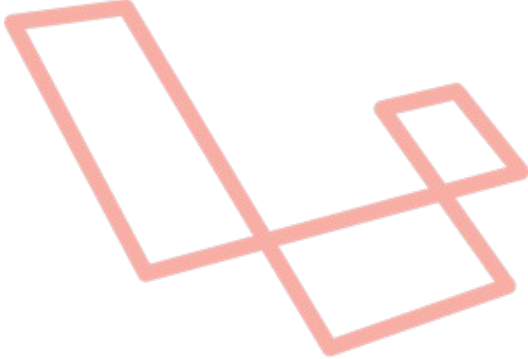
Introdução à PHP e ao framework Laravel

Adelino Lourenço
Alana Tenório
Eberson Santos

LMTS
Laboratório Multidisciplinar de Tecnologias Sociais

Pré-Requisitos

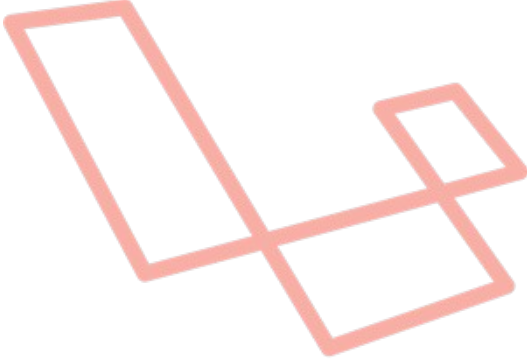


- HTML;
  - https://www.w3school.com/html/
- CSS;
  - https://www.w3school.com/css/
- JavaScript.
  - https://www.w3schools.com/js/
- PHP;
  - https://www.php.net/manual/pt_BR/

PHP: Hypertext Preprocessor

- Refere-se a uma linguagem de script open source de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser **embutida dentro do HTML**.
- **PHP é executado no servidor** (Apache), portanto, não é necessário instalar nada nas máquinas clientes (exceto um navegador para fins de execução).

PHP: Hypertext Preprocessor

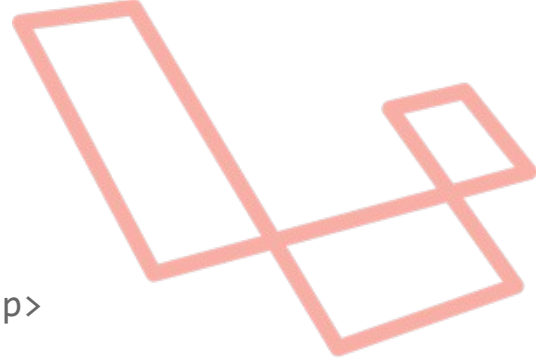


```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Exemplo</title>
    </head>
    <body>

        <?php
            echo "Olá, eu sou um script PHP!";
        ?>

    </body>
</html>
```

PHP: Hypertext Preprocessor

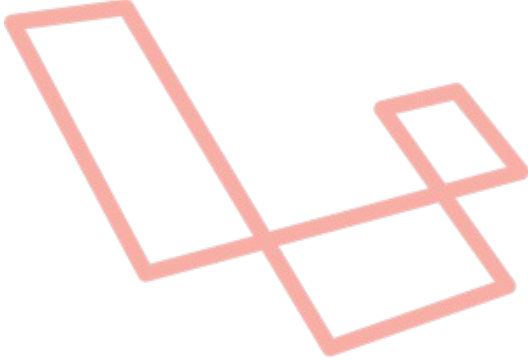
- Tudo que está fora das tags php é ignorado pelo interpretador do PHP.
- Isso permite embarcar o código PHP em “qualquer” tipo de documento.

PHP: Hypertext Preprocessor



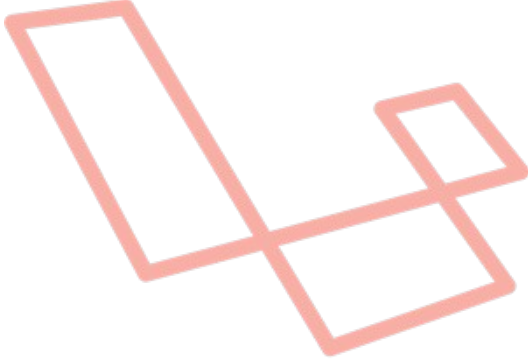
```
<p> Isto vai ser ignorado pelo PHP e enviado ao navegador. </p>
<?php echo ‘Enquanto isto vai ser interpretado’; ?>
<p>Isto também vai ser ignorado pelo PHP e enviado ao navegador. </p>


<?php $expression = true ?>

<?php if ($expression == true): ?>
    Isto irá aparecer se a expressão for verdadeira
<?php else: ?>
    Senão isto irá aparecer
<?php endif; ?>
```

PHP: Hypertext Preprocessor



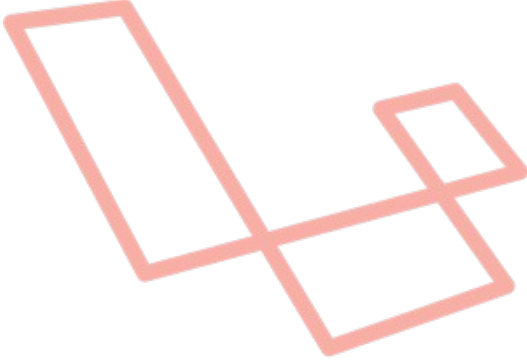
```php
<?php
    $a = 4;
    $b = 2;

    $c = "teste";
    echo $c;

    $c = 2*($a + $b);
    echo $c;

?>
```

PHP: Analisando variáveis



- **var_dump():** aloca informações sobre a variável.
- **dd():** mostra informações sobre a variável.
- **gettype():** obtém o tipo da variável.
- **funções is():** verifica se uma variável é de um determinado tipo.
    - is_int(), is_float(), is_string(), ...

