

# 1 Projeto Laravel

## 1.1 Criando o Projeto

De forma a organizar meus projetos, tenho no diretório \home do meu usuário, uma pasta: **ProjetosLaravel**. Após acessar essa pasta pelo terminal, executei o seguinte comando:

```
composer create-project - --prefer-dist laravel/laravel ProjetoExemplo
```

Esta etapa pode demorar um pouco, e ao concluí-la teremos o prompt de comando assim:

```
- Installing phar-io/version (3.2.1): Extracting archive
- Installing phar-io/manifest (2.0.3): Extracting archive
- Installing myclabs/deep-copy (1.11.0): Extracting archive
- Installing phpunit/phpunit (9.5.20): Extracting archive
74 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Package swiftmailer/swiftmailer is abandoned, you should avoid using it. Use symfony/mailer instead.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: laravel/ignition
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/sail
Discovered Package: laravel/sanctum
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
77 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
No publishable resources for tag [laravel-assets].
Publishing complete.
> @php artisan key:generate --ansi
Application key set successfully.
marcelo@Eddie:~/ProjetosLaravel$
```

Figura 1: Criação do projeto concluída

Criando desta forma o projeto **ProjetoExemplo**, com toda a estrutura do framework

## 1.2 Instalando NPM

O npm é o Gerenciador de Pacotes do Node (Node Package Manager) que vem junto com ele e que é muito útil no desenvolvimento Node.

O npm é uma ferramenta de linha de comando que ajuda a interagir com plataformas online, como navegadores e servidores. Essa utilidade auxilia na instalação e desinstalação de pacotes, gerenciamento da versões e gerenciamento de dependências necessárias para executar um projeto.

Acessando o terminal **na pasta criada** para o projeto, execute:

- npm install

```
marcelo@Eddie:~/ProjetosLaravel/ProjetoExemplo$ npm install
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was created with an old version of npm,
npm WARN old lockfile so supplemental metadata must be fetched from the registry.
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile

up to date, audited 744 packages in 7s

73 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
marcelo@Eddie:~/ProjetosLaravel/ProjetoExemplo$
```

Figura 2: Conclusão da instalação do NPM

### 1.3 Criando a base de dados MySQL

Através do terminal de comando, podemos executar:

- \$ mysql -u root -p
- mysql> create database Exemplo;

Ou através do phpmyadmin, se preferir.

Irei utilizar o editor VsCode, para editar os arquivos do projeto:

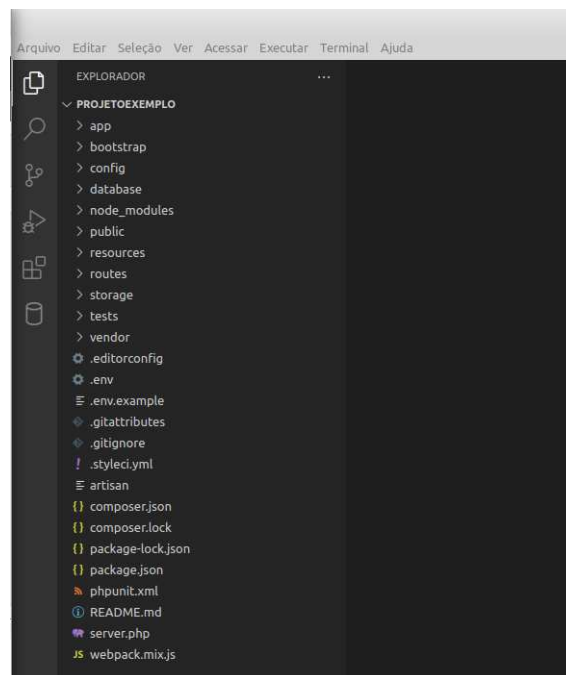


Figura 3: Projeto aberto através do VsCode

Edite o arquivo `.env` que está na raiz do seu projeto, informando: `DB_DATABASE` ; `DB_USERNAME` ; `DB_PASSWORD`

- `DB_CONNECTION=mysql`
- `DB_HOST=127.0.0.1`
- `DB_PORT=3306`
- `DB_DATABASE=Exemplo`
- `DB_USERNAME=root`
- `DB_PASSWORD=*****`

Você precisa informar o nome da base de dados criada, usuário do MySQL e senha.

