



Programação para a WEB - servidor (server-side)

Sérgio da Silva Nogueira

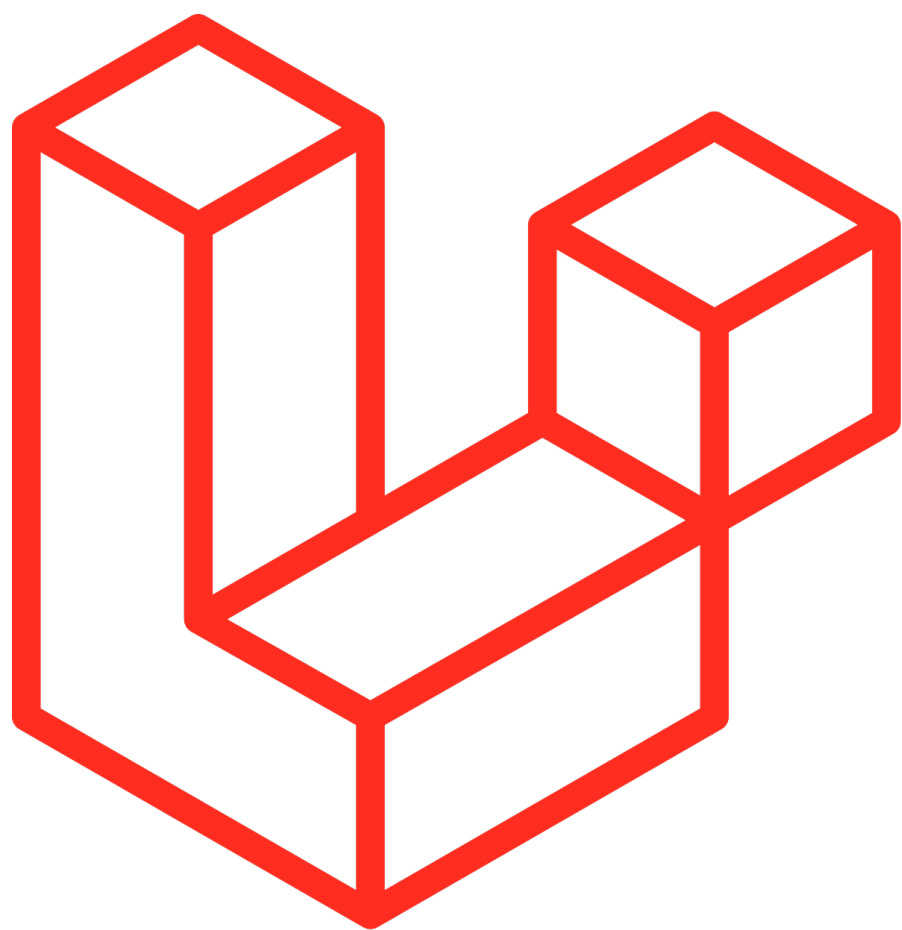
P H P

CRUD

CRUD (acrónimo do inglês Create, Read, Update and Delete) são as quatro operações básicas (criação, consulta, atualização e destruição de dados) utilizadas em bases de dados relacionais (RDBMS) fornecidas aos utilizadores do sistema.

A convenção de rotas e controllers pode ser vista na tabela seguinte:

Verb	URI	Typical Method Name	Route Name
GET	/photos	index()	photos.index
GET	/photos/create	create()	photos.create
POST	/photos	store()	photos.store
GET	/photos/{photo}	show()	photos.show
GET	/photos/{photo}/edit	edit()	photos.edit
PUT/PATCH	/photos/{photo}	update()	photos.update
DELETE	/photos/{photo}	destroy()	photos.destroy



P H P

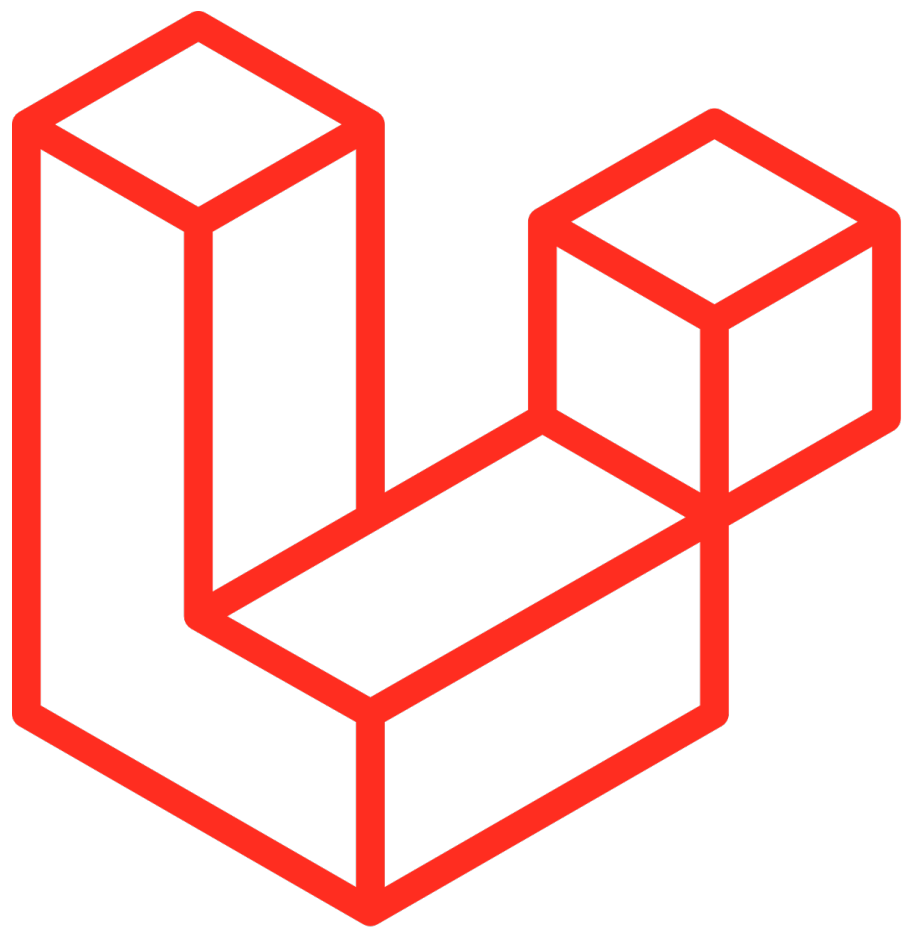
CRUD

Para o nosso CRUD vamos criar um projeto de raiz e criar um model Player:

```
php artisan make:model Player -a
```

A estrutura inicial da tabela players será a seguinte:

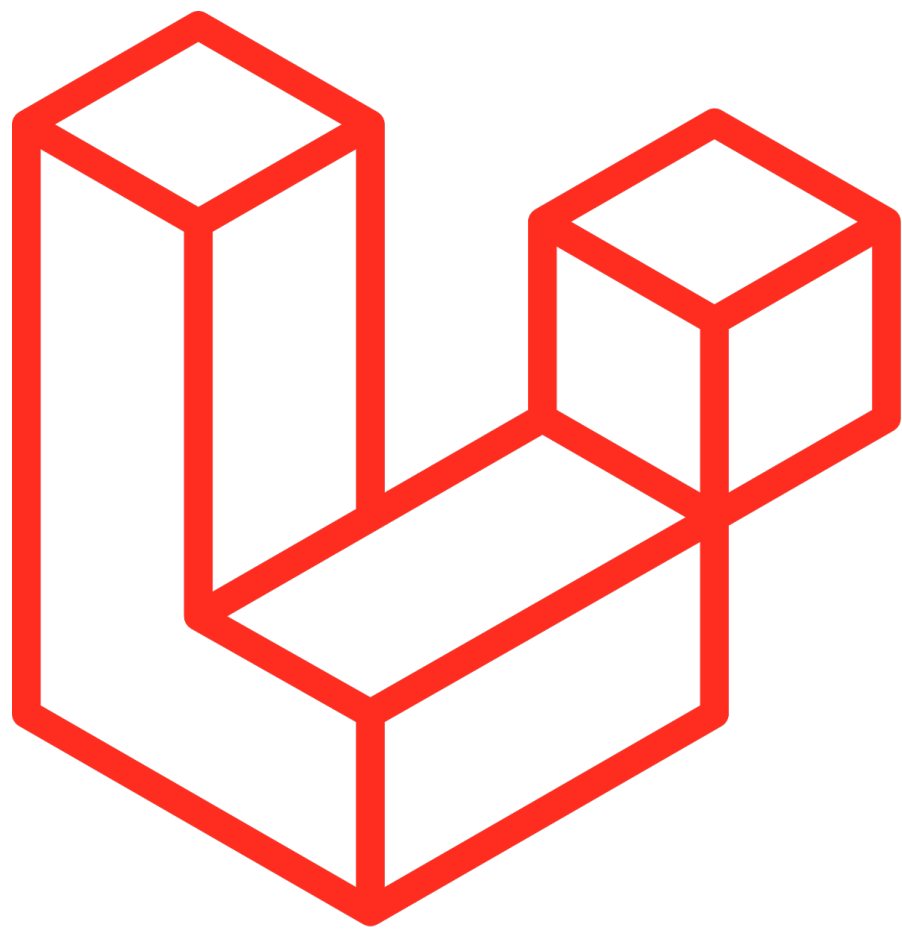
```
/**
 * Run the migrations.
 *
 * @return void
 */
public function up()
{
    Schema::create('players', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('address');
        $table->text('description');
        $table->boolean('retired');
        $table->timestamps();
    });
}
```



P H P

CRUD

No passo seguinte vamos criar uma Factory para o model Player:

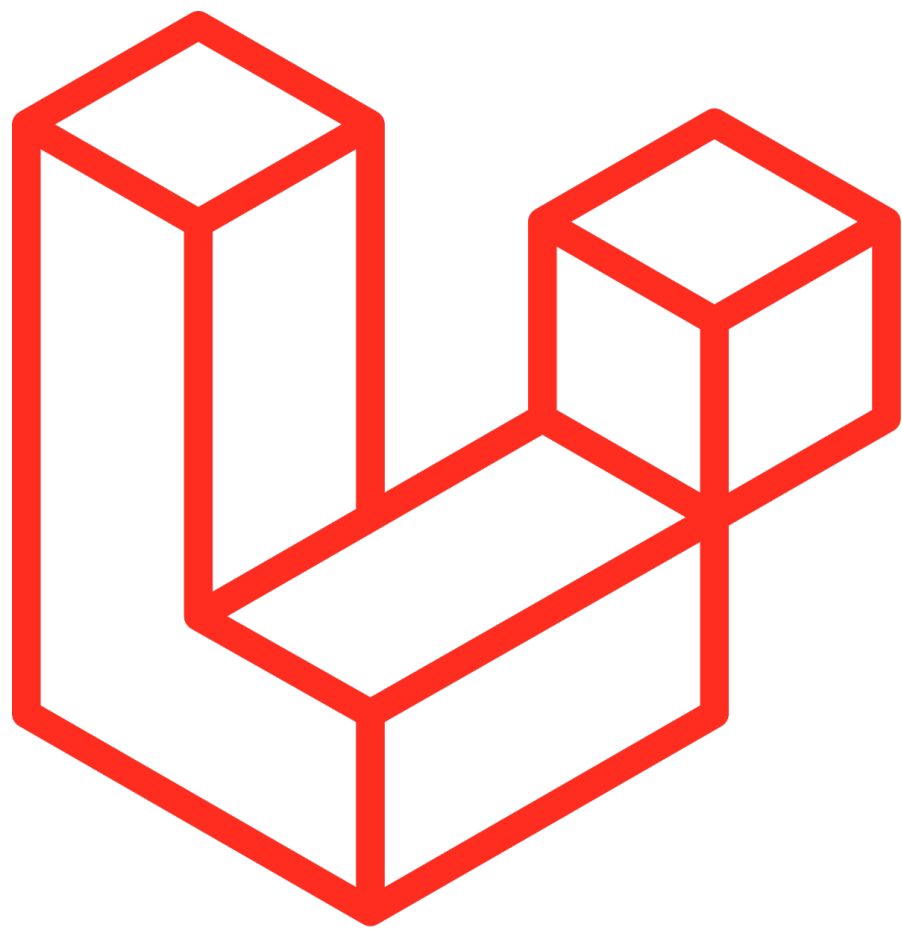


```
$factory->define(Player::class, function (Faker $faker) {  
    return [  
        'name'           => $faker->name,  
        'address'        => $faker->address,  
        'description'     => $faker->paragraph(50),  
        'retired'         => $faker->boolean,  
        'created_at'      => now(),  
        'updated_at'      => now()  
    ];  
});
```