PHP: Classe

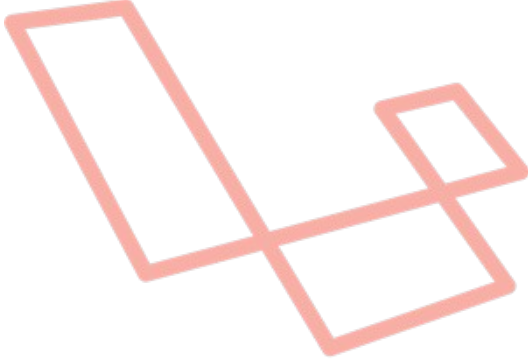


```php
<?php
class SimpleClass
{
    // declaração de propriedade
    public $var = 'um valor padrão';

    // declaração de método
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

PHP: Array

- Os arrays da linguagem PHP são implementados como mapas ordenados.
- Pode ser utilizado como um array tradicional, uma lista, um mapa, um dicionário, pilha, fila ou outras estruturas de dados.
- Existe ainda a possibilidade de utilizar arrays de arrays ou arrays multidimensionais.

PHP: Array

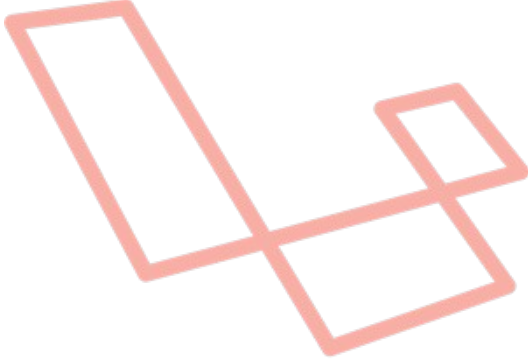


```php
$a = array(
    "a" => "b",
    "c" => "d",
    1 => 2,
    3 => 4
);

var_dump($a);
```

```php
$a = array(2,4,6,8);
$b = array(
    "p1" => "Fulano",
    "p2" => "Sicrano",
    "p3" => "Beltrano"
);
$c = array();
$d = array(
    "nome" => "Fulano",
    "notas" => array(7,8,9)
);
```

```php
// a partir do PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
```

PHP: Array



```php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
```

```php
$a = array(
    "a",
    "b",
    "c",
    "d"

);

unset($a[2]);

var_dump($a);
```

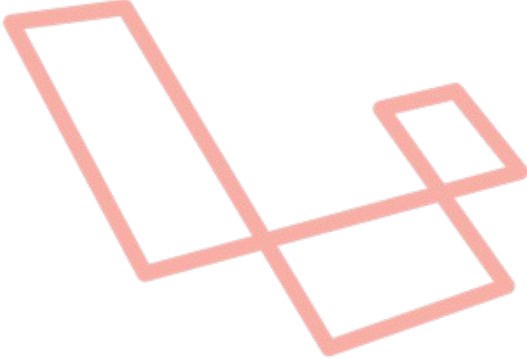
PHP: Array

```php
class Pessoa {
    public $nome;
    public $cpf;
    public $idade;

    public function __construct($n, $c, $i) {
        $this->nome = $n;
        $this->cpf = $c;
        $this->idade = $i;

    }

}
$p1 = new Pessoa("Fulano", "123", 18);

$a = (array) $p1;
var_dump($a);
```

PHP: Array

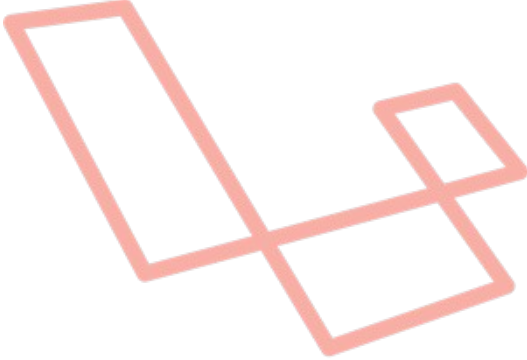
```php
$a = array("aaa", "bbb", "ccc");
$count = count($a);

for($i=0; $i<$count; $i++)
   echo "$a[$i] <br>";

echo "<br>";

foreach($a as $elemento) {
   echo "$elemento <br>";
}

echo "<br>";