## 2 Visualizando o projeto

Utilizando a opção Terminal - new terminal, do VsCode, execute o comando:

- `php artisan serve`

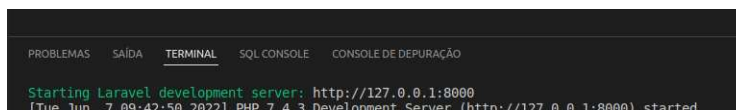


Figura 4: Terminal do VsCode

Artisan é o nome da interface da linha de comando incluída no Laravel. Esta interface fornece um bom número de comandos auxiliares para que você use durante o desenvolvimento de sua aplicação. O artisan é impulsionado pelo poderoso componente de Console do Symfony framework.

Seu servidor Laravel de desenvolvimento está rodando localmente (127.0.0.1) na porta 8000, para acessar o projeto vá até o endereço `http://127.0.0.1:8000` ou `http://localhost:8000/` em seu navegador.

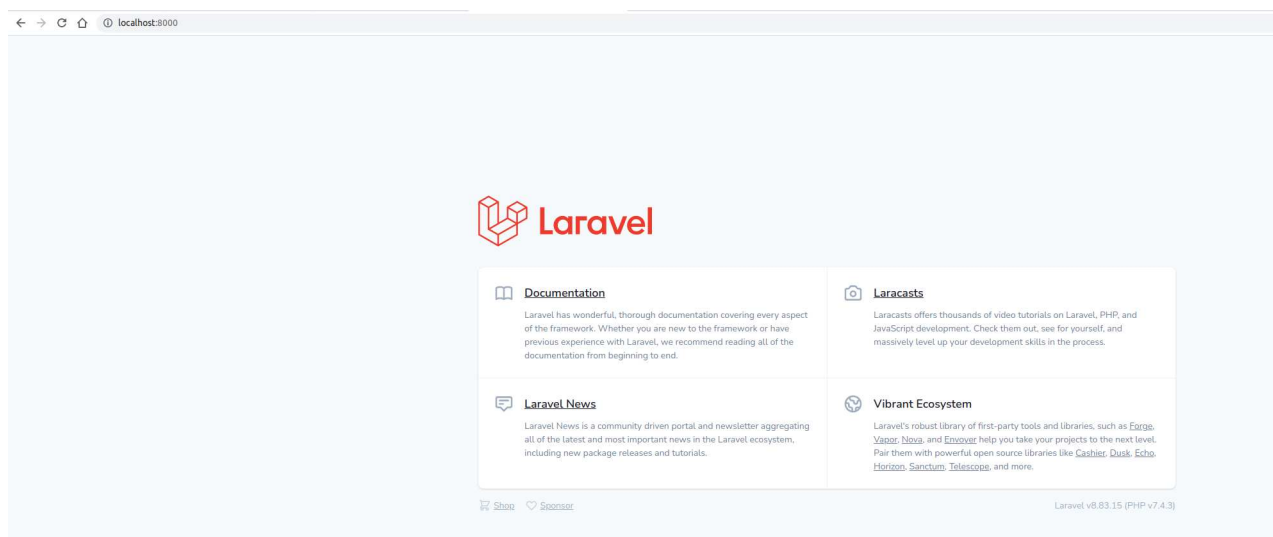


Figura 5: Tela padrão de um projeto inicial em Laravel

Para interromper a execução do servidor, no terminal do VsCode pressione: `CRTL + C`

## 3 Habilitando o Bootstrap

No terminal do VsCode, execute os seguintes comandos:

- `composer require laravel/ui - --dev`
- `php artisan ui bootstrap`
- `npm install`
- `npm run dev`
- `npm run production`
- Verificar pasta `\public\css` se contém o arquivo `bootstrap.min.css`

## 4 Habilitando o módulo de autenticação: Login

Laravel faz a implementação de autenticação de maneira muito simples. Na verdade, quase tudo está previamente configurado para você. O arquivo de configuração da autenticação está localizado no diretório `config/auth.php`, o qual contém muitas opções bem documentadas para adequar o comportamento dos serviços de autenticação.

Por Padrão, Laravel adiciona um modelo `App\User` em seu diretório `app`. Este modelo poderá ser usado com o Driver de Autenticação do Eloquent.

Para ativar o módulo de autenticação, execute os seguintes comandos no terminal do VsCode:

- `php artisan ui bootstrap - -auth`
- `npm install`
- `npm run dev`

Com isso já temos habilitado o módulo de autenticação padrão do Laravel, com opções de **Registro de usuários** e de **Login**.

A autenticação não está funcional, pois ainda não criamos as tabelas no nosso Banco de Dados. Etapa que faremos adiante.