P H P

CRUD

No passo seguinte vamos criar chamar a seed no DatabaseSeeder

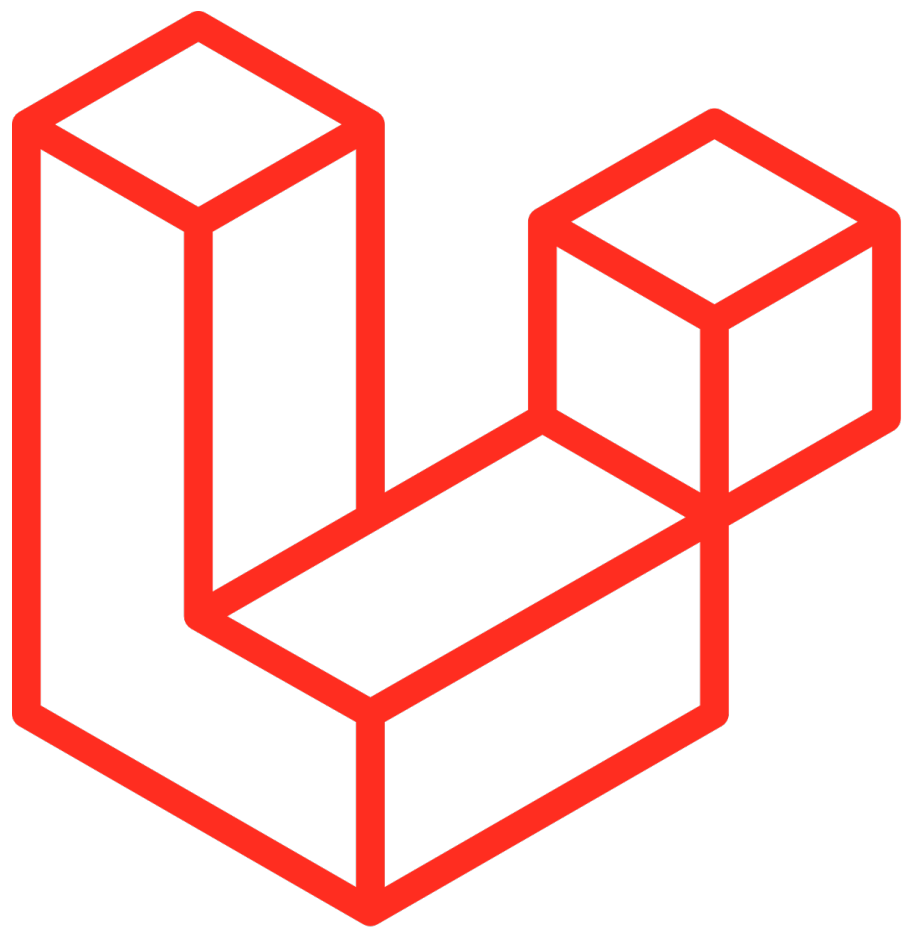


```
/**
 * Seed the application's database.
 *
 * @return void
 */
public function run()
{
    $this->call(PlayerSeeder::class);
}
```

P H P

CRUD

No passo seguinte vamos criar chamar a Factory respetiva na seed do Player:



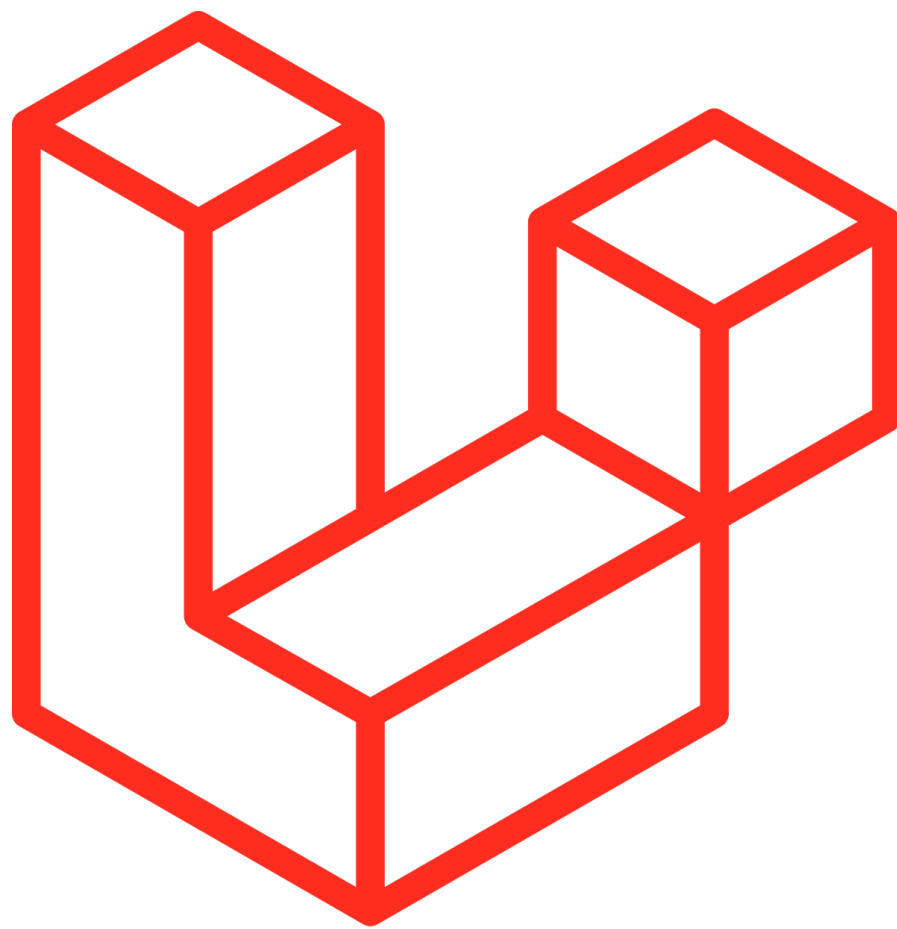
```
use Illuminate\Database\Seeder;
use App\Player;

class PlayerSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        factory(Player::class, 50)->create();
    }
}
```

P H P

CRUD

Ao criarmos o nosso model utilizando a flag -a, criamos um controller Player com os resources completos para o nosso CRUD:

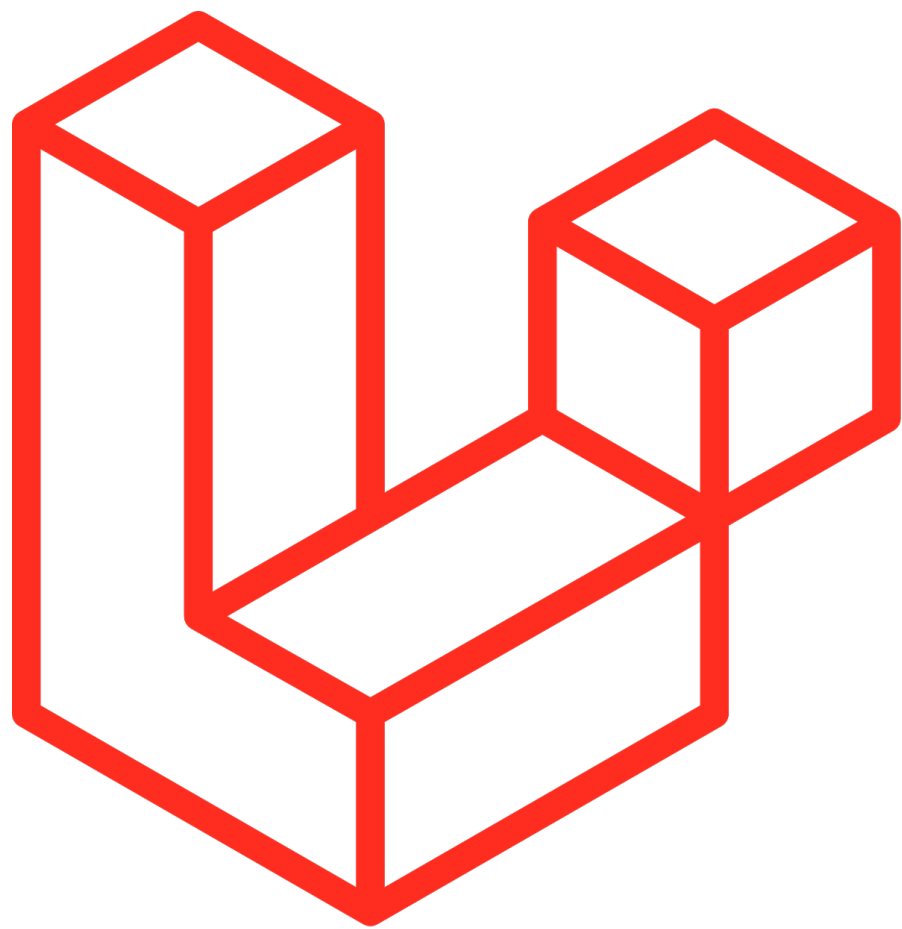


```
class PlayerController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }
}
```

P H P

CRUD



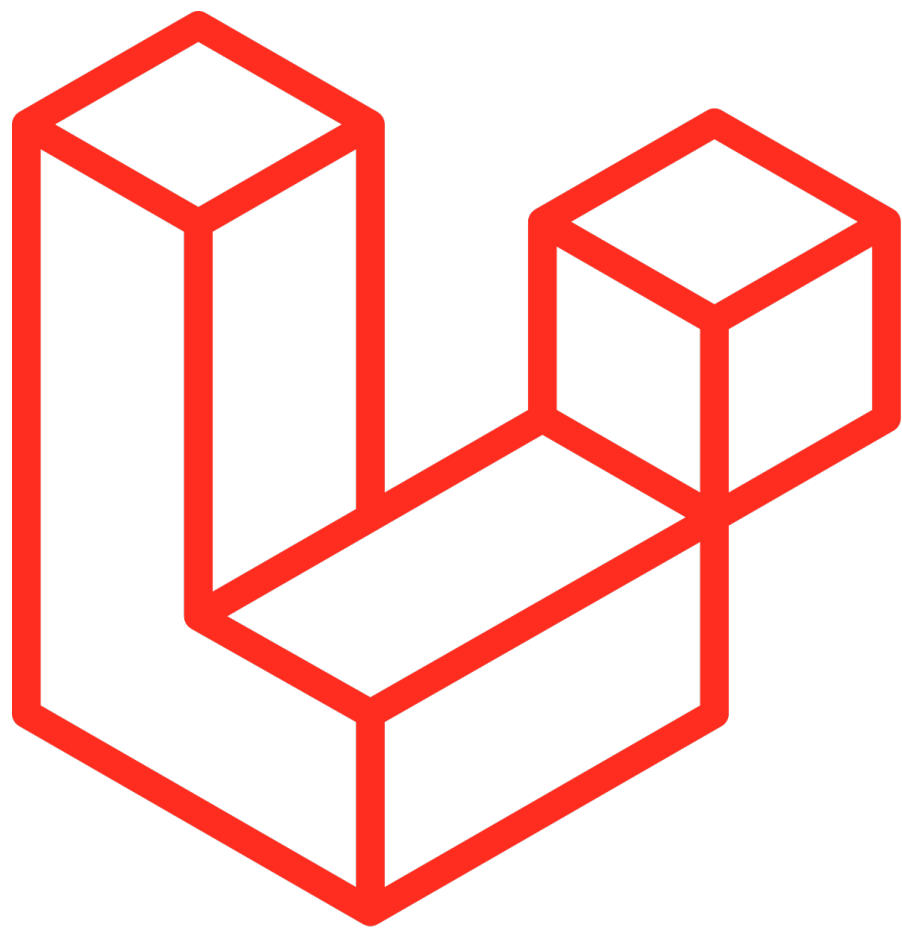
```
/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    //
}
```

```
/**
 * Display the specified resource.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function show(Player $player)
{
    //
}
```

```
/**
 * Show the form for editing the specified resource.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function edit(Player $player)
{
    //
}
```


P H P

CRUD



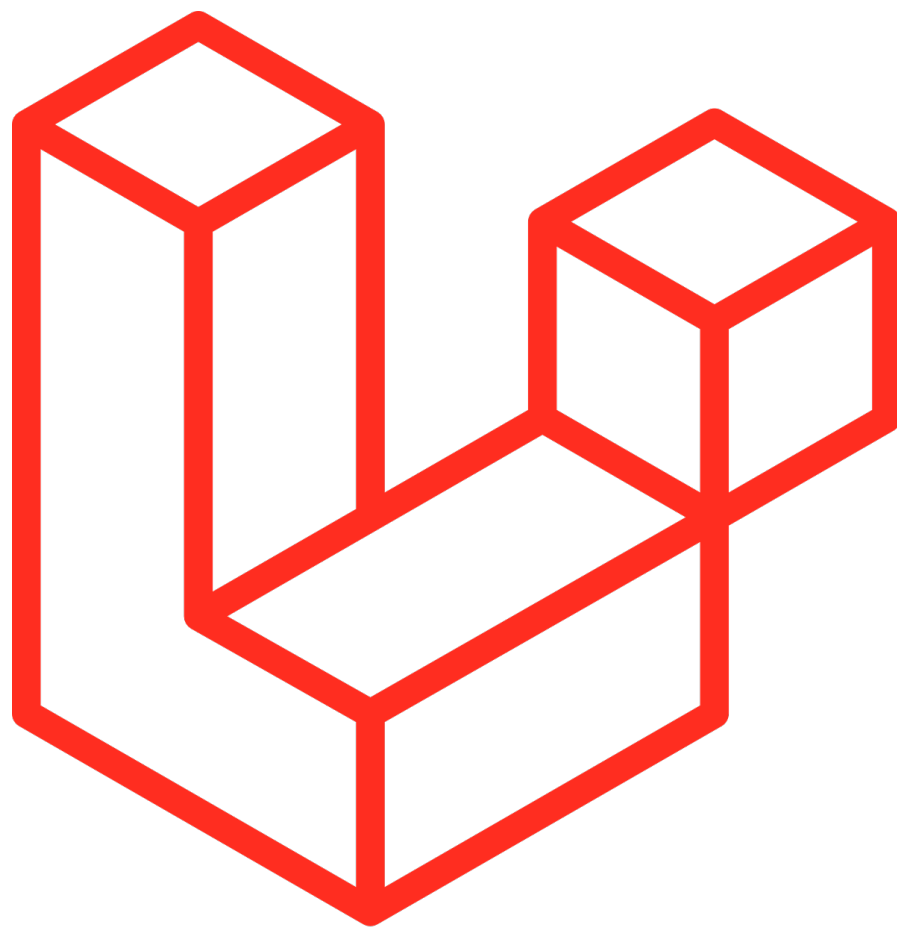
```
/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, Player $player)
{
    //
}

/**
 * Remove the specified resource from storage.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function destroy(Player $player)
{
    //
}
}
```