foreach($a as $k => $elemento) {
   echo "$k, $elemento <br>";
}
```

PHP: Array



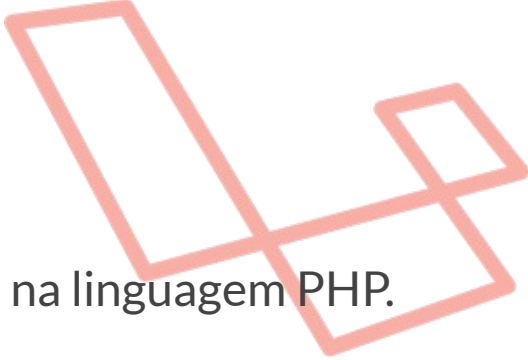
- Mais funções:
  - https://www.php.net/manual/pt_BR/ref.array.php

Laravel: Por que usar?

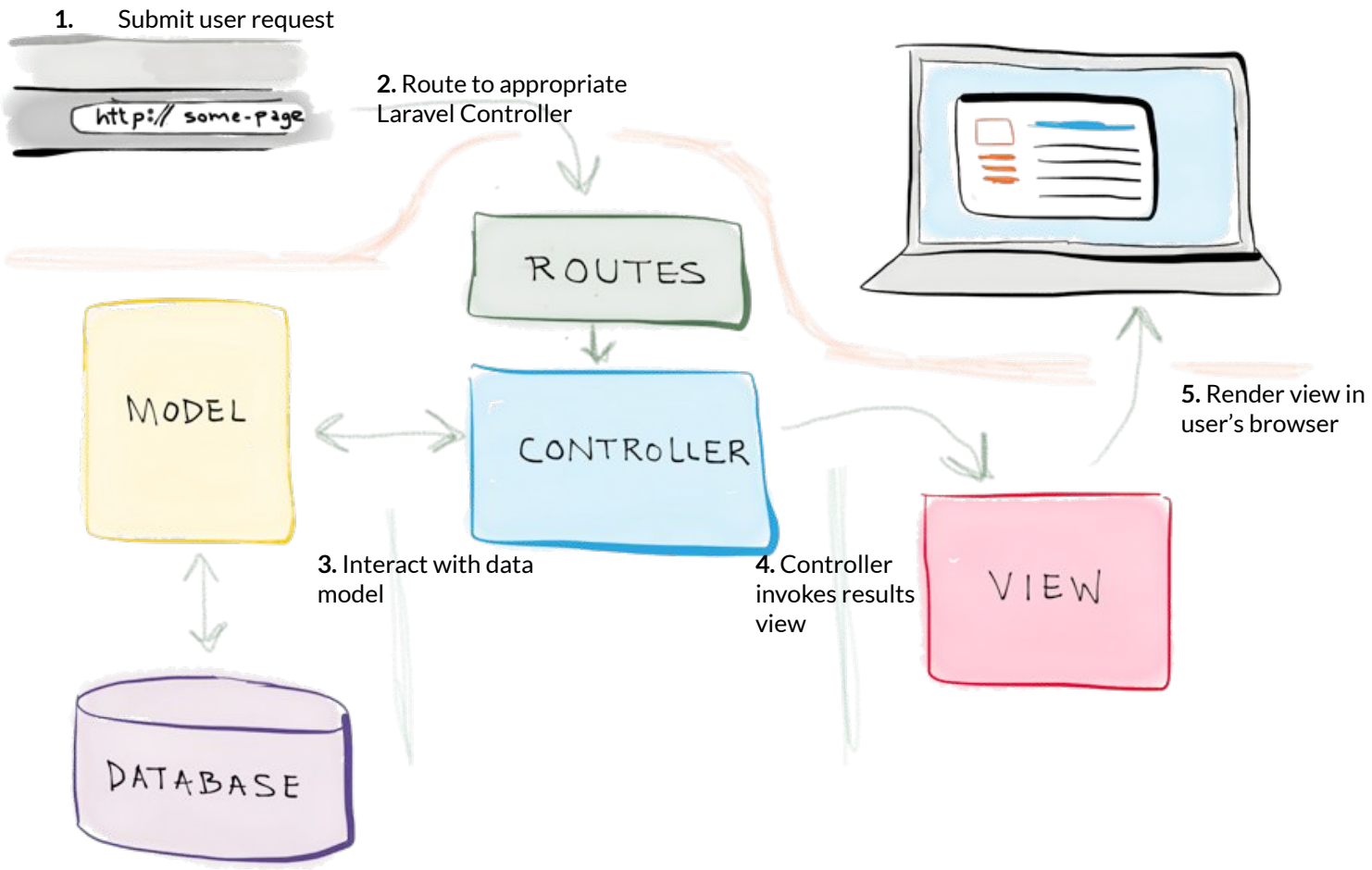


- Autenticação e Lógica de Aplicação;
- Blade;
- Artisan;
- Laracasts;
- Segurança;
- Migração de banco de dados.

Laravel: Introdução



- É um framework MVC para desenvolvimento WEB escrito na linguagem PHP.
- composer create-project laravel/laravel exemplo


1. Submit user request
http:// some-page
2. Route to appropriate Laravel Controller
ROUTES
MODEL
CONTROLLER
VIEW
DATABASE
3. Interact with data model
4. Controller invokes results view
5. Render view in user's browser

Laravel: Introdução

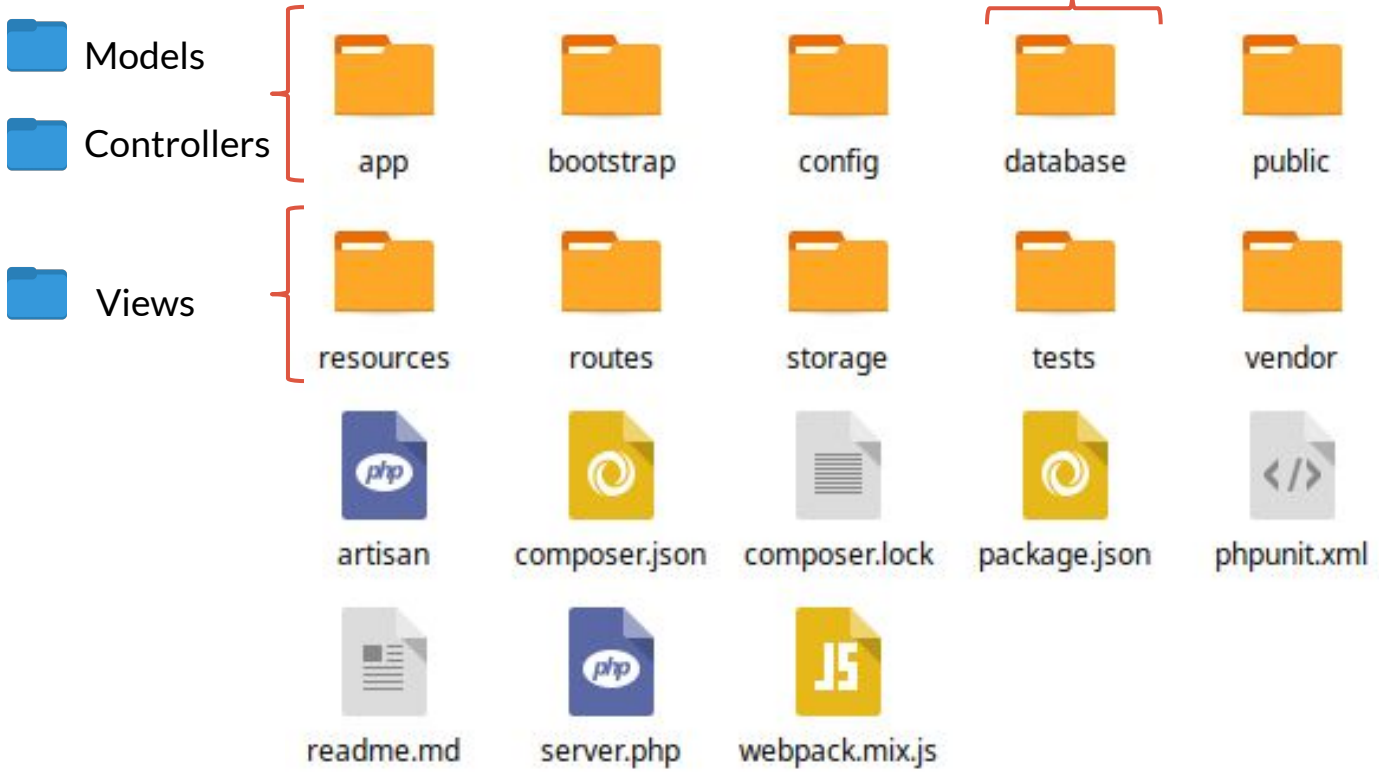
Migrations

Seeds

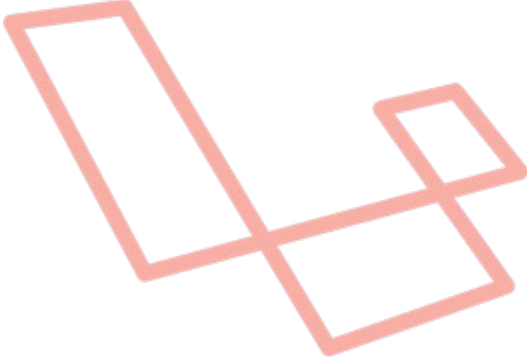
Models

Controllers

Views

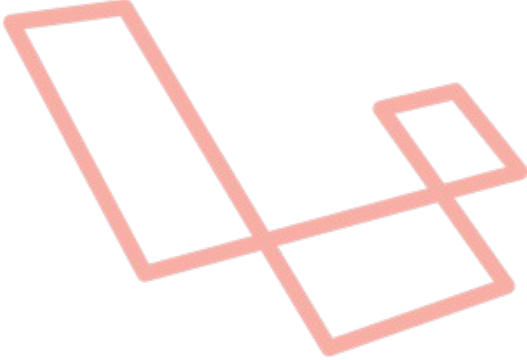


Laravel: Rotas



