### Prinsip *single responsibility*

Kelas dan metode seharusnya hanya memiliki satu tanggung jawab.

Contoh buruk:

```
public function getFullNameAttribute()
{
    if (auth()->user() && auth()->user()->hasRole('client') && auth()->user()->isVerified()) {
        return 'Mr. ' . $this->first_name . ' ' . $this->middle_name . ' ' . $this->last_name;
    } else {
        return $this->first_name[0] . '. ' . $this->last_name;
    }
}
```

Contoh terbaik:

```
public function getFullNameAttribute()
{
    return $this->isVerifiedClient() ? $this->getFullNameLong() : $this->getFullNameShort();
}

public function isVerifiedClient()
{
    return auth()->user() && auth()->user()->hasRole('client') && auth()->user()->isVerified();
}

public function getFullNameLong()
{
    return 'Mr. ' . $this->first_name . ' ' . $this->middle_name . ' ' . $this->last_name;
}

public function getFullNameShort()
{
    return $this->first_name[0] . '. ' . $this->last_name;
}
```

⬆️ Kembali ke konten

### Model tebal, *controller* tipis

Masukkan semua logika terkait DB ke model *eloquent* atau ke dalam kelas repositori jika anda menggunakan *Query Builder* atau kueri SQL mentah.

Contoh buruk:

```
public function index()
{
    $clients = Client::verified()
        ->with(['orders' => function ($q) {
            $q->where('created_at', '>', Carbon::today()->subWeek());
        }])
        ->get();
```

```
    return view('index', ['clients' => $clients]);
}
```

Contoh terbaik:

```
public function index()
{
    return view('index', ['clients' => $this->client->getWithNewOrders()]);
}

class Client extends Model
{
    public function getWithNewOrders()
    {
        return $this->verified()
            ->with(['orders' => function ($q) {
                $q->where('created_at', '>', Carbon::today()->subWeek());
            }])
            ->get();
    }
}
```

🔝 Kembali ke konten

### Validasi

Pindahkan validasi dari *controller* ke kelas *request*.

Contoh buruk:

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

    ....
}
```

Contoh terbaik:

```
public function store(PostRequest $request)
{
    ....
}

class PostRequest extends Request
{
    public function rules()
    {
        return [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
            'publish_at' => 'nullable|date',
        ];
    }
}
```

🔝 Kembali ke konten

### *Business logic* harus di dalam kelas *services*

*Controller* harus hanya memiliki satu tanggung jawab, jadi pindahkan *business logic* dari *controller* ke kelas *service*.

Contoh buruk:

```
public function store(Request $request)
{
    if ($request->hasFile('image')) {
        $request->file('image')->move(public_path('images') . 'temp');
    }

    ....
}
```

Contoh terbaik:

```php
public function store(Request $request)
{
    $this->articleService->handleUploadedImage($request->file('image'));

    ....
}

class ArticleService
{
    public function handleUploadedImage($image)
    {
        if (!is_null($image)) {
            $image->move(public_path('images') . 'temp');
        }
    }
}
```

🔝 Kembali ke konten

### *Don't repeat yourself* (DRY)

Gunakan kembali kode ketika anda bisa. PSR membantu anda menghindari duplikasi. Juga, gunakan kembali *template blade*, *scope eloquent*, dll.

Contoh buruk:

```php
public function getActive()
{
    return $this->where('verified', 1)->whereNotNull('deleted_at')->get();
}

public function getArticles()
{
    return $this->whereHas('user', function ($q) {
            $q->where('verified', 1)->whereNotNull('deleted_at');
        })->get();
}
```

Contoh terbaik:

```php
public function scopeActive($q)
{
    return $q->where('verified', 1)->whereNotNull('deleted_at');
}

public function getActive()
{
    return $this->active()->get();
}

public function getArticles()
{
    return $this->whereHas('user', function ($q) {
            $q->active();
        })->get();
}
```

🔝 Kembali ke konten

### Lebih memilih menggunakan *Eloquent* daripada menggunakan *Query Builder* dan query SQL mentah. Lebih memilih *collections* daripada *array*

*Eloquent* memungkinkan anda menulis kode yang dapat dibaca dan *maintainable*. Dan, *Eloquent* memiliki *built-in tools* yang bagus seperti *soft deletes*, *events*, *scopes*, dll.

Contoh buruk:

```sql
SELECT *
FROM `articles`
WHERE EXISTS (SELECT *
              FROM `users`
              WHERE `articles`.`user_id` = `users`.`id`
```

```
                AND EXISTS (SELECT *
                            FROM `profiles`
                            WHERE `profiles`.`user_id` = `users`.`id`)
                AND `users`.`deleted_at` IS NULL)
AND `verified` = '1'
AND `active` = '1'
ORDER BY `created_at` DESC
```

Contoh terbaik:

```
Article::has('user.profile')->verified()->latest()->get();
```

⬆️ Kembali ke konten

### *Mass assignment*

Contoh buruk:

```
$article = new Article;
$article->title = $request->title;
$article->content = $request->content;
$article->verified = $request->verified;
// Add category to article
$article->category_id = $category->id;
$article->save();
```

Contoh terbaik:

```
$category->article()->create($request->validated());
```

⬆️ Kembali ke konten

### Jangan mengeksekusi kueri dalam *template blade* dan gunakan *eager loading* (masalah N + 1)

Contoh buruk (untuk 100 *user*, 101 kueri DB akan dieksekusi):

```
@foreach (User::all() as $user)
    {{ $user->profile->name }}
@endforeach
```

Contoh terbaik (untuk 100 *user*, 2 kueri DB akan dieksekusi):

```
$users = User::with('profile')->get();

...

@foreach ($users as $user)
    {{ $user->profile->name }}
@endforeach
```

⬆️ Kembali ke konten

### Komentari kode anda, tetapi lebih baik *method* dan nama variabel yang deskriptif daripada komentar

Contoh buruk:

```
if (count((array) $builder->getQuery()->joins) > 0)
```

Contoh lebih baik:

```
// Determine if there are any joins.
if (count((array) $builder->getQuery()->joins) > 0)
```

Contoh terbaik:

```
if ($this->hasJoins())
```

⬆️ Kembali ke konten

### Jangan letakkan JS dan CSS di *template blade* dan jangan letakkan HTML apa pun di kelas PHP

Contoh buruk:

```
let article = `{{ json_encode($article) }}`;
```

Contoh lebih baik:

```
<input id="article" type="hidden" value='@json($article)'>

Atau

<button class="js-fav-article" data-article='@json($article)'>{{ $article->name }}<button>
```

Dalam file javascript:

```
let article = $('#article').val();
```

Cara terbaik adalah dengan menggunakan *package* `PHP to JS` khusus untuk mentransfer data.

⬆️ Kembali ke konten

### Gunakan file *config*, *language*, dan konstanta daripada teks dalam kode

Contoh buruk:

```
public function isNormal()
{
    return $article->type === 'normal';
}

return back()->with('message', 'Your article has been added!');
```

Contoh terbaik:

```
public function isNormal()
{
    return $article->type === Article::TYPE_NORMAL;
}

return back()->with('message', __('app.article_added'));
```

⬆️ Kembali ke konten

### Gunakan *tools* standar Laravel yang diterima oleh komunitas

Selalu gunakan fungsi *built-in* bawaan laravel dan *packages* komunitas daripada menggunakan *packages* dan *tools* pihak ke-3. *Developer* manapun yang akan bekerja dengan aplikasi anda di masa mendatang perlu mempelajari *tools* baru. Dan juga, peluang untuk mendapatkan bantuan dari komunitas Laravel jauh lebih rendah saat anda menggunakan *packages* atau *tools* pihak ke-3. Jangan membuat klien Anda membayar untuk itu.

| Task | Tools *standar* | *Tools* pihak ke-3 |
|---|---|---|
| Authorization | Policies | Entrust, Sentinel and other packages |
| Compiling assets | Laravel Mix | Grunt, Gulp, 3rd party packages |
| Development Environment | Laravel Sail, Homestead | Docker |
| Deployment | Laravel Forge | Deployer and other solutions |
| Unit testing | PHPUnit, Mockery | Phpspec |
| Browser testing | Laravel Dusk | Codeception |
| DB | Eloquent | SQL, Doctrine |
| Templates | Blade | Twig |
| Working with data | Laravel collections | Arrays |
| Form validation | Request classes | 3rd party packages, validation in controller |

| | | |
|---|---|---|
| Authentication | Built-in | 3rd party packages, your own solution |
| API authentication | Laravel Passport, Laravel Sanctum | 3rd party JWT and OAuth packages |
| Creating API | Built-in | Dingo API and similar packages |
| Working with DB structure | Migrations | Working with DB structure directly |
| Localization | Built-in | 3rd party packages |
| Realtime user interfaces | Laravel Echo, Pusher | 3rd party packages and working with WebSockets directly |
| Generating testing data | Seeder classes, Model Factories, Faker | Creating testing data manually |
| Task scheduling | Laravel Task Scheduler | Scripts and 3rd party packages |
| DB | MySQL, PostgreSQL, SQLite, SQL Server | MongoDB |

⬆️ Kembali ke konten

### Ikuti konvensi penamaan Laravel

Ikuti PSR standards. Dan juga, ikuti konvensi penamaan yang diterima oleh komunitas Laravel:

| What | How | Good | Bad |
|---|---|---|---|
| Controller | singular | ArticleController | ~~ArticlesController~~ |
| Route | plural | articles/1 | ~~article/1~~ |
| Named route | snake_case with dot notation | users.show_active | ~~users.show-active, show-active-users~~ |
| Model | singular | User | ~~Users~~ |
| hasOne or belongsTo relationship | singular | articleComment | ~~articleComments, article_comment~~ |
| All other relationships | plural | articleComments | ~~articleComment, article_comments~~ |
| Table | plural | article_comments | ~~article_comment, articleComments~~ |
| Pivot table | singular model names in alphabetical order | article_user | ~~user_article, articles_users~~ |
| Table column | snake_case without model name | meta_title | ~~MetaTitle; article_meta_title~~ |
| Model property | snake_case | $model->created_at | ~~$model->createdAt~~ |
| Foreign key | singular model name with _id suffix | article_id | ~~ArticleId, id_article, articles_id~~ |
| Primary key | - | id | ~~custom_id~~ |
| Migration | - | 2017_01_01_000000_create_articles_table | ~~2017_01_01_000000_articles~~ |
| Method | camelCase | getAll | ~~get_all~~ |
| Method in resource controller | table | store | ~~saveArticle~~ |
| Method in test class | camelCase | testGuestCannotSeeArticle | ~~test_guest_cannot_see_article~~ |
| Variable | camelCase | $articlesWithAuthor | ~~$articles_with_author~~ |
| Collection | descriptive, plural | $activeUsers = User::active()->get() | ~~$active, $data~~ |
| Object | descriptive, singular | $activeUser = User::active()->first() | ~~$users, $obj~~ |
| Config and language files index | snake_case | articles_enabled | ~~ArticlesEnabled; articles-enabled~~ |
| View | kebab-case | show-filtered.blade.php | ~~showFiltered.blade.php, show_filtered.blade.php~~ |
| Config | snake_case | google_calendar.php | ~~googleCalendar.php, google-calendar.php~~ |

| | | | |
|---|---|---|---|
| Contract (interface) | adjective or noun | AuthenticationInterface | ~~Authenticatable, IAuthentication~~ |
| Trait | adjective | Notifiable | ~~NotificationTrait~~ |

⬆️ Kembali ke konten

### Gunakan sintaks yang lebih pendek dan lebih mudah dibaca jika memungkinkan

Contoh buruk:

```
$request->session()->get('cart');
$request->input('name');
```

Contoh terbaik:

```
session('cart');
$request->name;
```

Contoh:

| Sintaks umum | Sintaks pendek dan mudah dibaca |
|---|---|
| `Session::get('cart')` | `session('cart')` |
| `$request->session()->get('cart')` | `session('cart')` |
| `Session::put('cart', $data)` | `session(['cart' => $data])` |
| `$request->input('name'), Request::get('name')` | `$request->name, request('name')` |
| `return Redirect::back()` | `return back()` |
| `is_null($object->relation) ? null : $object->relation->id` | `optional($object->relation)->id` |
| `return view('index')->with('title', $title)->with('client', $client)` | `return view('index', compact('title', 'client'))` |
| `$request->has('value') ? $request->value : 'default';` | `$request->get('value', 'default')` |
| `Carbon::now(), Carbon::today()` | `now(), today()` |
| `App::make('Class')` | `app('Class')` |
| `->where('column', '=', 1)` | `->where('column', 1)` |
| `->orderBy('created_at', 'desc')` | `->latest()` |
| `->orderBy('age', 'desc')` | `->latest('age')` |
| `->orderBy('created_at', 'asc')` | `->oldest()` |
| `->select('id', 'name')->get()` | `->get(['id', 'name'])` |
| `->first()->name` | `->value('name')` |

⬆️ Kembali ke konten

### Gunakan *IoC Container* atau *facades* daripada kelas baru

Sintaks Kelas baru membuat penggabungan yang sempit antar kelas dan memperumit proses pengujian. Gunakan *facades* atau *IoC container* sebagai gantinya.

Contoh buruk:

```
$user = new User;
$user->create($request->validated());
```

Contoh terbaik:

```
public function __construct(User $user)
{
    $this->user = $user;
}

....
```

```
$this->user->create($request->validated());
```

⬆️ Kembali ke konten

### Jangan mendapatkan data dari file `.env` secara langsung

Alihkan data ke file konfigurasi sebagai gantinya dan kemudian gunakan fungsi `config ()` untuk menggunakan data dalam aplikasi.

Contoh buruk:

```
$apiKey = env('API_KEY');
```

Contoh terbaik:

```
// config/api.php
'key' => env('API_KEY'),

// Use the data
$apiKey = config('api.key');
```

⬆️ Kembali ke konten

### Simpan tanggal dalam format standar. Gunakan *accessors* dan *mutators* untuk mengubah format tanggal

Contoh buruk:

```
{{ Carbon::createFromFormat('Y-d-m H-i', $object->ordered_at)->toDateString() }}
{{ Carbon::createFromFormat('Y-d-m H-i', $object->ordered_at)->format('m-d') }}
```

Contoh terbaik:

```
// Model
protected $dates = ['ordered_at', 'created_at', 'updated_at'];
public function getSomeDateAttribute($date)
{
    return $date->format('m-d');
}

// View
{{ $object->ordered_at->toDateString() }}
{{ $object->ordered_at->some_date }}
```

⬆️ Kembali ke konten

### Praktik bagus lainnya

Jangan pernah menaruh logika apa pun di file `route`.

Minimalkan penggunaan *vanilla* PHP di *template blade*.

⬆️ Kembali ke konten