

Criação de Gerador de CRUDs para o Laravel

Usando somente commands para as versões 8, 9, 10 e 11



Sumário

Criação de Gerador de CRUDs para o Laravel.....	1
0 – Introdução.....	3
Motivação.....	3
Ambiente de desenvolvimento.....	5
Preparativos para a criação do gerador.....	6
1 - Artisan.....	9
2 - Argumentos.....	17
3 – Cores.....	20
4 – Mensagens.....	21
5 - Métodos auxiliares.....	24
6 – Exemplos.....	25
6.1 - Chamar o artisan via programação.....	25
6.2 – Criação de um gerador de CRUDs.....	25
6.3 – Seeders.....	29
7 – Hospedagem/Download.....	30
8 – Prompt.....	31
9 - Referências sobre comandos no Laravel.....	32

0 – Introdução

CRUDs são as mais básicas aplicações e praticamente todas as aplicações tem CRUDs. Portanto este conteúdo é útil não somente para quem usa laravel, pois com apenas um único comando podemos criar um aplicativo.

Mesmo para quem não seja usuário do laravel, vale a pena ter ele por perto, para criar um CRUD para alguma necessidade, apenas digitando

```
sh crud.sh
```

Motivação

Este é um dos tipos de programação que mais me motivam, especialmente por dois motivos, pela complexidade e pela grande utilidade. No caso, criar um código que facilita a vida do usuário, neste caso, do programador.

Só para dar uma ideia, um CRUD simples criado com este gerador tem 9 arquivos com 335 linhas. Sem contar que é um código testado e que não contém erros. Usando este gerador, com apenas um único comando criamos tudo isso.

Dificuldade pode ensinar

Se um programador precisa executar um projeto parecido com vários que ele já executou e este programador gosta de se sentir confortável então gostará da incumbência, pois o projeto não o tirará de sua zona de conforto.

Mas se um programador gosta muito de desafios e de aprender coisas novas e precisa executar um projeto complexo e se decidir executar irá precisar estudar muito e raciocinar também. Em consequência aprenderá algo novo.

Commands

Os commands no Laravel são verdadeiros canivetes suíços, pois com eles podemos fazer qualquer coisa. Veja que o exemplo deste e-book cria um gerador de CRUDs usando somente commands. Nos commands podemos usar qualquer um dos helpers do Laravel, também podemos usar qualquer função nativo PHP, como também podemos usar comandos do PHP que interagem com o sistema operacional.

Criar um gerador de CRUDs para o laravel

Com suporte para as versões 8, 9, 10 e 11

Testado no Linux Debian 12 e Windows 11 com WSL2 e Xampp

Será composto pelos comandos:

- ModelCreate
- ControllerCreate
- MigrationCreate
- ViewsCreate
- RouteCreate

E pelos respectivos stubs

Autor

Ribamar FS/ribafs - <https://ribamar.net.br/>

<https://ribafs.github.io/sobre/curriculo/>

Ambiente de desenvolvimento

Para a criação e testes dos commands precisarei de um ambiente de desenvolvimento que possa testar algumas versões do laravel nos dois sistemas operacionais mais populares, que são o Linux e o Windows. Nunca usei o Mac mas sei que pode usar com compatibilidade com Linux, visto sua origem ser o FreeBSD.

Meu desktop usa Linux com Debian 12 como sistema principal, o que me permite com facilidade ter o PHP 8.2. Assim posso testar as versões 9, 10 e 11 do laravel. Tenho apache, php, mariadb, node e cia instalados pelos pacotes com apt.

Quero testar também nas versões 7 e 8 do laravel e para isso preciso do PHP 7.3 ou 7.4, que já não mais é mantido pela equipe.

Tanto no Linux quanto no Windows (uso a versão 11 em dual boot) podemos usar o Xampp

<https://sourceforge.net/projects/xampp/files/XAMPP%20Windows/7.4.33/>

<https://sourceforge.net/projects/xampp/files/XAMPP%20Linux/7.4.33/>

No Windows testei e funcionu bem com a versão 8 do laravel, mas não na versão 7. Imagino que para funcionar com a 7 requer ajustes.