```php
Route::get('/teste', function() {
    return "Hello Laravel";
});

Route::get('teste/{id}', function($id) {
    return "Hello Laravel " .   $id;
});

Route::get('teste/{id?}', function($id='0') {
    return "Hello Laravel " .   $id;
});
```

Laravel: Rotas

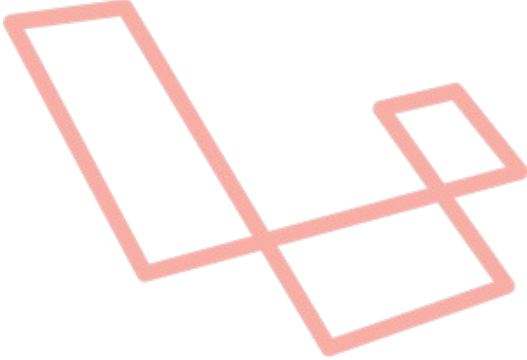


```php
Route::get('user/{name}', function ($name) {
    //
})->where('name', '[A-Za-z]+');

Route::get('user/{id}', function ($id) {
    //
})->where('id', '[0-9]+');

Route::get('user/{id}/{name}', function ($id, $name) {
    //
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Laravel: Controllers

- Em vez de a lógica para tratamento de todas as requisições no arquivo routes/web.php, é possível agrupar funcionalidades correlatas em classes Controller.
- Controller são armazenados no diretório app/Http/Controllers.

Laravel: Controllers



php artisan make:controller TesteController

```php
namespace exemplo\Http\Controllers;

use Illuminate\Http\Request;

class Teste extends Controller
{
      public function teste($id='0') {
         return "Hello Controller: " . $id;
      }
}
```

```php
Route::get('teste/{id?}', "Teste@teste");
```

Laravel: Views

- Views mantém o código html da aplicação separado da lógica de aplicação.

Laravel: Views

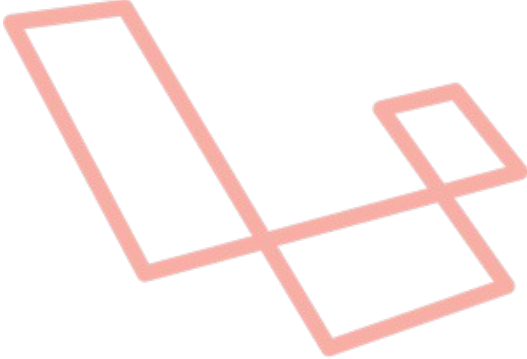
```blade
<!doctype html>
<html lang="{{ app()->getLocale() }}">
    <head>
      <title>Hello</title>
    </head>
    <body>
       <h1>Hello Laravel (View)</h1>
    </body>
</html>
```

View

```php
Route::get('/teste2/{nome}', function ($nome) {
    return view('hello', ['nome' => $nome]);
});
```

Route

Laravel: Views



```
<!doctype html>
<html lang="{{ app()->getLocale() }}">
    <head>
      <title>Hello</title>
    </head>
    <body>
        <h1>Hello Laravel (View) {{ $nome }}</h1>
    </body>
</html>
```

Laravel: Controller e View

```php
Route::get('/teste2/{nome}', "Teste@teste2");
```

Route

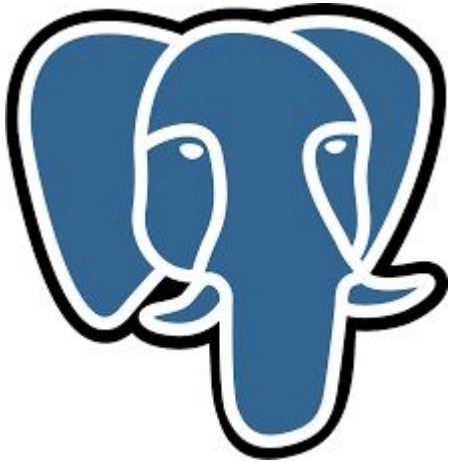
```php
class Teste extends Controller
{

      public function teste2($nome) {
         return view('hello', ['nome' => $nome]);
      }
}
```

