Laravel Artisan: صندوق أدواتك الاحترافي لسطر الأوامر

دليل شامل لأهم الأوامر مع أمثلة عملية

ما هو Artisan؟ شريكك في الإنجاز

شرح موجز: Artisan هو واجهة سطر الأوامر (CLI) المدمجة مع Laravel.

الهدف الأساسي: تم تصميمه لأتمتة المهام المتكررة والمعقدة، وتسريع عملية بناء مكونات التطبيق، وإدارة دورة حياة التطبيق بكل سهولة وكفاءة.

أوامر الاستكشاف الأساسية

- **php artisan list**: لاستعراض كل الأوامر المتاحة. هذا الأمر يفتح لك "صندوق الأدوات" بالكامل.
- **php artisan --help <command_name>**: للحصول على شرح تفصيلي لأي أمر، بما في ذلك الخيارات والـ arguments المتاحة له.

```
# استعراض كل الأوامر
php artisan list

# الحصول على مساعدة لأمر الـ migration
php artisan help migrate
```

```
Laravel Framework 12.x.x

Usage:
  command [options] [arguments]

Options:
  -h, --help       Display help for the given command.
  ...

Available commands:
  about            Displays information about the application
  ...
cache
  cache:clear      Flush the application cache
  cache:table      Create a migration for the cache database table
config
  config:cache     Create a cache file for faster configuration loading
  config:clear     Remove the configuration cache file
db
  db:seed          Seed the database with records
  ...
make
  make:command     Create a new Artisan command
  make:controller  Create a new controller class
  make:model       Create a new Eloquent model class
migrate
  migrate          Run the database migrations
  migrate:fresh    Drop all tables and re-run all migrations
  ...
queue
  queue:failed     List all of the failed queue jobs
  queue:flush      Flush all of the failed queue jobs
  queue:work       Start processing jobs on the queue as a daemon
...
```

إدارة التطبيق والبيئة المحيطة

أوامر للتحكم في حالة التطبيق وبيئته، من تشغيل سيرفر التطوير المحلي إلى وضع الصيانة.

`php artisan about`

الشرح: يعرض نظرة شاملة ومفصلة عن إعدادات التطبيق والبيئة، مثل تعريفات الـ Cache والـ Drivers المستخدمة وإعدادات PHP.

مثال: `php artisan about`

`php artisan down` / `php artisan up`

الشرح: `down` يضع التطبيق في وضع الصيانة، حيث يعرض صفحة 503 لجميع الطلبات. `up` يعيد التطبيق للعمل.

```
# تفعيل وضع الصيانة
php artisan down

# إيقاف وضع الصيانة
php artisan up
```

`php artisan serve`

الشرح: يقوم بتشغيل سيرفر تطوير محلي باستخدام سيرفر PHP المدمج.

مثال: `php artisan serve` (يبدأ السيرفر عادةً على `http://127.0.0.1:8000`).

التسريع والأداء: أوامر الـ Cache

الكاشينج هو أحد أهم عوامل تحسين أداء التطبيق في بيئة الإنتاج. Artisan يوفر مجموعة أوامر قوية لإدارة وتسريع مختلف أجزاء التطبيق.

php artisan config:cache

يدمج كل ملفات الإعدادات في ملف واحد لتقليل زمن تحميلها.

```
php artisan config:cache
```

php artisan route:cache

يقوم بتسجيل كل مسارات (Routes) التطبيق في ملف واحد لتسريع عملية توجيه الطلبات.

```
php artisan route:cache
```

php artisan view:clear

يمسح ملفات الـ Views المترجمة (compiled) لإجبار Blade على إعادة ترجمتها.

```
php artisan view:clear
```