Também instalei o Xampp no linux. Neste caso, antes paro o apache e o mariadb.

No linux ele instala em

`/opt/lampp`

Onde encontramos um script

`xampp`

Digitando `./xampp` e enter veremos suas opções.

Requer sudo para executar os comandos. Ex para parar todos os serviços

`sudo ./xampp stop`

Para instalar uma versão diferente da atual

`composer create-project laravel/laravel:8.* crud8`

`composer create-project laravel/laravel:7.* crud7`

`sudo ./xampp stop`

Após reiniciar o ambiente volta com os pacotes da distribuição e não do Xampp.

Também podemos ter paralelamente no desktop um container Docker usando o Debian buster, que traz a versão 7.3 do PHP e permite rodar o laravel 8 e o 7.

Conclusão dos testes: o gerador suporta bem, sem modificações as versões 8, 9, 10 e 11 do laravel.

Preparativos para a criação do gerador

Primeiramente gosto de criar, mesmo que seja simples, um projeto/planejamento sobre o mesmo. Defini o que quero fazer e definir um roteiro para a criação. Depois eu crio uma pasta para o projeto e copio para ela tudo que eu já tiver sobre o assunto e que considero que vá me ajudar. Depois eu faço uma boa pesquisa procurando sobre o assunto e exemplos, que testo. Os que funcionam guardo na pasta.

Neste caso eu procurei por geradores de CRUDs para laravel. Entre os existentes, o que mais gosto é deste <https://github.com/sohelamin/crud-generator> que foi o que mais tenho usado. Cheguei a criar um fork dele e depois um repositório para esclarecer seu uso:

<https://github.com/ribafs/crud-generator-laravel?tab=readme-ov-file>

Neste acima criei um tutorial que explica e mostra um exemplo do gerador de API. Por que fiz isso? Porque no projeto original praticamente não existia informação sobre o assunto.

Os geradores que encontrei todos usam packages e os packages tem um sério problema, que é a compatibilidade com novas versões. Gostava do gerador citado na versão 9 do laravel, quando apareceu a 10 ele não mais funcionava e assim vai. Os pacotes ajudam muito por um lado mas tem este problema.

A forma que encontrei é mais simples e não tão profissional, feita somente com comandos, mas veja a compatibilidade, funciona com todas as últimas versões, da 8 até a 11. Isso é muito importante e vou continuar assim.

Resumindo sobre os geradores, não encontrei nenhum que me agradasse e compatível com a versão 11 do laravel. Então procurei apenas alguns exemplos de CRUDs para a versão 11 e alguns para a 10. Estou com dificuldade de implementar ou encontrar um exemplo de ACL para a versão 11, mas CRUDs da versão 10 funcionam bem na 11 e na 8 e 9.

Minha principal fonte de consulta para exemplos é a busca do Github

<https://github.com/search>

Faço o download de alguns exemplos, experimento em meu desktop e quando funcionam e eu aprovo, guardo na pasta do projeto. Para começara eu selecionei o exemplo mais simples entre os que encontrei. Depois ainda comparei cada arquivo para ficar com o mais simples, visto que criar geradores não é tarefa simples.

Quando o exemplo funciona eu pego os arquivos base do CRUD e guardo em uma pasta para experimentara em outras versões do laravel (exemplo):

app/Http/Controllers/ProductController.php

app/Models/Product.php

database/migrations/2023_09_01_055442_create_products_table.php

routes/web.php

resources/views/products(os arquivos de views)

Com estes arquivos eu crio os stubs (no vídeo mostrarei detalhes);

Então criarei cada um dos commands e ao final criarei um command para gerar um CRUD com apenas um único comando.

Então começarei a criar e executar os commands. Quando concluo um passo para o próximo.

Como crio um command?

- Crio uma pasta para os stubs em Commands/stubs e nela coloco todos os stubs
- Um exemplo simples de stub é o do Model, que tem a maior parte de seu código sem variar, apenas crio duas variáveis que serão pesquisadas pelo command e substituídas e veja como elas são anotadas no stub: `{{ Model }}`. Assim o command procura essa string e troca pelo nome da tabela no singular e com inicial maiúscula.
- Outra variável representa a fillable: `{{ fields }}`, que o command trocará por uma sequência com os nomes dos campos da tabela.