Controller

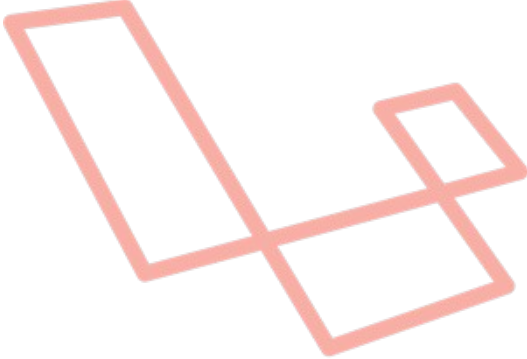
```php
<html lang="{{ app()->getLocale() }}">
    <head>
      <title>Hello</title>
    </head>
    <body>
       <h1>Hello Laravel (View) {{ $nome }}</h1>
    </body>
</html>
```

View

Laravel: Banco de Dados



Laravel: Banco de Dados



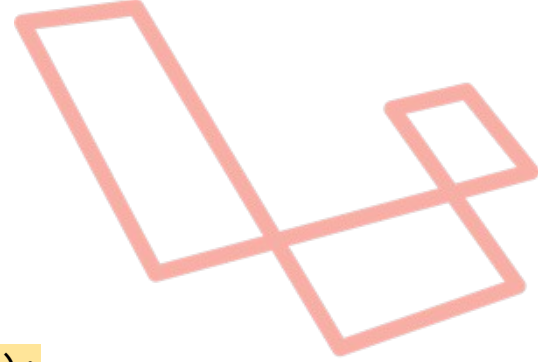
- /config/database.php
- .env

Laravel: Eloquent

- Provê uma forma simples de interagir com o Banco de Dados.
- Cada tabela do banco possui um Model correspondente, utilizado para manipular as tabelas.

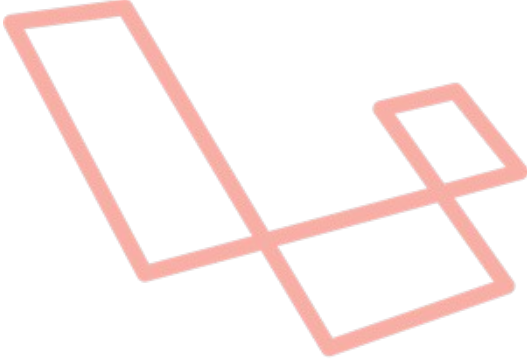
```
use exemplo\Categoria;
$categorias = \exemplo\Categoria::all();

foreach($categorias as $categoria){
     echo $categoria->descricao . “<br>”;
}
```

Laravel: Eloquent



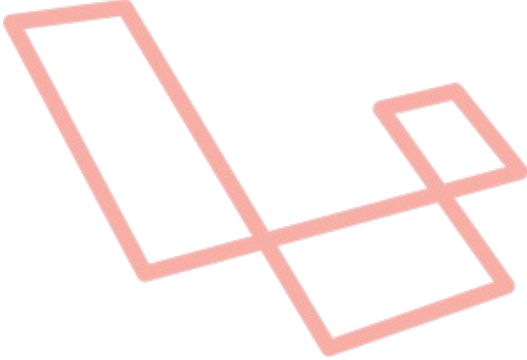
```
use exemplo\Categoria;
$categorias = \exemplo\Categoria::where('id', '<', 5)->get();

foreach($categorias as $categoria){
     echo $categoria->descricao . "<br>";
}


use exemplo\Categoria;
$categorias = \exemplo\Categoria::where(['id', '<', 5])
               ['descricao', 'like', '%i']])->get();

foreach($categorias as $categoria){
     echo $categoria->descricao . "<br>";
}
```

Laravel: Eloquent



```
use exemplo\Categoria;
$categorias = \exemplo\Categoria::where([‘id’, ‘<’, 5])
              [‘descricao’, ‘like’, ‘%i’]])
              ->orWhere(‘descricao’, ‘like’, ‘a%’)->get();

foreach($categorias as $categoria){
     echo $categoria->descricao . “<br>”;
}
```

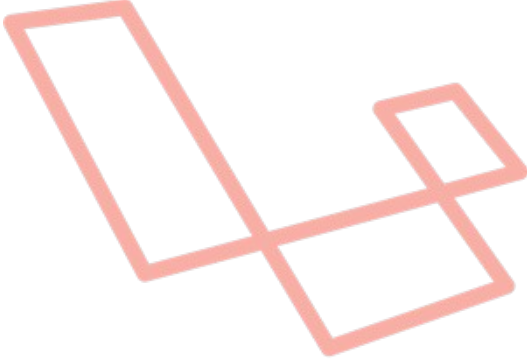
Laravel: Eloquent



```php
use exemplo\Categoria;
$categorias = \exemplo\Categoria::all()->take(4);
foreach($categorias as $categoria){
     echo $categoria->descricao . “<br>”;
}
```

```php
use exemplo\Categoria;
$categorias = \exemplo\Categoria::ordeBy(‘descricao’, ‘desc’);

foreach($categorias as $categoria){
     echo $categoria->descricao . “<br>”;
}
```

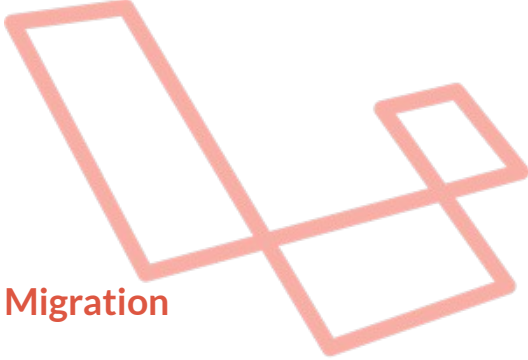
Laravel: Eloquent



```php
use exemplo\Categoria;
$categorias = \exemplo\Categoria::find($id);

if($categorias != NULL){
     return $categoria;
}
```