php artisan cache:clear

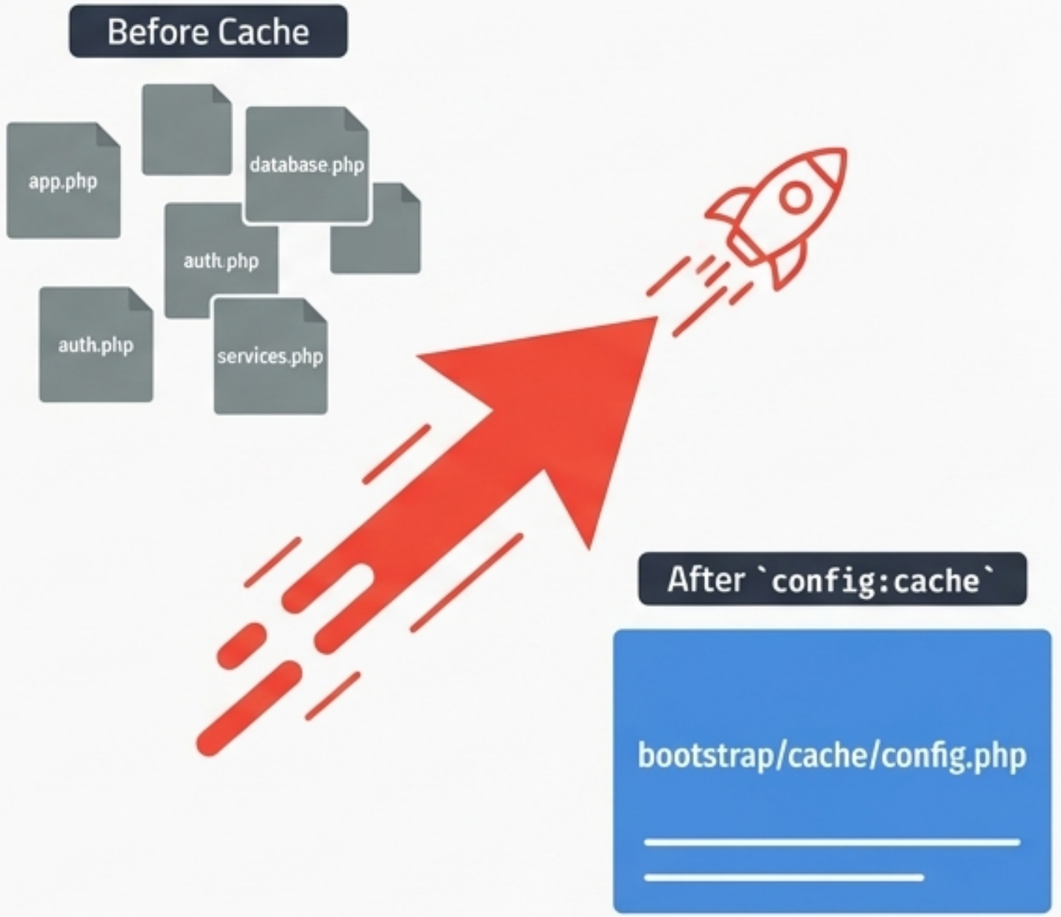
يمسح الكاش الخاص بالتطبيق (Application Cache) الذي تم تخزينه باستخدام `facade` `Cache`.

```
php artisan cache:clear
```

(Pro Tip) - php artisan optimize

أمر شامل يقوم بتنفيذ `config:cache` و `route:cache` معاً. (في الإصدارات الأحدث، تم تبسيط هذا الأمر، ولكنه لا يزال مفهوماً قوياً).

```
php artisan optimize
```



التوليد التلقائي للكود (الجزء الأول): الأساسيات

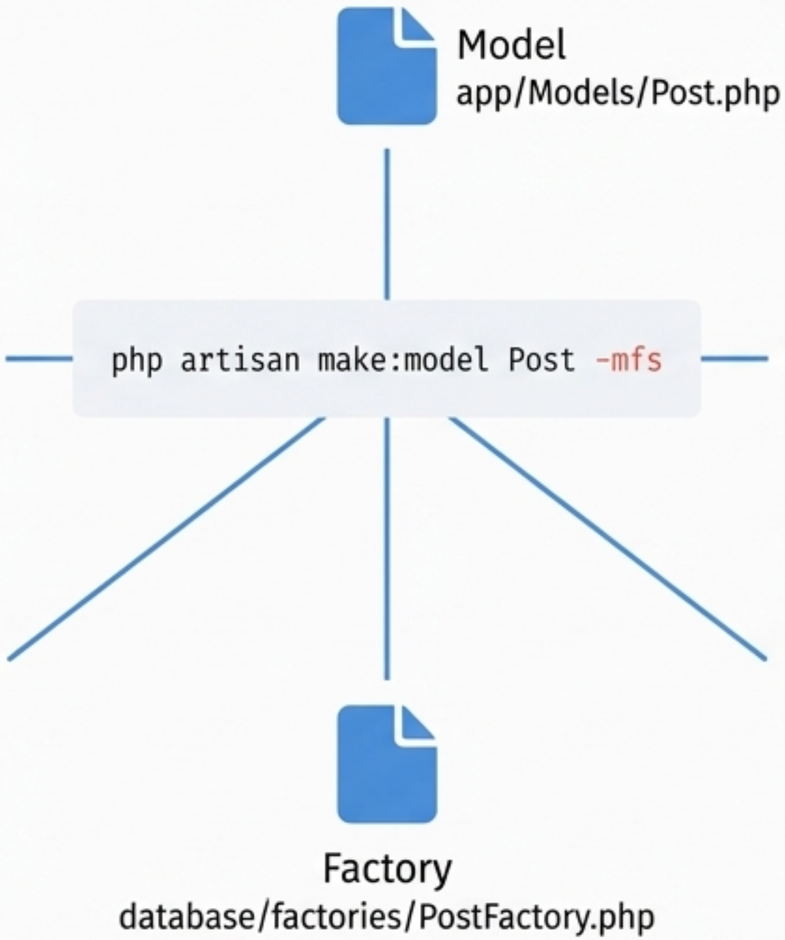
أوامر `make` هي قلب Artisan، تمكنك من بناء هيكل التطبيق بسرعة واتساق، مما يضمن اتباع أفضل الممارسات.

