25 порад і хитрощів в Laravel

15.04.2019 Cinex Веб-розробка 0

Не так уже й давно був такий період, коли PHP і його спільнота м'яко кажучи недолюблювали. І майже що кожен день головним жартом було сказати те, наскільки PHP жахливий. Давайте подивимося, які нові, що висміюють PHP, статті в блогах постять сьогодні?

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

> *Так, на жаль співтовариство і екосистема мови просто не були на тому ж рівні, що і інші сучасні мови.*

Так, на жаль співтовариство і екосистема мови просто не були на тому ж рівні, що і інші сучасні мови. Здавалося, що PHP судилося жити своїм окремим життям у вигляді брудної WordPress теми.

Але потім досить дивно як речі стали змінюватися, та до того ж дуже швидко. Прямо як при перемішування горщика відьми, один за іншим з нізвідки стали з'являтися інноваційні нові проекти. Напевно найбільш значущим з таких проектів став Composer: менеджер залежностей для PHP (не такий як Bundler для Ruby і NPM для Node). Якщо раніше в минулому PHP розробникам доводилося використовувати PEAR, то тепер завдяки Composer, ми можемо просто змінити JSON файл, і миттєво отримати необхідні залежності. Ось профайлер, ось фреймворк ... все готово за лічені секунди!

У переповненому світі PHP фреймворків, в той момент коли CodeIgniter почав видихатися, Taylor Otwell створює Laravel, який стає улюбленцем спільноти. Laravel своїм простим і елегантним синтаксисом зробив процес створення додатків на PHP дуже простим і цікавим! А в четвертій версії фреймворка, нарешті при використанні Composer, в співтоваристві все встало на свої місця.

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

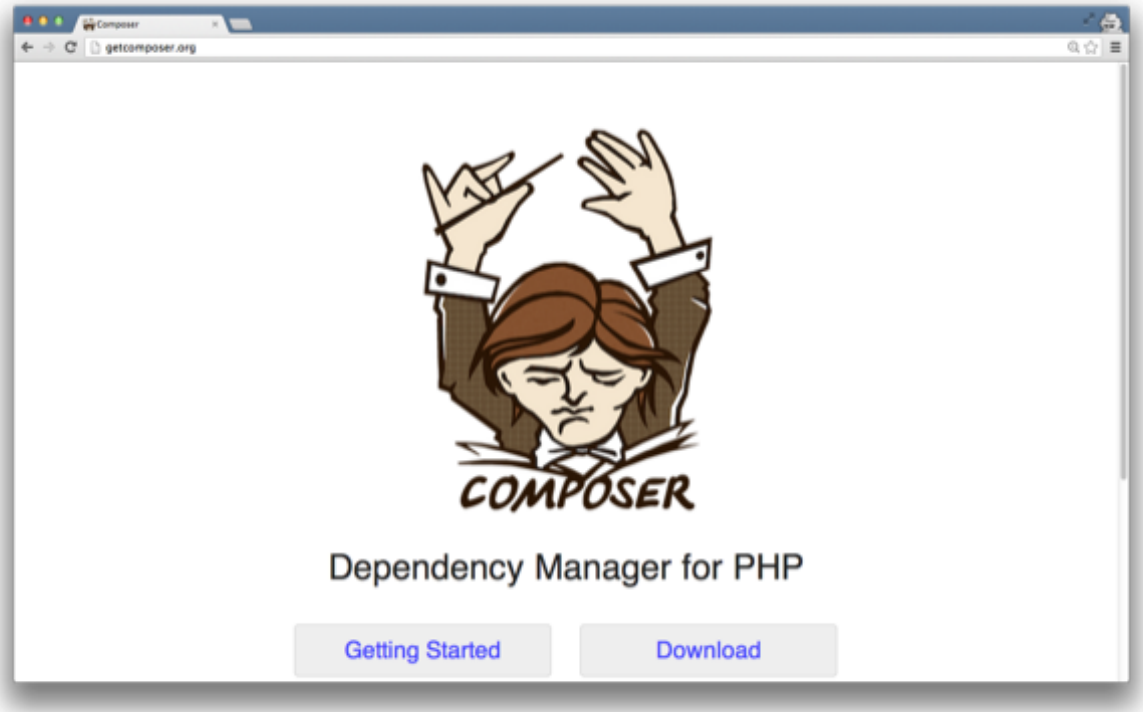
Як зробити скріншот сайту по URL на PHP
04.08.2019