Prática

Laravel: Model



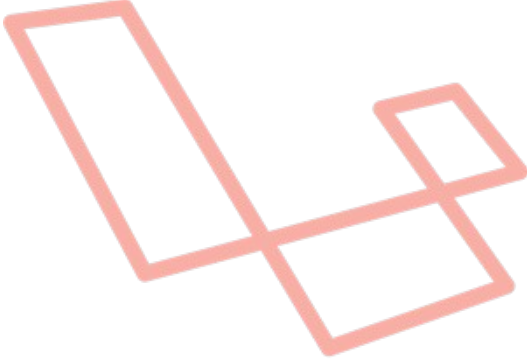
php artisan make:model -m Produto → Cria Model + Migration

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Produto extends Model

{

    protected $fillable = ['nome', 'preco'];

}
```

Laravel: Migration

```
class CreateProdutosTable extends Migration {

public function up() {

    Schema::create('produtos', function (Blueprint $table) {

    $table->bigIncrements('id');
    $table->timestamps();
    $table->string('nome');
    $table->float('preco');

    });

}
```

Laravel: Migration



- php artisan migrate
- php artisan migrate:refresh
- php artisan migrate:rollback

Laravel: Seed

- O Laravel oferta uma maneira simples de povoar seu banco de dados com dados pseudo reais utilizando, exemplo, a biblioteca Faker:
  - https://github.com/fzaninotto/Faker

CREATE READ UPDATE DELETE

C R U D

Laravel: Controller

```php
public function criar(Request $request){

    $produto = Produto::create([
            'nome' => $request->nome,
            'preco' => $request->preco,
    ]);


    return redirect()->route('exibir',$produto->id);

}
```

Laravel: Controller

```
public function exibir(Produto $produto){


     return view('produto.exibir', ['produto' => $produto]);

}

public function listar(){

     $produtos = Produto::all();
     return view('produto.listar', ['produtos' => $produtos]);

}
```

READ

Laravel: Controller

```
public function editar(Produto $produto){


     return view('produto.editar', ['produto' => $produto]);

}

public function salvar (Request $request, Produto $produto){

     $produto->nome = $request->nome;
     $produto->preco = $request->preco;
     return redirect()->route('exibir', $produto->id);

}
```

UPDATE

Laravel: Controller

```
public function remover(Produto $produto){


     return view('produto.remover', ['produto' => $produto]);

}

public function deletar(Produto $produto){

     $produto->delete();
     return redirect()->route('listar');

}
```

DELETE

Laravel: Rotas

```
Route::get('/produto/cadastrar','ProdutoController@cadastrar')->name('cadastrar');

Route::post('/produto/criar','ProdutoController@criar')->name('criar');

Route::get('/produto/exibir/{produto}','ProdutoController@exibir')->name('exibir');

Route::get('/produto/editar/{produto}','ProdutoController@editar')->name('editar');

Route::post('/produto/salvar/{produto}','ProdutoController@salvar')->name('salvar');

Route::get('/produto/remover/{produto}','ProdutoController@remover')->name('remover');

Route::post('/produto/deletar/{produto}','ProdutoController@deletar')->name('deletar');

Route::get('/produto/listar','ProdutoController@listar')->name('listar');
```

Laravel: View

```
<form action="{{route('criar')}}" method="post">
    @csrf
    <div class="form-group">
        <label for="nome" class="control-label">Nome</label>
        <input type="text" name="nome" class="form-control">
    </div>
    <div class="form-group">
        <label for="preco" class="control-label">Preço</label>
        <input type="number" step="0.01" name="preco" class="form-control">
    </div>
    <div class="form-group">
        <input type="submit" class="btn btn-primary" value="Criar">
    </div>
</form>
```

CREATE

Laravel

Login Register

Oficina LMTS

Introdução a PHP e ao framework Laravel

Cadastrar

Nome

Preço

Criar

Laravel: View

```
<table class="table table-hover">
    <tr>
        <th>Nome</th>
        <th>Preço</th>
    </tr>
    @foreach ($produtos as $produto)
        <tr>
            <td>
                <a href="{{route('exibir',[$produto->id])}}">
                    {{$produto->nome}}
                </a>
            </td>
            <td>{{$produto->preco}}</td>
        </tr>
    @endforeach
</table>
```

READ

Oficina LMTS

Introdução a PHP e ao framework Laravel

Lista de Produtos

Nome	Preço
Teclado	25
Mouse	10

Cadastrar

Laravel: View

```
<form action="{{route('salvar',$produto->id)}}" method="post">
    @csrf
    <div class="form-group">
        <label for="nome" class="control-label">Nome</label>
        <input type="text" name="nome" class="form-control" value="{{$produto->nome}}">
    </div>
    <div class="form-group">
        <label for="preco" class="control-label">Preço</label>
        <input type="text" name="preco" class="form-control" value="{{$produto->preco}}">
    </div>
    <input type="submit" class="btn btn-primary" value="Salvar">
</form>
```

UPDATE

Oficina LMTS

Introdução a PHP e ao framework Laravel

Teclado

Nome

Teclado

Preço

25

Salvar

Laravel: View

