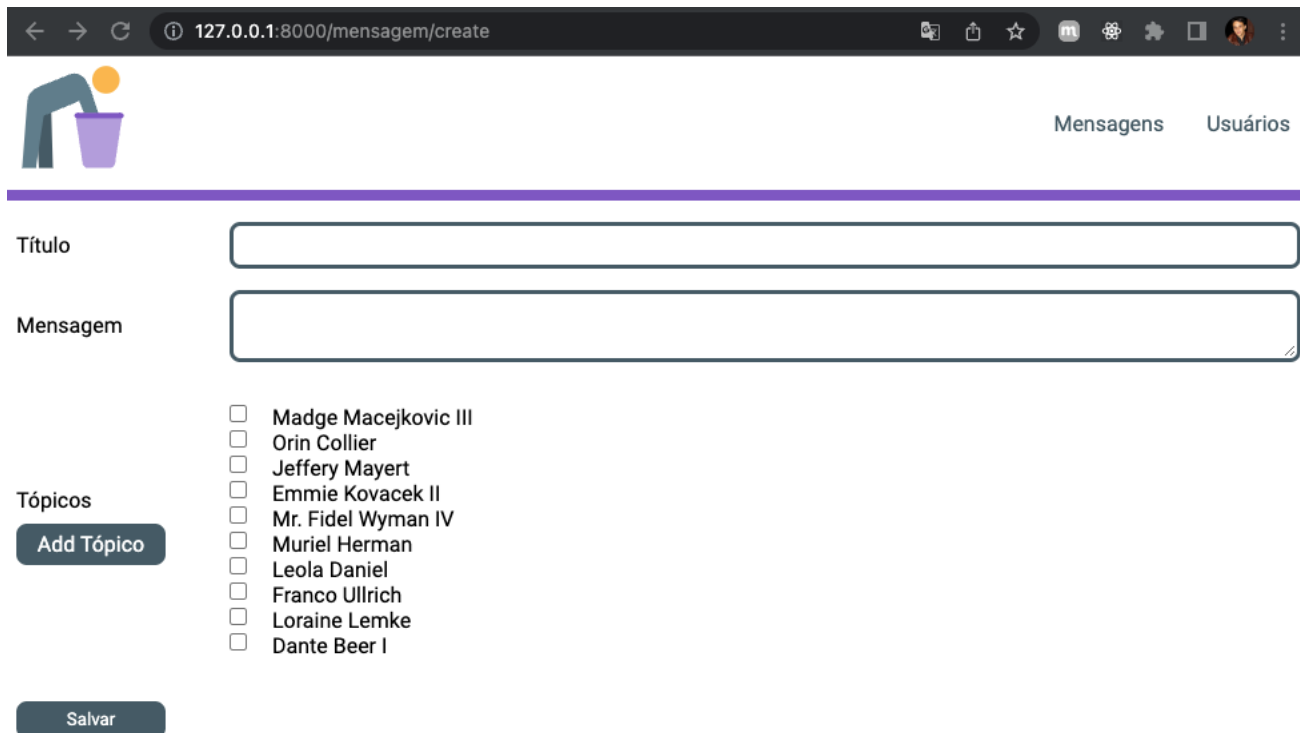
	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">CRUD com Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

A interface ficou como a imagem abaixo:



Adicionar					
Nome	Título	Mensagem	Tópicos	Editar	Remover
Lázaro	Aula de PHP	Implementação de um CRUD com Laravel	Madge Macejkovic III Emmie Kovacek II Franco Ullrich	Editar	Remover


Clicando no botão adicionar é apresentado o formulário com os campos título, mensagem e os tópicos gerados com o seeder:

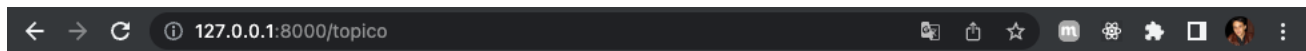


No botão Add Tópico é possível acrescentar um tópico:



Após salvar, você será redirecionado para a tela com todos os tópicos permitindo adicionar, editar e remover conforme imagem abaixo:

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">CRUD com Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>



Mensagens Usuários

Adicionar

Tópico	Editar	Remover
Madge Macejkovic III	Editar	Remover
Orin Collier	Editar	Remover
Jeffery Mayert	Editar	Remover
Emmie Kovacek II	Editar	Remover
Mr. Fidel Wyman IV	Editar	Remover
Muriel Herman	Editar	Remover
Leola Daniel	Editar	Remover
Franco Ullrich	Editar	Remover

Esse foi o CRUD implementado, vamos estudar este código para entender os recursos interessantes que foram utilizados.