Docker Cheat Sheet
05.07.2019

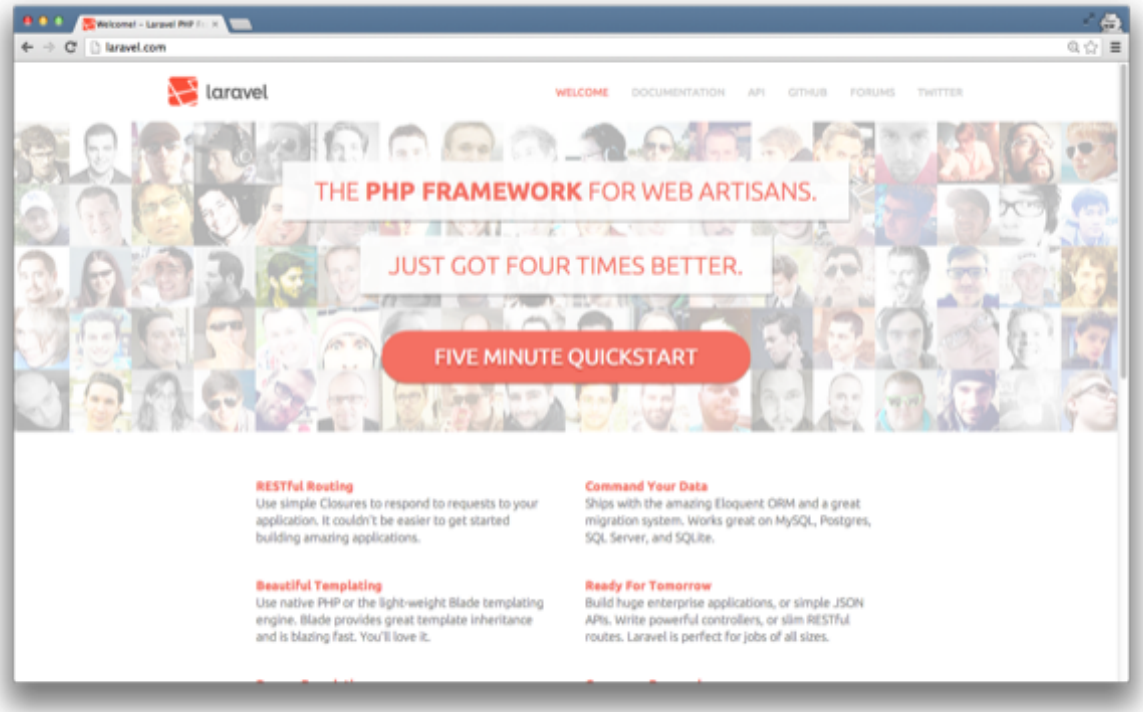
Гарячі клавіші Ubuntu Linux
01.07.2019

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

НЕДАВНІ КОМЕНТАРІ



Потрібні міграції (контроль версій для вашої бази даних)? Готово. А як на рахунок потужної реалізації Active-Record? Безумовно з цим успішно впорається **Eloquent**. Як на рахунок можливостей тестування? Звичайно ж. А маршрутизація? Будь ласка. А що ж на рахунок трестируємого HTTP прошарку? Завдяки Composer Laravel може використовувати велику кількість чудових компонентів Symfony. Якщо вам щось потрібно, то найімовірніше це вже є в Laravel.



У той час як PHP вже практично став нікому не цікавий - раптом завдяки Laravel і Composer майбутнє стає вже більш радісним. І так відкинемо зайве і заглянемо глибше всередину фреймворка, подивимося, що він може нам запропонувати.

1. Запити Eloquent

Laravel пропонує одну з найпотужніших реалізацій Active-Record в PHP світі. Припустимо у вас є таблиця **orders** разом з моделлю **Eloquent Order**:

```php
class Order extends Eloquent {}
```

Ми можемо легко виконати будь-яку кількість запитів до бази даних, використовуючи простий і елегантний PHP. Немає необхідності мати справу з брудним SQL. Давайте виберемо все замовлення.

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

Готово. Або можливо ці замовлення слід повернути відсортовані за датою. Це просто:

```php
$orders = Order::orderBy('release_date', 'desc')->get();
```

А що якщо замість отримання запису нам потрібно зберегти нове замовлення в базі. Звичайно ж, ми можемо зробити це.

```php
$order = new Order;
$order->title = 'Xbox One';
$order->save();
```

Готово! За допомогою Laravel завдання, які раніше здавалися громіздкими тепер стають до смішного простими.