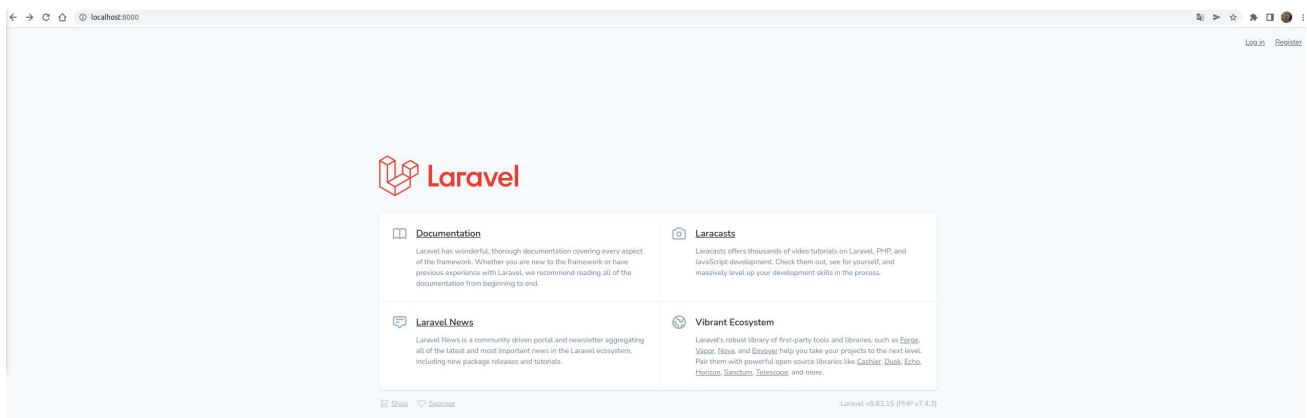


Figura 6: Autenticação no Laravel

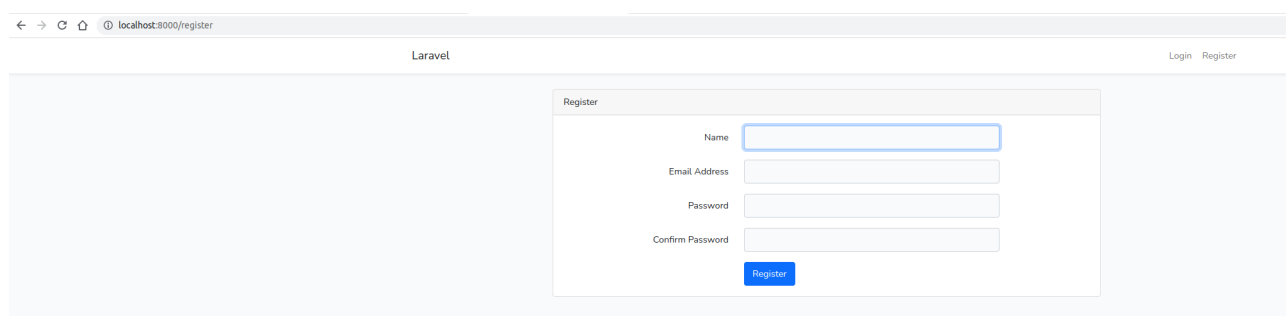


Figura 7: Tela de Registro

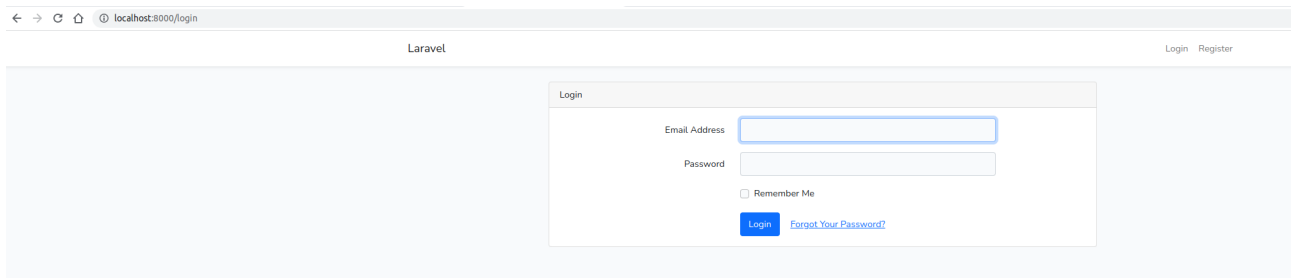


Figura 8: Tela de Login

## 5 Modelo do Banco de Dados

O Modelo do nosso Banco de Dados está definido assim:

