Kirill Devyatkin

It's not a Laravel alternative or SOLID principles, patterns etc. Here you'll find the best practices which are usually opened to real-life Laravel projects.

Table of Contents

- Follow Laravel naming conventions
- Use shorter and more readable syntax where possible
- Single responsibility principle
- Fat models, skinny controllers
- Validation
- Business logic should be in service class
- Don't repeat yourself (DRY)
- Prefer to use Eloquent over using Query Builder and raw SQL queries. Prefer collections over arrays
- Mass assignment
- Do not execute queries in Blade templates and use eager loading (N + 1 problem)
- Do not put JS and CSS in Blade templates and do not put any HTML in PHP classes
- Use config and language files, constants instead of text in the code
- Use IoC container or facades instead of new Class
- Do not get data from the `_env` -file directly
- Store dates in the standard format. Use accessors and mutators to modify date format

Follow Laravel naming conventions

From Kirill Devyatkin:

Follow Laravel naming conventions accepted by Laravel community:

What	Bad	Good
controller	singular	plural (resourceful)
route	plural	plural
hasMany() / belongsTo()	singular	plural
hasOne() / belongsToOne()	singular	singular
all other relationships	plural	plural (customize)
Table	singular	plural
Pivot table	singular model names in alphabetical order	singular model names in alphabetical order
Table column	snake_case	snake_case
Foreign Key	singular model name with _id suffix	snake_id
Primary Key	id	id
Migration	2017_12_31_235959_create_articles_table	2017_12_31_235959_articles
Method	camelCase	getFull
Variable	camelCase	$full_name
Method in resource controller	store	save
Method in view model	camelCase	formatLastUpdatedHumanly
Route parameter	camelCase	/articles/{articleId}
Config	kebab-case	google-calendar
Command line/Artisan	verb-noun	db:backup
Trait	adjective	Authorizable

Use shorter and more readable syntax where possible

Bad

```
$request = request();
$message = request('message');
```

Good

```
$request = request();
$message = $request->input('message');
```

Bad

```
Route::get('/users', function () {
    return User::where('active', 1)->orderBy('name', 'desc')->get();
});
```

Good

```
// routes/web.php
use App\Http\Controllers\UserController;
Route::get('/users', [UserController::class, 'index']);

// app/Http/Controllers/UserController.php
class UserController extends Controller
{
    /**
    * Display a listing of the resource.
    * GET /users
    * @return \Illuminate\Http\Response
    */
    public function index()
    {
        return User::active()->orderBy('name', 'desc')->get();
    }
}

// app/Models/User.php
class User extends Model
{
    /**
    * Scope a query to only include active users.
    * @return \Illuminate\Database\Eloquent\Builder
    */
    public function scopeActive($query)
    {
        return $query->where('active', 1);
    }
}
```

Bad (shorter syntax)

```
$user = new User;
$user->name = request('name');
$user->email = request('email');
$user->password = md5(request('password'));
$user->phone = request('full_phone');
$user->save();
```

Good (more readable syntax)

```
// app/Http/Controllers/UserController.php
use App\Models\User;
use App\Rules\ValidPhone;
use App\Rules\ValidPassword;
use Illuminate\Support\Facades\Hash;
use Illuminate\Http\Controllers\Controller;
class UserController extends Controller
{
    /**
    * Store a newly created resource in storage.
    * POST /users
    * @return \Illuminate\Http\Response
    */
    public function store()
    {
        $this->validate(request(), [
            'name' => 'required',
            'email' => 'required|email',
            'password' => 'required|min:6|max:255|'.(new ValidPassword(request('email'))),
            'full_phone' => (new ValidPhone(request('country')))->rules(),
        ]);
        $user = new User;
        $user->name = request('name');
        $user->email = request('email');
        $user->password = Hash::make(request('password'));
        $user->phone = request('full_phone');
        $user->save();
        return response()->redirectTo('/users');
    }
}
```

Bad is a contrib.

Single responsibility principle

In short, every command/model class only one responsibility.

Bad

```
class Function_Controller
{
    /**
    * Create a new controller instance.
    * @return void
    */
    public function __construct()
    {
        $this->middleware('auth');
    }
    /**
    * Show the application dashboard.
    * @return \Illuminate\Contracts\Support\Renderable|\Illuminate\Contracts\Foundation\Application|\Illuminate\Http\Response|\Illuminate\Contracts\View\View
    */
    public function index(Request $request)
    {
        $data = [];
        $data['total_users'] = \App\User::count();
        $data['total_posts'] = \App\Post::count();
        $data['total_comments'] = \App\Comment::count();
        $data['latest_users'] = \App\User::orderBy('id', 'desc')->limit(5)->get();
        $data['latest_posts'] = \App\Post::orderBy('id', 'desc')->limit(5)->get();
        $data['latest_comments'] = \App\Comment::orderBy('id', 'desc')->limit(5)->get();
        $data['popular_posts'] = \App\Post::orderBy('comments_count', 'desc')->limit(5)->get();
        $data['popular_users'] = \App\User::orderBy('posts_count', 'desc')->limit(5)->get();
        $data['mail_subject'] = request('mail_subject');
        $data['mail_from'] = request('mail_from');
        $data['mail_to'] = request('mail_to');
        $data['mail_text'] = request('mail_text');
        $data['mail_text_from_file'] = file_exists(storage_path('app/mail.txt')) ? file_get_contents(storage_path('app/mail.txt')) : null;
        $data['mail_sent'] = \Mail::send([], [], function ($message) use ($data) {
            $message->subject($data['mail_subject'])
                    ->from($data['mail_from'])
                    ->to($data['mail_to'])
                    ->setBody($data['mail_text'], 'text/plain');
            if ($data['mail_text_from_file']) {
                $message->setBody($data['mail_text_from_file'], 'text/plain');
            }
        });
        return view('home', compact('data'));
    }
}
```