P H P

CRUD

Todas as funções de CRUD do nosso controller necessitam de um rota, para criarmos as rotas precisamos de fazer a seguinte definição no ficheiro web.php

*// Exemplo 1*

```
Route::resource('players', 'PlayerController');
```

// Exemplo 2

```
Route::get('/players', 'PlayerController@index');
Route::get('/players/create', 'PlayerController@create');
Route::post('/players', 'PlayerController@store');
Route::get('/players/{player}', 'PlayerController@show');
Route::get('/players/{player}/edit', 'PlayerController@edit');
Route::put('/players/{player}', 'PlayerController@update');
Route::delete('/players/{player}', 'PlayerController@destroy');
```

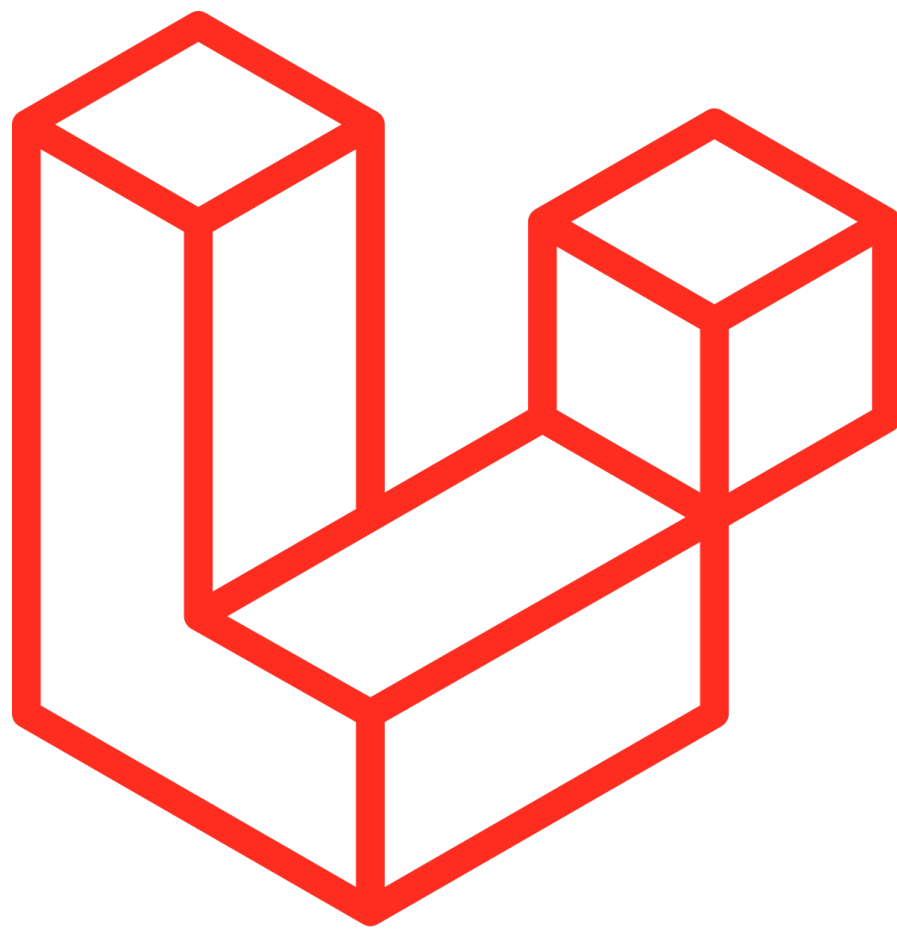
// Exemplo 3

```
Route::prefix('players')->group(function(){
    Route::get('', 'PlayerController@index');
    Route::get('create', 'PlayerController@create');
    Route::post('', 'PlayerController@store');
    Route::get('{player}', 'PlayerController@show');
    Route::get('{player}/edit', 'PlayerController@edit');
    Route::put('{player}', 'PlayerController@update');
    Route::delete('{player}', 'PlayerController@destroy');
});
```

P H P

CRUD

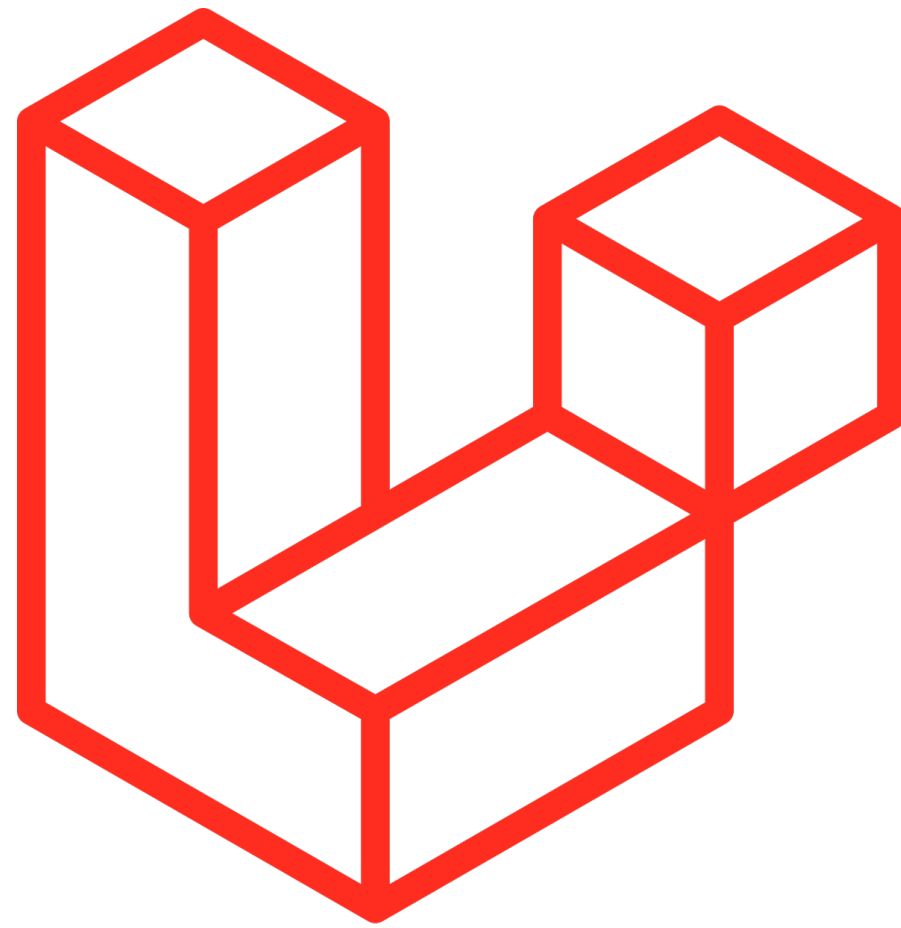
Crie as views master page e os seguintes components e pages:



- > app
- > bootstrap
- > config
- > database
- > node_modules library root
- > public
- ▼ resources
 - > js
 - > lang
 - > sass
 - ▼ views
 - > auth
 - ▼ components
 - ▼ players
 - player-form-create.blade.php
 - player-form-edit.blade.php
 - player-form-show.blade.php
 - players-list.blade.php
 - ▼ master
 - footer.blade.php
 - header.blade.php
 - main.blade.php
 - ▼ pages
 - ▼ players
 - create.blade.php
 - edit.blade.php
 - index.blade.php
 - show.blade.php

P H P

CRUD - READ

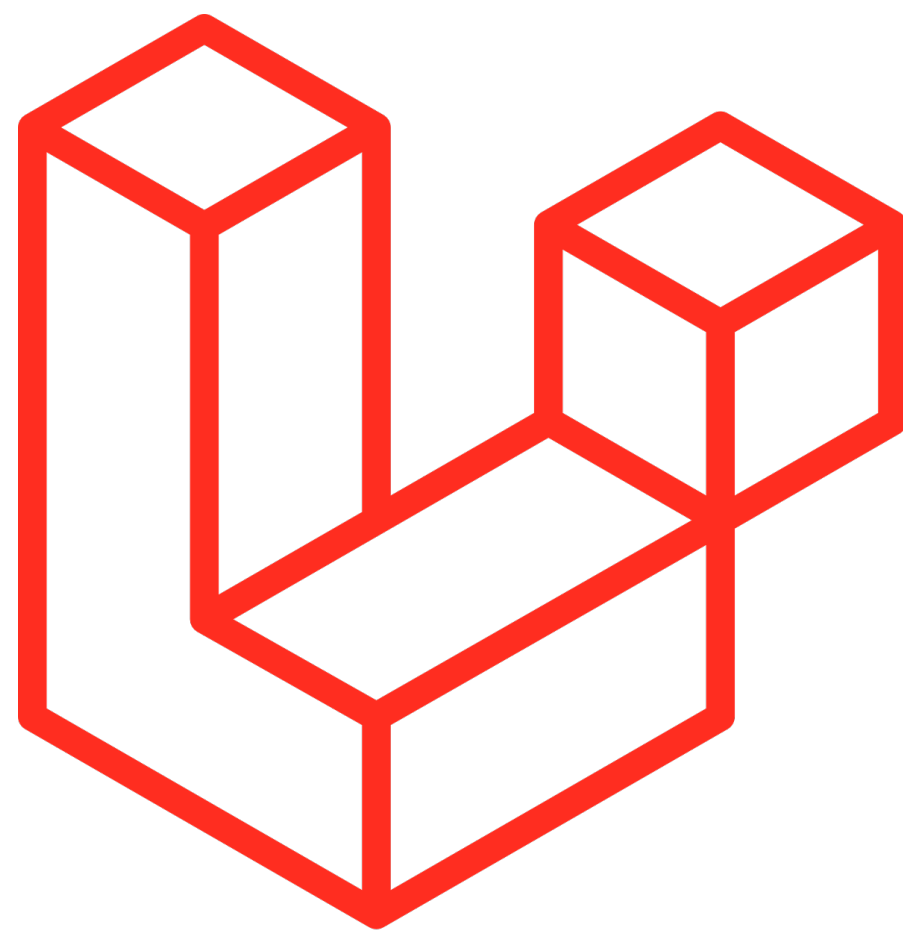


Read (Retrieve) | **SELECT**

Read (Retrieve) | Ler, recuperar ou ver entradas existentes

P H P

CRUD - READ



A primeira view a ser desenvolvida será a lista de Players, e deverá ter o seguinte layout:

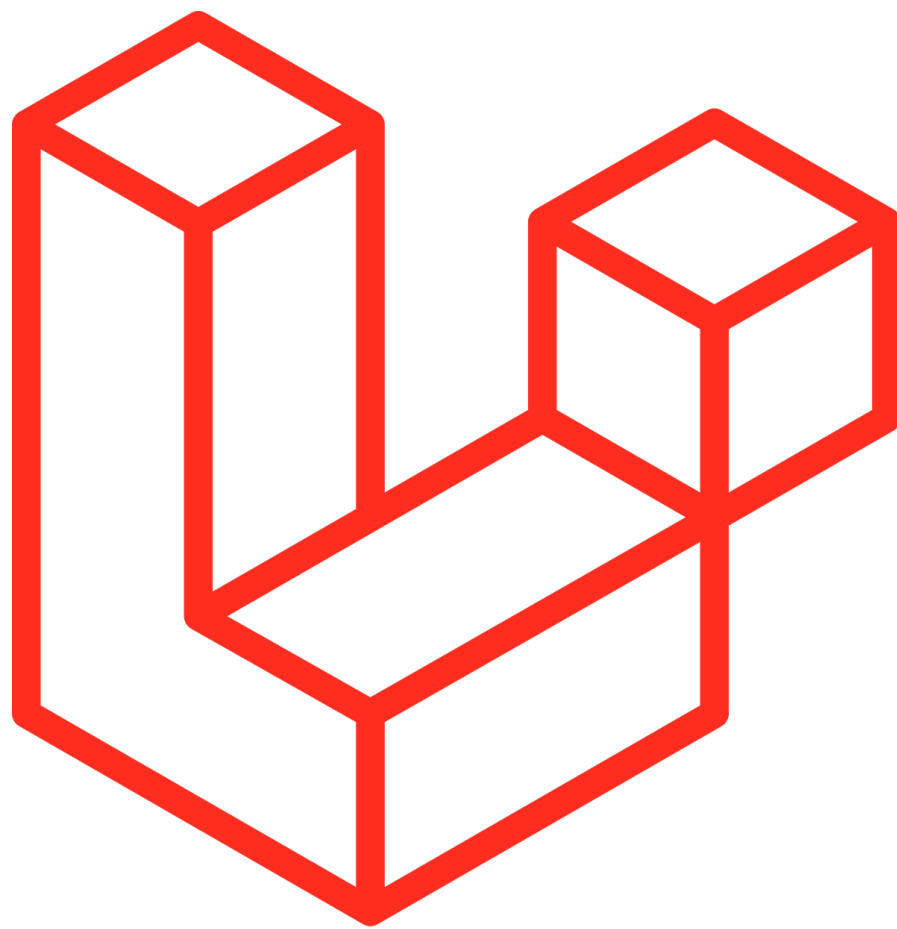
Players List

#	Name	Address	Retired	Actions
10	Prof. Maxime Stokes	603 Tremblay Valley Millsfurt, AR 37917-1156	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
9	Hulda Hudson V	612 Laurie Villages Apt. 574 Lake Dorian, LA 70399	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
8	Ray Lebsack	417 Swift Rue Effertzshire, RI 65302	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
7	Mrs. Dortha Shields PhD	928 Jaiden Key Apt. 215 Port Suzannestad, NJ 31109-6673	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
6	Kylee Doyle	98653 Octavia Lake Wunschborough, VT 79510	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
5	Mrs. Willow Hane	2483 Toy Crossroad Apt. 371 East Jarretberg, MN 64477	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
4	Dr. Lonny Nienow	24097 Prosacco Mills North Mariela, MS 81718	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
3	Prof. Ernesto Herman II	167 Francisca Passage Suite 178 Marquardtfort, MS 41155-2820	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
2	Mellie Towne III	76923 Lisette Rapid New Broderickfort, AK 84979	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>
1	Dina Johnston	909 Karianne Flat Suite 535 Lake Einarland, WI 12491	☹	<div>Show</div> <div>Edit</div> <div>Delete</div>

P H P

CRUD - READ

A função index do controller do nosso Player deve retornar uma lista de Players



```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    $players = Player::orderBy('id', 'desc')->get();

    return view('pages.players.index', ['players' => $players]);
}
```

O nosso index da view do resource Player deverá conter a seguinte definição:

```
@extends('master.main')
```

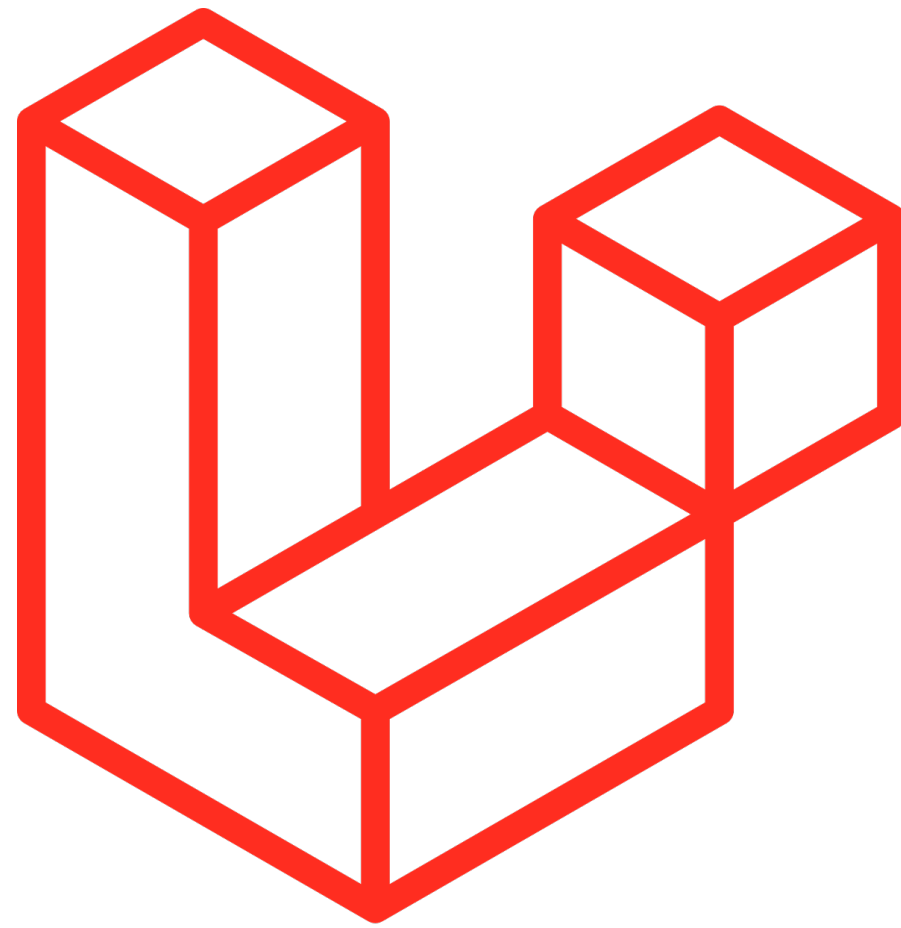
```
@section('content')
```

```
@component('components.players.players-list', ['players' => $players])
@endcomponent
```

```
@endsection
```

P H P

CRUD - CREATE

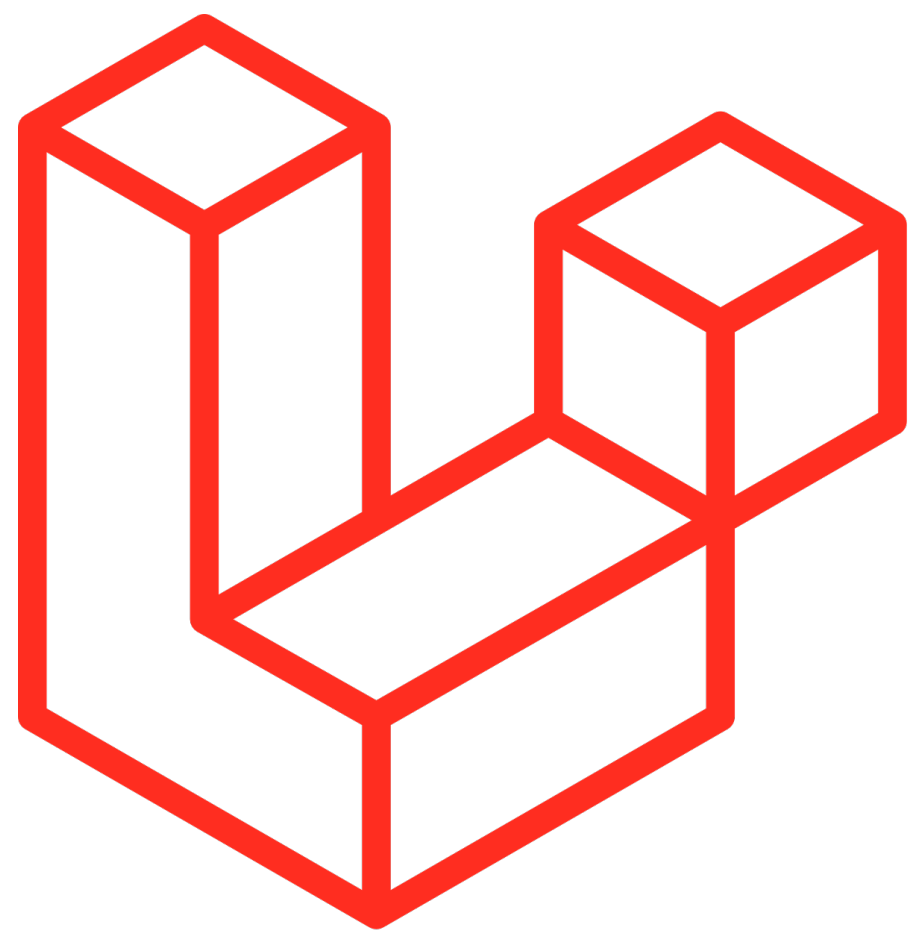


Create | INSERT

Create | Criar ou adicionar novas entradas

P H P

CRUD - CREATE



Após a listagem de players vamos desenvolver o formulário de inserção de Players que deverá respeitar o seguinte layout:

Add Player

Name

Type your name

We'll never share your data with anyone else.

Address

Type your address

Description

Type your description

Retired

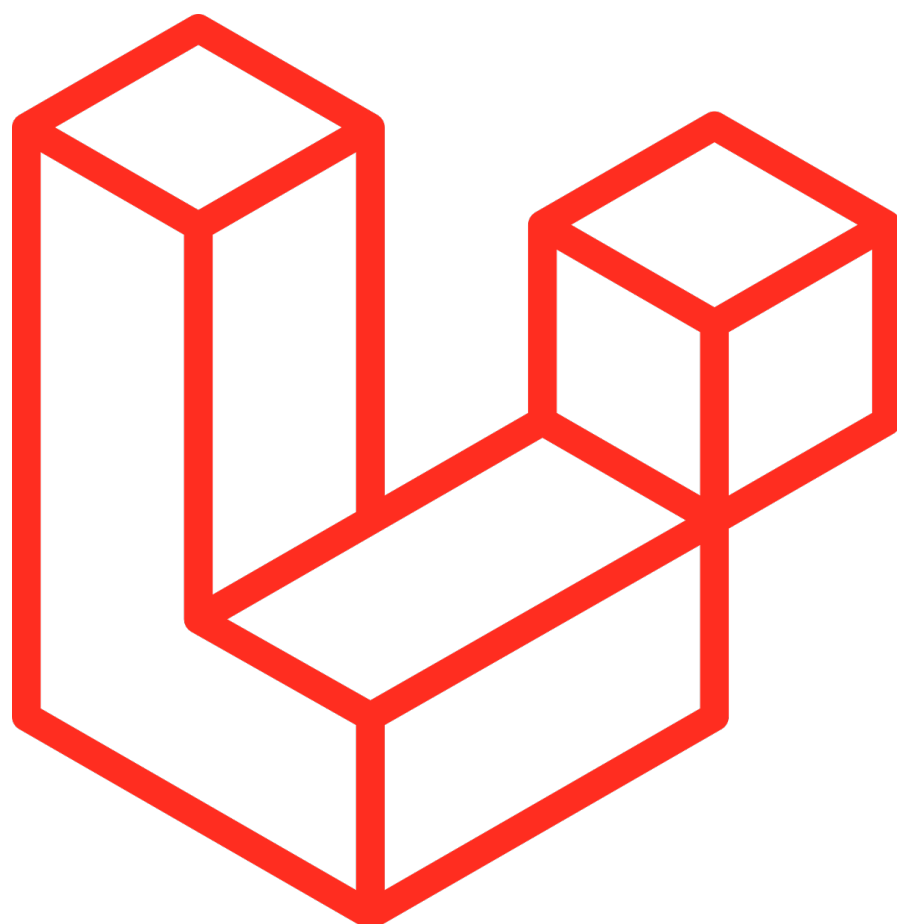
☐ Yes
 ☐ No

Submit

P H P

CRUD - CREATE

A função create do controller do nosso Player deve retornar a view do formulário:



```
/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    return view('pages.players.create');
}
```

O nosso create da view do resource Player deverá conter a seguinte definição:

```
@extends('master.main')

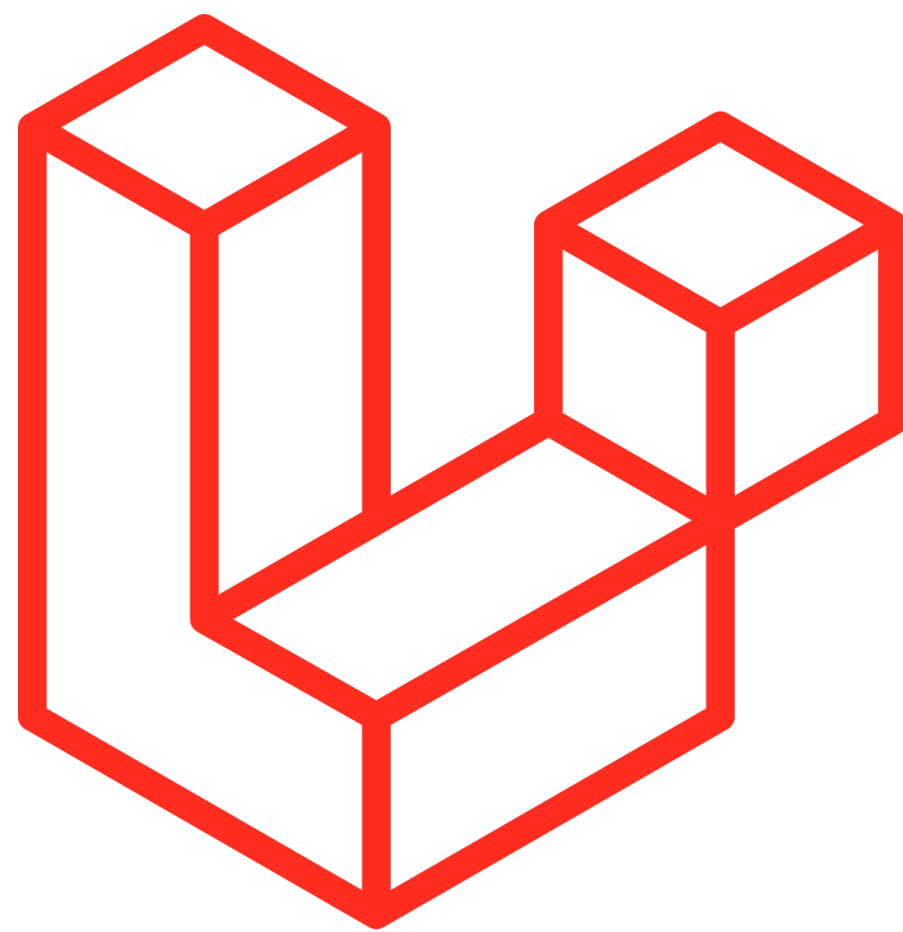
@section('content')

    @component('components.players.player-form-create')
    @endcomponent

@endsection
```

CRUD - CSRF

Quando enviamos dados via POST, PUT ou PATCH o Laravel faz uma validação na requisição para evitar que fontes externas enviem dados ou falsifiquem os nossos pedidos. Este controlo é chamado de CSRF.