`php artisan make:model <ModelName> -mfs`

أمر احترافي لإنشاء Model جديد مع كل الملفات المرتبطة به في خطوة واحدة.

- `-m` لإنشاء ملف Migration جديد للـ Model.
- `-f` لإنشاء ملف Factory جديد للـ Model.
- `-s` لإنشاء ملف Seeder جديد للـ Model.

```
php artisan make:model Post -mfs
```



`php artisan make:controller <ControllerName> --resource`

لإنشاء Controller من نوع Resource، والذي يحتوي على كل الدوال الأساسية للتعامل مع الـ Model (index, create, store, show, edit, update, destroy).

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

التوليد التلقائي للكود (الجزء الثاني): منطق التطبيق

توسيع استخدام أوامر `make` لتشمل المكونات التي تغلف منطق العمل وقواعد التحقق والاختبارات.

`php artisan make:job <JobName>`

لإنشاء Job جديد لتنفيذ المهام في الخلفية (background processing) عبر نظام الـ Queues.

```
php artisan make:job ProcessPodcast
```

`php artisan make:request <RequestName>`

لإنشاء Form Request مخصص يحتوي على قواعد التحقق (validation rules) والـ authorization.

```
php artisan make:request StorePostRequest
```

`php artisan make:notification <NotificationName>`

لإنشاء فئة إشعار (Notification) لإرسال التنبيهات عبر قنوات مختلفة (إيميل، SMS، Slack).

```
php artisan make:notification NewPostPublished
```

`php artisan make:test <TestName>`

لإنشاء ملف اختبار جديد في مجلد `tests`.

```
php artisan make:test UserTest
```

إدارة قاعدة البيانات: الـ Migrations

الـ Migrations هي بمثابة 'Version Control' لقاعدة بياناتك. تسمح لك بتعريف وتعديل ومشاركة بنية الجداول مع فريقك بشكل منظم وآمن.

`php artisan make:migration <migration_name>`

لإنشاء ملف migration جديد فارغ. يفضل أن يكون الاسم معبراً عن التغيير.

```
php artisan make:migration create_posts_table
```