Mais detalhes veja o vídeo.

Limitações

Várias vezes eu comecei o command para a criação da migration e desisti, pois não estava conseguindo. Então o usuário deveria criar antes a tabela. Não gostei e não me conformei. Acabei adotando uma solução intermediária. Eu crio a parte principal da migration e, se for o caso, o usuário completa. Lembro que certa vez eu consegui bonitinho, mas perdi a solução. Esta solução usa o tipo string para todos os campos e ao final avisa ao usuário que altere os tipos se for o caso e ainda deixa um link para ajudá-lo.

Segunda versão

Depois de tudo concluído, testado e funcionando, lembrei que pode ser o momento de melhorar e já entregar o melhor CRUD encontrado. O mais trabalhoso dos commands é o primeiro, depois de criar o primeiro, testar e executar já temos um roteiro e os demais seguem com menor dificuldade.

Detalhes importantes a serem considerados:

- Controller – aqui existem mensagens que podem ser capturadas nas views, também podemos usar validação dos campos e até mensagens de erro customizadas para elas
- Migration – Importante usar o tipo adequado e com as constraints mais importantes: controlar o tamanho do campo, unique, required, etc
- Views – Estas são a interface com o usuário final e nelas é importante tornar o uso do aplicativo o mais amigável possível, agradável a vista e também seguro para validar todas as entradas.

Depois de ver um ótimo vídeo sobre a criação de geradores de CRUD para laravel (<https://www.youtube.com/watch?v=hcMyXKloY6k>) eu criei a versão 2, sem scripts e chamando apenas um commando.

<https://ribamar.net.br/down/laravel/commands/crud-gen2.zip>

Terceira versão

Depois disso resolvi criar um pacote para a versão 2. Então baixei o pacote

<https://github.com/ribafs/laravel-acl>

E adaptei para crud-gen, publicando em

<https://github.com/ribafs2/crud-gen>

Com ele basta executar o comando, ou melhor, seguir os comandos do README.md. Fica aí mais uma versão.

Quarta versão

Depois tive vontade de criar um gerador de APIs para laravel, mas não andei muito.

Tá tudo aqui:

<https://ribamar.net.br/laravel/gerador-de-cruds>

Quinta versão

Agora adicionei um seeder ao pacote para melhorar a saída

<https://github.com/ribafs2/crud-gen>

1 - Artisan

Lista de comandos

php artisan list

Help sobre um comando

php artisan model:create -h

php artisan help migrate

Criar command

php artisan make:command NomeCommand

Executar servidor web

php artisan serve

php artisan serve [--host [HOST]] [--port [PORT]] [--tries [TRIES]] [--no-reload]

Limpar o cache das rotas

php artisan route:cache

php artisan list which gives us all the commands, like this:

make:channel	Create a new channel class
make:command	Create a new Artisan command
make:controller	Create a new controller class
make:event	Create a new event class
make:exception	Create a new custom exception class
make:factory	Create a new model factory
make:job	Create a new job class
make:listener	Create a new event listener class
make:mail	Create a new email class
make:middleware	Create a new middleware class
make:migration	Create a new migration file
make:model	Create a new Eloquent model class
make:notification	Create a new notification class
make:observer	Create a new observer class
make:policy	Create a new policy class
make:provider	Create a new service provider class
make:request	Create a new form request class
make:resource	Create a new resource
make:rule	Create a new validation rule
make:seeder	Create a new seeder class
make:test	Create a new test class

Criar migration

php artisan make:migration create_projects_table

Criar seeder

```
php artisan make:seeder BooksTableSeeder
```

```
php artisan make:request StoreBlogPost
```

```
php artisan make:middleware CheckAge
```

```
php artisan make:policy PostPolicy
```

```
php artisan make:exception UserNotFoundException
```

```
php artisan make:resource PostResource
```

Exibir que tipo de variável de ambiente

```
php artisan env
```

Display an inspiring quote

```
php artisan inspire
```

Cache the framework bootstrap files: config, events, routes e views

```
php artisan optimize
```