CSRF é um dos ataques mais conhecidos, existe desde a “criação” da Web. Ele ocorre quando um pedido HTTP é feita entre sites na tentativa de se passar por um utilizador legítimo. Quem se utiliza desse tipo de ataque normalmente foca em fazê-lo esperando que o utilizador alvo esteja autenticado no site onde a requisição fraudulenta será realizada, a fim de se ter mais privilégios e acessos a operações.

As credenciais do utilizador é validada, uma cookie é enviada na resposta do pedido HTTP. A partir desse momento, o browser tem salvo no disco o cookie que nos mantém autenticado.

CSRF Tokens

A proteção classicamente utilizada nos formulários é a de criar um campo oculto com um token único por utilizador. Este token fica salvo na sessão do utilizador no servidor e, quando o formulário é enviado, o token enviado pelo formulário é comparado com o que se tem na sessão, no servidor. Sendo iguais, o pedido é aceite. Caso contrário, é recusada.

P H P

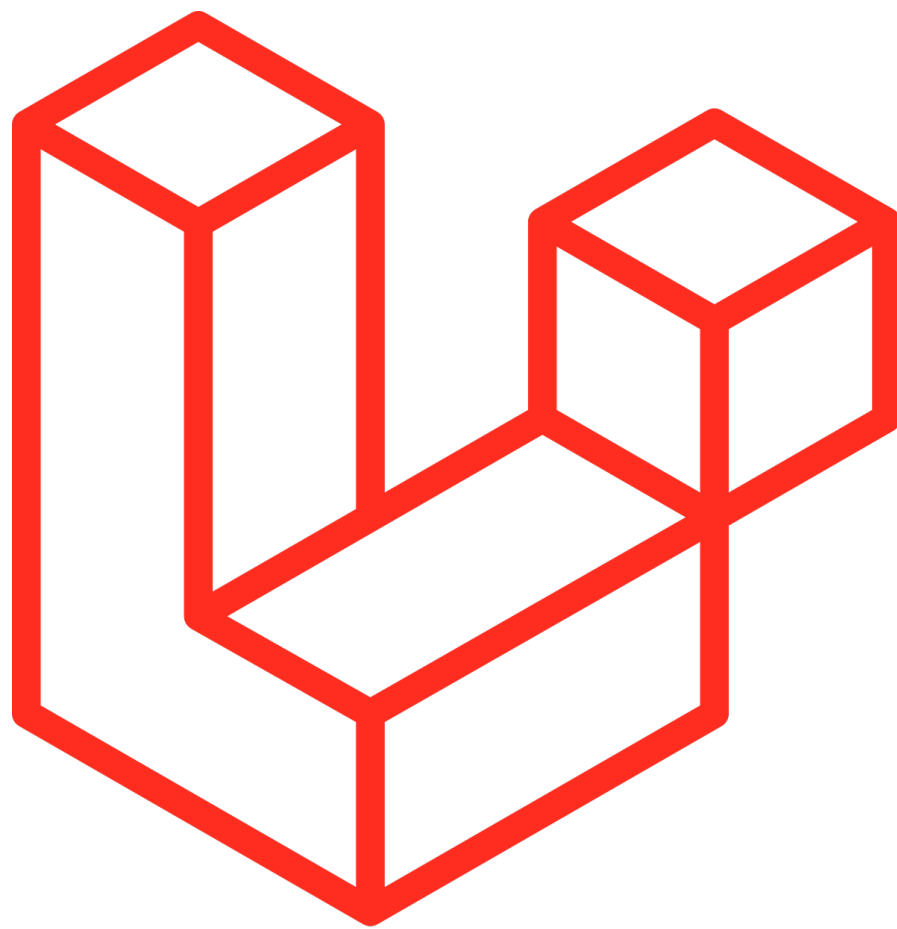
CRUD - CSRF

Na framework Laravel, este controlo é bastante simples de implementar, para isso, apenas é necessário adicionar a tag do exemplo

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

```
@csrf
```

Agora neste momento vamos criar o formulário de inserção de registos, para isso vamos utilizar o formulário de registo de utilizador que está em [views/auth/register](#) como referência.



CRUD - CREATE - Form Example

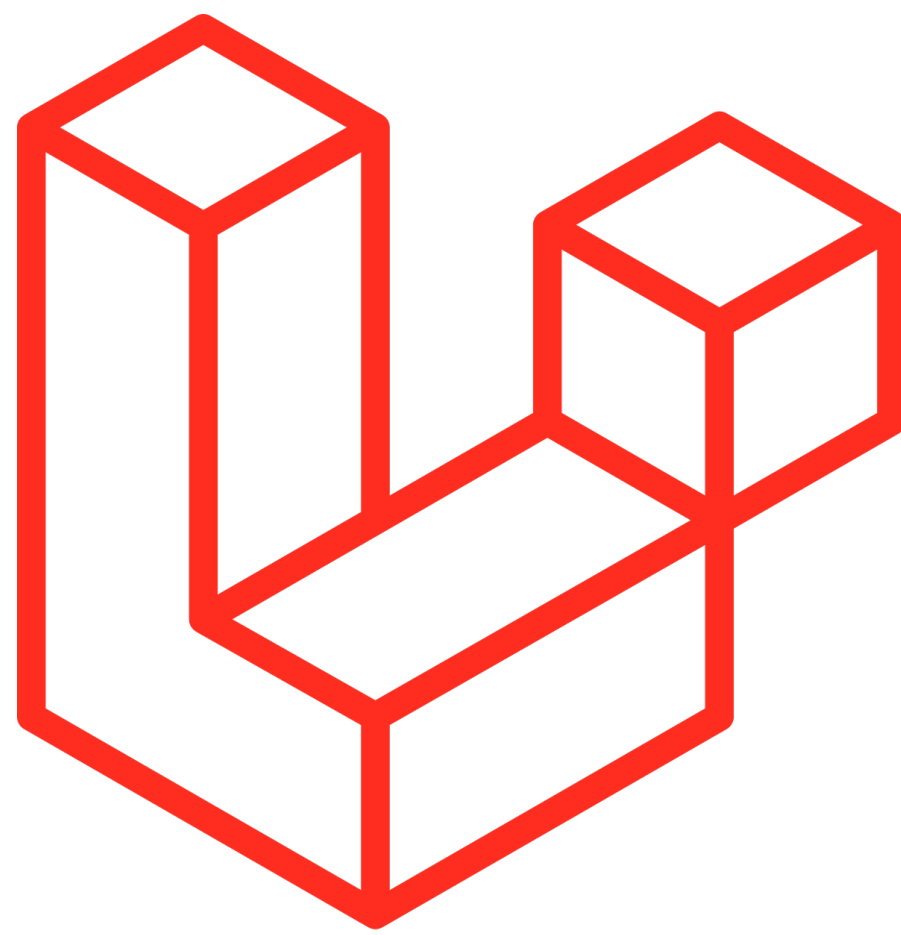
O component components.players.player-form-create vai conter o layout do formulário de criação de um player

```
<form method="POST" action="{{ url('players') }}">
  @csrf
  <div class="form-group">
    <label for="name">Name</label>
    <input
      type="text"
      id="name"
      name="name"
      autocomplete="name"
      placeholder="Type your name"
      class="form-control"
      @error('name') is-invalid @enderror"
      value="{{ old('name') }}"
      required
      aria-describedby="nameHelp">

    <small id="nameHelp" class="form-text text-muted">We'll never share your data with anyone else.</small>

    @error('name')
      <span class="invalid-feedback" role="alert">
        <strong>{{ $message }}</strong>
      </span>
    @enderror
  </div>

  <button type="submit" class="mt-2 mb-5 btn btn-primary">Submit</button>
</form>
```



<https://laravel.com/docs/7.x/validation>

<https://laravel.com/docs/7.x/eloquent>

```
class Player extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'address',
        'description',
        'retired'
    ];
}
```

P H P

CRUD - CREATE

23

```
/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
```

```
    $this->validate($request, [
        'name'      => 'required',
        'address'    => 'required',
        'description' => 'required',
        'retired'    => 'required'
    ]);
```

```
    Player::create($request->all());
```

```
/* Example 2
$input = $request->all();
Player::create($input);
*/
```

```
/* Example 3
Player::create([
    'name'      => $request->name,
    'address'    => $request->address,
    'description' => $request->description,
    'retired'    => $request->retired
]);
*/
```

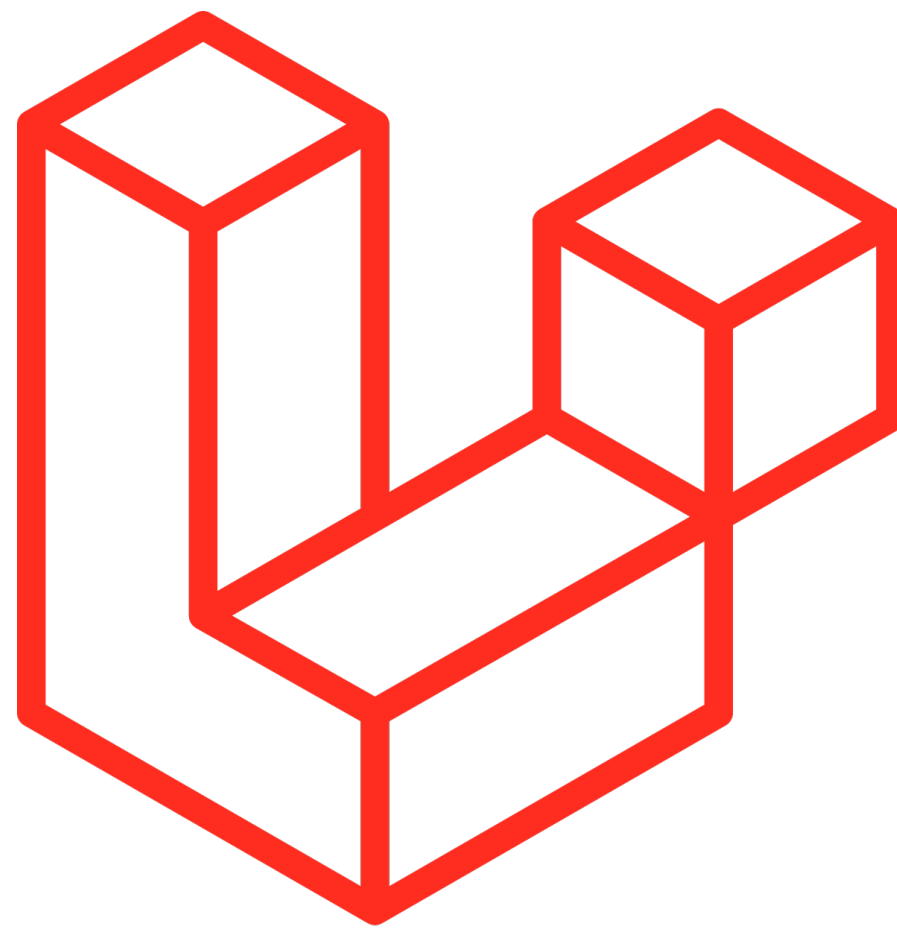
```
/* Example 4
$player      = new Player();
$player->name  = $request->name;
$player->address = $request->address;
$player->description = $request->description;
$player->retired = $request->retired;
$player->save();
*/
```

```
return redirect('players')->with('status','Item created successfully!');
```

```
}
```


P H P

CRUD - CREATE



<https://laravel.com/docs/7.x/eloquent>

```
<h1> Players List </h1>
```

```
@if (session('status'))
```

```
<div class="alert alert-success alert-dismissible fade show" role="alert">
```

```
    {{ session('status') }}
```

```
<button type="button" class="close" data-dismiss="alert" aria-label="Close">
```

```
<span aria-hidden="true">&times;</span>
```

```
</button>
```