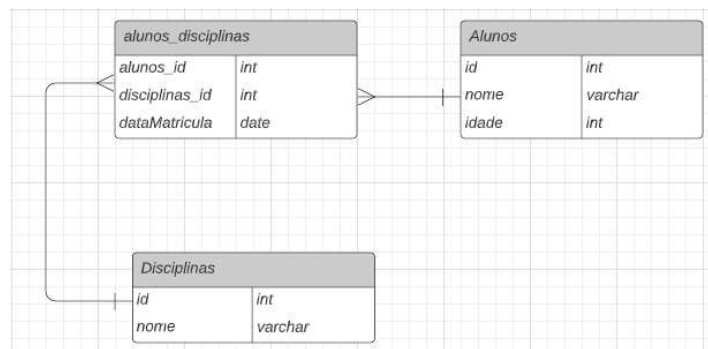


Figura 9: Modelo do Banco de Dados

Teremos então as seguintes tabelas:

- alunos
- disciplinas
- aluno\_disciplinas

Além destas o Laravel irá criar as tabelas para serem utilizadas no módulo de autenticação.

- users
- password\_resets

Para cada tabela do nosso BD, deveremos criar um **Model** específico, exceto para as tabelas criadas pelo Laravel.

### 5.1 Criação dos Models

- php artisan make:model Aluno -m
- php artisan make:model Disciplina -m
- php artisan make:model AlunoDisciplina -m

Observe que existe um padrão para criação dos nomes. Os Models devem ter o nome iniciado por letra maiuscula e estarem no singular. Assim, o Laravel criará as tabelas com nomes em minúsculas e no plural.

Após executar os comandos acima, o Laravel irá gerar os arquivos necessários para criar as tabelas no Banco de Dados.

Esse arquivos ficam localizados na pasta: database\migrations

Edite o arquivo database\migrations\[timestamp]\_create\_alunos\_table, informando os atributos que deverão ser criados na tabela do BD.

```
1 public function up()
2     {
3         Schema::create('alunos', function (Blueprint $table) {
4             $table->bigIncrements('id');
5             $table->string('nome');
6             $table->integer('idade');
7             $table->timestamps();
8         });
9     }
```

Edite o arquivo database\migrations\[timestamp]\_create\_alunos\_table, informando os atributos que deverão ser criados na tabela do BD.

```
1 public function up()
2     {
3         Schema::create('disciplinas', function (Blueprint $table) {
4             $table->bigIncrements('id');
5             $table->string('nome');
6             $table->timestamps();
7         });
8     }
```

Edite o arquivo database\migrations\[timestamp]\_create\_aluno\_disciplinas\_table, informando os atributos que deverão ser criados na tabela do BD.

```
1 public function up()
2     {
3         Schema::create('aluno_disciplinas', function (Blueprint $table) {
4             $table->bigIncrements('id');
5             $table->unsignedBigInteger('aluno_id');
6             $table->unsignedBigInteger('disciplina_id');
7             $table->date('dataMatricula');
8             $table->foreign('aluno_id')->references('id')->on('alunos');
9             $table->foreign('disciplina_id')->references('id')->on('disciplinas');
10            $table->timestamps();
11        });
12    }
```

Execute o comando no terminal:

- \$ php artisan migrate

As tabelas configuradas foram criadas em sua base de dados.

O próximo passo será alterarmos os Models acrescentando os atributos de nossas tabelas:

app/Models/**Aluno.php**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Aluno extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['nome', 'idade'];
12 }
```

app/Models/**Disciplina.php**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Disciplina extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['nome'];
12 }
```

app/Models/**AlunoDisciplina.php**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class AlunoDisciplina extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['aluno_id', 'disciplina_id', 'dataMatricula'];
12 }
```

## 6 Relacionamentos no Eloquent ORM

O Eloquent é ORM(Object Relational Mapper) embutido no Laravel, que fornece uma implementação do ActiveRecord Pattern, para interagir com um banco de dados. Cada Model do Eloquent cria um wrapper sobre a tabela no banco de dados que a ele está associado. Isso faz com que cada instância de um Model seja uma representação de uma linha da tabela associada.

Em nosso sistema o relacionamento existente entre as tabelas **alunos** e **disciplinas** é de  $N \rightarrow N$ .

Para implementar esse relacionamento, criamos a tabela auxiliar **aluno\_disciplinas**, ficando desta forma com um relacionamento  $1 \rightarrow N$  entre alunos e aluno\_disciplinas e outro relacionamento  $1 \rightarrow N$  entre disciplinas e aluno\_disciplinas.

Para que o sistema reconheça esse relacionamento, acrescentaremos em nossos **Models**:

app/Models/**Aluno.php**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Aluno extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['nome', 'idade'];
12     public function alunoDisciplina(){
13         return $this->hasMany('App\Models\AlunoDisciplina', 'aluno_id');
14     }
15 }
```