2. Гнучка маршрутизація

Laravel є унікальним в тому, що його можна абсолютно по-різному використовувати. Віддаєте перевагу просту схожу на Sinatra систему маршрутизації? Звичайно, Laravel може досить легко це надати, використовуючи замикання.

```php
Route::get('orders', function()
{
    return View::make('orders.index')
        ->with('orders', Order::all());
});
```

Такий підхід може бути зручний для маленьких проектів, але швидше за все в більшості ваших проектів ви будете використовувати контролери. І нормально, Laravel і це вміє!

```php
Route::get('orders', 'OrdersController@index');
```

Готово. Чи помічаєте як Laravel ростет разом з вашими потребами? Така властивість пристосовуватися до потреб розробників і робить цей фреймворк таким популярним.

3. Легкі зв'язки

Що ми робимо, коли нам потрібно визначити зв'язки? Наприклад, завдання швидше за все буде належати користувачеві. Як це уявити в Laravel? Якщо ми уявімо, що необхідні таблиці в базі вже присутні, то нам залишиться тільки налаштувати відповідні моделі Eloquent.

```php
class Task extends Eloquent {
    public function user()
    {
        return $this->belongsTo('User');
    }
}

class User extends Eloquent {
    public function tasks()
    {
        return $this->hasMany('Task');
    }
}
```

І готово! Давайте спробуємо отримати всі завдання користувача з id 1. Ми можемо зробити це в два рядки коду.

```php
$user = User::find(1);
$tasks = $user->tasks;
```

Однак оскільки ми визначили відносини в обидва кінці, якщо ми замість цього хочемо, отримати користувача, пов'язаного із завданням, то ми теж зможемо легко це зробити.

```php
$task = Task::find(1);
$user = $task->user;
```

4. Прив'язка моделі до форми

Часто може виявитися корисним прив'язати форму до моделі. Очевидним прикладом цього є, коли ви хочете змінити деякі записи в базі даних. За допомогою прив'язки моделі до форми, ми можемо миттєво заповнити поля форми значеннями з відповідного запису в таблиці.

```
{{ Form::model($order) }}
    <div>
        {{ Form::label('title', 'Title:') }}
        {{ Form::text('title') }}
    </div>

    <div>
        {{ Form::label('description', 'Description:') }}
        {{ Form::textarea('description') }}
    </div>
{{ Form::close() }}
```

Так як форма тепер прив'язана до певного примірника **Order**, то поля введення будуть відображати коректні дані з таблиці. Просто!

5. Кешування запитів до бази даних

Занадто багато запитів до бази, і дуже швидко ваш додаток стане повільним як равлик. На щастя Laravel пропонує простий механізм для кешування цих запитів, використовуючи для цього лише один виклик метода.

Давайте отримаємо всі questions з бази даних, але закешуємо запит, так як навряд чи ця таблиця буде часто оновлюватися.

```php
$questions = Question::remember(60)->get();
```

От і все! Тепер весь наступний час цей запит буде залишатися закешованим і ми не будемо смикати базу.

У Laravel версії 5 і вище даний функціонал, на жаль вирізаний!

6. Компонувальники View

Ви обов'язково зустрінете ситуацію, коли в кількох різних відображеннях потрібно отримати доступ до одних і тих самих даних. Хорошим прикладом цього є панель навігації, яка відображає список тегів.

Щоб контролери містили якомога менше коду, Laravel пропонує компонувальник відображень для управління подібного роду речами.

```php
View::composer('layouts.nav', function($view)
{
    $view->with('tags', ['tag1', 'tag2']);
});
```

За допомогою цього коду тепер в будь-який час, коли буде завантажений файл **layouts/nav.blade.php**, то ми отримаємо доступ до змінної **$tags** з певним значенням.

7. Проста Аутентифікація

Laravel надає максимально простий підхід до аутентифікації. Просто беремо масив з обліковими даними, які зазвичай були отримані через форму логіна, і передаємо їх в Auth :: attempt(). Якщо

передані значення відповідають даним, які містяться в таблиці users, то користувач буде залогінений.

```php
$user = [
    'email' => 'email',
    'password' => 'password'
];

if (Auth::attempt($user))
{
    // user is now logged in!
    // Access user object with Auth::user()
}
```

А що якщо нам потрібно розлогінити користувача, коли наприклад буде відкритий URL **/logout**? Це так само дуже просто.

```php
Route::get('logout', function()
{
    Auth::logout();

    return Redirect::home();
});
```

8. Ресурси

Працювати за принципом REST в Laravel дуже просто. Щоб зареєструвати контролер з ресурсами, потрібно просто викликати **Route :: resource()** наступним чином:

```php
Route::resource('orders', 'OrdersController');
```

За допомогою цього коду Laravel зареєструє 8 маршрутів.

- GET /orders

- GET /orders/:order
- GET /orders/create
- GET /orders/:order/edit
- POST /orders
- PUT /orders/:order
- PATCH /orders/:order
- DELETE /orders/:order

Потім відповідний контролер можна буде згенерувати за допомогою такої команди з командного рядка:

```
1 php artisan controller:make OrdersController
```

У цьому новому створеному контролері кожен метод буде відповідати кожному маршруту, який ми зробили вище. Наприклад маршрут **/orders** буде відповідати методу **index**, **/orders/create** - методу **create** і т.д.

Тепер у нас є потужні можливості, які дозволяють з легкістю створювати RESTful додатки і API.

9. Шаблонізатор Blade

Хоча звичайно ж PHP за своєю природою сам є мовою шаблонів, але він особливо не змінився, щоб стати дійсно хорошим в цьому відношенні. Але і це нормально, так як Laravel пропонує власний шаблонізатор Blade, щоб заповнити цю прогалину. Просто називаємо свої файли з розширенням **.blade.php** і вони автоматично будуть парситись шаблонізатором. Тепер ми можемо робити подібні речі:

```
1 @if ($orders->count())
2     <ul>
```

```
        @foreach($orders as $order)
            <li>{{ $order->title }}</li>
        @endforeach
    </ul>
@endif
```

10. Можливості для тестування

Так як Laravel використовує Composer, то у нас одразу ж з коробки є підтримка фреймворку для тестування PHPUnit. Встановлюємо фреймворк і просто виконує **phpunit** з командного рядка.

І навіть більше того, Laravel надає набір хелперів для тестування найзагальніших ситуацій в функціональних тестах.

Давайте перевіримо, що домашня сторінка повертає 200 код відповіді.

```php
public function test_home_page()
{
    $this->call('GET', '/');
    $this->assertResponseOk();
}
```

Або ж нам потрібно впевнитися, що коли відправляється форма контактів, то користувача перенаправлять назад на домашню сторінку з відповідним спливаючих повідомленням.

```
public function test_contact_page_redirects_user_to_home_page()
{
    $postData = [
        'name' => 'Joe Example',
        'email' => 'email-address',
        'message' => 'I love your website'
    ];

    $this->call('POST', '/contact', $postData);

    $this->assertRedirectedToRoute('home', null, ['flash_message']);
```

```
12 }
```

11. Компонент Remote

Починаючи з Laravel 4.1 є можливість легко написати Artisan команду для доступу до сервера по SSH, і виконати там будь-які команди. Все це легко можна здійснити за допомогою **SSH** фасаду:

```php
SSH::into('production')->run([
    'cd /var/www',
    'git pull origin master'
]);
```

Просто передаємо масив команд в метод run(), а Laravel вже сам все зробить! Тепер, залишається тільки виконати Artisan команду, запускаємо **php artisan command: make DeployCommand**, і вставляємо потрібний код в метод команди **fire** для швидкого створення команди деплоя.

12. Події

Laravel пропонує елегантну реалізацію шаблону спостерігач, який ви можете використовувати в своїх додатках. Можна прослуховувати власні події фреймворка, такі як **illuminate.query**, або навіть викликати і відловлювати свої власні.

Правильне використання подій в додатку матиме величезний ефект на його структуру і можливість його подальшої підтримки.

```powershell
Event::listen('user.signUp', function()
{
    // do whatever needs to happen
    // when a new user signs up
});
```

Як і більшість речей у Laravel, якщо ви віддаєте перевагу використовувати ім'я класу замість замикання, то без проблем можна використовувати його. Laravel просто отримає його з IoC контейнера.

```php
Event::listen('user.signUp', 'UserEventHandler');
```

13. Перегляд всіх маршрутів

```
~ casts [master] php artisan routes
+--------+-----------------------------------------------+-----------------------+-----------------------------------+---------------+--------------+
| Domain | URI                                           | Name                  | Action                            | Before Filters | After Filters |
+--------+-----------------------------------------------+-----------------------+-----------------------------------+---------------+--------------+
|        | POST /admin/receivables                       |                       | Closure                           |               |              |
|        | GET /free                                     |                       | Closure                           | force.ssl     |              |
|        | GET /support/create                           | support               | SupportController@create          | force.ssl     |              |
|        | POST /support/store                           |                       | SupportController@store           | force.ssl     |              |
|        | GET /blog                                     | blog.index            | BlogController@index              | force.ssl     |              |
|        | GET /blog/{blog}                              | blog.show             | BlogController@show               | force.ssl     |              |
|        | GET /feed                                     |                       | RssController@index               | force.ssl     |              |
|        | GET /testimonials                             |                       | PagesController@testimonials      | force.ssl     |              |
|        | GET /                                         | home                  | PagesController@home              | force.ssl     |              |
|        | GET /about                                    | about                 | PagesController@about             | force.ssl     |              |
|        | GET /faq                                      | faq                   | PagesController@faq               | force.ssl     |              |
|        | GET /lessons                                  | lessons.index         | LessonsController@index           | force.ssl     |              |
|        | GET /lessons/{lessons}                        | lessons.show          | LessonsController@show            | force.ssl     |              |
|        | GET /tags                                     | tags.index            | TagsController@index              | force.ssl     |              |
|        | GET /tags/{tags}                              | tags.show             | TagsController@show               | force.ssl     |              |
|        | GET /search                                   |                       | SearchController@index            | force.ssl     |              |
|        | GET /search/{query}                           |                       | SearchController@search           | force.ssl     |              |
|        | GET /series                                   | series.index          | SeriesController@index            | force.ssl     |              |
|        | GET /series/{series}                          | series.show           | SeriesController@show             | force.ssl     |              |
|        | GET /series/{seriesId}/episodes/{episodeId}   | episode               | EpisodesController@show           | force.ssl     |              |
|        | GET /join                                     |                       | SubscriptionsController@join      | force.ssl     |              |
|        | GET /subscriptions/create                     | subscriptions.create  | SubscriptionsController@create    | force.ssl     |              |
|        | POST /subscriptions                           | subscriptions.store   | SubscriptionsController@store     | force.ssl     |              |
|        | PUT /subscriptions/{subscriptions}            | subscriptions.update  | SubscriptionsController@update    | force.ssl     |              |
|        | PATCH /subscriptions/{subscriptions}          |                       | SubscriptionsController@update    | force.ssl     |              |
```

У міру зростання додатка, простір всіх зареєстрованих маршрутів може виявитися досить важким заняттям. Це особливо справедливо, коли ви не дуже дбайливо ставитеся до свого **routes.php** файлу.

Laravel пропонує зручну команду **routes**, яка покаже всі зареєстровані маршрути разом з відповідними їм методами контролерів.

```
1 php artisan routes
```

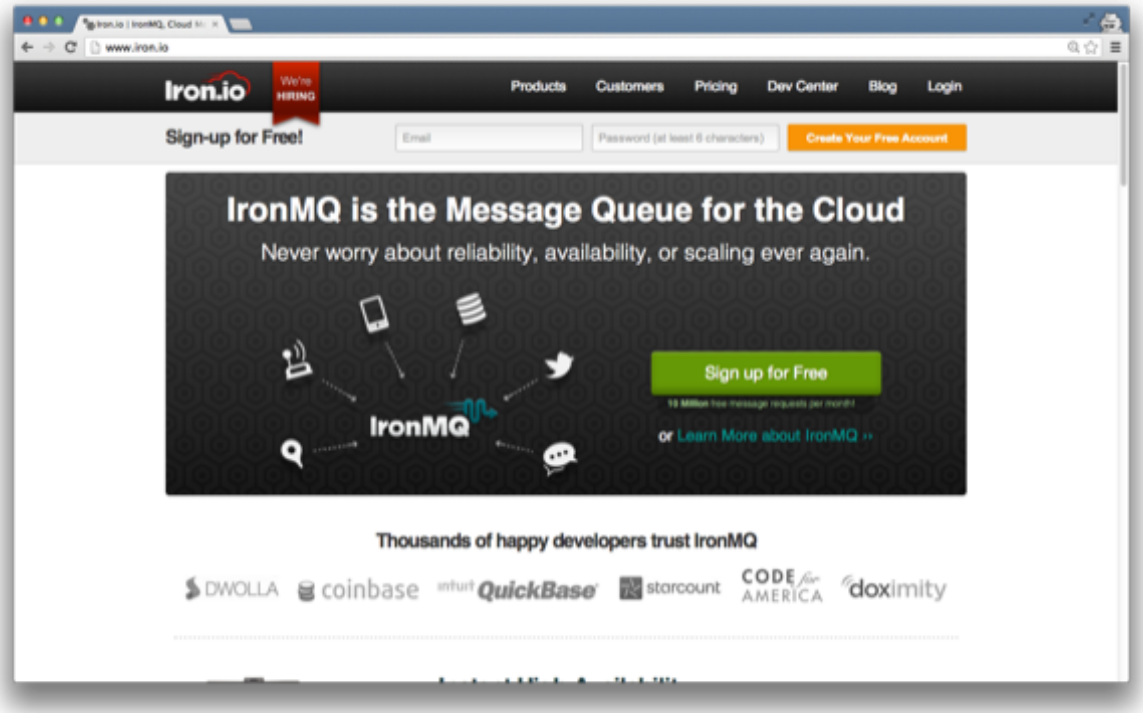
14. Черги

Розглянь процес реєстрації користувача в нашому додатку. Швидше за все тут буде задіяно деяку кількість подій. Потрібно буде оновити дані в базі даних, додати нового користувача в список розсилок, повинен відкритися новий рахунок для клієнта, відправлено лист-запрошення і так далі. На жаль такі дії зазвичай можуть зайняти тривалий час.

Навіщо змушувати користувача чекати, поки все це виконатися? Замість це ми можемо виконати ці дії в тлі.

```
1 Queue::push('SignUpService', compact('user'));
```

Напевно найцікавішою частиною є те, що Laravel прекрасно підтримує черги iron.io. Це означає, що ми можемо використовувати всю міць черг навіть не маючи ніякого досвіду роботи з "worker" або "daemon". Просто реєструємо URL для точки входу використовуючи зручну команду **php artisan queue:subscribe** і iron.io буде пінгувати цей URL, кожен раз коли в чергу буде додана нова задача.



Прості кроки для швидкої продуктивності!

15. Проста валідація

Laravel приходить нам на допомогу так само коли потрібно валідація. Ми можемо використовувати клас **Validator** з інтуїтивно зрозумілим інтерфейсом. Просто передаємо об'єкт, який потрібно затверджувати і список правил в метод **make**, про решту подбає Laravel.

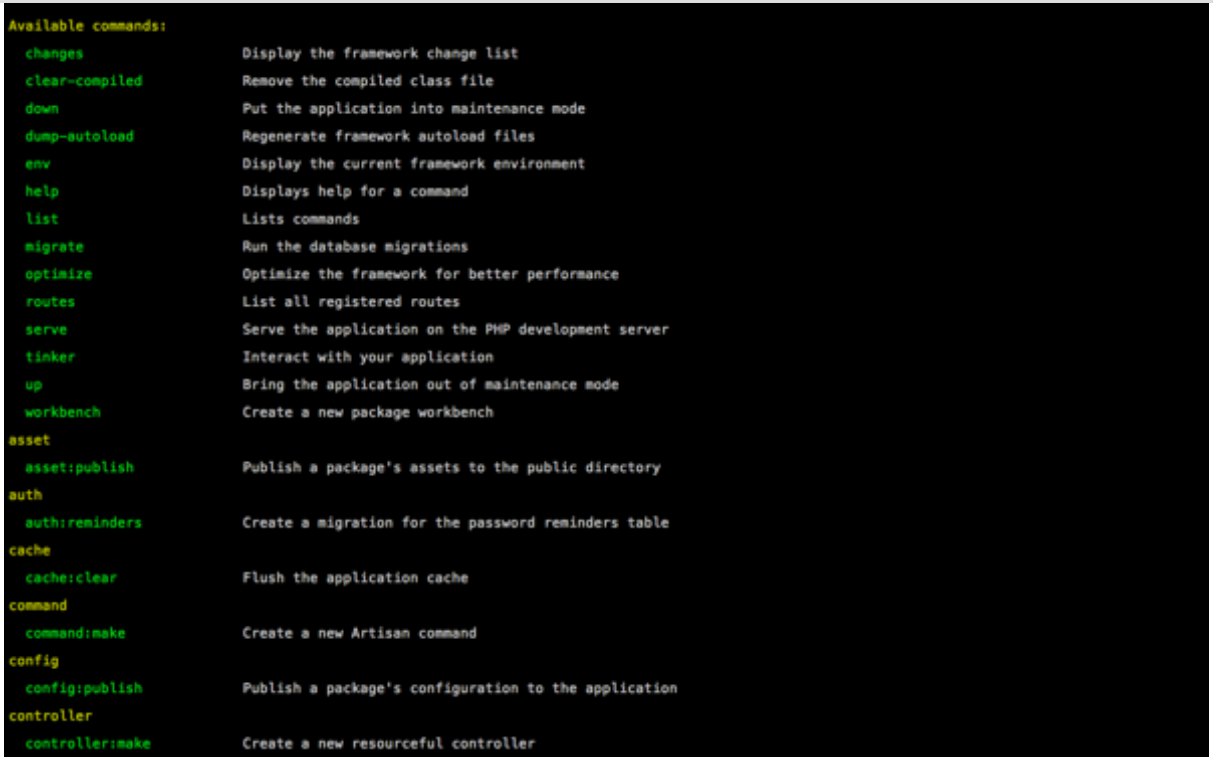
```php
$order = [
    'title' => 'Wii U',
    'description' => 'Game console from Nintendo'
];

```

```php
$rules = [
    'title' => 'required',
    'description' => 'required'
];

$validator = Validator::make($order, $rules);

if ($validator->fails())
{
    var_dump($validator->messages()); // validation errors array
}
```

Зазвичай такий код слід розташовувати в моделі, щоб викликати валідацію можна було одним методом:

```php
$order->isValid();
```

16. Тінкер



Особливо в перші моменти вивчення Laravel, може виявитися вельми корисним можливість повозитися з його ядром і можливостями. Artisan команда **tinker** допоможе нам в цьому.

> *Починаючи з версії 4.1 команда tinker стала ще більш потужною, тепер вона використовує популярний компонент Boris.*

```
$ php artisan tinker

> $order = Order::find(1);
> var_dump($order->toArray());
> array(...)
```

17. Міграції

Міграції слід розглядати, як систему контролю версій вашої бази даних. У будь-який момент можна відкотити всі міграції назад, перезапустити їх і навіть більше. Можливо справжня міць криється в можливості викочування додатки на продакшн, і виконання всього лише однієї команди **php artisan migrate**, щоб оновити схему бази даних.

Щоб підготувати схему для нової таблиці users, просто виконуємо:

```
1 php artisan migrate:make create_users_table
```

Ця команда створить новий файл з міграцією, який ви потім зможете заповнити відповідно до ваших потреб. Як тільки закінчите, виконуємо команду **php artisan migrate** і таблиця буде створена. Готово! Потрібно відкинути назад створення? Легко! **php artisan migrate:rollback**.

Ось приклад схеми для таблиці з часто вживаними питаннями.

```php
public function up()
{
    Schema::create('faqs', function(Blueprint $table) {
        $table->integer('id', true);
        $table->text('question');
        $table->text('answer');
        $table->timestamps();
    });
}

public function down()
{
    Schema::drop('faqs');
}
```

Зверніть увагу, що метод **drop()** є інверсією методу **up()**. Ось що легко нам дозволяє відкотити міграції. Це ж набагато простіше ніж мати справу з купою SQL запитів?

18. Генератори

Хоча Laravel і пропонує кілька зручних генераторів, але неймовірно корисним виявився пакет "Laravel 4 Generators". Можна генерувати ресурси, файли сідів, проміжні таблиці, а також міграції.

У попередньому розділі ми мусили самим писати схему. Однак в допомогою пакета генераторів замість цього ми можемо зробити так:

```
1 php artisan generate:migration create_users_table --fields="username:string, password:str
```

Про інше подбає генератор. Виходить, що за допомогою всього двох команд ви можете підготувати і створити нову таблицю в базі даних.

> *Пакет Laravel 4 Generators **може бути встановлений** через Composer.*

19. Команди

Як було помічено раніше, є кілька сценаріїв, коли може знадобитися написати свої власні команди. Вони можуть використовуватися для генерації файлів, розгортання додатків і іншого.

Так як це досить поширене завдання, то Laravel максимально спрощує процес створення власних команд.

```
1 php artisan command:make MyCustomCommand
```

Ця команда створить весь необхідний код, готовий для написання своєї власної команди. Потім в новій створеній команді **app/commands/MyCustomCommand.php** заповнюємо ім'я команди і її опис:

```php
protected $name = 'command:name';
protected $description = 'Command description.';
```

І нарешті в методі **fire()** виконуємо будь-які дії, які нам потрібні. Як тільки буде готове, залишиться тільки зареєструвати нову команду в Artisan, в файлі **app/start/Artisan.php**.

```php
Artisan::add(new MyCustomCommand);
```

Вірите чи ні, але це все! Тепер можна викликати вашу власну команду з терміналу.

20. Моки Фасадів

Laravel рясно використовує шаблон проектування фасад. Це дозволяє отримати чистий синтаксис в стилі статичних викликів, який ви безсумнівно полюбите (**Route::get()**, **Config::get()** і т.д.), який як і раніше надає за лаштунками всі можливості для тестування.

Так як ці класи для фасадів дістаються з IoC контейнера, ми можемо легко підмінити з екземпляри моками для тестування. Це дозволяє робити такі речі:

```php
Validator::shouldReceive('make')->once();
```

Так, ми викликаємо shouldReceive прямо біля фасаду. Але за лаштунками Laravel використовує чудовий фреймворк для тестування Mockery. Це означає, що ми можемо вільно використовувати фасади в нашому коді, і він як і раніше залишається на 100% тестованим.

21. Помічники форм

Оскільки створення форми часто може виявитися досить трудомістким завданням, будівник форм в Laravel може полегшити цей процес. Ось декілька прикладів:

```
{{ Form::open() }}
    {{ Form::text('name') }}
    {{ Form::textarea('bio') }}
    {{ Form::selectYear('dob', date('Y') - 80, date('Y')) }}
{{ Form::close() }}
```

Як на рахунок таких завдань, як запам'ятовування введених даних з останньої відправки форми? Laravel робить це автоматично!

22. IoC контейнер

В ядрі Laravel знаходиться його потужний IoC контейнер, який допомагає в управлінні залежностями класів. Відзначимо також, що контейнер може автоматично створювати екземпляри класів без будь-якої конфігурації!

Просто вказуємо типи залежностей в конструкторі і під час створення об'єкта, Laravel використовує PHP Reflection API щоб прочитати ці залежності і спробувати з вставити для нас.

```php
public function __construct(MyDependency $thing)
{
    $this->thing = $thing;
}
```

Коли ви запитуєте клас з IoC контейнера, то дозвіл всіх його залежностей відбувається автоматично.

```php
$myClass = App::make('MyClass');
```

> *Важливо відзначити, що контролери завжди дістаються з IoC контейнера. Тому можна вказати тип залежностей для вашого контролера і Laravel зробить все можливе, щоб вбудувати їх для вас.*

23. Оточення

У той час як для маленьких проектів цілком згодиться і одне оточення, для додатків більш серйозних розмірів буде потрібно вже кілька різних оточень. Development, testing, production ... всі вони є необхідними і вимагає свою окрему конфігурацію.

Можливо ваше тестове оточення використовує для тестування бази даних, яка знаходиться в пам'яті. А оточення для розробки використовує інші ключі API. Швидше за все ваше бойове оточення використовує інший рядок з'єднання з базою.

На щастя Laravel спрощує для нас і це завдання. Погляньмо на файл **bootstrap/start.php** в нашому додатку.

Це базова демонстрація установки як **local** так і **production** оточень на основі адресного рядка в браузері.

```php
$env = $app->detectEnvironment(array(
    'local' => array('localhost'),
    'production' => array('*.com')
));
```

Хоча це і працює, але чесно кажучи, краще використовувати для таких речей змінні оточення. Не турбуйтесь; Це все теж здійснимо в Laravel! Замість цього просто повертаємо функцію з методу **detectEnvironment** в об'єкті контейнера.

```php
$env = $app->detectEnvironment(function()
{
    return getenv('ENV_NAME') ?: 'local';
});
```

Тепер якщо не була встановлена змінна оточення (що ви повинні будете робити для production), оточення за замовчуванням **local**.

24. Проста конфігурація

І знову Laravel робить підхід до конфігурації додатка максимально простим. Створюємо папку всередині **app/config**, яка збігається з необхідним оточенням, і будь-які файли конфігурації всередині її будуть використані, якщо використовується відповідне оточення. Таким чином щоб, скажімо, встановити різні API ключі білінгу для розробки, можна зробити так:

```php
<?php app/config/development/billing.php

return [
    'api_key' => 'your-development-mode-api-key'
];
```

Конфігурація переключиться автоматично. Просто пишемо **Config::get ('billing.api_key')** і Laravel коректно визначить який файл потрібно буде прочитати.

25. Подальше вивчення

Коли справа доходить до освіти, Laravel співтовариство, незважаючи на свій порівняно молодий вік, є дуже доброзичливим. У трохи більше ніж за рік було випущено півдюжини різних книг - з усіх питань розробки Laravel.

Можна вивчити все, починаючи від основ, і тестування і закінчуючи підтримкою великих додатків!

COMPOSER ELOQUENT LARAVEL TIPS AND TRICKS

« ПОПЕРЕДНІЙ
Популярні плагіни для Laravel

ДАЛІ »
20 хитрощів в Laravel Eloquent про які ви не знали

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