`php artisan migrate`

لتنفيذ جميع الـ migrations التي لم يتم تنفيذها بعد.

```
php artisan migrate
```

`php artisan migrate:rollback --step=1`

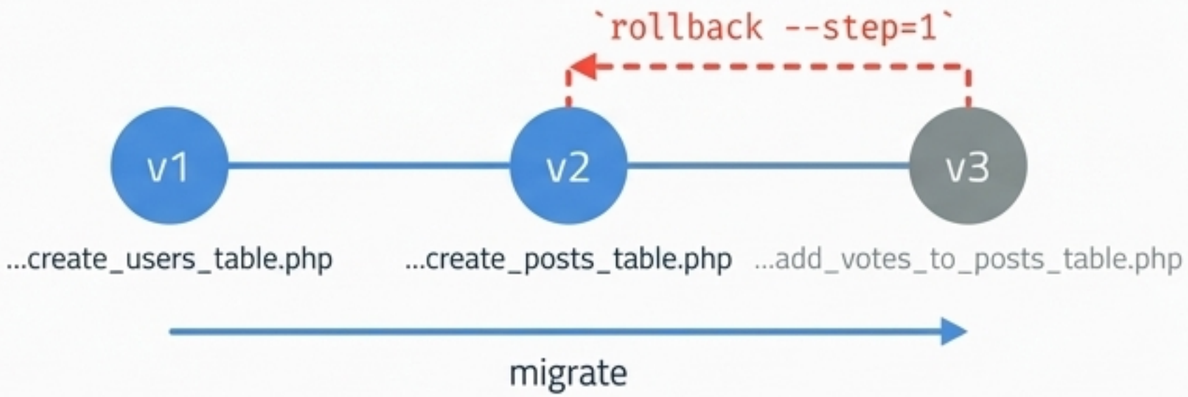
للتراجع عن آخر دفعة (batch) من الـ migrations.
الـ flag --step=1 يتراجع عن آخر migration فقط.

```
php artisan migrate:rollback --step=1
```

`php artisan migrate:fresh --seed`

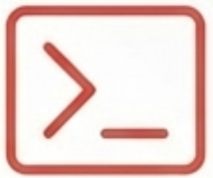
أمر قوي جداً في بيئة التطوير. يقوم بحذف كل الجداول وإعادة بنائها من جديد عبر تنفيذ كل الـ migrations، ثم يقوم بتشغيل الـ Seeders لملء البيانات.

```
php artisan migrate:fresh --seed
```



ملء البيانات والتفاعل المباشر

أدوات لملء قاعدة البيانات ببيانات وهمية للاختبار والتطوير، والتفاعل المباشر مع مكونات التطبيق عبر سطر الأوامر.

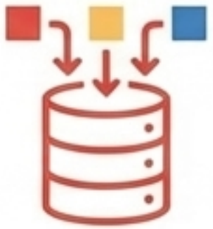


`php artisan tinker`

أداة Read-Eval-Print LoPL) قوية تتيح لك التفاعل المباشر مع تطبيق Laravel. يمكنك يمكنك تنفيذ أي كود PHP في سياق التطبيق.

```
$ php artisan tinker
Psy Shell v0.11.x (PHP 8.2.x — CLI) by Justin Hileman
------------------------------------------------
>>> User::factory()->count(10)->create();
 => Illuminate\Database\Eloquent\Collection {#...}

>>> User::count();
 => 10
>>>
```



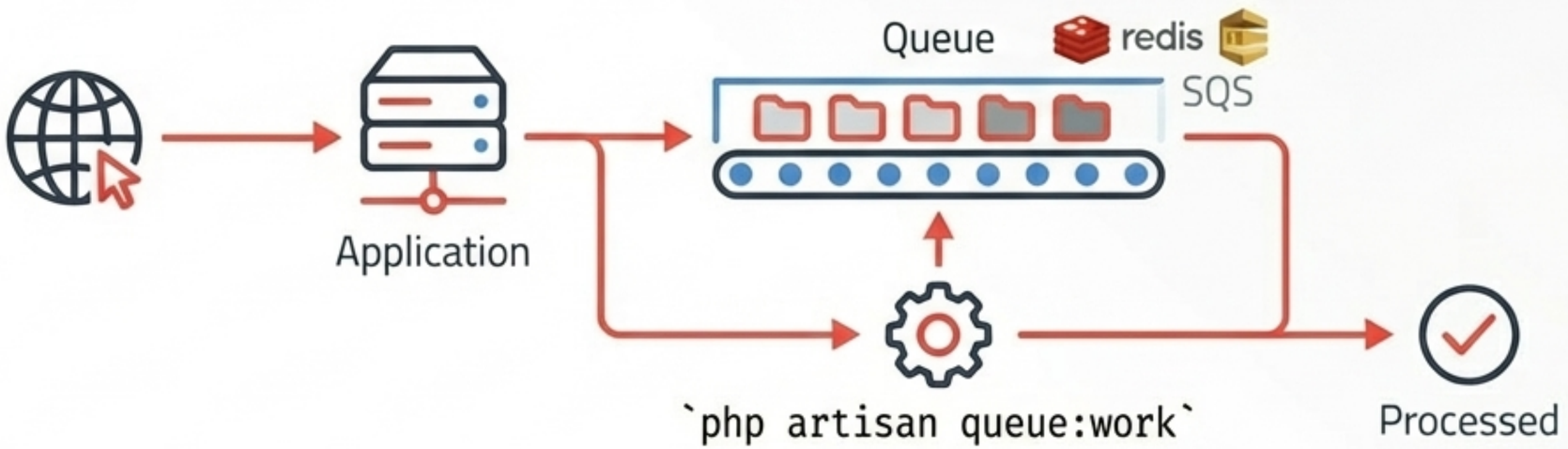
`php artisan db:seed`

لتشغيل كل فئات الـ Seeders لملء الجداول ببيانات أولية أو تجريبية.

```
php artisan db:seed
```