Limpar o cache das configurações

```
php artisan config:clear
```

Clear cache

```
php artisan view:clear  
php artisan cache:clear  
php artisan route:cache  
php artisan route:clear  
php artisan optimize  
php artisan config:cache
```

```
//Clear route cache:  
Route::get('/route-cache', function() {  
    $exitCode = Artisan::call('route:cache');  
    return 'Routes cache cleared';  
});  
  
//Clear config cache:  
Route::get('/config-cache', function() {  
    $exitCode = Artisan::call('config:cache');  
    return 'Config cache cleared';  
});  
  
// Clear application cache:  
Route::get('/clear-cache', function() {
```

```
$exitCode = Artisan::call('cache:clear');
return 'Application cache cleared';
});

// Clear view cache:
Route::get('/view-clear', function() {
    $exitCode = Artisan::call('view:clear');
    return 'View cache cleared';
});

//Clear Cache facade value:
Route::get('/clear-cache', function() {
    $exitCode = Artisan::call('cache:clear');
    return '<h1>Cache facade value cleared</h1>';
});

//Reoptimized class loader:
Route::get('/optimize', function() {
    $exitCode = Artisan::call('optimize');
    return '<h1>Reoptimized class loader</h1>';
});

//Route cache:
Route::get('/route-cache', function() {
    $exitCode = Artisan::call('route:cache');
    return '<h1>Routes cached</h1>';
});

//Clear Route cache:
Route::get('/route-clear', function() {
    $exitCode = Artisan::call('route:clear');
    return '<h1>Route cache cleared</h1>';
});

//Clear View cache:
Route::get('/view-clear', function() {
    $exitCode = Artisan::call('view:clear');
    return '<h1>View cache cleared</h1>';
});

//Clear Config cache:
Route::get('/config-cache', function() {
    $exitCode = Artisan::call('config:cache');
    return '<h1>Clear Config cleared</h1>';
});
```

```
Route::get('/clear', function() {
    Artisan::call('cache:clear');
    Artisan::call('config:clear');
    Artisan::call('config:cache');
    Artisan::call('view:clear');
    return "Cleared!";
});

Route::get('/artisan/{cmd}', function($cmd) {
    $cmd = trim(str_replace("-", ":", $cmd));
    $validCommands = ['cache:clear', 'optimize', 'route:cache', 'route:clear', 'view:clear',
'config:cache'];
    if (in_array($cmd, $validCommands)) {
        Artisan::call($cmd);
        return "<h1>Ran Artisan command: {$cmd}</h1>";
    } else {
        return "<h1>Not valid Artisan command</h1>";
    }
});
```

In schedule function:

```
$schedule->command('clear:data')->dailyAt('07:00');

namespace App\Console\Commands\Admin;

use Illuminate\Console\Command;

class ClearAll extends Command
{
    protected $signature = 'traqza:clear-everything';

    protected $description = 'Clears routes, config, cache, views, compiled, and caches config.';

    public function __construct()
    {
        parent::__construct();
    }

    public function handle()
    {
        $validCommands = array('route:clear', 'config:clear', 'cache:clear', 'view:clear', 'clear-
compiled', 'config:cache');
        foreach ($validCommands as $cmd) {
            $this->call(" . $cmd . ");
        }
    }
}
```

```
php artisan traqza:clear-all

use Illuminate\Support\Facades\Artisan;

Artisan::call('some:command');
Artisan::call('list');
\Artisan::call('config:clear');
\Artisan::call('migrate');
dd(Artisan::output());
```

Auto-Completion

The anticipate method can be used to provide auto-completion for possible choices. The user can still provide any answer, regardless of the auto-completion hints:

```
$name = $this->anticipate('What is your name?', ['Taylor', 'Dayle']);
```

Saída

```
public function handle()
{
    $this->info('The command was successful!');
}

$this->error('Something went wrong!');
You may use the line method to display plain, uncolored text:
$this->line('Display this on the screen');

$this->newLine(3);
```

Tables

The table method makes it easy to correctly format multiple rows / columns of data. All you need to do is provide the column names and the data for the table and Laravel will automatically calculate the appropriate width and height of the table for you:

```
use App\Models\User;

$this->table(
    ['Name', 'Email'],
    User::all(['name', 'email'])->toArray()
);
```