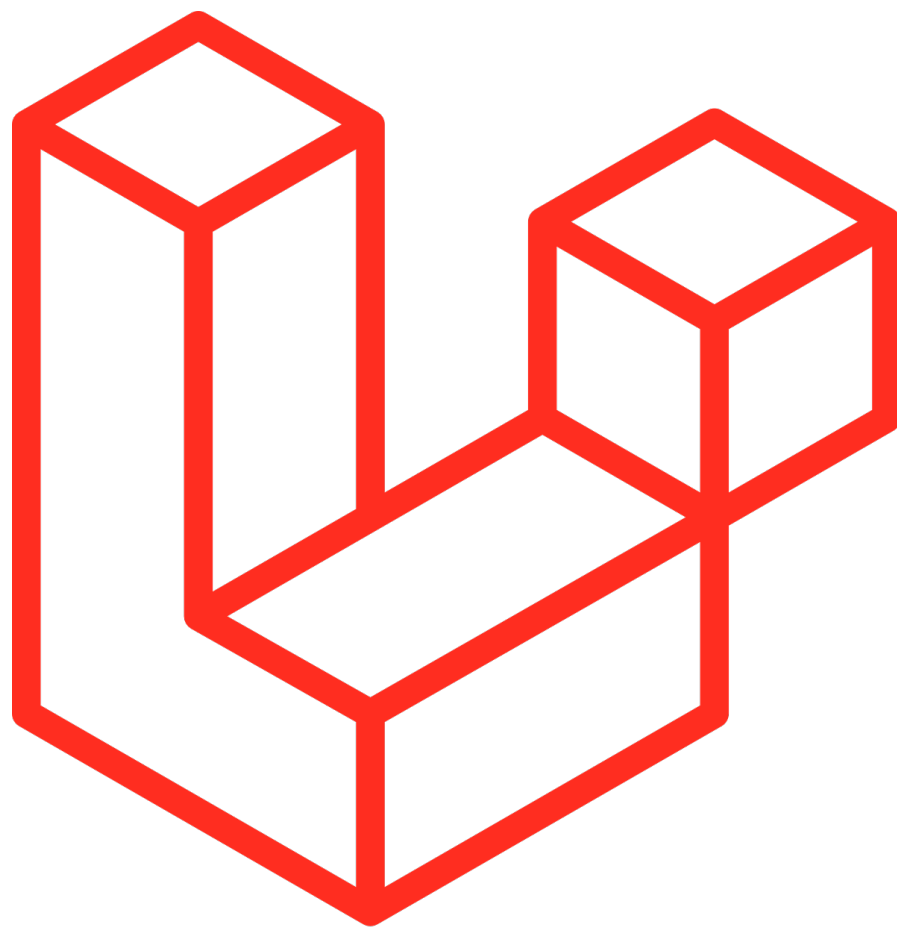
```
</div>
```

```
@endif
```


P H P

CRUD - SHOW

Após a nossa listagem e inserção de registros estar finalizada, vamos criar uma pagina de detalhe do nosso player, esta página será responsável por mostrar todos os dados de um Player incluindo a description. Para isso temos que fazer alguma alterações ao button de show, para no redirecionar para uma rota de show onde passamos por parâmetro o id do nosso player.



```
<div class="pr-1">
  <a href="{{url('players/' . $player->id)}}" type="button"
    class="btn btn-success">Show</a>
</div>
```

O formulário de show Player deverá respeitar o seguinte layout:

Show Player

Name

Prof. Maxime Stokes

We'll never share your data with anyone else.

Address

603 Tremblay ValleyMillsfurt, AR 37917-1156

Description

Ut qui aspernatur quis explicabo velit est. Est dolores explicabo eos sit nostrum modi. Eveniet quod atque nisi ut. Explicabo cum tempora excepturi corporis. Amet nostrum porro explicabo earum voluptates deserunt et. Ratione autem at vero consequuntur iusto sint iure. Beatae aliquam laudantium repellat consequuntur. Optio aut consequatur corrupti enim maiores modi. Ut totam.

Retired

☒ Yes

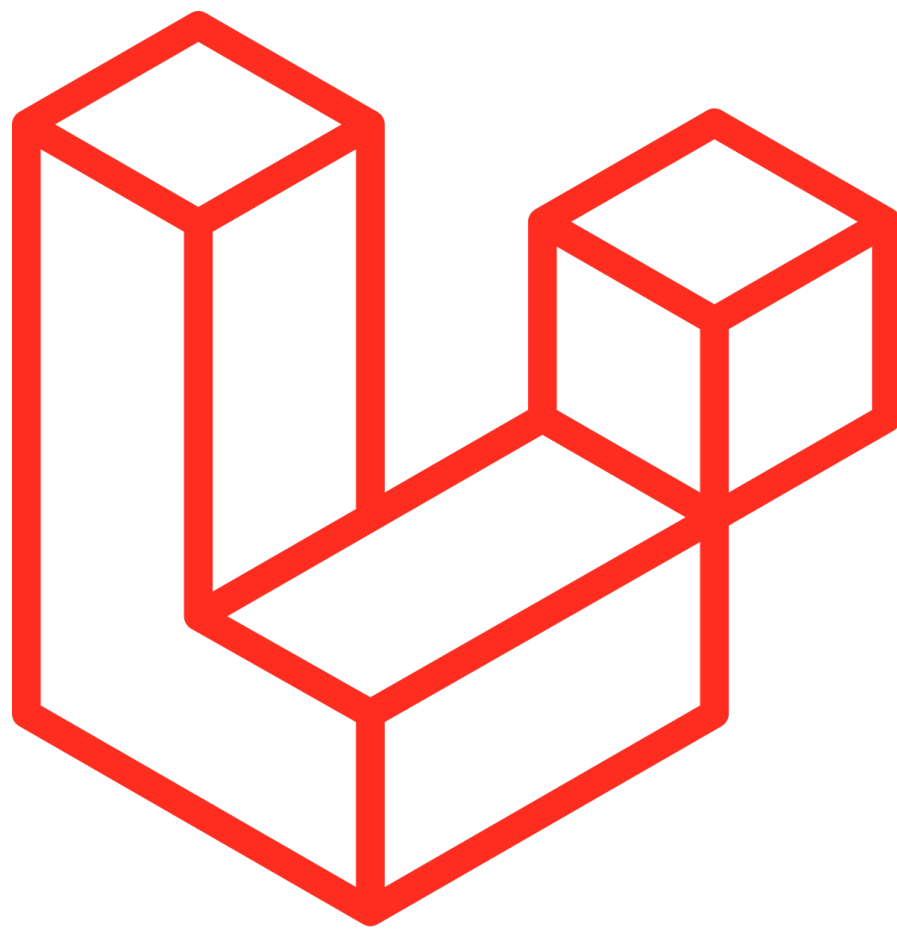
☐ No

Back

P H P

CRUD - SHOW

O nosso controller show apenas irá devolver o player em questão



```
/**
 * Display the specified resource.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function show(Player $player)
{
    return view('pages.players.show', ['player' => $player]);
}
```

O nosso show da view do resource Player deverá conter a seguinte definição:

```
@extends('master.main')

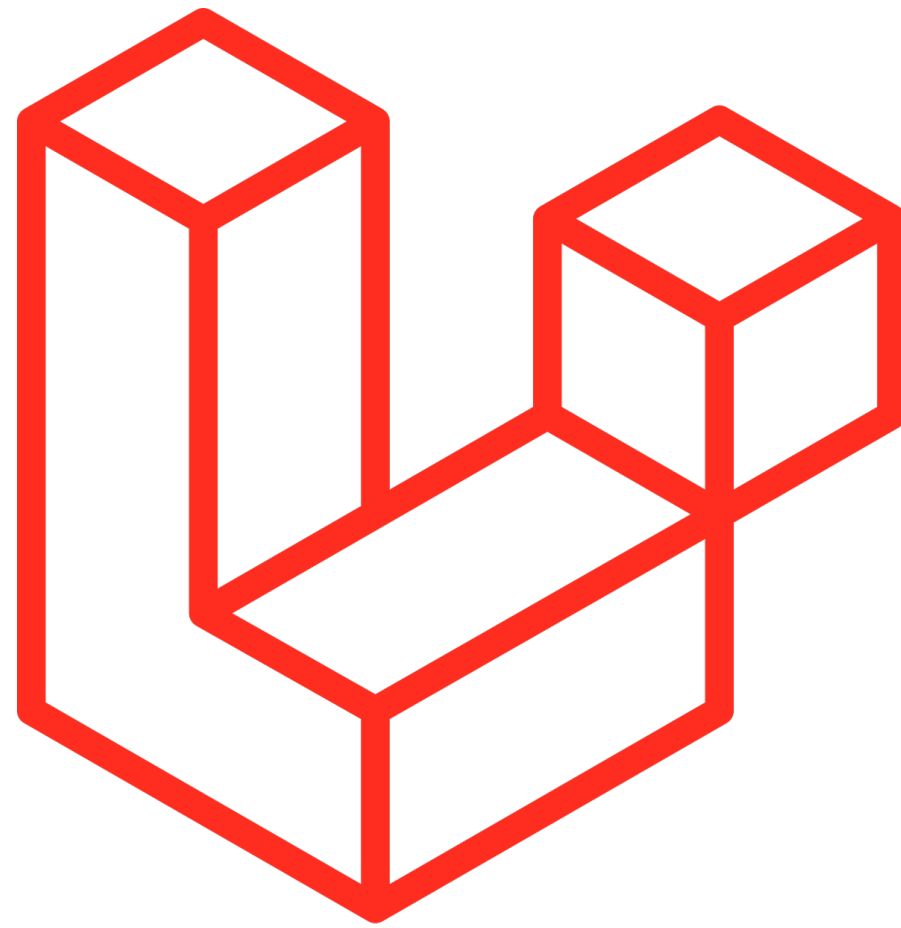
@section('content')

    @component('components.players.player-form-show', ['player' => $player])
    @endcomponent

@endsection
```

P H P

CRUD - UPDATE



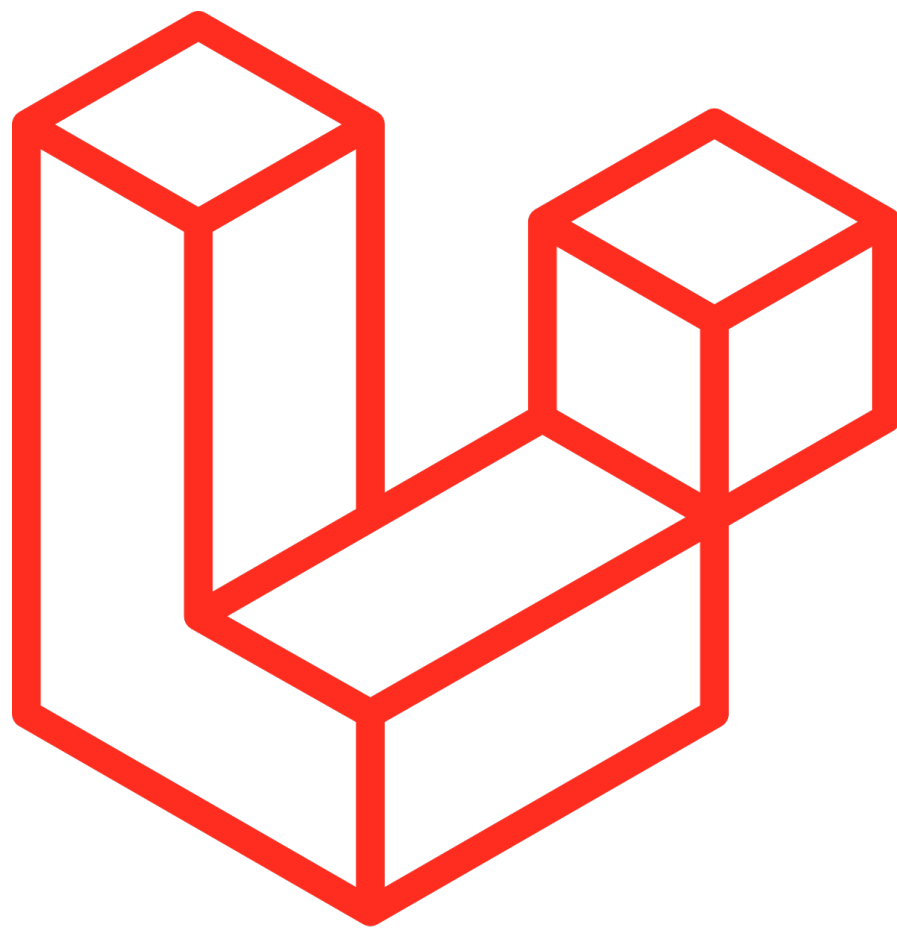
Update | UPDATE

Update | Atualizar ou editar entradas existentes

P H P

CRUD - UPDATE

Após a nossa vista detalhada estar finalizada, vamos criar uma pagina de edição do nosso player, esta página será responsável por editar os dados de um Player. Para isso temos que fazer alguma alterações ao button de edit, para no redirecionar para uma rota de edit onde passamos por parâmetro o id do nosso player.



```
<div class="pr-1">  
  <a href="{{url('players/' . $player->id . '/edit')}}" type="button"  
    class="btn btn-primary">Edit</a>  
</div>
```

O formulário de edit Player deverá respeitar o seguinte layout:

Edit Player

Name

Prof. Maxime Stokes

We'll never share your data with anyone else.

