**Laravel:**

Para criar o projeto usa-se:

composer create/Project laravel/laravel nome-do-projeto;

Para iniciar seu projeto:

php artisan serve;

Criar controllers:

php artisan make:controller nome-controller;

Criar models:

php artisan make:model nome-model;

Exemplo de uma implementação de um model, criamos o model Store e adicionamos o nome da tabela e da chave primária e os campos que são editáveis:

<?php

namespace App\Models\core;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Store extends Model

{

    use HasFactory;

    protected $table = "Store";

    protected $primary = "id"; //Por padrão o laravel já pega o id, mas caso seja outro campo tem que alterar na variável

    protected $fillable = [

        'name\_fantasy',

        'cnpj',

        'street',

        'number',

        'complement',

        'city',

        'state',

        'cellphone',

        'site',

        'email'

    ];

}

Para criarmos uma migration:

php artisan make:migrations nome-migration;

Bom lembrar que para criar as migrations, diferente dos demais, devemos colocar um nome padrão como por exemplo:

php artisan make:migration create\_store\_table;

Neste exemplo vamos criar uma tabela Store no banco.

É para que as criações e modificações do banco funcionem, precisamos rodar o comando:

php artisan migrate;

Caso tenha algum problema pode voltar para a última modificação usando:

php artisan migrate:rollback;

Consultas do banco:

Para pegar todos os dados podemos usar, get() ou all(), e para limitar os dados temos essas duas condições, porém “limit” só funciona com get();

$data = Store::limit(1)->get();

        $data = Store::all()->take(1);

Outra forma de trazer todos os dados e usando o chunk, essa função se torna mais eficiente quando se deseja carregar muitos dados, pois pode limitar o total de dados por vez, exemplo:

Store::chunk(1, function($data){

        //     echo <<<'HTML'

        //         <hr><br>

        //     HTML;

        //     print\_r($data);

        // });

**Geração de Tokens JWT:**

Seguindo os passos dessa documentação e bem simples:

<https://jwt-auth.readthedocs.io/en/develop/quick-start/>

Único detalhe está na classe AuthController onde se der erro nas funções auth(),

Você troca auth() por auth:: .

Agora para poder usar o token nas suas rotas, deve-se criar uma classe middleware, exemplo:

php artisan make::middleware JwtMiddleware.

Feito isso pode adicionar o esse código:

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Tymon\JWTAuth\Facades\JWTAuth;

class JwtMiddleware

{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next)

{

try

{

$user = JWTAuth::parseToken()->authenticate();

}

catch(\Exception $e)

{

if($e instanceof \Tymon\JWTAuth\Exceptions\TokenInvalidException)

{

return response()->json(['message' => 'Token inválido']);

}

elseif($e instanceof \Tymon\JWTAuth\Exceptions\TokenExpiredException)

{

return response()->json(['message' => 'Token expirado']);

}else{

return response()->json(['message' => 'Token de autorização não encontrado']);

}

}

return $next($request);

}

}

Ai nas suas rotas você cria um grupo de rotas com middleware de segurança, exemplo:

Route::group([

    'middleware' => ['jwt.verify']

], function ($router) {

    Route::get('store', [StoreController::class, 'index']);

    Route::post('create-store', [StoreController::class, 'store']);

    Route::put('update-store/{id}', [StoreController::class, 'update']);

    Route::delete('delete-store/{id}', [StoreController::class, 'destroy']);

});

Ai vai em App/Middleware/Kernel.php e adiciona o middleware da rota e aponta para sua classe que acabou de criar, procura

  protected $routeMiddleware

e adiciona o Middleware criado na última linha:

'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

'jwt.verify' => \App\Http\Middleware\JwtMiddleware::class,

**Validações:**

Para realizar validações podemos fazer direto no controller, como por exemplo:

Antes do exemplo, vou passar só uma forma de mudar as messagens de erro para português:

1. Instale o pacote

composer require lucascudo/laravel-pt-br-localization --dev

1. Publique as traduções

php artisan vendor:publish --tag=laravel-pt-br-localization

1. Configure o Framework para utilizar 'pt\_BR' como linguagem padrão

// Altere Linha 83 do arquivo config/app.php para:

'locale' => 'pt\_BR',

1. Configure o Framework para utilizar 'America/Sao\_Paulo' como data hora padrão

// Altere Linha 70 do arquivo config/app.php para:

'timezone' => 'America/Sao\_Paulo',

<https://github.com/lucascudo/laravel-pt-BR-localization/blob/master/README.md>;

Agora vamos para o exemplo de validações:

use Illuminate\Support\Facades\Validator;

$validation = Validator::make(

        $request->all(),

        [

            'cnpj' => 'required|unique:store',

            'name\_fantasy' => 'nullable|unique:store'

        ]);

        if($validation->fails())

        {

            return response()->json([$validation->errors(), 422]);

        }

Neste exemplo estou validando o cnpj e o name\_fantasy.

Outra forma de fazer, quando você está em um projeto grande e deseja manter a organização do código usa dessa forma, primeiro criamos um request:

php artisan make:request Core/StoreRequest;

Feito isso ele cria o request com duas funções authorize() e rules(), a função authorize() serve

Para validar ações do usuário, como por exemplo, se ele pode cadastrar em determinada rota,

Já a rules() são as regras de validações que deseja, é nesse exemplo adicionei mais uma função attributes(), que serve para renomear as colunas no momento de apresentar na tela para o usuário.