```
<form action="{{route('deletar',$produto->id)}}" method="post">
    @csrf
    <div class="form-group">
        <label for="nome" class="control-label">Nome</label>
        <input type="text" name="nome" class="form-control" value="{{$produto->nome}}" disabled>
    </div>
    <div class="form-group">
        <label for="preco" class="control-label">Preço</label>
        <input type="text" name="preco" class="form-control" value="{{$produto->preco}}" disabled>
    </div>
    <input type="submit" class="btn btn-primary" value="Deletar">
</form>
```

DELETE

Oficina LMTS

Introdução a PHP e ao framework Laravel

Teclado

Nome

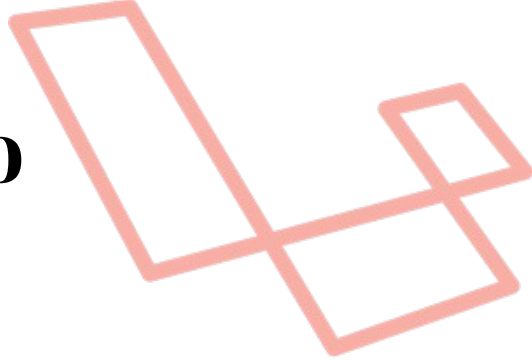
Teclado

Preço

25

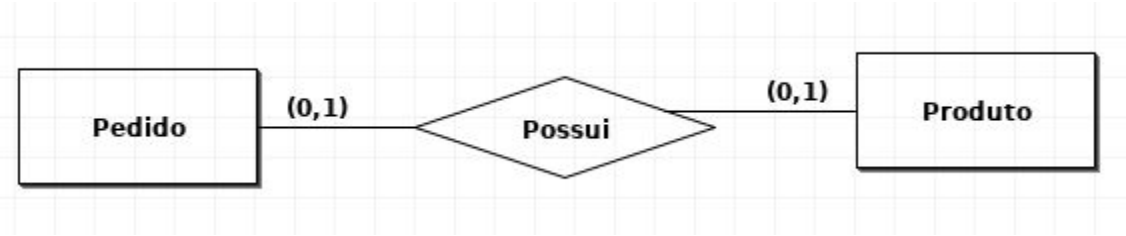
Deletar

Laravel: Tipos de Relacionamento



- One to One
  - Pessoa → Endereço
- One to Many
  - Pessoa → Telefone
- Many to Many
  - Pessoa → Disciplina

Laravel: Relacionamento One to One



Laravel: Relacionamento One to One

```php
class Pedido {

    public function produto() {

    return $this->hasOne(Produto::class);

    }

}
```

```php
class Produto {

    public function pedido() {

    return
    $this->belongsTo(Pedido::class);

    }

}
```

Laravel: Relacionamento One to One

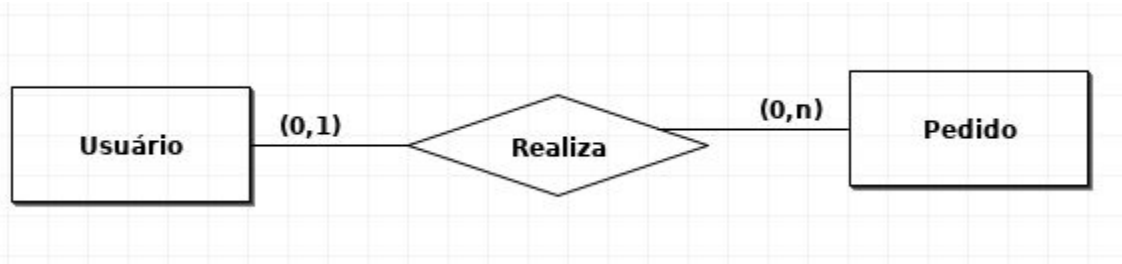
Migration **Pedido**:

```
$table->integer('produto_id');

$table->foreign('produto_id')->references('id')->on('produtos');
```

Laravel: Relacionamento One to One

```php
public function criar(Request $request){

    $pedido = Pedido::create([

        'produto_id' => $request->idProduto,

        'user_id' => Auth::user()->id,

        'data' => new DateTime(),

    ]);

}
```

Laravel: Relacionamento One to Many



Laravel: Relacionamento One to Many

```php
Class User {

    public function pedidos() {

        return $this->hasMany(Pedido::class);
    }
}
```

```php
Class Pedido {

    public function user() {

        return $this->belongsTo(User::class);

    }

}
```

Laravel: Relacionamento One to Many

Migration **Pedido**:

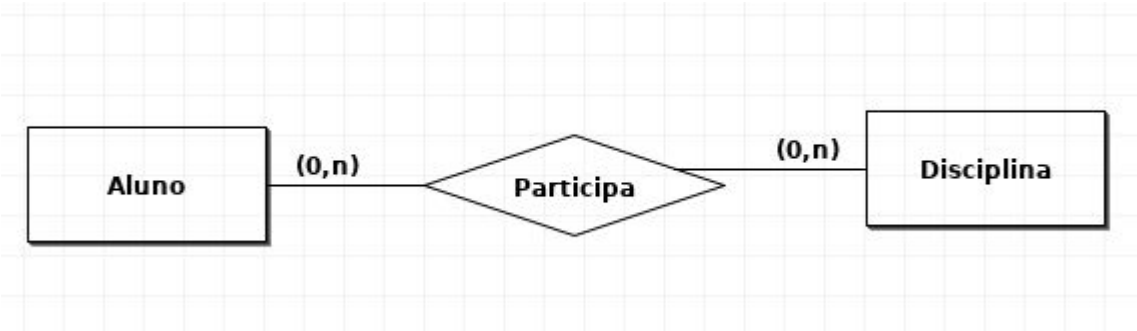
```
$table->integer('user_id');

$table->foreign('user_id')->references('id')->on('users');
```

Laravel: Relacionamento One to Many

```php
public function criar(Request $request){

    $pedido = Pedido::create([

        'produto_id' => $request->idProduto,

        'user_id' => Auth::user()->id,

        'data' => new DateTime(),

    ]);

}
```

Laravel: Relacionamento Many to Many

- É preciso criar uma terceira tabela para armazenar os dados deste tipo de relacionamento.
- Esta tabela deve conter uma chave estrangeira para cada Model envolvido no relacionamento.
- Exemplo:



Laravel: Relacionamento Many to Many

```php
Class Aluno {

    public function disciplinas() {

        return $this->belongsToMany(Disciplina::class);
    }
}
```

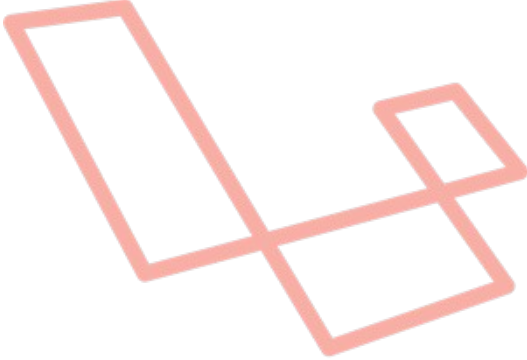
```php
Class Disciplina {

    public function aluno() {

        return $this->hasMany(Aluno::class);