app/Models/**Disciplina.php**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Disciplina extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['nome'];
12     public function alunoDisciplina(){
13         return $this->hasMany('App\Models\AlunoDisciplina', 'disciplina_id')
14         ;
15     }
16 }
```

app/Models/**AlunoDisciplina.php**

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class AlunoDisciplina extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['aluno_id', 'disciplina_id', 'dataMatricula'];
12     public function aluno(){
13         return $this->belongsTo('App\Models\Aluno');
14     }
15     public function disciplina(){
16         return $this->belongsTo('App\Models\Disciplina');
17     }
18 }
```



Nos Models **Aluno** e **Disciplina** acrescentamos um método com o nome do modelo a qual ele se relaciona com retorno da função **hasMany**, indicando que esse é o "lado um" do relacionamento.

No model **AlunoDisciplina** acrescentamos os métodos com os nomes dos modelos aos quais ele se relaciona com retorno da função **belongsToMany**, indicando que esse é o "lado N" do relacionamento.

## 7 Controllers

O controller faz parte do design pattern MVC. A função dos Controllers é organizar e agrupar requests relacionadas, manipulando sua lógica em uma única classe.

Controladores podem agrupar solicitações HTTP relacionadas a manipulação lógica de uma classe. Controladores ficam localizados no diretório `app/Http/Controllers`.

### 7.1 Criando nossos Controllers

Primeiramente temos que criar os Controllers que iremos trabalhar.

Voltando ao nosso exemplo, vamos criar o Controller para **Aluno**.

- `php artisan make:controller ControladorAluno - --resource`

este comando irá criar o arquivo **ControladorAluno.php** no diretório: `app/Http/Controllers`.

O parâmetro - **--resource** irá criar a estrutura padrão do nosso controlador, com seus métodos principais.

- `public function index()`
- `public function create()`
- `public function store(Request $request)`
- `public function show($id)`
- `public function edit($id)`
- `public function update(Request $request, $id)`
- `public function destroy($id)`

### 7.2 Implementando as funcionalidades do Controller

Primeiro devemos indicar o caminho para o Model a ser utilizado por esse controlador e o caminho para o módulo de autenticação:

```
1 use Illuminate\Support\Facades\Auth;  
2 use App\Models\Aluno;
```

Para que o módulo de autenticação possa impedir que algum usuário não logado acesse diretamente nosso controlador, incluiremos o método construtor.

```
1 public function __construct(){  
2     $this->middleware('auth');  
3 }
```

Em seguida implementaremos o método **index**.

Este método será responsável por "buscar" os dados armazenados em nosso BD para exibí-los na tela de alunos.

```
1 public function index()  
2 {  
3     $alunos = Aluno::all();  
4     return view('alunos', compact('alunos'));  
5 }
```

Estamos criando um array \$alunos que está recebendo uma instância do nosso Model e executando a função **all()**.

Esta função corresponde ao comando SQL:

```
1 select * from alunos;
```

No comando seguinte estamos enviando os dados do nosso array para a view "alunos" que será criada mais adiante.

Em seguida implementaremos o método **create**.

este método será responsável por exibir a view para inclusão de novos registros no BD.

```
1 public function create()  
2 {  
3     return view('novoAluno');  
4 }
```

A implementação do método **store**.

este método será responsável pelo envio dos dados ao Model para inclusão no BD.

```
1 public function store(Request $request)  
2 {  
3     $dados = new Aluno();  
4     $dados->nome = $request->input('nomeAluno');  
5     $dados->idade = $request->input('idadeAluno');  
6     $dados->save();  
7     return redirect('/alunos');  
8 }
```

Instanciamos um objeto \$dados para o nosso Model.

Recebemos os dados que foram preenchidos no formulário(view). Onde \$dados->nome corresponde ao atributo da nossa tabela e \$request->input('nomeAluno') corresponde ao nome do input type text do formulário.

O mesmo se aplica à \$dados->idade corresponde ao atributo da nossa tabela e \$request->input('idadeAluno') corresponde ao nome do input type text do formulário.

\$dados->save(); executamos a gravação dos dados no BD, correspondendo ao comando SQL:

```
1 insert into alunos (nome, idade) values ('valores recebidos do formulário');
```

retornamos para a view alunos, exibindo os dados do BD com a nova inserção realizada.