إدارة مهام الخلفية: الـ Queues

نظام الـ Queues يسمح بتأجيل تنفيذ المهام التي تستغرق وقتاً طويلاً لتتم في الخلفية، مما يحسن من سرعة استجابة التطبيق للمستخدم.



`php artisan queue:work`

لبدء worker يقوم بمعالجة المهام من الـ queue. يظل هذا الأمر يعمل في الخلفية.

`php artisan queue:failed`

لعرض قائمة بالمهام التي فشلت أثناء المعالجة.

`php artisan queue:retry <id|all>`

لإعادة محاولة تنفيذ مهمة فاشلة معينة (عبر الـ ID) أو كل المهام الفاشلة.

`php artisan queue:flush`

لحذف كل المهام من جدول المهام الفاشلة.

أوامر متقدمة: القوة الكاملة في أمر واحد

اجمع قوة العديد من أوامر `make` في أمر واحد لتوليد الهيكل الكامل لأي Resource في تطبيقك.

The Power Command: `php artisan make:model <ModelName> -a`

الـ `a flag` (أو `--all`) هو اختصار قوي يقوم بإنشاء كل ما يتعلق بالـ Model دفعة واحدة.

ماذا ينشئ هذا الأمر؟

- **Model**: `app/Models/ModelName.php`
- **Migration**:`database/migrations`
- **Factory**: `database/factories/ModelNameFactory.php`
- **Seeder**:`database/seeders/ModelNameSeeder.php`
- **Resource Controller**: `app/Http/Controllers/ModelNameController.php`
- **Policy**: `app/Policies/ModelNamePolicy.php` (حسب الإصدار)

1. `app/Models/ModelName.php`
2. ملف إنشاء الجدول في `database/migrations`
3. `database/factories/ModelNameFactory.php`
4. `database/seeders/ModelNameSeeder.php`
5. مع دوال الـ CRUD.
6. `app/Policies/ModelNamePolicy.php` تحدا.php

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

صناعة أدواتك الخاصة: إنشاء أوامر مخصصة

Artisan ليس صندوق أدوات مغلق. يمكنك توسيعه بسهولة عبر إنشاء أوامرك الخاصة لأتمتة المهام الفريدة لمشروعك.

The Creation Command

```
php artisan make:command <CommandName>
```

هذا الأمر يقوم بإنشاء ملف Class جديد للأمر في `app/Console/Commands`.

```
php artisan make:command CleanupOldUsers
```

Structure of a Command Class

- `protected $signature` تعريف اسم الأمر والـ arguments/options.
- `protected $description` وصف الأمر الذي يظهر في `php artisan list`.
- `public function handle()` المكان الذي يتم فيه تنفيذ منطق الأمر.

```php
// app/Console/Commands/CleanupOldUsers.php

class CleanupOldUsers extends Command
{
    // highlight-start
    protected $signature = 'app:cleanup-old-users {--days=30}';
    // highlight-end

    // highlight-start
    protected $description = 'Deletes user accounts older than a specified number of days.';
    // highlight-end

    // highlight-start
    public function handle(): void
    {
        $days = $this->option('days');
        $cutoffDate = now()->subDays($days);

        // Your logic to delete old users...

        $this->info("Successfully cleaned up users older than {$days} days.");
    }
    // highlight-end
}
```

تخصيص أوامرك: المدخلات (Arguments & Options)

اجعل أوامرك المخصصة ديناميكية وقابلة للتكوين عبر تعريف الـ arguments والـ options. يتم تعريف كل ذلك في متغير `$signature`.

