


 [LaravelDaily](#) / [laravel-tips](#)

Awesome tips for Laravel

☆ 1.7k stars    🍴 327 forks

 Star Watch ▾

&lt;&gt; Code

 Issues Pull requests Actions Security Insights master ▾

...



PovilasKorop Merge pull request #18 from idemonbd/patch-1 ...

20 days ago

 98[View code](#)

# Laravel Tips

Awesome Laravel tips and tricks for all artisans. PR and ideas are welcome!

An idea by [PovilasKorop](#) and [MarceauKa](#).

Update 06 July 2021: Currently there are **129 tips** divided into 14 sections.

## Table of Contents

- [DB Models and Eloquent](#) (31 tips)
- [Models Relations](#) (23 tips)
- [Migrations](#) (8 tips)
- [Views](#) (8 tips)
- [Routing](#) (13 tips)
- [Validation](#) (7 tips)
- [Collections](#) (6 tips)
- [Auth](#) (5 tips)
- [Mail](#) (4 tips)

- [Artisan](#) (5 tips)
- [Factories](#) (2 tips)
- [Log and debug](#) (2 tips)
- [API](#) (2 tips)
- [Other](#) (13 tips)

## DB Models and Eloquent

---

[↑](#) Go to top [→](#) Next (Models Relations)

- [Eloquent where date methods](#)
- [Increments and decrements](#)
- [No timestamp columns](#)
- [Soft-deletes: multiple restore](#)
- [Model all: columns](#)
- [To Fail or not to Fail](#)
- [Column name change](#)
- [Map query results](#)
- [Change Default Timestamp Fields](#)
- [Quick Order by created\\_at](#)
- [Automatic Column Value When Creating Records](#)
- [DB Raw Query Calculations Run Faster](#)
- [More than One Scope](#)
- [No Need to Convert Carbon](#)
- [Grouping by First Letter](#)
- [Never Update the Column](#)
- [Find Many](#)
- [Find by Key](#)
- [Use UUID instead of auto-increment](#)
- [Sub-selects in Laravel Way](#)
- [Hide Some Columns](#)
- [Exact DB Error](#)
- [Soft-Deletes with Query Builder](#)
- [Good Old SQL Query](#)
- [Use DB Transactions](#)
- [Update or Create](#)

- [Forget Cache on Save](#)
- [Change Format of Created\\_at and Updated\\_at](#)
- [Storing Array Type into JSON](#)
- [Make a Copy of the Model](#)
- [Reduce Memory](#)

## Eloquent where date methods

In Eloquent, check the date with functions `whereDay()` , `whereMonth()` , `whereYear()` , `whereDate()` and `whereTime()` .

```
$products = Product::whereDate('created_at', '2018-01-31')->get();
$products = Product::whereMonth('created_at', '12')->get();
$products = Product::whereDay('created_at', '31')->get();
$products = Product::whereYear('created_at', date('Y'))->get();
$products = Product::whereTime('created_at', '=', '14:13:58')->get();
```

## Increments and decrements

If you want to increment some DB column in some table, just use `increment()` function. Oh, and you can increment not only by 1, but also by some number, like 50.

```
Post::find($post_id)->increment('view_count');
User::find($user_id)->increment('points', 50);
```

## No timestamp columns

If your DB table doesn't contain timestamp fields `created_at` and `updated_at` , you can specify that Eloquent model wouldn't use them, with `$timestamps = false` property.

```
class Company extends Model
{
    public $timestamps = false;
}
```

## Soft-deletes: multiple restore

When using soft-deletes, you can restore multiple rows in one sentence.

```
Post::onlyTrashed()->where('author_id', 1)->restore();
```

## Model all: columns

When calling Eloquent's `Model::all()`, you can specify which columns to return.

```
$users = User::all(['id', 'name', 'email']);
```

## To Fail or not to Fail

In addition to `findOrFail()`, there's also Eloquent method `firstOrFail()` which will return 404 page if no records for query are found.

```
$user = User::where('email', 'povilas@laraveldaily.com')->firstOrFail();
```

## Column name change

In Eloquent Query Builder, you can specify "as" to return any column with a different name, just like in plain SQL query.

```
$users = DB::table('users')->select('name', 'email as user_email')->get();
```

## Map query results

After Eloquent query you can modify rows by using `map()` function in Collections.

```
$users = User::where('role_id', 1)->get()->map(function (User $user) {  
    $user->some_column = some_function($user);  
    return $user;  
});
```

## Change Default Timestamp Fields

What if you're working with non-Laravel database and your timestamp columns are named differently? Maybe, you have `create_time` and `update_time`. Luckily, you can specify them in the model, too:

```
class Role extends Model  
{  
    const CREATED_AT = 'create_time';
```

```
const UPDATED_AT = 'update_time';  
}
```

## Quick Order by created\_at

Instead of:

```
User::orderBy('created_at', 'desc')->get();
```

You can do it quicker:

```
User::latest()->get();
```

By default, `latest()` will order by `created_at`.

There is an opposite method `oldest()` which would order by `created_at` ascending:

```
User::oldest()->get();
```

Also, you can specify another column to order by. For example, if you want to use `updated_at`, you can do this:

```
$lastUpdatedUser = User::latest('updated_at')->first();
```

## Automatic Column Value When Creating Records

If you want to generate some DB column value when creating record, add it to model's `boot()` method. For example, if you have a field "position" and want to assign the next available position to the new record (like `Country::max('position') + 1`), do this:

```
class Country extends Model {  
    protected static function boot()  
    {  
        parent::boot();  
  
        Country::creating(function($model) {  
            $model->position = Country::max('position') + 1;  
        });  
    }  
}
```

## DB Raw Query Calculations Run Faster

Use SQL raw queries like `whereRaw()` method, to make some DB-specific calculations directly in query, and not in Laravel, usually the result will be faster. Like, if you want to get users that were active 30+ days after their registration, here's the code:

```
User::where('active', 1)
    ->whereRaw('TIMESTAMPDIFF(DAY, created_at, updated_at) > ?', 30)
    ->get();
```

## More than One Scope

You can combine and chain Query Scopes in Eloquent, using more than one scope in a query.

Model:

```
public function scopeActive($query) {
    return $query->where('active', 1);
}

public function scopeRegisteredWithinDays($query, $days) {
    return $query->where('created_at', '>=', now()->subDays($days));
}
```

Some Controller:

```
$users = User::registeredWithinDays(30)->active()->get();
```

## No Need to Convert Carbon

If you're performing `whereDate()` and check today's records, you can use Carbon's `now()` and it will automatically be transformed to date. No need to do `->toDateString()`.

```
// Instead of
$todayUsers = User::whereDate('created_at', now()->toDateString())->get();
// No need to convert, just use now()
$todayUsers = User::whereDate('created_at', now())->get();
```

## Grouping by First Letter

You can group Eloquent results by any custom condition, here's how to group by first letter of user's name:

```
$users = User::all()->groupBy(function($item) {  
    return $item->name[0];  
});
```

## Never Update the Column

If you have DB column which you want to be set only once and never updated again, you can set that restriction on Eloquent Model, with a mutator:

```
class User extends Model  
{  
    public function setEmailAttribute($value)  
    {  
        if ($this->email) {  
            return;  
        }  
  
        $this->attributes['email'] = $value;  
    }  
}
```

## Find Many

Eloquent method `find()` may accept multiple parameters, and then it returns a Collection of all records found, not just one Model:

```
// Will return Eloquent Model  
$user = User::find(1);  
// Will return Eloquent Collection  
$users = User::find([1,2,3]);
```

## Find by Key

You can also find multiple records with `whereKey()` method which takes care of which field is exactly your primary key ( `id` is the default but you may override it in Eloquent model):

```
$users = User::whereKey([1,2,3])->get();
```

## Use UUID instead of auto-increment

You don't want to use auto incrementing ID in your model?