Bad is a contrib.

Fat models, skinny controllers

It's all SO-related logic into Eloquent models or into Repository classes if you're using Query Builder or raw SQL queries.

Bad

```
public function index()
{
    $user = Auth::user();
    $profile = $user->profile ?: new Profile;
    $profile->fill(request()->all('first_name', 'last_name', 'middle_name'));
    $profile->save();
    return view('home', ['profile' => $profile]);
}
```

Good

```
public function index()
{
    return view('home', ['profile' => Auth::user()->profile()->firstOrNew()]);
}

// app/Http/Controllers/ProfileController.php
public function store(ProfileRequest $request)
{
    $profile = Auth::user()->profile ?: new Profile;
    $profile->fill($request->all());
    $profile->save();
    return view('home', ['profile' => $profile]);
}

// app/Models/Profile.php
class Profile extends Model
{
    /**
    * Fillable fields for the profile.
    * @var array
    */
    protected $fillable = ['first_name', 'last_name', 'middle_name'];
}
```

Bad is a contrib.

Validation

Move validation from controllers to Request classes.

Bad

```
public function storePost(Request $request)
{
    $this->validate(request(), [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
        'publish_at' => 'nullable|date',
    ]);
    ...
}
```

Good

```
// app/Http/Controllers/PostController.php
public function storePost(StorePostRequest $request)
{
    ...
}

// app/Http/Requests/StorePostRequest.php
class StorePostRequest extends FormRequest
{
    /**
    * Get the validation rules that apply to the request.
    * @return array
    */
    public function rules()
    {
        return [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
            'publish_at' => 'nullable|date',
        ];
    }
}
```

Bad is a contrib.

Business logic should be in service class

It controller class can be a hundred lines long, it happens. Big fat controllers are security climbers.

Bad

```
public function index()
{
    $articles = \App\Article::where('active', 1)->orderBy('created_at', 'desc')->get();
    $cache = \Cache::get('articles');
    if (!$articles && !$cache) {
        \Log::error('Articles and cache are empty!');
    }
    return view('articles', compact('articles', 'cache'));
}
```

Good

```
// app/Http/Controllers/ArticleController.php
use App\Services\ArticleService;
class ArticleController extends Controller
{
    /**
    * Display a listing of articles.
    * GET /articles
    * @return \Illuminate\Http\Response
    */
    public function index(ArticleService $articleService)
    {
        return view('articles', compact('articles', 'cache'));
    }
}

// app/Services/ArticleService.php
class ArticleService
{
    /**
    * Get all articles.
    * @return array
    */
    public function getAll()
    {
        $articles = \App\Article::where('active', 1)->orderBy('created_at', 'desc')->get();
        $cache = \Cache::get('articles');
        if (!$articles && !$cache) {
            \Log::error('Articles and cache are empty!');
        }
        return compact('articles', 'cache');
    }
}
```

Bad is a contrib.

Don't repeat yourself (DRY)

Reuse code whenever you can. DRY is a fitting goal in Laravel applications. Also, Blade templates, and Eloquent scopes etc.

Bad

```
public function getFullNameAttribute()
{
    return $this->attributes['first_name'] . ' ' . $this->attributes['middle_name'] . ' ' . $this->attributes['last_name'];
}

public function getFullNameAttribute()
{
    return $this->attributes['first_name'] . ' ' . $this->attributes['middle_name'] . ' ' . $this->attributes['last_name'];
}
```

Good

```
// app/Models/User.php
class User extends Model
{
    /**
    * Get the full name of the user.
    * @return string
    */
    public function getFullNameAttribute()
    {
        return $this->attributes['first_name'] . ' ' . $this->attributes['middle_name'] . ' ' . $this->attributes['last_name'];
    }
}
```

Bad is a contrib.

Prefer to use Eloquent over using Query Builder and raw SQL queries. Prefer collections over arrays

Eloquent models are quite readable and maintainable code. Also, Eloquent has great built-in tools like soft deletes, events, scopes, etc.

Bad

```
SELECT
    DISTINCT t.id, t.title, t.body
    FROM forum_topics AS t
    LEFT JOIN forum_posts AS p
        ON p.topic_id = t.id
        AND p.created
        AND p.deleted = 0
        AND p.id = (SELECT MAX(id) FROM forum_posts AS lp
            WHERE lp.topic_id = t.id
            AND lp.created
            AND lp.deleted = 0
            ORDER BY lp.created DESC LIMIT 1)
    WHERE t.deleted = 0
    ORDER BY t.created DESC, p.id DESC
    LIMIT 0, 20;
```

Good

```
$articles = Article::with('author.profile')->latest()->paginate(20);
```

Bad is a contrib.

Mass assignment

Bad

```
// app/Http/Controllers/ArticleController.php
use App\Article;
use App\Http\Requests\ArticleRequest;
class ArticleController extends Controller
{
    /**
    * Store a newly created resource in storage.
    * POST /articles
    * @return \Illuminate\Http\Response
    */
    public function store(ArticleRequest $request)
    {
        $article = new Article;
        $article->title = request('title');
        $article->slug = request('slug');
        $article->category_id = request('cat');
        $article->body = request('body');
        $article->save();
        return response()->redirectTo('/articles');
    }
}
```

Bad is a contrib.

Do not execute queries in Blade templates and use eager loading (N + 1 problem)

Bad for both reasons. Fix DB queries with the Eloquent:

Bad (slow + bad MVC)

```
@if (User::count() > 0)
    Hello!
@endif
```

Good (faster, but still violates MVC and SRP principles)

```
@if ($user_count && $user_count > 0)
    Hello!
@endif
```

Bad is a contrib.

Comment your code, but prefer descriptive method and variable names over comments

Bad

```
$ch = curl_init($to_fetch);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HEADER, false);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_MAXREDIRS, 10);
curl_setopt($ch, CURLOPT_TIMEOUT, 120);
curl_setopt($ch, CURLOPT_USERAGENT, "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko");
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, false);
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
curl_setopt($ch, CURLOPT_ENCODING, "");
curl_setopt($ch, CURLOPT_AUTOREFERER, true);
curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 120);
curl_setopt($ch, CURLOPT_PROTOCOLS, 2);
curl_setopt($ch, CURLOPT_SSLVERSION, 4);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post_data);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Expect:"));
curl_setopt($ch, CURLOPT_URL, $to_fetch);
$data = curl_exec($ch);
$error = curl_error($ch);
$errno = curl_errno($ch);
if (is_resource(curl_handle($ch))) {
    trace("Trying to close a resource which is still open");
}
curl_close($ch);
```

Bad is a contrib.

Do not put JS and CSS in Blade templates and do not put any HTML in PHP classes

Bad

```
$js_minified = 'var a = 1;';
$this->header('X-Js-Minified', $js_minified);
```

Bad is a contrib.

Use config and language files, constants instead of text in the code

Bad

```
// app/Http/Controllers/ArticleController.php
class ArticleController extends Controller
{
    /**
    * Store a newly created resource in storage.
    * POST /articles
    * @return \Illuminate\Http\Response
    */
    public function store(ArticleRequest $request)
    {
        $article = new Article;
        $article->title = request('title');
        $article->slug = request('slug');
        $article->category_id = request('cat');
        $article->body = request('body');
        $article->save();
        return response()->redirectTo('/articles');
    }
}
```

Bad is a contrib.

Use IoC container or facades instead of new Class

Their Eloq Series is using Static Eloquent Models and Unmaintainable Binding. Use IoC container or facades instead.

Bad

```
$post = new Post;
$post->save(['title' => 'Hello']);
```

Good

```
// app/Http/Controllers/PostController.php
use App\Post;
class PostController extends Controller
{
    /**
    * Store a newly created resource in storage.
    * POST /posts
    * @return \Illuminate\Http\Response
    */
    public function store(PostRequest $request)
    {
        $post = new Post;
        $post->save(['title' => 'Hello']);
        return response()->redirectTo('/posts');
    }
}
```

Bad is a contrib.

Do not get data from the `_env` -file directly

Place this data to config files instead and then use the `config()` helper function to use this data in an application.

Bad

```
$key = $_ENV['APP_KEY'];

$driver = $_SERVER['MAIL_DRIVER'];
```

Good

```
// config/app.php
'key' => env('APP_KEY'),

// .env file entry
MAIL_DRIVER=smtp

// app/Mail/Transport.php
$driver = config('mail.driver');
```

Bad is a contrib.

Store dates in the standard format. Use accessors and mutators to modify date format

Bad

```
// app/Models/Article.php
class Article extends Model
{
    /**
    * Get the attribute for the given index.
    * @param string $key
    * @return mixed
    */
    public function offset($key)
    {
        return $this->attributes[$key];
    }
}
```

Good

```
// app/Models/Article.php
class Article extends Model
{
    /**
    * Get the attribute for the given index.
    * @param string $key
    * @return mixed
    */
    public function offset($key)
    {
        return $this->attributes[$key];
    }
}
```

Bad is a contrib.