$signature

```
'mail:send {user : The ID of the user} {--Q|queue=default : The queue to send on}'
```

Command Name | Argument | Option

Arguments

- Required: {user} → php artisan command:name 1
- Optional: {user?} → php artisan command:name (user is null)
- Optional with Default: {user=1} → php artisan command:name (user defaults to 1)
- Array: {user*} → php artisan command:name 1 2 3

Options

- → Boolean Flag: {--force} → php artisan command:name --force (returns true/false)
- → Value Required: {--queue=} → php artisan command:name --queue=default
- → Value with Default: {--queue=default} → php artisan command:name (queue defaults to 'default')

Accessing Input in `handle()` method

```php
$this->argument('user');
$this->option('queue');
```

التفاعل مع المستخدم: المدخلات والمخرجات

الأوامر الاحترافية لا تقوم بتنفيذ المهام فقط، بل تتواصل مع المستخدم بوضوح. Artisan يوفر طرقاً سهلة لعرض المخرجات الملونة، والجداول، وطلب المدخلات.

```
[INFO] Success message in green.
[COMMENT] Comment message in yellow.
[QUESTION] Question message in cyan.
[ERROR] Error message in red.

Standard uncolored message.

+----------+------------------+
| Header 1 | Header 2         |
+----------+------------------+
| Value A  | Description for A |
| Value B  | Description for B |
+----------+------------------+

Do you wish to continue? [y/N]
```

Output Methods

```
$this->info('...')
$this->comment('...')
$this->question('...')
$this->error('...')
$this->line('...')
```

Displaying Data as Tables

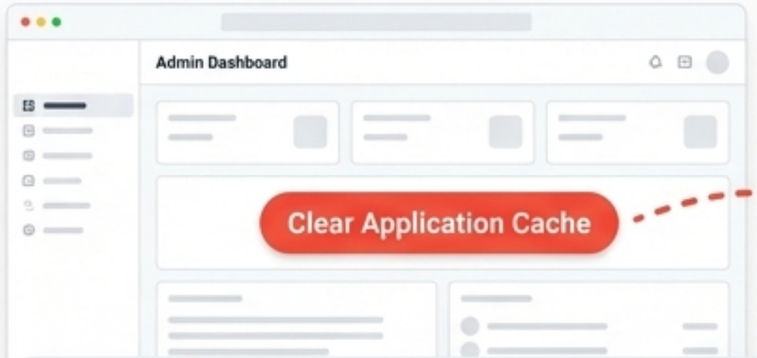
```
$this->table(['Header 1', 'Header 2'], $dataArray);
```

Input Methods

```
$name = $this->ask('What is your name?');
$password = $this->secret('Enter your password:');
if ($this->confirm('Do you wish to continue?')) { … }
```

التشغيل البرمجي: Artisan داخل الكود

يمكنك استدعاء أوامر Artisan برمجياً من أي مكان في تطبيقك، مثل الـ Controllers أو الـ Jobs. هذا يتيح لك إعادة إعادة استخدام منطق الأوامر ودمجها في واجهات المستخدم الإدارية.



Use Case Example

```
// Inside a Controller method for an admin panel

public function clearApplicationCache()
{
    // Execute the cache:clear command
    Artisan::call('cache:clear');

    return back()->with('message', 'Application cache has been cleared!');
}
```

Key Methods (`Artisan` Facade)

Artisan::call('command:name', [args])

الشرح: لتنفيذ أمر بشكل متزامن (synchronously).

مثال:

```
Artisan::call('cache:clear');
```

Artisan::queue('command:name', [args])

الشرح: لوضع تنفيذ الأمر في الـ queue ليتم معالجته في الخلفية.

مثال:

```
Artisan::queue('app:process-report',
['--user' => 1]);
```

صندوق أدواتك الآن مكتمل

لقد استكشفت كل ركن من أركان صندوق أدوات Artisan. من إدارة التطبيق وتوليد الكود، إلى التعامل مع قاعدة البيانات وأتمتة المهام المعقدة، وحتى صناعة أدواتك الخاصة. أنت الآن تملك المعرفة اللازمة لتسريع وتيرة تطويرك وبناء تطبيقات Laravel باحترافية وكفاءة.

ملخص القدرات

- 🔧 الإدارة والتحكم: `serve`, `down`, `cache:clear`
- 🛠️ التوليد والبناء: `make:model -a`, `make:controller`
- 🗄️ قاعدة البيانات: `migrate:fresh`, `db:seed`, `tinker`
- 💻 الأتمتة والمهام المتقدمة: `queue:work`, `make:command`

الآن، اذهب وابنِ شيئًا رائعًا!

لمزيد من التفاصيل، راجع التوثيق الرسمي لـ Laravel Artisan Console