Migration:

```
Schema::create('users', function (Blueprint $table) {  
    // $table->increments('id');  
    $table->uuid('id')->unique();  
});
```

Model:

```
class User extends Model  
{  
    public $incrementing = false;  
    protected $keyType = 'string';  
  
    protected static function boot()  
    {  
        parent::boot();  
  
        User::creating(function ($model) {  
            $model->setId();  
        });  
    }  
  
    public function setId()  
    {  
        $this->attributes['id'] = Str::uuid();  
    }  
}
```

## Sub-selects in Laravel Way

From Laravel 6, you can use addSelect() in Eloquent statement, and do some calculation to that added column.

```
return Destination::addSelect(['last_flight' => Flight::select('name')  
    ->whereColumn('destination_id', 'destinations.id')  
    ->orderBy('arrived_at', 'desc')  
    ->limit(1)  
)->get();
```



## Hide Some Columns

When doing Eloquent query, if you want to hide specific field from being returned, one of the quickest ways is to add `->makeHidden()` on Collection result.

```
$users = User::all()->makeHidden(['email_verified_at', 'deleted_at']);
```

## Exact DB Error

If you want to catch Eloquent Query exceptions, use specific `QueryException` instead default Exception class, and you will be able to get the exact SQL code of the error.

```
try {
    // Some Eloquent/SQL statement
} catch (\Illuminate\Database\QueryException $e) {
    if ($e->getCode() === '23000') { // integrity constraint violation
        return back()->withErrors('Invalid data');
    }
}
```

## Soft-Deletes with Query Builder

Don't forget that soft-deletes will exclude entries when you use Eloquent, but won't work if you use Query Builder.

```
// Will exclude soft-deleted entries
$users = User::all();

// Will NOT exclude soft-deleted entries
$users = User::withTrashed()->get();

// Will NOT exclude soft-deleted entries
$users = DB::table('users')->get();
```

## Good Old SQL Query

If you need to execute a simple SQL query, without getting any results - like changing something in DB schema, you can just do `DB::statement()`.

```
DB::statement('DROP TABLE users');
DB::statement('ALTER TABLE projects AUTO_INCREMENT=123');
```

## Use DB Transactions

If you have two DB operations performed, and second may get an error, then you should rollback the first one, right?

For that, I suggest to use DB Transactions, it's really easy in Laravel:

```
DB::transaction(function () {  
    DB::table('users')->update(['votes' => 1]);  
  
    DB::table('posts')->delete();  
});
```

## Update or Create

If you need to check if the record exists, and then update it, or create a new record otherwise, you can do it in one sentence - use Eloquent method `updateOrCreate()` :

```
// Instead of this  
$flight = Flight::where('departure', 'Oakland')  
    ->where('destination', 'San Diego')  
    ->first();  
if ($flight) {  
    $flight->update(['price' => 99, 'discounted' => 1]);  
} else {  
    $flight = Flight::create([  
        'departure' => 'Oakland',  
        'destination' => 'San Diego',  
        'price' => 99,  
        'discounted' => 1  
    ]);  
}  
  
// Do it in ONE sentence  
$flight = Flight::updateOrCreate(  
    ['departure' => 'Oakland', 'destination' => 'San Diego'],  
    ['price' => 99, 'discounted' => 1]  
);
```

## Forget Cache on Save

Tip given by [@pratiksh404](#)

If you have cache key like `posts` that gives collection, and you want to forget that cache key on new store or update, you can call static `saving` function on your model:

```
class Post extends Model
{
    // Forget cache key on storing or updating
    public static function boot()
    {
        parent::boot();
        static::saving(function () {
            Cache::forget('posts');
        });
    }
}
```

## Change Format Of Created\_at and Updated\_at

Tip given by [@syofyanzuhad](#)

To change the format of `created_at` you can add a method in your model like this:

```
public function getCreatedAtFormattedAttribute()
{
    return $this->created_at->format('H:i d, M Y');
}
```

So you can use it `$entry->created_at_formatted` when it's needed. It will return the `created_at` attribute like this: 04:19 23, Aug 2020 .

And also for changing format of `updated_at` attribute, you can add this method :

```
public function getUpdatedAtFormattedAttribute()
{
    return $this->updated_at->format('H:i d, M Y');
}
```

So you can use it `$entry->updated_at_formatted` when it's needed. It will return the `updated_at` attribute like this: 04:19 23, Aug 2020 .

## Storing Array Type into JSON

Tip given by [@pratiksh404](#)

If you have input field which takes an array and you have to store it as a JSON, you can use `$casts` property in your model. Here `images` is a JSON attribute.

```
protected $casts = [  
    'images' => 'array',  
];
```

So you can store it as a JSON, but when retrieved from DB, it can be used as an array.

## Make a Copy of the Model

If you have two very similar Models (like shipping address and billing address) and you need to make a copy of one to another, you can use `replicate()` method and change some properties after that.

Example from the [official docs](#):

```
$shipping = Address::create([  
    'type' => 'shipping',  
    'line_1' => '123 Example Street',  
    'city' => 'Victorville',  
    'state' => 'CA',  
    'postcode' => '90001',  
]);  
  
$billing = $shipping->replicate()->fill([  
    'type' => 'billing'  
]);  
  
$billing->save();
```

## Reduce Memory

Sometimes we need to load a huge amount of data into memory. For example:

```
$orders = Order::all();
```

But this can be slow if we have really huge data, because Laravel prepares objects of the Model class. In such cases, Laravel has a handy function `toBase()`

```
$orders = Order::toBase()->get();  
// $orders will contain `Illuminate\Support\Collection` with objects `stdClass`.
```

By calling this method, it will fetch the data from the database, but it will not prepare the Model class. Keep in mind it is often a good idea to pass an array of fields to the get method, preventing all fields to be fetched from the database.

## Models Relations

---

[↑ Go to top](#) [← Previous \(DB Models and Eloquent\)](#) [→ Next \(Migrations\)](#)

- [OrderBy on Eloquent relationships](#)
- [Conditional relationships](#)
- [Raw DB Queries: havingRaw\(\)](#)
- [Eloquent has\(\) deeper](#)
- [Has Many. How many exactly?](#)
- [Default model](#)
- [Use hasMany to create Many](#)
- [Multi level Eager Loading](#)
- [Eager Loading with Exact Columns](#)
- [Touch parent updated\\_at easily](#)
- [Always Check if Relationship Exists](#)
- [Use withCount\(\) to Calculate Child Relationships Records](#)
- [Extra Filter Query on Relationships](#)
- [Load Relationships Always, but Dynamically](#)
- [Instead of belongsTo, use hasMany](#)
- [Rename Pivot Table](#)
- [Update Parent in One Line](#)
- [Laravel 7+ Foreign Keys](#)
- [Combine Two "whereHas"](#)
- [Check if Relationship Method Exists](#)
- [Pivot Table with Extra Relations](#)
- [Load Count on-the-fly](#)
- [Randomize Relationship Order](#)

### OrderBy on Eloquent relationships

You can specify orderBy() directly on your Eloquent relationships.

```
public function products()
{
    return $this->hasMany(Product::class);
}

public function productsByName()
{
    return $this->hasMany(Product::class)->orderBy('name');
}
```

## Conditional relationships

If you notice that you use same relationship often with additional "where" condition, you can create a separate relationship method.