    }

}
```

Laravel: Relacionamento Many to Many

Migration **Aluno_disciplina**:

```
$table->integer('aluno_id');

$table->foreign('aluno_id')->references('id')->on('aluno');


$table->integer('disciplina_id');

$table->foreign('disciplina_id')->references('id')->on('disciplinas');
```

Laravel: Auth

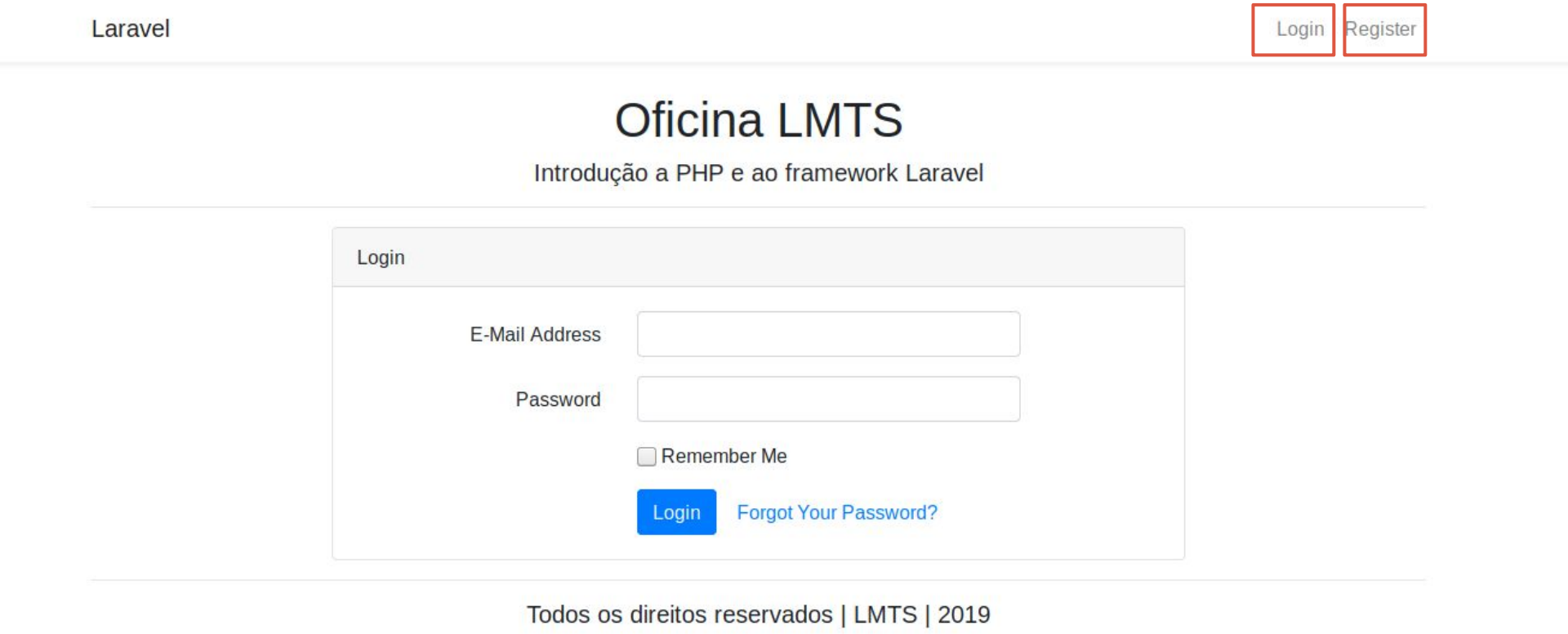


- php artisan make:auth

Métodos:

```
Auth::user();

Auth::id();

Auth::check();
```

Laravel: Auth



Laravel: Middlewares

- Fornece um mecanismo conveniente para filtrar solicitações HTTP que entram na aplicação
  - Exemplo: verificar se o usuário está autenticado

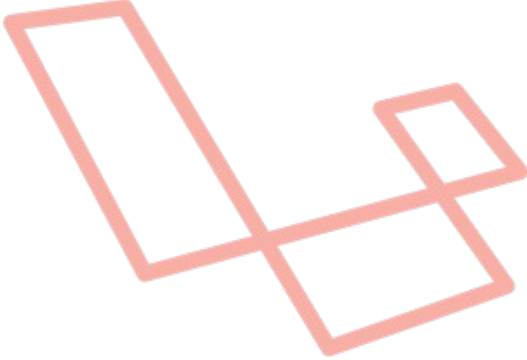
- **php artisan make:middleware nomeDoMiddleware**

```
Route::get('/produto/cadastrar','ProdutoController@cadastrar')
        ->name('cadastrar')->middleware('auth');
```

Laravel: Validator

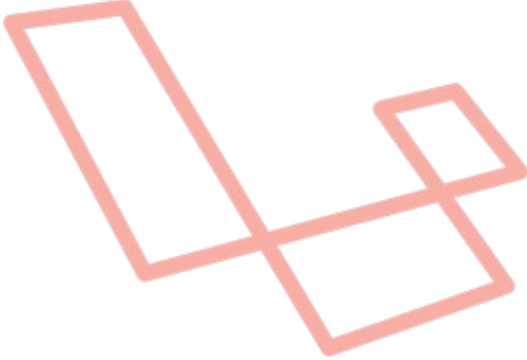
- O Laravel fornece várias abordagens para validar os dados recebidos pela aplicação
- Por padrão, a classe de controlador base do Laravel usa uma característica **ValidatesRequests**, que fornece um método conveniente para validar a solicitação HTTP uma variedade de regras de validação

Laravel: Validator



```php
public function store(Request $request){
    $validatedData = $request->validate([
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);
}
```

Laravel: Documentação



- https://laravel.com/docs/5.8/seeding
- https://github.com/fzaninotto/Faker
- https://laravel.com/docs/5.8/validation
- https://laravel.com/docs/5.8/eloquent
- https://laravel.com/docs/4.2/eloquent
- Github do projeto: https://github.com/AlanaTenorio/OficinaLaravel
- Instalação do Laravel: