# Laravel 5. Хороші практики

05.05.2019 Cinex Веб-розробка 0


ПОШУК

ПОШУК ...

ПОЗНАЧКИ

APACHE AUTORUN BASH BITCOIN BLOCKCHAIN COMPOSER CRYPTOCURRENCY CSS DATABASE DJANGO DOCKER ELOQUENT ENGLISH ENUM GAMES GIMP GIT HTML JAVA JAVASCRIPT JQUERY LARAVEL MERCURIAL PARSING PHOTOSHOP PHP PHPSTORM

Це не переказ кращих практик на кшталт SOLID, патернів і ін. З адаптацією під Laravel. Тут зібрані саме практики, які ігноруються в реальних Laravel проектах. Також, рекомендую ознайомитися з хорошими практиками в контексті PHP. Дивіться також обговорення хороших практик Laravel.

## Принцип єдиної відповідальності (Single responsibility principle)

Кожен клас і метод повинні виконувати лише одну функцію.

Погано:

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

Добре:

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

public function isVerfiedClient()
{
    return auth()->user() && auth()->user()->hasRole('client') && auth()->user()->isVeri
}

public function getFullNameLong()
{
```

PLAYONLINUX PLUGINS PROBLEM SOLVING PYTHON REGEXP SEO SOFTWARE ENGINEERING SPRING SQL TELEGRAM TIPS AND TRICKS TIPS AND TRICKS UNIT TESTING WORDPRESS ВІРШІ ШПАРГАЛКИ

### ОСТАННІ НОТАТКИ

20 Laravel Eloquent порад і трюків
15.12.2019

Як зробити скріншот сайту по URL на PHP
04.08.2019

Docker Cheat Sheet
05.07.2019

Гарячі клавіші Ubuntu Linux
01.07.2019

Git happens! 6 типових помилок Git і як їх виправити
22.06.2019

### НЕДАВНІ КОМЕНТАРІ

```
13      return 'Mr. ' . $this->first_name . ' ' . $this->middle_name . ' ' . $this->last_nam
14  }
15
16  public function getFullNameShort()
17  {
18      return $this->first_name[0] . '. ' . $this->last_name;
19  }
```

## Тонкі контролери, товсті моделі

За своєю суттю, це лише один з окремих випадків принципу єдиної відповідальності. Виносьте роботу з даними в моделі при роботі з Eloquent або в репозиторії при роботі з Query Builder або "сирими" SQL запитами.

Погано:

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

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

Добре:

```php
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();
    }
}
```

## Валідація

Дотримуючись принципів тонкого контролера і SRP, виносьте валідацію з контролера в Request класи.

Погано:

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

    ....
}
```

Добре:

```php
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',
        ];
    }
}
```

## Бізнес логіка в сервіс-класах

Контролер повинен виконувати тільки свої прямі обов'язки, тому виносьте всю бізнес логіку в окремі класи та сервіс класи.

Погано:

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

    ....
}
```

Добре:

```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');
        }
    }
}
```

## Не повторюйся (DRY)

Цей принцип закликає вас перевикористати код всюди, де це можливо. Якщо ви дотримуєтеся принципу SRP, ви вже уникаєте повторень, але Laravel дозволяє вам також перевикористати уявлення, частини Eloquent запитів і т.д.

Погано:

```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();
}
```

Добре:

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

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

```

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

## Віддавайте перевагу Eloquent конструктору запитів (query builder) і сирим запитам в БД. Віддавайте перевагу роботу з колекціями роботі з масивами

Eloquent дозволяє писати максимально читабельний код, а змінювати функціонал додатка незрівнянно легше. У Eloquent також є ряд зручних і потужних інструментів.

Погано:

```mysql
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
```

Добре:

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

## Використовуйте масове заповнення (mass assignment)

Погано:

```php
$article = new Article;
$article->title = $request->title;
$article->content = $request->content;
$article->verified = $request->verified;
// Прив'язати статтю до категорії.
$article->category_id = $category->id;
$article->save();
```

Добре:

```php
$category->article()->create($request->all());
```

## Не виконуйте запити в поданнях і використовуйте нетерпляче завантаження (проблема N + 1)

Погано (буде виконаний 101 запит в БД для 100 користувачів):

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

Добре (буде виконано 2 запити в БД для 100 користувачів):

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

...
```

```
4
5 @foreach ($users as $user)
6     {{ $user->profile->name }}
7 @endforeach
```

Коментуйте код, віддайте перевагу читабельності імен методів коментарям

Погано:

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

Краще:

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

Добре:

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

## Виносьте JS і CSS з шаблонів Blade і HTML з PHP коду

Погано:

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

Добре:

```
<input id="article" type="hidden" value="{{ json_encode($article) }}">

Або

```

```
5 <button class="js-fav-article" data-article="{{ json_encode($article) }}">{{ $article->no
```

В Javascript файлі:

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

Ще краще використовувати спеціалізований пакет для передачі даних з бекенд у фронтенд.

## Конфіги, мовні файли і константи замість тексту в коді

Безпосередньо в коді не повинно бути ніякого тексту.

Погано:

```php
1 public function isNormal()
2 {
3     return $article->type === 'normal';
4 }
5
6 return back()->with('message', 'Ваша стаття була успішно додана');
```

Добре:

```php
1 public function isNormal()
2 {
3     return $article->type === Article::TYPE_NORMAL;
4 }
5
6 return back()->with('message', __('app.article_added'));
```

# Використовуйте інструменти і практики прийняті спільнотою

Laravel має вбудовані інструменти для вирішення завдань, які часто зустрічаються. Віддавайте перевагу користуватися ними використанню сторонніх пакетів і інструментів. Laravel розробнику, який прийшов в проект після вас, доведеться вивчати і працювати з новим для нього інструментом, з усіма наслідками, що випливають. Отримати допомогу від спільноти буде також набагато важче. Не змушуйте клієнта або роботодавця платити за ваші велосипеди.

| Задача | Стандартні інструменти | Нестандартні інструменти |
|---|---|---|
| Авторизація | Політики | Entrust, Sentinel і др. пакети, особисті рішення |
| Робота с JS, CSS и пр. | Laravel Mix | Grunt, Gulp, сторонні пакети |
| Середовище розробки | Homestead | Docker |
| розгортання додатків | Laravel Forge | Deployer і багато інших |
| Тестування | Phpunit, Mockery | Phpspec |
| e2e тестування | Laravel Dusk | Codeception |
| Робота з БД | Eloquent | SQL, будівник запитів, Doctrine |
| Шаблони | Blade | Twig |
| Рибота з даними | Коллекції Laravel | Массиви |
| Валідація форм | Request класи | Сторонні пакети, валідація в контролері |
| Аутентифікація | Вбудований функціонал | Сторонні пакети, власне рішення |

| Задача | Стандартні інструменти | Нестандартні інструменти |
|---|---|---|
| Аутентифікація API | Laravel Passport | Сторонні пакети, що використовують JWT, OAuth |
| Створення API | Вбудований функціонал | Dingo API і інші пакети |
| Робота зі структурою БД | Миграції | Робота з БД напряму |
| Локалізація | Вбудований функціонал | Сторонні пакети |
| Обмін даними в реальному часі | Laravel Echo, Pusher | Пакети і робота з веб сокетами безпосередньо |
| Генерація тестових даних | Seeder класи, фабрики моделей, Faker | Ручне заповнення і пакети |
| Планування завдань | Планувальник задач Laravel | Скрипти і сторонні пакети |
| БД | MySQL, PostgreSQL, SQLite, SQL Server | MongoDb |

## Дотримуйтесь угоди спільноти про іменування

Дотримуйтесь стандартів PSR при написанні коду.

Також, дотримуйтесь інші Угоди про іменування:

| Що | Правило | Принято | Не прин |
|---|---|---|---|
| Контролер | од. ч. | ArticleController | ~~ArticlesController~~ |

| Що | Правило | Принято | Не прин |
|---|---|---|---|
| Маршрути | мн. ч. | articles/1 | ~~article/1~~ |
| Імена маршрутів | snake_case | users.show_active | ~~users.show-active~~ ~~active-users~~ |
| Модель | од. ч. | User | ~~Users~~ |
| Відносини hasOne и belongsTo | од. ч. | articleComment | ~~articleComments,~~ ~~article_comment~~ |
| Всі інші відносини | мн. ч. | articleComments | ~~articleComment,~~ ~~article_comments~~ |
| Таблиця | мн. ч. | article_comments | ~~article_comment,~~ ~~articleComments~~ |
| Pivot таблиця | імена моделей в алфавітному порядку в од. ч. | article_user | ~~user_article, articl~~ |
| Стовпець в таблиці | snake_case без імені моделі | meta_title | ~~MetaTitle; article_~~ |
| Зовнішній ключ | им'я моделі од. ч. і _id | article_id | ~~ArticleId, id_article~~ |
| Первинний ключ | - | id | ~~custom_id~~ |

| Що | Правило | Принято | Не прин |
|---|---|---|---|
| Миграція | - | 2017_01_01_000000_create_articles_table | ~~2017_01_01_0000~~ |
| Метод | camelCase | getAll | ~~get_all~~ |
| Метод в контролері ресурсів | таблиця | store | ~~saveArticle~~ |
| Метод в тесті | camelCase | testGuestCannotSeeArticle | ~~test_guest_canno~~ |
| Переменні | camelCase | $articlesWithAuthor | ~~$articles_with_aut~~ |
| Колекція | описове, мн. ч. | $activeUsers = User::active()->get() | ~~$active, $data~~ |
| Об'єкт | описове, ед. ч. | $activeUser = User::active()->first() | ~~$users, $obj~~ |
| Індекси в конфіги і мовних файлах | snake_case | articles_enabled | ~~ArticlesEnabled; a enabled~~ |
| Подання | snake_case | show_filtered.blade.php | ~~showFiltered.blad show-filtered.blac~~ |
| Конфігураційний файл | snake_case | google_calendar.php | ~~googleCalendar.p calendar.php~~ |
| Контракт (інтерфейс) | прикметник або іменник | Authenticatable | ~~AuthenticationInt IAuthentication~~ |
| Трейт | прикметник | Notifiable | ~~NotificationTrait~~ |

## Короткий і читається синтаксис там, де це можливо

Погано:

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

Добре:

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

Ще приклади:

| Часто використовується синтаксис | Більш короткий і читається синтаксис |
|---|---|
| Session::get('cart') | session('cart') |
| $request->session()->get('cart') | session('cart') |
| Session::put('cart', $data) | session(['cart' => $data]) |
| $request->input('name') | $request->name |
| Request::get('name') | request('name') |
| return Redirect::back() | return back() |
| return view('index')->with('title', $title)->with('client', $client) | return view('index', compact('title', 'client')) |

# Використовуйте IoC або фасади замість new Class

Впровадження класів через синтаксис new Class створює сильне поєднання між частинами програми та ускладнює тестування. Використовуйте контейнер або фасади.

Погано:

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

Добре:

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

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

Інші поради та практики

Не ставте логіку в маршрутах.

Намагайтеся не використовувати сирого PHP в шаблонах Blade.

Джерело

LARAVEL TIPS AND TRICKS ШПАРГАЛКИ

« ПОПЕРЕДНІЙ ДАЛІ »


Case Styles: Camel, Pascal, Snake, and Kebab Case

Laravel 5. Маловідомі можливості Eloquent, про які ви повинні знати

## ЗАЛИШТЕ ПЕРШИЙ КОМЕНТАР

# Залишити коментар

Вашу адресу електронної пошти не буде опубліковано

Коментувати

Ім'я*

Email *

Збережіть моє ім'я, електронну пошту у цьому веб-переглядачі наступного разу, коли я коментуватиму.

ОПУБЛІКУВАТИ КОМЕНТАР