Executando commands programaticamente

```
use Illuminate\Support\Facades\Artisan;

Route::post('/user/{user}/mail', function ($user) {
    $exitCode = Artisan::call('mail:send', [
        'user' => $user, '--queue' => 'default'
    ]);

    //
});
```

Stubs

Stub Customization

The Artisan console's make commands are used to create a variety of classes, such as controllers, jobs, migrations, and tests. These classes are generated using "stub" files that are populated with values based on your input. However, you may want to make small changes to files generated by Artisan. To accomplish this, you may use the stub:publish command to publish the most common stubs to your application so that you can customize them:

```
php artisan stub:publish
```

The published stubs will be located within a stubs directory in the root of your application. Any changes you make to these stubs will be reflected when you generate their corresponding classes using Artisan's make commands.

```
$defaultIndex = 1;

$name = $this->choice(
    'Qual o seu nome?',
    ['Ribamar', 'Tiago'],
    $defaultIndex,
    $maxAttempts = null,
    $allowMultipleSelections = true
);

$this->info('The command was successful!');
$this->error('Something went wrong!');
$this->line('Display this on the screen');
```

Tabelas

```
use App\Models\User;

$this->table(
    ['Name', 'Email'],
    User::all(['name', 'email'])->toArray()
);
```

Array de inputs

```
$input['name'] = $this->ask('What is your name?');  
$input['email'] = $this->ask('What is your email address?');  
$input['password'] = $this->ask('Provide your secret password?');  
$input['password'] = Hash::make($input['password']);  
User::create($input);  
  
$this->info("User created successfully");
```


2 - Argumentos

Valor default de Argumentos

```
protected $signature = 'alloted:shares {user} {age} [--difficulty=1] [--istest=4]';
```

Default de difficulty é 1

De istesté 4

Argumento opcional (use uma interrogação)

```
{field?}
```

Options

Os options, como os argumentos, são outra forma de entrada do usuário. São prefixados por dois hífen (--) quando são especificados na linha de comando

```
protected $signature = 'command:name  
    {argument}  
    {optionalArgument?}  
    {argumentWithDefault=default}  
    {--booleanOption}  
    {--optionWithValue=}  
    {--optionWithValueAndDefault=default}  
';
```

Exemplo de uso

```
do:thing {awesome}
```

O usuário roda

```
php artisan do:thing fantastic
```

No código

```
$this->argument('awesome'); // Deve retornar fantastic
```

```
jump:on {thing1} {thing2}
```

Usuário roda

```
php artisan jump:on rock boulder
```

`$this->argument();// deve retornar o array:`

```
[  
  'command': 'jump:on',  
  'thing1': 'rock',  
  'thing2': 'boulder'  
]
```

Recebendo um argumento

`$this->argument('nome_arg');`

Receber todos os argumentos

`$this->arguments();`

Recebdo um option

`$this->option('nome');`

Receber todos os options

`$this->options();`

Dica: mantenha sempre os argumentos cercados de chaves {}.

Valor default para Argumentos

`protected $signature = 'alloted:shares {user} {age} {--difficulty=1} {--istest=3}';`

`protected $signature = 'order:check {--order=7}'`

Testando

`php artisan order:check --order=7`

ou

`php artisan order:check --order 7`

`$orderNumber = $this->option('order'); // 7`

Argumento opcional

Para tornar um argumento opcional use uma interrogação

`{field?*`

Como lidar com arrays juntamente com argumentos simples?

```
// Optional argument...  
email:send {user?}
```

```
// Optional argument with default value...  
email:send {user=foo}
```

Argumento com array

```
email:send {user*}
```

Argumento com array e opcional

```
email:send {user?*}
```

3 – Cores

```
function handle()
{
    $this->line('<fg=red>A simple line.</>');
}
```

```
$this->line('<fg=red;bg=yellow>A simple line.</>');
```

```
$this->line('<fg=black>Black <fg=red>Red <fg=green>Green <fg=yellow>Yellow
    <fg=blue>Blue <fg=magenta>Magenta <fg=cyan>Cyan
    <fg=white;bg=black>White <fg=default;bg=black>Default</>');
```