Address

603 Tremblay ValleyMillsfurt, AR 37917-1156

Description

Ut qui aspernatur quis explicabo velit est. Est dolores explicabo eos sit nostrum modi. Eveniet quod atque nisi ut. Explicabo cum tempora excepturi corporis. Amet nostrum porro explicabo earum voluptates deserunt et. Ratione autem at vero consequuntur iusto sint iure. Beatae aliquam laudantium repellat consequuntur. Optio aut consequatur corrupti enim maiores modi. Ut totam.

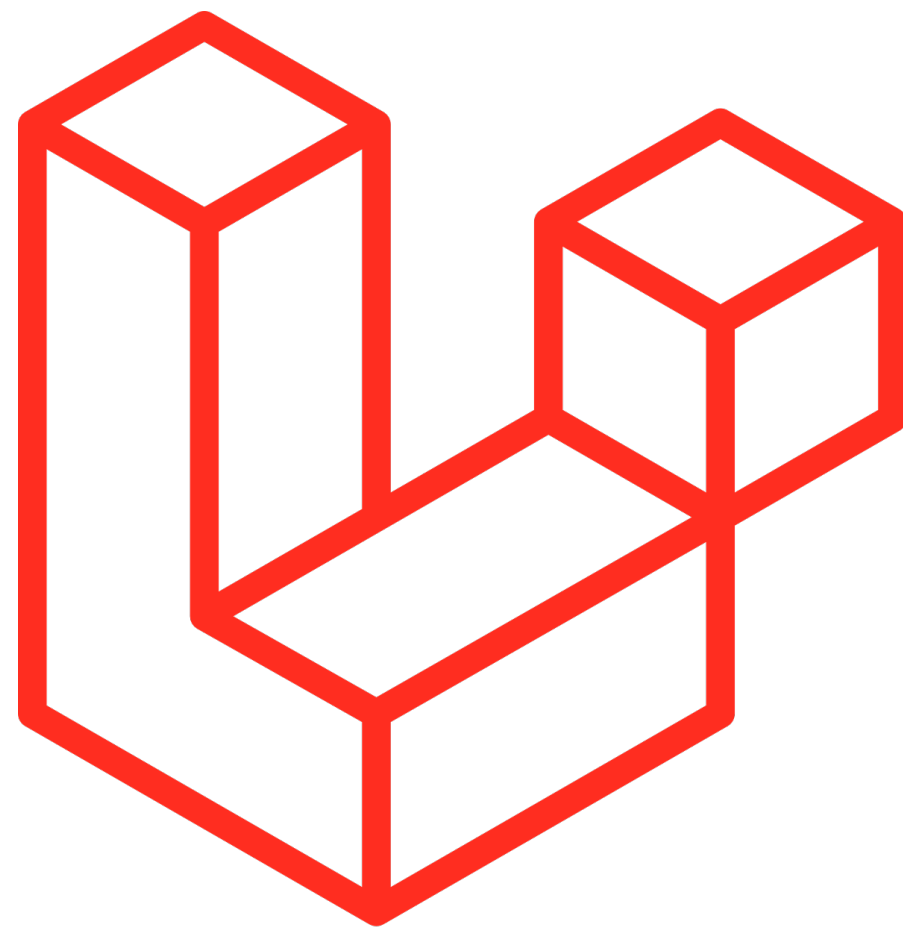
Retired

☒ Yes ☐ No

Submit

P H P

CRUD - UPDATE



O nosso controller edit apenas irá devolver o player em questão

```
/**
 * Show the form for editing the specified resource.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function edit(Player $player)
{
    return view('pages.players.edit', ['player' => $player]);
}
```

O nosso edit da view do resource Player deverá conter a seguinte definição:

```
@extends('master.main')

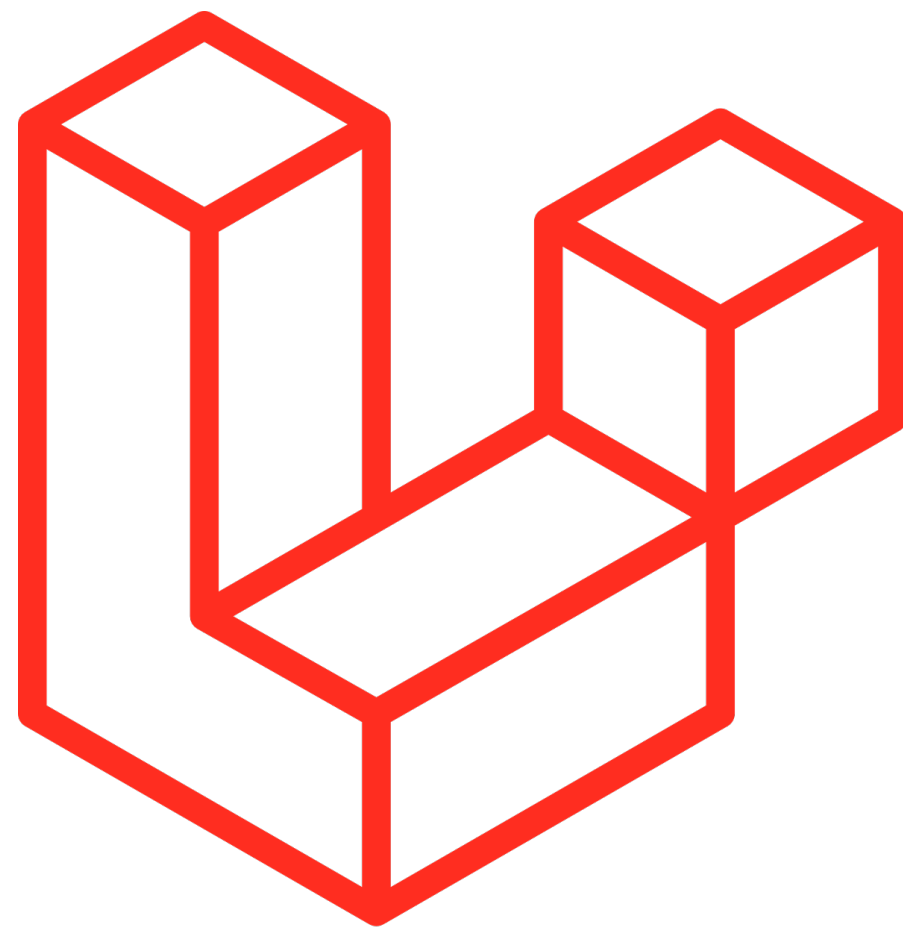
@section('content')

    @component('components.players.player-form-edit', ['player' => $player])
    @endcomponent

@endsection
```

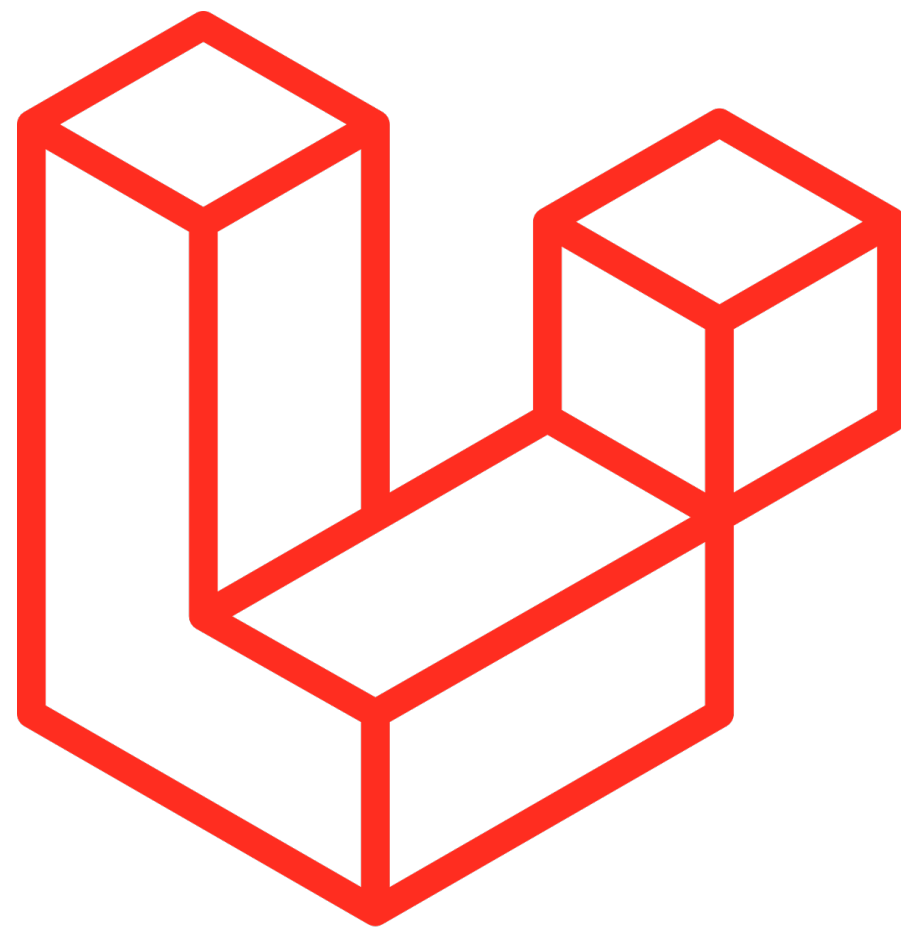
P H P

CRUD - UPDATE



Na framework Laravel, quando temos um formulário e pretendemos fazer update a um registo único devemos utilizar o method PUT para respeitar a convenção, para isso devemos utilizar a seguinte tag:

```
<form method="POST" action="{{ url('players/' . $player->id) }}">  
    @csrf  
    @method( 'PUT' )
```

<https://laravel.com/docs/7.x/validation>

<https://laravel.com/docs/7.x/eloquent>

```
class Player extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'address',
        'description',
        'retired'
    ];
}
```

P H P

CRUD - UPDATE

33

```
/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, Player $player)
{
    $player->update($request->all());

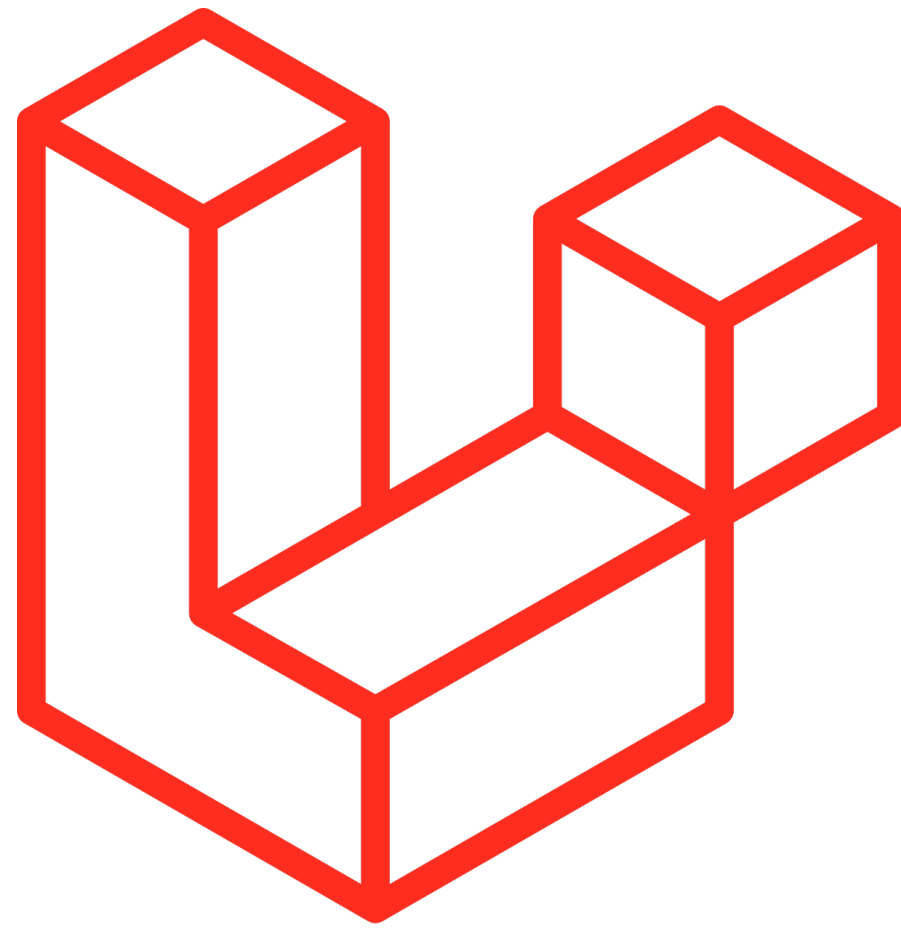
    return redirect('players')->with('status', 'Item edited successfully!');
}
```

```
/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, Player $player)
{
    $player          = Player::find($player->id);
    $player->name      = $request->name;
    $player->address    = $request->address;
    $player->description = $request->description;
    $player->retired     = $request->retired;
    $player->save();

    return redirect('players')->with('status', 'Item edited successfully!');
}
```