A implementação do método **edit**.

este método será responsável por localizar no BD o registro a ser alterado e exibí-lo na view para edição.

```
1 public function edit($id)
2     {
3         $dados = Aluno::find($id);
4         if(isset($dados))
5             return view('editarAluno', compact('dados'));
6         return redirect('/alunos');
7     }
```

O método receberá como parâmetro o id do registro a ser editado.

em `$dados = Aluno::find($id);` instanciamos o objeto `$dados` com o registro localizado no BD. A função `find($id)` corresponde ao comando SQL:

```
1 select * from alunos where id = $id;
```

verificamos se o registro foi encontrado e enviamos os dados para a view `editarAluno`.

caso não seja encontrado o registro correspondente ao `$id`, retornamos para a view `alunos`.

A implementação do método **update**.

este método será responsável por atualizar o registro no BD.

```
1 public function update(Request $request, $id)
2     {
3         $dados = Aluno::find($id);
4         if(isset($dados)){
5             $dados->nome = $request->input('nomeAluno');
6             $dados->idade = $request->input('idadeAluno');
7             $dados->save();
8         }
9         return redirect('/alunos');
10    }
```

Ele recebe o id como parâmetro, realiza a busca no BD e se encontrar realiza o update.

A implementação do método **destroy**.

este método será responsável por deletar o registro do BD.

```
1 public function destroy($id)
2     {
3         $dados = Aluno::find($id);
4         if(isset($dados)){
5             $dados->delete();
6         }else{
7             return response('Aluno não encontrado', 404);
8         }
9         return redirect('/alunos');
10    }
```

Ele busca o registro no BD, se localizá-lo apaga o registro, se não localizá-lo devolve uma mensagem de erro, se conseguir deletar retorna para a view `alunos`.

## 8 Criando as Views

O Laravel utiliza um sistema de template chamado Blade, que se diferencia de outras soluções PHP por não se restringir ao uso dessa linguagem em suas páginas. Além disso, no Blade, cada view é compilada e armazenada em cache até sofrer alguma alteração, deixando assim seus templates mais leves.

Então, todas as nossas views deverão ter a extensão **.blade.php**.

Inicialmente criaremos o layout base para nosso sistema. Criaremos então a pasta **layout** em `/resources/views`

Nesta pasta criaremos a view: **app.blade.php**

```

1 <html>
2   <head>
3     <link href="{{ asset('css/app.css') }}" rel="stylesheet">
4     <link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet">
5     <!-- DatePicker Files -->
6     <link rel="stylesheet" href="{{asset('datePicker/css/bootstrap-
7       datePicker3.css')}}">
8     <link rel="stylesheet" href="{{asset('datePicker/css/bootstrap-
9       datePicker.standalone.css')}}">
10    <meta name="csrf-token" content="{{ csrf_token() }}">
11    <title>Projeto Exemplo</title>
12    <style>
13      body{
14        padding: 20px;
15      }
16      .navbar{
17        margin-bottom: 20px;
18      }
19    </style>
20  </head>
21  <body>
22    <div class="container">
23      @component('navbar', ["current" => $current])
24      @endcomponent
25      <main role="main">
26        @hasSection('body')
27          @yield('body')
28        @endif
29      </main>
30    </div>
31
32    <script src="{{ asset('js/app.js') }}" type="text/javascript"></
33      script>
34    <script src="{{ asset('js/jquery-3.6.0.min.js') }}" type="text/
35      javascript"></script>
36    <!-- DatePicker Files -->
37    <script src="{{asset('datePicker/js/bootstrap-datepicker.js')}}"></
38      script>
39    <!-- Language -->
40    <script src="{{asset('datePicker/locales/bootstrap-datepicker.pt-BR.
41      min.js')}}"></script>
42    @hasSection('javascript')
43      @yield('javascript')
44    @endif
45  </body>
46 </html>