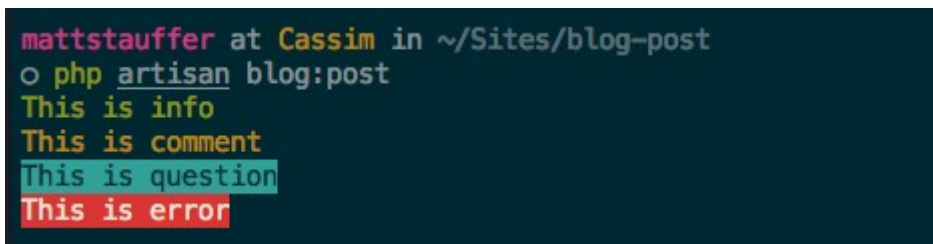
The following table lists the valid foreground and background colors:

Color	Value	Example
Black	black	fg=black;bg=black
Red	red	fg=red;bg=red
Green	green	fg=green;bg=green
Yellow	yellow	fg=yellow;bg=yellow
Blue	blue	fg=blue;bg=blue
Magenta	magenta	fg=magenta;bg=magenta
Cyan	cyan	fg=cyan;bg=cyan
White	white	fg=white;bg=white
Default	default	fg=default;bg=default

Options can also be used to change various aspects of the text as it is displayed. The following table lists each of the various options and a description of each:

bold
reverse
blink
underscore

```
$style = new OutputFormatterStyle('white', 'blue', ['bold']);
$this->output->getFormatter()->setStyle('bigBlue', $style);
```



```
mattstauffer at Cassim in ~/Sites/blog-post
o php artisan blog:post
This is info
This is comment
This is question
This is error
```

4 – Mensagens

Escrevendo mensagem no handle:

```
$this->writeFile('Frase desejada aqui');
```

Adicionando uma quebra de linha

```
$this->info(PHP_EOL);
```

Mensagens

```
$this->line("Some text");//Uma única linha
$this->info("Hey, watch this !");
$this->comment("Just a comment passing by");
$this->question("Why did you do that?");// Fica em fundo azul claro
$this->error("Ops, that should not happen.");
```

```
public function inlineInfo($string)
{
    $this->output->write("<info>$string</info>");// <info> fica com fontes verdes
}
```

```
$this->output->write('my inline message', true);
$this->output->write('my inline message continues', false);// Com false puxa a próxima linha para o final desta
```

```
$this->line("Some text");//Uma única linha
$this->info("Hey, watch this !");
$this->comment("Just a comment passing by");
$this->question("Why did you do that?");// Fica em fundo azul claro
$this->error("Ops, that should not happen.");
```

```
public function inlineInfo($string)
{
    $this->output->write("<info>$string</info>");// <info> fica com fontes verdes
}
```

```
$this->output->write('my inline message', true);
$this->output->write('my inline message continues', false);// Com false puxa a próxima linha para o final desta
```

Perguntas/Questions

Pára a execução e faz uma pergunta

```
$answer = $this->ask('What is your name?');

// Ask for sensitive information
$password = $this->secret('What is the password?');

// Choices
$name = $this->choice('What is your name?', ['Taylor', 'Dayle'], $default);

// Confirmation

if ($this->confirm('Is '.$name.' correct, do you wish to continue? [y|N]')) {
    //
}
```

Exemplo

```
$questions = [
    'easy' => [
        'How old are you?', 'What is the name of your mother?',
        'Do you have 3 parents?', 'Do you like Javascript?',
        'Do you know what is a JS promise?'
    ],
    'hard' => [
        'Why the sky is blue?', 'Can a kangaroo jump higher than a house?',
        'Do you think i am a bad father?', 'why the dinosaurs disappeared?',
        "why don't whales have gills?"
    ]
];

$questionsToAsk = $questions[$difficulty];
$answers = [];

foreach($questionsToAsk as $question){
    $answer = $this->ask($question);
    array_push($answers,$answer);
}

$this->info("Thanks for do the quiz in the console, your answers : ");

for($i = 0;$i <= (count($questionsToAsk) -1 );$i++){
    $this->line(($i + 1).) ' $answers[$i]';
}
```