Model:

```
public function comments()
{
    return $this->hasMany(Comment::class);
}

public function approved_comments()
{
    return $this->hasMany(Comment::class)->where('approved', 1);
}
```

## Raw DB Queries: havingRaw()

You can use RAW DB queries in various places, including `havingRaw()` function after `groupBy()` .

```
Product::groupBy('category_id')->havingRaw('COUNT(*) > 1')->get();
```

## Eloquent has() deeper

You can use Eloquent `has()` function to query relationships even two layers deep!

```
// Author -> hasMany(Book::class);
// Book -> hasMany(Rating::class);
$authors = Author::has('books.ratings')->get();
```

## Has Many. How many exactly?

In Eloquent `hasMany()` relationships, you can filter out records that have X amount of children records.

```
// Author -> hasMany(Book::class)
$authors = Author::has('books', '>', 5)->get();
```

## Default model

You can assign a default model in `belongsTo` relationship, to avoid fatal errors when calling it like `{{ $post->user->name }}` if `$post->user` doesn't exist.

```
public function user()
{
    return $this->belongsTo('App\User')->withDefault();
}
```

## Use hasMany to create Many

If you have `hasMany()` relationship, you can use `saveMany()` to save multiple "child" entries from your "parent" object, all in one sentence.

```
$post = Post::find(1);
$post->comments()->saveMany([
    new Comment(['message' => 'First comment']),
    new Comment(['message' => 'Second comment']),
]);
```

## Multi level Eager Loading

In Laravel you can Eager Load multiple levels in one statement, in this example we not only load the author relation but also the country relation on the author model.

```
$users = App\Book::with('author.country')->get();
```

## Eager Loading with Exact Columns

You can do Laravel Eager Loading and specify the exact columns you want to get from the relationship.

```
$users = App\Book::with('author:id,name')->get();
```

You can do that even in deeper, second level relationships:

```
$users = App\Book::with('author.country:id,name')->get();
```

## Touch parent updated\_at easily

If you are updating a record and want to update the `updated_at` column of parent relationship (like, you add new post comment and want `posts.updated_at` to renew), just use `$touches = ['post'];` property on child model.

```
class Comment extends Model
{
    protected $touches = ['post'];
}
```

## Always Check if Relationship Exists

Never **ever** do `$model->relationship->field` without checking if relationship object still exists.

It may be deleted for whatever reason, outside your code, by someone else's queued job etc. Do `if-else`, or `{{ $model->relationship->field ?? '' }}` in Blade, or `{{ optional($model->relationship)->field }}`. With php8 you can even use the nullsafe operator `{{ $model->relationship?->field }}`

## Use withCount() to Calculate Child Relationships Records

If you have `hasMany()` relationship, and you want to calculate “children” entries, don’t write a special query. For example, if you have posts and comments on your User model, write this `withCount()` :

```
public function index()
{
    $users = User::withCount(['posts', 'comments'])->get();
    return view('users', compact('users'));
}
```



And then, in your Blade file, you will access those number with `{relationship}_count` properties:

```
@foreach ($users as $user)
<tr>
    <td>{{ $user->name }}

```

⋮ README.md

```
</tr>
@endforeach
```

You may also order by that field:

```
User::withCount('comments')->orderBy('comments_count', 'desc')->get();
```

## Extra Filter Query on Relationships

If you want to load relationship data, you can specify some limitations or ordering in a closure function. For example, if you want to get Countries with only three of their biggest cities, here's the code.

```
$countries = Country::with(['cities' => function($query) {
    $query->orderBy('population', 'desc');
    $query->take(3);
}])->get();
```

## Load Relationships Always, but Dynamically

You can not only specify what relationships to ALWAYS load with the model, but you can do it dynamically, in the constructor method:

```
class ProductTag extends Model
{
    protected $with = ['product'];

    public function __construct() {
        parent::__construct();
        $this->with = ['product'];

        if (auth()->check()) {
            $this->with[] = 'user';
        }
    }
}
```



```
}  
}
```

## Instead of belongsTo, use hasMany

For `belongsTo` relationship, instead of passing parent's ID when creating child record, use `hasMany` relationship to make a shorter sentence.

```
// if Post -> belongsTo(User), and User -> hasMany(Post)...  
// Then instead of passing user_id...  
Post::create([  
    'user_id' => auth()->id(),  
    'title' => request()->input('title'),  
    'post_text' => request()->input('post_text'),  
]);  
  
// Do this  
auth()->user()->posts()->create([  
    'title' => request()->input('title'),  
    'post_text' => request()->input('post_text'),  
]);
```

## Rename Pivot Table

If you want to rename "pivot" word and call your relationship something else, you just use `->as('name')` in your relationship.

Model:

```
public function podcasts() {  
    return $this->belongsToMany('App\Podcast')  
        ->as('subscription')  
        ->withTimestamps();  
}
```

Controller:

```
$podcasts = $user->podcasts();  
foreach ($podcasts as $podcast) {  
    // instead of $podcast->pivot->created_at ...  
    echo $podcast->subscription->created_at;  
}
```

## Update Parent in One Line

If you have a `belongsTo()` relationship, you can update the Eloquent relationship data in the same sentence:

```
// if Project -> belongsTo(User::class)
$project->user->update(['email' => 'some@gmail.com']);
```

## Laravel 7+ Foreign Keys

From Laravel 7, in migrations you don't need to write two lines for relationship field - one for the field and one for foreign key. Use method `foreignId()`.

```
// Before Laravel 7
Schema::table('posts', function (Blueprint $table) {
    $table->unsignedBigInteger('user_id');
    $table->foreign('user_id')->references('id')->on('users');
})

// From Laravel 7
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('user_id')->constrained();
})

// Or, if your field is different from the table reference
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('created_by_id')->constrained('users', 'column');
})
```

## Combine Two "whereHas"

In Eloquent, you can combine `whereHas()` and `orWhereDoesntHave()` in one sentence.

```
User::whereHas('roles', function($query) {
    $query->where('id', 1);
})
->orWhereDoesntHave('roles')
->get();
```

## Check if Relationship Method Exists

If your Eloquent relationship names are dynamic and you need to check if relationship with such name exists on the object, use PHP function `method_exists($object, $methodName)`

```
$user = User::first();  
if (method_exists($user, 'roles')) {  
    // Do something with $user->roles()->...  
}
```

## Pivot Table with Extra Relations

In many-to-many relationship, your pivot table may contain extra fields, and even extra relationships to other Model.

Then generate a separate Pivot Model:

```
php artisan make:model RoleUser --pivot
```

Next, specify it in `belongsToMany()` with `->using()` method. Then you could do magic, like in the example.

```
// in app/Models/User.php  
public function roles()  
{  
    return $this->belongsToMany(Role::class)  
        ->using(RoleUser::class)  
        ->withPivot(['team_id']);  
}  
  
// app/Models/RoleUser.php: notice extends Pivot, not Model  
use Illuminate\Database\Eloquent\Relations\Pivot;  
  
class RoleUser extends Pivot  
{  
    public function team()  
    {  
        return $this->belongsTo(Team::class);  
    }  
}  
  
// Then, in Controller, you can do:  
$firstTeam = auth()->user()->roles()->first()->pivot->team->name;
```

## Load Count on-the-fly

In addition to Eloquent's `withCount()` method to count related records, you can also load the count on-the-fly, with `loadCount()` :

```
// if your Book hasMany Reviews...
$book = App\Book::first();

$book->loadCount('reviews');
// Then you get access to $book->reviews_count;

// Or even with extra condition
$book->loadCount(['reviews' => function ($query) {
    $query->where('rating', 5);
}]);
```

## Randomize Relationship Order

You can use `inRandomOrder()` to randomize Eloquent query result, but also you can use it to randomize the **relationship** entries you're loading with query.

```
// If you have a quiz and want to randomize questions...

// 1. If you want to get questions in random order:
$questions = Question::inRandomOrder()->get();

// 2. If you want to also get question options in random order:
$questions = Question::with(['answers' => function($q) {
    $q->inRandomOrder();
}])->inRandomOrder()->get();
```

## Migrations

[↑ Go to top](#) [← Previous \(Models Relations\)](#) [→ Next \(Views\)](#)

- [Unsigned Integer](#)
- [Order of Migrations](#)
- [Migration fields with timezones](#)
- [Database migrations column types](#)
- [Default Timestamp](#)
- [Migration Status](#)
- [Create Migration with Spaces](#)
- [Create Column after Another Column](#)

## Unsigned Integer

For foreign key migrations instead of `integer()` use `unsignedInteger()` type or `integer()->unsigned()` , otherwise you may get SQL errors.

```
Schema::create('employees', function (Blueprint $table) {  
    $table->unsignedInteger('company_id');  
    $table->foreign('company_id')->references('id')->on('companies');  
    // ...  
});
```

You can also use `unsignedBigInteger()` if that other column is `bigInteger()` type.

```
Schema::create('employees', function (Blueprint $table) {  
    $table->unsignedBigInteger('company_id');  
});
```

## Order of Migrations

If you want to change the order of DB migrations, just rename the file's timestamp, like from `2018_08_04_070443_create_posts_table.php` to `2018_07_04_070443_create_posts_table.php` (changed from `2018_08_04` to `2018_07_04` ).

They run in alphabetical order.

## Migration fields with timezones

Did you know that in migrations there's not only `timestamps()` but also `timestampsTz()` , for the timezone?

```
Schema::create('employees', function (Blueprint $table) {  
    $table->increments('id');  
    $table->string('name');  
    $table->string('email');  
    $table->timestampsTz();  
});
```

Also, there are columns `dateTimeTz()` , `timeTz()` , `timestampTz()` , `softDeletesTz()` .

## Database migrations column types

There are interesting column types for migrations, here are a few examples.

```
$table->geometry('positions');
$table->ipAddress('visitor');
$table->macAddress('device');
$table->point('position');
$table->uuid('id');
```

See all column types on the [official documentation](#).

## Default Timestamp

While creating migrations, you can use `timestamp()` column type with option `useCurrent()` and `useCurrentOnUpdate()`, it will set `CURRENT_TIMESTAMP` as default value.

```
$table->timestamp('created_at')->useCurrent();
$table->timestamp('updated_at')->useCurrentOnUpdate();
```

## Migration Status

If you want to check what migrations are executed or not yet, no need to look at the database "migrations" table, you can launch `php artisan migrate:status` command.

Example result:

```
+-----+-----+-----+-----+
| Ran? | Migration                                          | Batch |
+-----+-----+-----+-----+
| Yes  | 2014_10_12_000000_create_users_table            | 1      |
| Yes  | 2014_10_12_100000_create_password_resets_table  | 1      |
| No   | 2019_08_19_000000_create_failed_jobs_table      |        |
+-----+-----+-----+-----+
```

## Create Migration with Spaces

When typing `make:migration` command, you don't necessarily have to use underscore `_` symbol between parts, like `create_transactions_table`. You can put the name into quotes and then use spaces instead of underscores.

```
// This works
php artisan make:migration create_transactions_table

// But this works too
php artisan make:migration "create transactions table"
```

Source: [Steve O on Twitter](#)

## Create Column after Another Column

*Notice: Only MySQL*

If you're adding a new column to the existing table, it doesn't necessarily have to become the last in the list. You can specify, after which column it should be created:

```
Schema::table('users', function (Blueprint $table) {  
    $table->string('phone')->after('email');  
});
```

If you want your column to be the first in your table, then use the before method.

```
Schema::table('users', function (Blueprint $table) {  
    $table->string('uuid')->first();  
});
```

## Views

[↑ Go to top](#) [← Previous \(Migrations\)](#) [→ Next \(Routing\)](#)

- [\\$loop variable in foreach](#)
- [Does view file exist?](#)
- [Error code Blade pages](#)
- [View without controllers](#)
- [Blade @auth](#)
- [Two-level \\$loop variable in Blade](#)
- [Create Your Own Blade Directive](#)
- [Blade Directives: IncludeIf, IncludeWhen, IncludeFirst](#)

### \$loop variable in foreach

Inside of foreach loop, check if current entry is first/last by just using \$loop variable.

```
@foreach ($users as $user)  
    @if ($loop->first)  
        This is the first iteration.  
    @endif
```



```

        @if ($loop->last)
            This is the last iteration.
        @endif

        <p>This is user {{ $user->id }}</p>
    @endforeach

```

There are also other properties like `$loop->iteration` or `$loop->count`. Learn more on the [official documentation](#).

## Does view file exist?

You can check if View file exists before actually loading it.

```

if (view()->exists('custom.page')) {
    // Load the view
}

```

You can even load an array of views and only the first existing will be actually loaded.

```

return view()->first(['custom.dashboard', 'dashboard'], $data);

```

## Error code Blade pages

If you want to create a specific error page for some HTTP code, like 500 - just create a blade file with this code as filename, in `resources/views/errors/500.blade.php`, or `403.blade.php` etc, and it will automatically be loaded in case of that error code.

## View without controllers

If you want route to just show a certain view, don't create a Controller method, just use `Route::view()` function.

```

// Instead of this
Route::get('about', 'TextsController@about');
// And this
class TextsController extends Controller
{
    public function about()
    {
        return view('texts.about');
    }
}

```

```
}  
// Do this  
Route::view('about', 'texts.about');
```

## Blade @auth

Instead of if-statement to check logged in user, use `@auth` directive.

Typical way:

```
@if(auth()->user()  
    // The user is authenticated.  
@endif
```

Shorter:

```
@auth  
    // The user is authenticated.  
@endauth
```

The opposite is `@guest` directive:

```
@guest  
    // The user is not authenticated.  
@endguest
```

## Two-level \$loop variable in Blade

In Blade's foreach you can use `$loop` variable even in two-level loop to reach parent variable.

```
@foreach ($users as $user)  
    @foreach ($user->posts as $post)  
        @if ($loop->parent->first)  
            This is first iteration of the parent loop.  
        @endif  
    @endforeach  
@endforeach
```

## Create Your Own Blade Directive

It's very easy - just add your own method in `app/Providers/AppServiceProvider.php`. For example, if you want to have this for replace `<br>` tags with new lines:

```
<textarea>@br2nl($post->post_text)</textarea>
```

Add this directive to AppServiceProvider's `boot()` method:

```
public function boot()
{
    Blade::directive('br2nl', function ($string) {
        return "<?php echo preg_replace('/<br(\s*)?\/?>/i', '\n', $string); ?>";
    });
}
```

## Blade Directives: IncludeIf, IncludeWhen, IncludeFirst

If you are not sure whether your Blade partial file actually would exist, you may use these condition commands:

This will load header only if Blade file exists

```
@includeIf('partials.header')
```

This will load header only for user with `role_id 1`

```
@includeWhen(auth()->user()->role_id == 1, 'partials.header')
```

This will try to load `adminlte.header`, if missing - will load `default.header`

```
@includeFirst('adminlte.header', 'default.header')
```

## Routing

[↑ Go to top](#) [← Previous \(Views\)](#) [→ Next \(Validation\)](#)

- [Route group within a group](#)
- [Wildcard subdomains](#)
- [What's behind the routes?](#)

- [Route Model Binding: You can define a key](#)
- [Quickly Navigate from Routes file to Controller](#)
- [Route Fallback: When no Other Route is Matched](#)
- [Route Parameters Validation with RegExp](#)
- [Rate Limiting: Global and for Guests/Users](#)
- [Query string parameters to Routes](#)
- [Separate Routes by Files](#)
- [Translate Resource Verbs](#)
- [Custom Resource Route Names](#)
- [More Readable Route List](#)

## Route group within a group

In Routes, you can create a group within a group, assigning a certain middleware only to some URLs in the "parent" group.

```
Route::group(['prefix' => 'account', 'as' => 'account.'], function() {
    Route::get('login', 'AccountController@login');
    Route::get('register', 'AccountController@register');

    Route::group(['middleware' => 'auth'], function() {
        Route::get('edit', 'AccountController@edit');
    });
});
```

## Wildcard subdomains

You can create route group by dynamic subdomain name, and pass its value to every route.

```
Route::domain('{username}.workspace.com')->group(function () {
    Route::get('user/{id}', function ($username, $id) {
        //
    });
});
```

## What's behind the routes?

Want to know what routes are actually behind `Auth::routes()` ? From Laravel 7, it's in a separate package, so check the file `/vendor/laravel/ui/src/AuthRouteMethods.php` .

```

public function auth()
{
    return function ($options = []) {
        // Authentication Routes...
        $this->get('login', 'Auth\LoginController@showLoginForm')->name('login');
        $this->post('login', 'Auth\LoginController@login');
        $this->post('logout', 'Auth\LoginController@logout')->name('logout');
        // Registration Routes...
        if ($options['register'] ?? true) {
            $this->get('register', 'Auth\RegisterController@showRegistrationForm')->
            $this->post('register', 'Auth\RegisterController@register');
        }
        // Password Reset Routes...
        if ($options['reset'] ?? true) {
            $this->resetPassword();
        }
        // Password Confirmation Routes...
        if ($options['confirm'] ?? class_exists($this->prependGroupNamespace('Auth\C
            $this->confirmPassword();
        }
        // Email Verification Routes...
        if ($options['verify'] ?? false) {
            $this->emailVerification();
        }
    };
}

```

Before Laravel 7, check the file

`/vendor/laravel/framework/src/illuminate/Routing/Router.php` .

## Route Model Binding: You can define a key

You can do Route model binding like `Route::get('api/users/{user}', function (App\User $user) { ... })` - but not only by ID field. If you want `{user}` to be a `username` field, put this in the model:

```

public function getRouteKeyName() {
    return 'username';
}

```

## Quickly Navigate from Routes file to Controller

This thing was optional before Laravel 8, and became a standard main syntax of routing in Laravel 8.

Instead of routing like this:

```
Route::get('page', 'PageController@action');
```

You can specify the Controller as a class:

```
Route::get('page', [\App\Http\Controllers\PageController::class, 'action']);
```

Then you will be able to click on **PageController** in PhpStorm, and navigate directly to Controller, instead of searching for it manually.

Or, to make it shorter, add this to top of Routes file:

```
use App\Http\Controllers\PageController;

// Then:
Route::get('page', [PageController::class, 'action']);
```

## Route Fallback: When no Other Route is Matched

If you want to specify additional logic for not-found routes, instead of just throwing default 404 page, you may create a special Route for that, at the very end of your Routes file.

```
Route::group(['middleware' => ['auth'], 'prefix' => 'admin', 'as' => 'admin.'], func
    Route::get('/home', 'HomeController@index');
    Route::resource('tasks', 'Admin\TasksController');
});

// Some more routes....
Route::fallback(function() {
    return 'Hm, why did you land here somehow?';
});
```

## Route Parameters Validation with RegExp

We can validate parameters directly in the route, with “where” parameter. A pretty typical case is to prefix your routes by language locale, like `fr/blog` and `en/article/333`. How do we ensure that those two first letters are not used for some other than language?

routes/web.php :

```
Route::group([
    'prefix' => '{locale}',
    'where' => ['locale' => '[a-zA-Z]{2}']
], function () {
    Route::get('/', 'HomeController@index');
    Route::get('article/{id}', 'ArticleController@show');
});
```

## Rate Limiting: Global and for Guests/Users

You can limit some URL to be called a maximum of 60 times per minute, with  
throttle:60,1:

```
Route::middleware('auth:api', 'throttle:60,1')->group(function () {
    Route::get('/user', function () {
        //
    });
});
```

But also, you can do it separately for public and for logged-in users:

```
// maximum of 10 requests for guests, 60 for authenticated users
Route::middleware('throttle:10|60,1')->group(function () {
    //
});
```

Also, you can have a DB field users.rate\_limit and limit the amount for specific user:

```
Route::middleware('auth:api', 'throttle:rate_limit,1')->group(function () {
    Route::get('/user', function () {
        //
    });
});
```

## Query string parameters to Routes

If you pass additional parameters to the route, in the array, those key / value pairs will automatically be added to the generated URL's query string.

```
Route::get('user/{id}/profile', function ($id) {
    //
})->name('profile');
```

```
$url = route('profile', ['id' => 1, 'photos' => 'yes']); // Result: /user/1/profile?
```

## Separate Routes by Files

If you have a set of routes related to a certain "section", you may separate them in a special `routes/XXXXX.php` file, and just include it in `routes/web.php`

Example with `routes/auth.php` in [Laravel Breeze](#) by Taylor Otwell himself:

```
Route::get('/', function () {
    return view('welcome');
});

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth'])->name('dashboard');

require __DIR__.'/auth.php';
```

Then, in `routes/auth.php`:

```
use App\Http\Controllers\Auth\AuthenticatedSessionController;
use App\Http\Controllers\Auth\RegisteredUserController;
// ... more controllers

use Illuminate\Support\Facades\Route;

Route::get('/register', [RegisteredUserController::class, 'create'])
    ->middleware('guest')
    ->name('register');

Route::post('/register', [RegisteredUserController::class, 'store'])
    ->middleware('guest');

// ... A dozen more routes
```

But you should use this `include()` only when that separate route file has the same settings for prefix/middlewares, otherwise it's better to group them in

`app/Providers/RouteServiceProvider`:

```
public function boot()
{
    $this->configureRateLimiting();
```



```

$this->routes(function () {
    Route::prefix('api')
        ->middleware('api')
        ->namespace($this->namespace)
        ->group(base_path('routes/api.php'));

    Route::middleware('web')
        ->namespace($this->namespace)
        ->group(base_path('routes/web.php'));

    // ... Your routes file listed next here
});
}

```

## Translate Resource Verbs

If you use resource controllers, but want to change URL verbs to non-English for SEO purposes, so instead of `/create` you want Spanish `/crear`, you can configure it by using `Route::resourceVerbs()` method in `App\Providers\RouteServiceProvider`:

```

public function boot()
{
    Route::resourceVerbs([
        'create' => 'crear',
        'edit' => 'editar',
    ]);

    // ...
}

```

## Custom Resource Route Names

When using Resource Controllers, in `routes/web.php` you can specify `->names()` parameter, so the URL prefix in the browser and the route name prefix you use all over Laravel project may be different.

```
Route::resource('p', ProductController::class)->names('products');
```

So this code above will generate URLs like `/p`, `/p/{id}`, `/p/{id}/edit`, etc. But you would call them in the code by `route('products.index')`, `route('products.create')`, etc.

## More Readable Route List

Have you ever run "php artisan route:list" and then realized that the list takes too much space and hard to read?

Here's the solution: `php artisan route:list --compact`

Then it shows 3 columns instead of 6 columns: shows only Method / URI / Action.

```
+-----+-----+-----+
| Method | URI | Action |
+-----+-----+-----+
| GET|HEAD | / | Closure |
| GET|HEAD | api/user | Closure |
| POST | confirm-password | App\Http\Controllers\Auth\ConfirmablePasswordController@store |
| GET|HEAD | confirm-password | App\Http\Controllers\Auth\ConfirmablePasswordController@show |
| GET|HEAD | dashboard | Closure |
| POST | email/verification-notification | App\Http\Controllers\Auth>EmailVerificationNotificationController@store |
| POST | forgot-password | App\Http\Controllers\Auth>PasswordResetLinkController@store |
| GET|HEAD | forgot-password | App\Http\Controllers\Auth>PasswordResetLinkController@create |
| POST | login | App\Http\Controllers\Auth\AuthenticatedSessionController@store |
| GET|HEAD | login | App\Http\Controllers\Auth\AuthenticatedSessionController@create |
| POST | logout | App\Http\Controllers\Auth\AuthenticatedSessionController@destroy |
| POST | register | App\Http\Controllers\Auth\RegisteredUserController@store |
| GET|HEAD | register | App\Http\Controllers\Auth\RegisteredUserController@create |
| POST | reset-password | App\Http\Controllers\Auth\NewPasswordController@store |
| GET|HEAD | reset-password/{token} | App\Http\Controllers\Auth\NewPasswordController@create |
| GET|HEAD | verify-email | App\Http\Controllers\Auth>EmailVerificationPromptController@__invoke |
| GET|HEAD | verify-email/{id}/{hash} | App\Http\Controllers\Auth\VerifyEmailController@__invoke |
+-----+-----+-----+
```

You can also specify the exact columns you want:

```
php artisan route:list --columns=Method,URI,Name
```

Method	URI	Name
GET HEAD	/	
GET HEAD	api/user	
POST	confirm-password	
GET HEAD	confirm-password	password.confirm
GET HEAD	dashboard	dashboard
POST	email/verification-notification	verification.send
POST	forgot-password	password.email
GET HEAD	forgot-password	password.request
POST	login	
GET HEAD	login	login
POST	logout	logout
POST	register	
GET HEAD	register	register
POST	reset-password	password.update
GET HEAD	reset-password/{token}	password.reset
GET HEAD	verify-email	verification.notice
GET HEAD	verify-email/{id}/{hash}	verification.verify

## Validation

[↑ Go to top](#)
[← Previous \(Routing\)](#)
[→ Next \(Collections\)](#)

- [Image validation](#)
- [Custom validation error messages](#)
- [Validate dates with "now" or "yesterday" words](#)
- [Validation Rule with Some Conditions](#)
- [Change Default Validation Messages](#)
- [Prepare for Validation](#)
- [Stop on First Validation Error](#)

### Image validation

While validating uploaded images, you can specify the dimensions you require.

```
['photo' => 'dimensions:max_width=4096,max_height=4096']
```

## Custom validation error messages

You can customize validation error messages per **field**, **rule** and **language** - just create a specific language file `resources/lang/xx/validation.php` with appropriate array structure.

```
'custom' => [
    'email' => [
        'required' => 'We need to know your e-mail address!',
    ],
],
```

## Validate dates with "now" or "yesterday" words

You can validate dates by rules before/after and passing various strings as a parameter, like: `tomorrow`, `now`, `yesterday`. Example: `'start_date' => 'after:now'`. It's using `strtotime()` under the hood.

```
$rules = [
    'start_date' => 'after:tomorrow',
    'end_date' => 'after:start_date'
];
```

## Validation Rule with Some Conditions

If your validation rules depend on some condition, you can modify the rules by adding `withValidator()` to your `FormRequest` class, and specify your custom logic there. Like, if you want to add validation rule only for some user role.

```
use Illuminate\Validation\Validator;
class StoreBlogCategoryRequest extends FormRequest {
    public function withValidator(Validator $validator) {
        if (auth()->user()->is_admin) {
            $validator->addRules(['some_secret_password' => 'required']);
        }
    }
}
```

## Change Default Validation Messages

If you want to change default validation error message for specific field and specific validation rule, just add a `messages()` method into your `FormRequest` class.

```
class StoreUserRequest extends FormRequest
{
    public function rules()
    {
        return ['name' => 'required'];
    }

    public function messages()
    {
        return ['name.required' => 'User name should be real name'];
    }
}
```

## Prepare for Validation

If you want to modify some field before default Laravel validation, or, in other words, "prepare" that field, guess what - there's a method `prepareForValidation()` in `FormRequest` class:

```
protected function prepareForValidation()
{
    $this->merge([
        'slug' => Illuminate\Support\Str::slug($this->slug),
    ]);
}
```

## Stop on First Validation Error

By default, Laravel validation errors will be returned in a list, checking all validation rules. But if you want the process to stop after the first error, use validation rule called `bail` :

```
$request->validate([
    'title' => 'bail|required|unique:posts|max:255',
    'body' => 'required',
]);
```

## Collections

[↑ Go to top](#) [← Previous \(Validation\)](#) [→ Next \(Auth\)](#)

- [Don't Filter by NULL in Collections](#)
- [Use groupBy on Collections with Custom Callback Function](#)

- [Multiple Collection Methods in a Row](#)
- [Calculate Sum with Pagination](#)
- [Serial no. in foreach loop with pagination](#)
- [Higher order collection methods](#)

## Don't Filter by NULL in Collections

You can filter by NULL in Eloquent, but if you're filtering the **collection** further - filter by empty string, there's no "null" in that field anymore.

```
// This works
$messages = Message::where('read_at is null')->get();

// Won't work - will return 0 messages
$messages = Message::all();
$unread_messages = $messages->where('read_at is null')->count();


// Will work
$unread_messages = $messages->where('read_at', '')->count();
```

## Use groupBy on Collections with Custom Callback Function

If you want to group result by some condition which isn't a direct column in your database, you can do that by providing a closure function.

For example, if you want to group users by day of registration, here's the code:

```
$users = User::all()->groupBy(function($item) {
    return $item->created_at->format('Y-m-d');
});
```

 Notice: it is done on a `Collection` class, so performed **AFTER** the results are fetched from the database.

## Multiple Collection Methods in a Row

If you query all results with `->all()` or `->get()`, you may then perform various Collection operations on the same result, it won't query database every time.

```
$users = User::all();
echo 'Max ID: ' . $users->max('id');
```

```
echo 'Average age: ' . $users->avg('age');
echo 'Total budget: ' . $users->sum('budget');
```

## Calculate Sum with Pagination

How to calculate the sum of all records when you have only the PAGINATED collection? Do the calculation BEFORE the pagination, but from the same query.

```
// How to get sum of post_views with pagination?
$post = Post::paginate(10);
// This will be only for page 1, not ALL posts
$sum = $post->sum('post_views');

// Do this with Query Builder
$query = Post::query();
// Calculate sum
$sum = $query->sum('post_views');
// And then do the pagination from the same query
$post = $query->paginate(10);
```

## Serial no in foreach loop with pagination

We can use foreach collection items index as serial no (SL) in pagination.

```
...
<th>Serial</th>
...
@foreach ($products as $product)
<tr>
    <td>{{ $loop->index + $product->firstItem() }}</td>
    ...
@endforeach
```

it will solve the issue of next pages(?page=2&...) index count from continue.

## Higher order collection methods

Collections have higher order methods, this are methods that can be chained , like `groupBy()` , `map()` ... Giving you a fluid syntax. This example calculates the price per group of products on an offer.

```
$offer = [
    'name' => 'offer1',
    'lines' => [
```

```

        ['group' => 1, 'price' => 10],
        ['group' => 1, 'price' => 20],
        ['group' => 2, 'price' => 30],
        ['group' => 2, 'price' => 40],
        ['group' => 3, 'price' => 50],
        ['group' => 3, 'price' => 60]
    ];

```

```
$totalPerGroup = collect($offer->lines)->groupBy('group')->map(fn($group) => $group-
```

## Auth

[↑ Go to top](#)
[← Previous \(Collections\)](#)
[→ Next \(Mail\)](#)

- [Check Multiple Permissions at Once](#)
- [More Events on User Registration](#)
- [Did you know about Auth::once\(\)?](#)
- [Change API Token on users password update](#)
- [Override Permissions for Super Admin](#)

## Check Multiple Permissions at Once

In addition to `@can` Blade directive, did you know you can check multiple permissions at once with `@canany` directive?

```

@canany(['update', 'view', 'delete'], $post)
    // The current user can update, view, or delete the post
@elsecanany(['create'], \App\Post::class)
    // The current user can create a post
@endcanany

```

## More Events on User Registration

Want to perform some actions after new user registration? Head to `app/Providers/EventServiceProvider.php` and add more Listeners classes, and then in those classes implement `handle()` method with `$event->user` object

```

class EventServiceProvider extends ServiceProvider
{
    protected $listen = [
        Registered::class => [

```



```

        SendEmailVerificationNotification::class,

        // You can add any Listener class here
        // With handle() method inside of that class
    ],
];

```

## Did you know about Auth::once()?

You can login with user only for ONE REQUEST, using method `Auth::once()` .  
No sessions or cookies will be utilized, which means this method may be helpful when building a stateless API.

```

if (Auth::once($credentials)) {
    //
}

```

## Change API Token on users password update

It's convenient to change the user's API Token when its password changes.

Model:

```

public function setPasswordAttribute($value)
{
    $this->attributes['password'] = $value;
    $this->attributes['api_token'] = Str::random(100);
}

```

## Override Permissions for Super Admin

If you've defined your Gates but want to override all permissions for SUPER ADMIN user, to give that superadmin ALL permissions, you can intercept gates with `Gate::before()` statement, in `AuthServiceProvider.php` file.

```

// Intercept any Gate and check if it's super admin
Gate::before(function($user, $ability) {
    if ($user->is_super_admin == 1) {
        return true;
    }
});

// Or if you use some permissions package...

```

```
Gate::before(function($user, $ability) {  
    if ($user->hasPermission('root')) {  
        return true;  
    }  
});
```

## Mail

---

[↑ Go to top](#) [← Previous \(Auth\)](#) [→ Next \(Artisan\)](#)

- [Testing email into laravel.log](#)
- [Preview Mailables](#)
- [Default Email Subject in Laravel Notifications](#)
- [Send Notifications to Anyone](#)

### Testing email into laravel.log

If you want to test email contents in your app but unable or unwilling to set up something like Mailgun, use `.env` parameter `MAIL_DRIVER=log` and all the email will be saved into `storage/logs/laravel.log` file, instead of actually being sent.

### Preview Mailables

If you use Mailables to send email, you can preview the result without sending, directly in your browser. Just return a Mailable as route result:

```
Route::get('/mailable', function () {  
    $invoice = App\Invoice::find(1);  
    return new App\Mail\InvoicePaid($invoice);  
});
```

### Default Email Subject in Laravel Notifications

If you send Laravel Notification and don't specify subject in `toMail()`, default subject is your notification class name, CamelCased into Spaces.

So, if you have:

```
class UserRegistrationEmail extends Notification {  
    //  
}
```

Then you will receive an email with subject **User Registration Email**.

## Send Notifications to Anyone

You can send Laravel Notifications not only to a certain user with `$user->notify()`, but also to anyone you want, via `Notification::route()`, with so-called "on-demand" notifications:

```
Notification::route('mail', 'taylor@example.com')
    ->route('nexmo', '5555555555')
    ->route('slack', 'https://hooks.slack.com/services/...')
    ->notify(new InvoicePaid($invoice));
```

## Artisan

---

[↑ Go to top](#) [← Previous \(Mail\)](#) [→ Next \(Factories\)](#)

- [Artisan command parameters](#)
- [Maintenance Mode](#)
- [Artisan command help](#)
- [Exact Laravel version](#)
- [Launch Artisan command from anywhere](#)

## Artisan command parameters

When creating Artisan command, you can ask the input in variety of ways: `$this->confirm()`, `$this->anticipate()`, `$this->choice()`.

```
// Yes or no?
if ($this->confirm('Do you wish to continue?')) {
    //
}

// Open question with auto-complete options
$name = $this->anticipate('What is your name?', ['Taylor', 'Dayle']);

// One of the listed options with default index
$name = $this->choice('What is your name?', ['Taylor', 'Dayle'], $defaultIndex);
```

## Maintenance Mode

If you want to enable maintenance mode on your page, execute the down Artisan command:

```
php artisan down
```

Then people would see default 503 status page.

You may also provide flags, in Laravel 8:

- the path the user should be redirected to
- the view that should be prerendered
- secret phrase to bypass maintenance mode
- status code during maintenance mode
- retry page reload every X seconds

```
php artisan down --redirect="/" --render="errors::503" --secret="1630542a-246b-4b66-
```

Before Laravel 8:

- message that would be shown
- retry page reload every X seconds
- still allow the access to some IP address

```
php artisan down --message="Upgrading Database" --retry=60 --allow=127.0.0.1
```

When you've done the maintenance work, just run:

```
php artisan up
```

## Artisan command help

To check the options of artisan command, Run artisan commands with `--help` flag. For example, `php artisan make:model --help` and see how many options you have:

Options:

<code>-a, --all</code>	Generate a migration, seeder, factory, and resource controller for the model
<code>-c, --controller</code>	Create a new controller for the model

```

-f, --factory          Create a new factory for the model
--force               Create the class even if the model already exists
-m, --migration        Create a new migration file for the model
-s, --seed             Create a new seeder file for the model
-p, --pivot            Indicates if the generated model should be a custom
intermediate table model
-r, --resource         Indicates if the generated controller should be a
resource controller
--api                 Indicates if the generated controller should be an API
controller
-h, --help             Display this help message
-q, --quiet            Do not output any message
-V, --version          Display this application version
--ansi                Force ANSI output
--no-ansi              Disable ANSI output
-n, --no-interaction   Do not ask any interactive question
--env[=ENV]           The environment the command should run under
-v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output,
2 for more verbose output and 3 for debug

```

## Exact Laravel version

Find out exactly what Laravel version you have in your app, by running command `php artisan --version`

## Launch Artisan command from anywhere

If you have an Artisan command, you can launch it not only from Terminal, but also from anywhere in your code, with parameters. Use `Artisan::call()` method:

```

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

    //
});

```

## Factories

[↑ Go to top](#)
[← Previous \(Artisan\)](#)
[→ Next \(Log and debug\)](#)

- [Factory callbacks](#)
- [Generate Images with Seeds/Factories](#)

## Factory callbacks

While using factories for seeding data, you can provide Factory Callback functions to perform some action after record is inserted.

```
$factory->afterCreating(App\User::class, function ($user, $faker) {  
    $user->accounts()->save(factory(App\Account::class)->make());  
});
```

## Generate Images with Seeds/Factories

Did you know that Faker can generate not only text values but also IMAGES? See `avatar` field here - it will generate 50x50 image:

```
$factory->define(User::class, function (Faker $faker) {  
    return [  
        'name' => $faker->name,  
        'email' => $faker->unique()->safeEmail,  
        'email_verified_at' => now(),  
        'password' => bcrypt('password'),  
        'remember_token' => Str::random(10),  
        'avatar' => $faker->image(storage_path('images'), 50, 50)  
    ];  
});
```

## Log and debug

[↑ Go to top](#) [← Previous \(Factories\)](#) [→ Next \(API\)](#)

- [Logging with parameters](#)
- [More convenient DD](#)

### Logging with parameters

You can write `Log::info()`, or shorter `info()` message with additional parameters, for more context about what happened.

```
Log::info('User failed to login.', ['id' => $user->id]);
```

### More convenient DD

Instead of doing `dd($result)` you can put `->dd()` as a method directly at the end of your Eloquent sentence, or any Collection.

```
// Instead of
$users = User::where('name', 'Taylor')->get();
dd($users);
// Do this
$users = User::where('name', 'Taylor')->get()->dd();
```

## API

[↑ Go to top](#) [← Previous \(Log and debug\)](#) [→ Next \(Other\)](#)

- [API Resources: With or Without "data"?](#)
- [API Return "Everything went ok"](#)

### API Resources: With or Without "data"?

If you use Eloquent API Resources to return data, they will be automatically wrapped in 'data'. If you want to remove it, add `JsonResource::withoutWrapping();` in `app/Providers/AppServiceProvider.php`.

```
class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        JsonResource::withoutWrapping();
    }
}
```

### API Return "Everything went ok"

If you have API endpoint which performs some operations but has no response, so you wanna return just "everything went ok", you may return 204 status code "No content". In Laravel, it's easy: `return response()->noContent();`.

```
public function reorder(Request $request)
{
    foreach ($request->input('rows', []) as $row) {
        Country::find($row['id'])->update(['position' => $row['position']]);
    }
}
```

```
    return response()->noContent();  
}
```

## Other

---

[↑ Go to top](#) [← Previous \(API\)](#)

- [Localhost in .env](#)
- [When \(NOT\) to run "composer update"](#)
- [Composer: check for newer versions](#)
- [Auto-Capitalize Translations](#)
- [Carbon with Only Hours](#)
- [Single Action Controllers](#)
- [Redirect to Specific Controller Method](#)
- [Use Older Laravel Version](#)
- [Add Parameters to Pagination Links](#)
- [Repeatable Callback Functions](#)
- [Request: has any](#)
- [Simple Pagination](#)
- [Data Get Function](#)

### Localhost in .env

Don't forget to change `APP_URL` in your `.env` file from `http://localhost` to the real URL, cause it will be the basis for any links in your email notifications and elsewhere.

```
APP_NAME=Laravel  
APP_ENV=local  
APP_KEY=base64:9PHz3TL5C4YrdV6Gg/Xkkmx9btaE93j7rQTUZWm2MqU=  
APP_DEBUG=true  
APP_URL=http://localhost
```

### When (NOT) to run "composer update"

Not so much about Laravel, but... Never run `composer update` on production live server, it's slow and will "break" repository. Always run `composer update` locally on your computer, commit new `composer.lock` to the repository, and run `composer install` on the live server.



## Composer: Check for Newer Versions

If you want to find out which of your `composer.json` packages have released newer versions, just run `composer outdated`. You will get a full list with all information, like this below.

```
phpdocumentor/type-resolver 0.4.0 0.7.1
phpunit/php-code-coverage 6.1.4 7.0.3 Library that provides collection,
processing, and rende...
phpunit/phpunit 7.5.9 8.1.3 The PHP Unit Testing framework.
ralouphie/getallheaders 2.0.5 3.0.3 A polyfill for getallheaders.
sebastian/global-state 2.0.0 3.0.0 Snapshotting of global state
```

## Auto-Capitalize Translations

In translation files ( `resources/lang` ), you can specify variables not only as `:variable`, but also capitalized as `:VARIABLE` or `:Variable` - and then whatever value you pass - will be also capitalized automatically.

```
// resources/lang/en/messages.php
'welcome' => 'Welcome, :Name'

// Result: "Welcome, Taylor"
echo __('messages.welcome', ['name' => 'taylor']);
```

## Carbon with Only Hours

If you want to have a current date without seconds and/or minutes, use Carbon's methods like `setSeconds(0)` OR `setMinutes(0)`.

```
// 2020-04-20 08:12:34
echo now();

// 2020-04-20 08:12:00
echo now()->setSeconds(0);

// 2020-04-20 08:00:00
echo now()->setSeconds(0)->setMinutes(0);

// Another way - even shorter
echo now()->startOfHour();
```

## Single Action Controllers

If you want to create a controller with just one action, you can use `__invoke()` method and even create "invokable" controller.

Route:

```
Route::get('user/{id}', 'ShowProfile');
```

Artisan:

```
php artisan make:controller ShowProfile --invokable
```

Controller:

```
class ShowProfile extends Controller
{
    public function __invoke($id)
    {
        return view('user.profile', [
            'user' => User::findOrFail($id)
        ]);
    }
}
```

## Redirect to Specific Controller Method

You can `redirect()` not only to URL or specific route, but to a specific Controller's specific method, and even pass the parameters. Use this:

```
return redirect()->action('SomeController@method', ['param' => $value]);
```

## Use Older Laravel Version

If you want to use OLDER version instead of the newest Laravel, use this command:

```
composer create-project --prefer-dist laravel/laravel project "7.*"
```

Change 7.\* to whichever version you want.

## Add Parameters to Pagination Links

In default Pagination links, you can pass additional parameters, preserve the original query string, or even point to a specific `#xxxxx` anchor.

```
{{ $users->appends(['sort' => 'votes'])->links() }}
```

```
{{ $users->withQueryString()->links() }}
```

```
{{ $users->fragment('foo')->links() }}
```

## Repeatable Callback Functions

If you have a callback function that you need to re-use multiple times, you can assign it to a variable, and then re-use.

```
$userCondition = function ($query) {  
    $query->where('user_id', auth()->id());  
};  
  
// Get articles that have comments from this user  
// And return only those comments from this user  
$articles = Article::with(['comments' => $userCondition])  
    ->whereHas('comments', $userCondition)  
    ->get();
```

## Request: has any

You can check not only one parameter with `$request->has()` method, but also check for multiple parameters present, with `$request->hasAny()` :

```
public function store(Request $request)  
{  
    if ($request->hasAny(['api_key', 'token'])) {  
        echo 'We have API key passed';  
    } else {  
        echo 'No authorization parameter';  
    }  
}
```

## Simple Pagination

In pagination, if you want to have just "Previous/next" links instead of all the page numbers (and have fewer DB queries because of that), just change `paginate()` to `simplePaginate()` :

```
// Instead of
$users = User::paginate(10);

// You can do this
$users = User::simplePaginate(10);
```

## Data Get Function

If you have an array complex data structure, for example a nested array with objects. You can use `data_get()` helper function retrieves a value from a nested array or object using "dot" notation and wildcard:

```
// We have an array
[
  0 =>
    ['user_id' => 'some user id', 'created_at' => 'some timestamp', 'product' =>
  1 =>
    ['user_id' => 'some user id', 'created_at' => 'some timestamp', 'product' =>
  2 => etc
]

// Now we want to get all products ids. We can do like this:

data_get($yourArray, '*.product.id');

// Now we have all products ids [1, 2, 3, 4, 5, etc...]
```

## Releases

No releases published

## Packages

No packages published

## Contributors 13



[+ 2 contributors](#)