```

Em seguida, criaremos nosso menu do sistema. Em: /resources/views crie o arquivo **navbar.blade.php**

```

1 <nav class="navbar navbar-expand-lg navbar-dark bg-dark rounded">
2   <button class="navbar-toggler" type="button" data-toggle="collapse"
3     data-target="#navbar" aria-controls="navbar" aria-expanded="
4       false"
5     aria-label="Toggle navigation">
6   <span class="navbar-toggler-icon"></span>
7 </button>
8 <div class="collapse navbar-collapse" id="navbar">
9   <ul class="navbar-nav mr-auto">
10    <li @if($current=="home") class="nav-item active" @else class="
11      nav-item" @endif>
12      <a class="nav-link" href="/">Início</a>
13    </li>
14    <li @if($current=="alunos") class="nav-item active" @else class=
15      "nav-item" @endif>
16      <a class="nav-link" href="/alunos">Alunos</a>
17    </li>
18    <li @if($current=="disciplinas") class="nav-item active" @else
19      class="nav-item" @endif>
20      <a class="nav-link" href="/disciplinas">Disciplinas</a>
21    </li>
22    <li @if($current=="matriculas") class="nav-item active" @else
23      class="nav-item" @endif>
24      <a class="nav-link" href="/matriculas">Matrículas</a>
25    </li>
26  </ul>
27
28  <div class="collapse navbar-collapse" id="navbarSupportedContent">
29    <!-- Left Side Of Navbar -->
30    <ul class="navbar-nav me-auto">
31
32    </ul>
33
34    <!-- Right Side Of Navbar -->
35    <ul class="navbar-nav ms-auto">
36      <!-- Authentication Links -->
37      @guest
38        @if (Route::has('login'))
39          <li class="nav-item">
40            <a class="nav-link" href="{{ route('
41              login' ) }}">{{ __( 'Acessar' ) }}</a>
42          </li>
43        @endif
44
45        @if (Route::has('register'))
46          <li class="nav-item">
47            <a class="nav-link" href="{{ route('
48              register' ) }}">{{ __( 'Registrar' )

```

```

49
50         <div class="dropdown-menu dropdown-menu-end"
51             aria-labelledby="navbarDropdown">
52             <a class="dropdown-item" href="{{ route
53                 ('logout') }}"
54                 onclick="event.preventDefault();
55                     document.getElementById(
56                         'logout-form').
57                         submit();">
58                 {{ __('Logout') }}
59             </a>
60
61             <form id="logout-form" action="{{ route
62                 ('logout') }}" method="POST" class="d
63                 -none">
64                 @csrf
65             </form>
66         </div>
67     </li>
68 @endguest
69 </ul>
70 </div>
71 </div>
72 </nav>

```

Agora criaremos a view principal do nosso sistema: em /resources/views criaremos index.blade.php

```

1 @extends('layout.app', ["current" => "home"])
2
3 @section('body')
4     <div class="jumbotron bg-light border border-secondary">
5         <div class="row">
6             <div class="card-deck">
7                 <div class="card border border-primary">
8                     <div class="card-body">
9                         <h5 class="card-title">Cadastro de Alunos</h5>
10                        <p class="card-text">
11                            Visualize os alunos cadastrados, inclua novos ou
12                            edite os dados já existentes
13                        </p>
14                        <a href="/alunos" class="btn btn-primary">Cadastro
15                            de Alunos</a>
16                    </div>
17                </div>
18                <div class="card border border-primary">
19                    <div class="card-body">
20                        <h5 class="card-title">Cadastro de Disciplinas</
21                        h5>
22                        <p class="card-text">
23                            Visualize as disciplinas cadastradas, inclua
24                            novas ou edite os dados já existentes
25                        </p>
26                        <a href="/disciplinas" class="btn btn-primary">
27                            Cadastro de Disciplinas</a>
28                    </div>
29                </div>
30                <div class="card border border-primary">
31                    <div class="card-body">
32                        <h5 class="card-title">Matrículas</h5>
33                        <p class="card-text">
34                            Visualize as matrículas cadastradas, inclua
35                            novas ou edite os dados já existentes
36                        </p>
37                    </div>
38                </div>
39            </div>
40        </div>
41    </div>

```

```

30         </p>
31         <a href="/matriculas" class="btn btn-primary">
            Cadastro de Matriculas</a>
32     </div>
33 </div>
34 </div>
35 </div>
36 </div>
37 @endsection

```

view: alunos.blade.php

```

1  @extends('layout.app', ["current" => "alunos"])
2  @section('body')
3      <div class="card border">
4          @if(session()->get('danger'))
5              <div class="alert alert-danger">
6                  {{ session()->get('danger') }}
7              </div><br />
8          @endif
9          <div class="card-body">
10             <h5 class="card-title" style="text-align: center">Cadastro de
                Alunos</h5>
11             <table class="table table-ordered table-hover" id="
                tabelaAlunos">
12                 <thead>
13                     <tr>
14                         <th>Código</th>
15                         <th>Nome do Aluno</th>
16                         <th>Idade</th>
17                         <th style="text-align:center" colspan="2">Ações
                            </th>
18                     </tr>
19                 </thead>
20                 <tbody>
21                     @foreach ($alunos as $item)
22                         <tr>
23                             <td>{{ $item->id }}</td>
24                             <td>{{ $item->nome }}</td>
25                             <td>{{ $item->idade }}</td>
26                             <td style="text-align:center">
27                                 <a href="/alunos/editar/{{ $item->id }}" class
                                    ="btn btn-primary">Editar</a>
28                             </td>
29                             <td style="text-align:center">
30                                 <a href="/alunos/apagar/{{ $item->id }}" class
                                    ="btn btn-danger">Deletar</a>
31                             </td>
32                         </tr>
33                     @endforeach
34                 </tbody>
35             </table>
36         </div>
37         <div class="card-footer">
38             <a href="/alunos/novo" class="btn btn-primary btn-sm" role="
                button">Novo Cadastro</a>
39         </div>
40     </div>
41
42 @endsection