Escrevendo mensagem no handle:

```
$this->writeFile('Frase desejada aqui');
```

Namespace Artisan

```
use Illuminate\Support\Facades\Artisan;
```

Executando comandos do sistema operacional linux

```
exec('comando');
```

```
exec('nohup php artisan some:command > /dev/null &');
```

How to run an artisan command from a controller

Apart from within another command, I am not really sure I can think of a good reason to do this. But if you really want to call a Laravel command from a controller (or model, etc.) then you can use `Artisan::call()`

```
Artisan::call('email:send', [  
    'user' => 1, '--queue' => 'default'  
]);
```

One interesting feature that I wasn't aware of until I just Googled this to get the right syntax is `Artisan::queue()`, which will process the command in the background (by your queue workers):

```
Route::get('/foo', function () {  
    Artisan::queue('email:send', [  
        'user' => 1, '--queue' => 'default'  
    ]);  
  
    //  
});
```

If you are calling a command from within another command you don't have to use the `Artisan::call` method - you can just do something like this:

```
public function handle()  
{  
    $this->call('email:send', [  
        'user' => 1, '--queue' => 'default'  
    ]);  
  
    //  
}
```

5 - Métodos auxiliares

```
private function clear(){  
    if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {  
        system('cls');  
    } else {  
        system('clear');  
    }  
}
```

Usando no handle():

```
$this->clear();
```


6 – Exemplos

6.1 - Chamar o artisan via programação

```
use Illuminate\Support\Facades\Artisan;

Artisan::call('some:command');
Artisan::call('list');
\Artisan::call('config:clear');
\Artisan::call('migrate');
dd(Artisan::output());
```

6.2 – Criação de um gerador de CRUDs

Será composto pelos comandos:

- ModelCreate
- ControllerCreate
- MigrationCreate
- ViewsCreate
- RouteCreate
- CRUDCreate

E somente por comandos.

```
php artisan make:command ModelCreate
php artisan make:command ControllerCreate
php artisan make:command MigrationCreate
php artisan make:command ViewsCreate // Este criará 5 views
php artisan make:command RouteCreate
php artisan make:command CRUDCreate
```

Quando executamos os comandos acima no raiz do aplicativo eles criam um arquivo em `app/Console/Commands`.

Criação dos stubs

Stubs são templates dos arquivos, usados para facilitar a criação dos commands.

Criarei os stubs na pasta
`app/Console/Commands/stubs`

```
stubs/views (aqui teremos create.stub, edit.stub, index.stub, layout.stub e show.stub)
stubs/controller.stub
stubs/migration.stub
stubs/model.stub
stubs/route.stub
```

Exemplificando stub

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class {{ Model }} extends Model
{
    use HasFactory;

    protected $fillable = [ {{ fields }} ];
}
```

Este stub terá apenas duas substituições:

- {{ Model }} - o commando substituirá pelo nome do model
- {{ fields }} - o comando substituirá pelo array de campos

Comando para a criação do Model

```
<?php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use Illuminate\Support\Str;

class ModelCreate extends Command
{
    protected $signature = 'model:create {tableName} {fields}';
    protected $description = 'Command create model';

    public function handle()
    {
        // Receber variáveis
        $tableName = $this->argument('tableName');
        $fields = $this->argument("fields");

        // Converter string em variável
        $fields = explode(',', $fields);

        // Adicionar delimitadores em cada campo
        $flds = "";
        foreach($fields as $field){
            $flds .= "".$field." ";
        }

        // Definir arquivo de stub
        $stub = app_path('Console/Commands/stubs/model.stub');
        $string = file_get_contents($stub);

        // Mudar tabela para inicial maiúscula
        $tableUc = Str::of($tableName)->ucfirst();
        // Mudar tabela para seu singular
        $tableUcSing = Str::of($tableUc)->singular();

        // Substituir {{ Model }} por $tableUcSing no stub
    }
}
```

```

$string = str_replace('{{ Model }}', $tableUcSing, $string);
$string = str_replace('{{ fields }}', $flds, $string);
$model = app_path('Models/'.$tableUcSing.'.php');
file_put_contents($model, $string);
$this->info(PHP_EOL.'Model criado'.PHP_EOL);