P H P

CRUD - DELETE



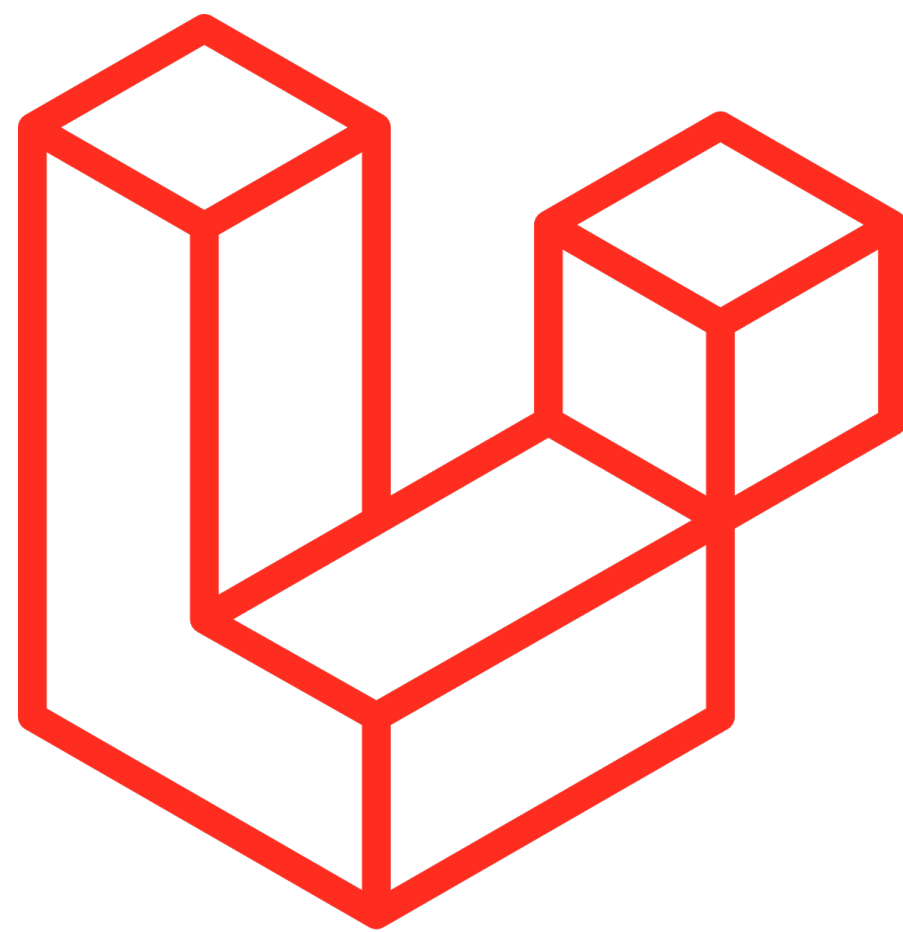
Delete (Destroy) | DELETE

Delete (Destroy) | Remover entradas existentes

P H P

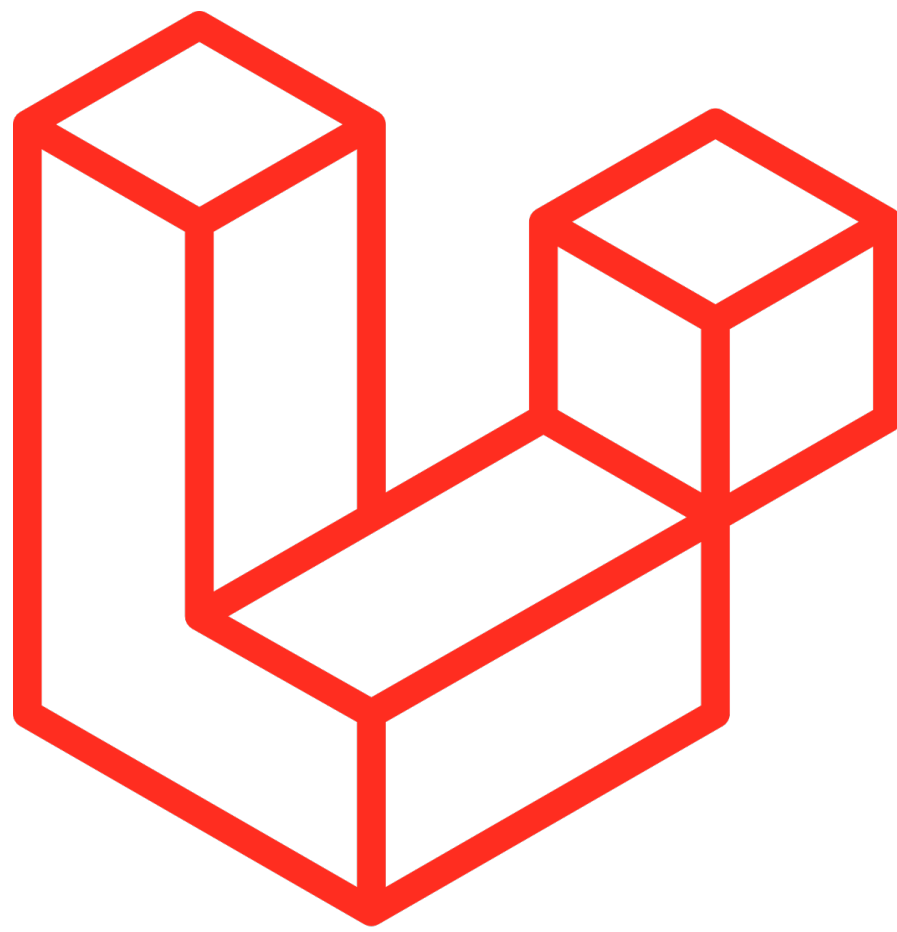
CRUD - DELETE

Após a nossa vista de edit estar finalizada, por fim vamos fazer a nossa instrução de delete. Para isso temos que fazer algumas alterações ao button de Delete, para nos fazer um request DELETE.



```
<form action="{{url('players/' . $player->id)}}" method="POST">
    @csrf
    @method('DELETE')

    <button type="submit" class="btn btn-danger">Delete</button>
</form>
```



<https://laravel.com/docs/7.x/validation>

<https://laravel.com/docs/7.x/eloquent>

```
class Player extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'address',
        'description',
        'retired'
    ];
}
```

P H P

CRUD - DELETE

36

```
/**
 * Remove the specified resource from storage.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function destroy(Player $player)
{
    $player->delete();

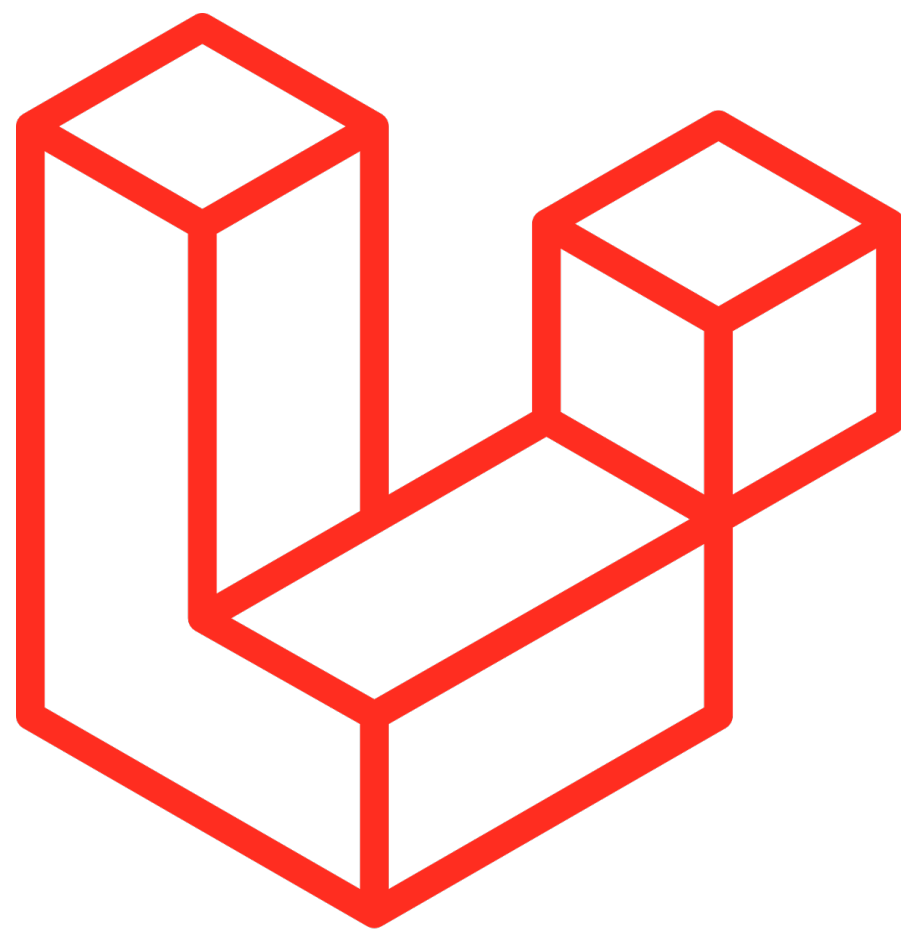
    return redirect('players')->with('status', 'Item deleted successfully!');
}
```

```
/**
 * Remove the specified resource from storage.
 *
 * @param  \App\Player  $player
 * @return \Illuminate\Http\Response
 */
public function destroy(Player $player)
{
    $player = Player::find($player->id);
    $player->delete();

    return redirect('players')->with('status', 'Item deleted successfully!');
}
```

P H P

CRUD - Soft Deleting



<https://laravel.com/docs/7.x/eloquent#soft-deleting>

Utilizando Soft Deleting invés de realmente removermos registros da base de dados, o Eloquent também pode “excluir” modelos. Com o SoftDelete, quando os modelos são excluídos, eles não são realmente removidos da base de dados. Em vez disso, um atributo `deleted_at` é definido no modelo e inserido na base de dados. Se um modelo tiver um `deleted_at` não nulo, o modelo é excluído na query. Veja o exemplo de implementação:

```
<?php

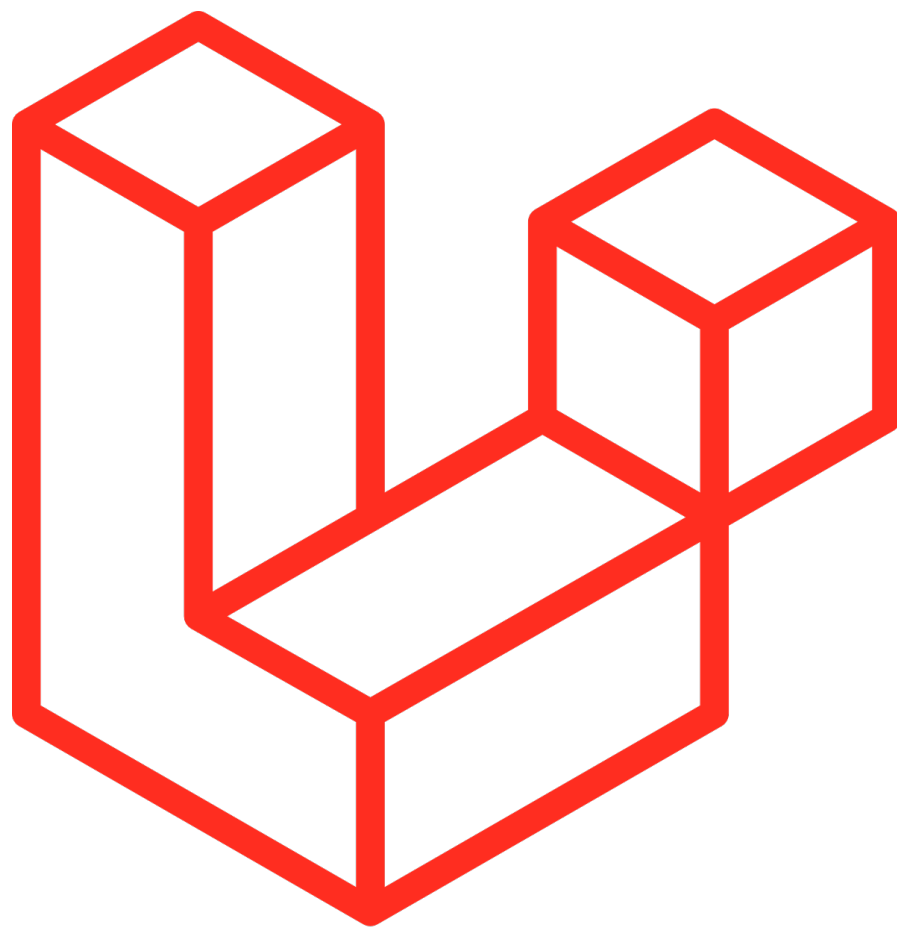
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Flight extends Model
{
    use SoftDeletes;
}
```

P H P

CRUD - Soft Deleting



Migration:

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Facades\Schema;

Schema::table('flights', function (Blueprint $table) {
    $table->softDeletes();
});

Schema::table('flights', function (Blueprint $table) {
    $table->dropSoftDeletes();
});
```