```

view: novoAluno.blade.php

```

1 @extends('layout.app', ["current"=>"alunos"])
2 @section('body')
3 <div class="card border">
4     <div class="card-body">
5         <form action="/alunos" method="POST">
6             @csrf
7             <div class="form-group">
8                 <label for="nomeAluno">Nome do Aluno</label>
9                 <input type="text" class="form-control" name="nomeAluno"
10                     id="nomeAluno" placeholder="Informe o nome do aluno">
11             </div>
12             <div class="form-group">
13                 <label for="idadeAluno">Idade do Aluno</label>
14                 <input type="text" class="form-control" name="idadeAluno"
15                     id="idadeAluno" placeholder="Informe a idade do aluno">
16             </div>
17             <button type="submit" class="btn btn-primary btn-sm">Salvar</
18                 button>
19             <button onclick="window.location.href='/alunos';" type="button"
20                 class="btn btn-danger btn-sm">Cancelar</button>
21         </form>
22     </div>
23 </div>
24 @endsection

```

view: editarAluno.blade.php

```

1 @extends('layout.app', ["current"=>"alunos"])
2 @section('body')
3 <div class="card border">
4     <div class="card-body">
5         <form action="/alunos/{{ $dados->id }}" method="POST">
6             @csrf
7             <div class="form-group">
8                 <label for="nomeAluno">Nome do Aluno</label>
9                 <input type="text" class="form-control" name="nomeAluno"
10                     value="{{ $dados->nome }}"
11                     id="nomeAluno">
12             </div>
13             <div class="form-group">
14                 <label for="idadeAluno">Idade do Aluno</label>
15                 <input type="text" class="form-control" name="idadeAluno"
16                     value="{{ $dados->idade }}"
17                     id="idadeAluno">
18             </div>
19             <button type="submit" class="btn btn-primary btn-sm">Salvar</
20                 button>
21             <button onclick="window.location.href='/alunos';" type="button"
22                 class="btn btn-danger btn-sm">Cancelar</button>
23         </form>
24     </div>
25 </div>
26 @endsection

```



## 9 Definindo as Rotas

De uma forma simplificada, rotas podem ser entendidas como URL AMIGÁVEIS.

Imagine que temos uma aplicação que acessamos ela pelo seu endereço de domínio `www.exemplo.com.br/`

Quando acessamos `www.exemplo.com.br` estamos utilizando a rota `"/`

Quando acessamos `www.exemplo.com.br/alunos` estamos utilizando a rota `"/alunos"`

Este é o conceito de rotas, no laravel, ao receber uma requisição em uma determinada URL o sistema de rotas define o que fazer, como por exemplo redirecionar ou enviar para o controlador (Controller) decidir o que fazer.

### 9.1 Editando o arquivo `web.php`

Em `/routes` está localizado o arquivo **`web.php`** que deve ser alterado.

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('index');
7 });
8
9 Auth::routes();
10 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->
    name('home');
11
12 Route::get('/alunos', [App\Http\Controllers\ControladorAluno::class, 'index'
    ]);
13 Route::get('/alunos/novo', [App\Http\Controllers\ControladorAluno::class, '
    create']);
14 Route::post('/alunos', [App\Http\Controllers\ControladorAluno::class, 'store
    ']);
15 Route::get('/alunos/editar/{id}', [App\Http\Controllers\ControladorAluno::
    class, 'edit']);
16 Route::post('/alunos/{id}', [App\Http\Controllers\ControladorAluno::class, '
    update']);
17 Route::get('/alunos/apagar/{id}', [App\Http\Controllers\ControladorAluno::
    class, 'destroy']);
```

Para finalizar esta etapa devemos ajustar o **`HomeController.php`** indicando qual deve ser a view inicial do nosso projeto.

```
1 public function index()
2     {
3         return view('index');
4     }
```