<?php

namespace App\Http\Requests\Core;

use Illuminate\Foundation\Http\FormRequest;

class StoreRequest extends FormRequest

{

    /\*\*

     \* Determine if the user is authorized to make this request.

     \*

     \* @return bool

     \*/

    public function authorize()

    {

        return true;

    }

    /\*\*

     \* Get the validation rules that apply to the request.

     \*

     \* @return array

     \*/

    public function rules()

    {

        return [

             'cnpj' => 'required|unique:store',

             'name\_fantasy' => 'nullable|unique:store'

        ];

    }

    public function attributes()

    {

        return [

            'name\_fantasy' => 'Nome'

        ];

    }

}

Com essa classe criada basta apenas ir no código e alterar o Request padrão da função para o Request criado, seguindo esse exemplo:

public function store(StoreRequest $request)

    {

        $data = $request->all();

        // $validation = Validator::make(

        // $request->all(),

        // [

        //     'cnpj' => 'required|unique:store',

        //     'name\_fantasy' => 'nullable|unique:store'

        // ]);

        // if($validation->fails())

        // {

        //     return response()->json([$validation->errors(), 422]);

        // }

        Store::create($data);

        return response()->json(['data' => $data]);

    }

Feito isso, a validação já ocorre antes mesmo de entrar nessa função, só que quando ele for validar os dados você deve deixar configurado no Handler.php um regrinha, porque senão você sempre vai ser direcionado para alguma página da aplicação, vai em App/Exceptions/Handler.php e adiciona essa função:

public function render($request, Throwable $e)

    {

        if($request->is('api/\*'))

        {

            if($e instanceof \Illuminate\Validation\ValidationException){

                return response()->json([$e->errors(), $e->status]);

            }

        }

    }

Feito isso a aplicação vai estar certinha.

**Upload arquivos:**

Para realizar o upload de arquivos no laravel e bem simples, primeiro vai no .arquivo .env e localiza “FILESYSTEM\_DRIVER” se tiver show, senão adicione ele em qualquer lugar do arquivo “FILESYSTEM\_DRIVER = public”, caso esteja com ‘local” mude para “public”, todos as configurações de arquivos do laravel ficam em config/filesystems.php. Feito esse processo execute

php artisan storage:link.

Esse comando gera um link para public.

Feito isso e só gerar o upload dos arquivos passando para dentro do public.

public function updateTwo(Request $request, $id)

    {

        $data = Store::findOrFail($id);

        $dataNew = $request->except(['\_token', '\_method']);

        if($request->hasFile('image')){

if($request->hasFile('image')){

    $nameFile = Str::of($request->name\_fantasy.rand(0,1000))->slug('-'). '.' .$request->image->getClientOriginalExtension();

     $dataNew['image'] = $request->image->storeAs('public', $nameFile);

        }

        $data->update($dataNew);

        echo 'aqui';

        return redirect()->route('lista2');

        echo 'depois';

        return response()->json(['data' => $data]);

    }

Obs: no momento de ler essas imagens sempre coloque o “storage/” antes do arquivo, exemplo:

<img src="{{ url("storage/{$datas->image}") }}" alt="" style="width: 60px;" />

ORM:

**Create->([]):** Método para criar registros no banco;

**CreateMany->([]):** Método para criar vários registros no banco;

**Update->([]):** Método para atualizar no banco;

**updateOrCreate->([]):** Método para atualizar dados no banco, caso não exista os dados ele criar os dados. Exemplo:

$flight = Flight::updateOrCreate(

['departure' => 'Oakland', 'destination' => 'San Diego'],

['price' => 99, 'discounted' => 1]

);

**Upsert->([]):** Método para atualizar vários dados no banco. Exemplo:

Flight::upsert([

['departure' => 'Oakland', 'destination' => 'San Diego', 'price' => 99],

['departure' => 'Chicago', 'destination' => 'New York', 'price' => 150]

], ['departure', 'destination'], ['price']);

**delete->():** Método para deletar registro do banco;

**truncate->():** Método para deletar todos os registros do banco associado ao modelo usado;

**destroy->():** Método para deletar registros do banco pela chave primária;

**replicate():** Método para replicar registros para outro modelo. Exemplo:

$shipping = Address::create([

'type' => 'shipping',

'line\_1' => '123 Example Street',

'city' => 'Victorville',

'state' => 'CA',

'postcode' => '90001',

]);

$billing = $shipping->replicate()->fill([

'type' => 'billing'

]);

$billing->save();

Caso precise replicar alguns campos específicos, não todos, você pode passar os campos como array dentro da função replicate(). Exemplo:

$flight = $flight->replicate([

'last\_flown',

'last\_pilot\_id'

]);

**TRABALHANDO COM FILAS:**

Primeiro deve-se alterar a variável de ambiente **QUEUE\_CONNECTION** para **QUEUE\_CONNECTION=database.** Feito isso precisa criar a tabela que vai controlar a fila de processamento e a tabela que vai controlar os erros:

php artisan queue:table

php artisan queue:failed-table

php artisan migrate

Para ver o processamento da fila, pode usar o tinker **php artisan tinker.** Depois roda **php artisan queue:listen –tries=1** (Para processar a fila). Se for em produção deixa acionado o **php artisan queue:work.**