// php artisan model:create clients name,email
}
}

```

Como usar

Criar instalação limpa do laravel
 laravel new crud
 cd crud

No raiz do aplicativo

wget <https://ribamar.net.br/down/laravel/commands/crud-gen.zip>

unzip -o crud-gen.zip

sh crud.sh clients name,email Cadastro_de_Clientes

php artisan migrate --seed

php artisan serve

<http://127.0.0.1:8000/>

Cadastro_de_Clientes

Create New

ID	Name	Email	Action
1	Nakia Conn	damon10@gulgowski.com	Show Edit Delete
2	Jennyfer Kihn	korbin87@damore.com	Show Edit Delete
3	Connie Lehner	williamson.kelsi@hotmail.com	Show Edit Delete
4	Margarette Dooley Sr.	mwest@gmail.com	Show Edit Delete
5	Mckenna Stroman	arunte@schuppe.com	Show Edit Delete

« Previous [Next »](#)
 Showing 1 to 5 of 101 results

Problema contornado

Quando executo o comando

php artisan crud:create products name,price "Cadastro de Produtos"
 Ele não reconhece o título por ter espaços e não executa.

Mas assim funciona bem:

```
php artisan crud:create products name,price  
php artisan views/:create products name,price "Cadastro de Produtos"
```

Ou seja, primeiro o CRUD, depois as views, então funciona bem.

Então temos três alternativas para executar o criador de CRUDs:

- command com todos os comandos usando

Artisan::call(), mas assim não podemos usar espaços no título do CRUD

- Usando os comandos com Artisan::call() e executando as views em separado, mas assim precisamos de dois commands

- Criando um script crud.sh no linux ou crud.bat no windows no raiz do aplicativo, que executa todos os comandos com php artisan command:create args. Esta me parece a forma mais flexível, pois executamos apenas um comando e podemos usar espaços no título.

No linux
sh crud.sh

No Windows
.\crud.bat

Solução adotada

Adotei o uso do script.

Exemplo de uso – este exemplo é para a tabela clients, campos name e e-mail, title "Cadastro de Clientes". Faça adaptações para outros tipos, especialmente no seeder

```
composer create-project laravel/laravel crud  
cd crud  
wget https://ribamar.net.br/download/laravel/commands/crud-gen.zip  
unzip -o crud-gen.zip  
sh crud.sh clients name,email Cadastro_de_Clientes (No linux, mac ou WSL)  
php artisan migrate --seed  
php artisan serve  
http://127.0.0.1:8000
```

Observe que o route não é o padrão do laravel. Foi alterado para já abrir a view index por padrão.

Encontrará uma grande quantidade de exemplos no site

<https://ribamar.net.br/download/laravel/commands/Exemplos.zip>

6.3 – Seeders

Adicionei um seeder apenas como exemplo a ser usado e adaptado para sua tabela e assim conseguimos melhor visualização das virws. Não criei um command para o seeder.

7 – Hospedagem/Download

No meu site (aqui atualizo com mais frequência)

<https://ribamar.net.br/laravel/gerador-de-cruds>

No Github

<https://github.com/ribafs2/laravel-crud-generator>

8 – Prompt

A partir do laravel 10 foi lançado um excelente recurso para a interface CLI do Linux e Mac, que também pode ser usado no Windows com o WSL.

Segue um arquivo texto com quase todos os exemplos da documentação oficial

<https://laravel.com/docs/11.x/prompts#main-content>

<https://ribamar.net.br/down/laravel/commands/prompts.txt>

9 - Referências sobre comandos no Laravel

<https://mattstauffer.com/blog/advanced-input-output-with-artisan-commands-tables-and-progress-bars-in-laravel-5.1/>

<https://medium.com/@flyinglucas/laravel-criando-comandos-904454cae849>

<https://appdividend.com/2017/09/21/laravel-5-5-artisan-console-tutorial/>

<https://ourcodeworld.com/articles/read/248/how-to-create-a-custom-console-command-artisan-for-laravel-5-3>

<https://laravelpackageboilerplate.com/#/>