Користувальницький поштовий драйвер в Laravel 5.8

31.05.2019 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

Одна з багатьох смакот, яку пропонує Laravel, - це розсилка. Ви можете легко налаштовувати і відправляти електронні листи через кілька популярних сервісів, і в них навіть є помічник по веденню журналу для розробки.

```php
Mail::send('emails.welcome', ['user' => $user], function ($m) use ($user) {
    $m->to($user->email, $user->name)->subject('Welcome to the website');
});
```

Це відправить електронного листа новому зареєстрованому користувачу на веб-сайті, використовуючи уявлення emails.welcome. Це стало ще простіше з Laravel 5.8, які використовують поштові повідомлення (але старий синтаксис все ще діє).

Ось приклад:

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

```
# Generate a new mailable class
php artisan make:mail WelcomeMail
```

```php
// app/Mail/WelcomeMail.php

class WelcomeUser extends Mailable
{

    use Queueable, SerializesModels;

    public $user;

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

    public function build()
    {
        return $this->view('emails.welcome');
    }
}
```

```php
// routes/web.php

Route::get('/', function () {
    $user = User::find(2);

    \Mail::to($user->email)->send(new WelcomeUser($user));

    return "done";
});
```

Laravel також забезпечує хорошу відправну точку для відправки листів на етапі розробки з використанням драйвера **log**, а у виробництві - за допомогою **smtp**, **sparkpost**, **mailgun** і т. д. У більшості випадків це нормально, але не може охопити всі доступні служби! У цьому уроці ми дізнаємося, як розширити існуючу систему драйверів пошти, щоб додати нашу власну.

Щоб наш приклад був простим і зрозумілим, ми збираємося записувати наші електронні листи в таблицю БД.

Створення Service Provider

Переважний спосіб зробити це - створити постачальника послуг, який буде взаємодіяти з нашим додатком під час завантаження і реєструвати наші сервіси. Давайте почнемо з створення нового постачальника послуг, використовуючи помічника командного рядка **artisan**.

```
1 php artisan make:provider DBMailProvider
```

Це створить новий клас в нашій папці **app/Providers**. Якщо ви знайомі з постачальниками послуг Laravel, ви знаєте, що ми розширюємо клас **ServiceProvider** і визначаємо методи **boot** і **register**. Детальніше про провайдерів ви можете **прочитати в документації**.

Використання Mail Provider

Замість використання батьківського класу постачальника послуг, ми можемо скористатися ярликом і розширити існуючий клас **Illuminate\Mail\MailServiceProvider**. Це означає, що метод реєстрації вже реалізований.

```php
1 // vendor/Illuminate/Mail/MailServiceProvider.php
2
3 public function register()
4 {
5     $this->registerSwiftMailer();
6
7     // ...
8 }
```

Метод **registerSwiftMailer** повертає відповідний драйвер транспорту на основі значення конфігурації **mail.driver**. Тут ми можемо виконати перевірку перед викликом батьківського методу **registerSwiftMailer** і повернути наш власний менеджер транспорту.

```php
// app/Providers/DBMailProvider.php

function registerSwiftMailer()
{
    if ($this->app['config']['mail.driver'] == 'db') {
        $this->registerDBSwiftMailer();
    } else {
        parent::registerSwiftMailer();
    }
}
```

Использование Transport Manager

Laravel дозволяє екземпляр **swift.mailer** з IOC, який повинен повертати екземпляр **Swift_Mailer** **SwiftMailer**. Нам потрібно прив'язати наш екземпляр поштового Swift до контейнера.

```php
private function registerMailSwiftMailer()
{
    $this->app->singleton('swift.mailer', function () {
        return new Swift_Mailer(new DBTransport());
    });
}
```

Ви можете думати про транспортний об'єкт, як про фактичний драйвер. Якщо ви перевірите простір імен **Illuminate\Mail\Transport**, ви знайдете різні класи транспорту для кожного драйвера (наприклад: **LogTransport**, **SparkPostTransport** і т. д.).

Клас **Swift_Mailer** вимагає примірника **Swift_Transport**, який ми можемо задовольнити, розширюючи клас **Illuminate\Mail\Transport\Transport**. Це має виглядати приблизно так.

```php
```

```php
namespace App\Mail\Transport;

use Illuminate\Mail\Transport\Transport;
use Illuminate\Support\Facades\Log;
use Swift_Mime_Message;
use App\Emails;

class DBTransport extends Transport
{

    public function __construct()
    {
    }


    public function send(Swift_Mime_Message $message, &$failedRecipients = null)
    {
        Emails::create([
            'body'    => $message->getBody(),
            'to'      => implode(',', array_keys($message->getTo())),
            'subject' => $message->getSubject()
        ]);
    }

}
```

Єдиний метод, який ми повинні реалізувати тут - це метод **send**. Він відповідає за логіку відправки пошти, і в цьому випадку він повинен реєструвати наші електронні листи в базі даних. Що стосується нашого конструктора, ми можемо залишити його порожнім на даний момент, тому що нам не потрібні ніякі зовнішні залежності.

Метод **$message->getTo()** завжди повертає асоціативний масив листів і імен одержувачів. Ми отримуємо список електронних листів, використовуючи функцію **array_keys**, а потім поділяємо їх, щоб отримати строку.

Log Email to DB

Наступним кроком є створення необхідної міграції для нашої таблиці бази даних.

```
php artisan make:migration create_emails_table --create="emails"
```

```php
class CreateEmailsTable extends Migration
{

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('emails', function (Blueprint $table) {
            $table->increments('id');
            $table->text('body');
            $table->string('to');
            $table->string('subject');
            $table->timestamps();
        });
    }


    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('emails');
    }
}
```

Наша міграція містить тільки текст повідомлення, тему і адресу електронної пошти одержувача, але ви можете додати стільки деталей, скільки захочете. Перевірте визначення класу **Swift_Mime_Message**, щоб побачити список доступних полів.

Тепер нам потрібно створити нову модель для взаємодії з нашою таблицею і додати необхідні поля в масив з можливістю заповнення.

```
php artisan make:model Emails
```

```php
class Emails extends Model
{

    protected $fillable = [
        'body',
        'to',
        'subject'
    ];
}
```

Відправка листів

Добре, прийшов час перевірити, що у нас так далеко. Ми почнемо з додавання нашого провайдера в список провайдерів в файлі **config/app.php**.

```php
return [
    // ...

    'providers' => [
        // ...

        App\Providers\DBMailProvider::class,

        // ...
    ],

    // ...
];
```

Потім ми реєструємо нашу поштову драйвер для **db** всередині файлу **config/mail.php**.

```php
return [
    'driver' => 'db',

```

```
    // ...
];
```

Єдина частина, що залишилася - відправити тестове повідомлення електронної пошти, щоб переглянути, чи було воно зареєстровано в базі даних. Я збираюся відправити електронний лист, коли URL відпрацює. ось код:

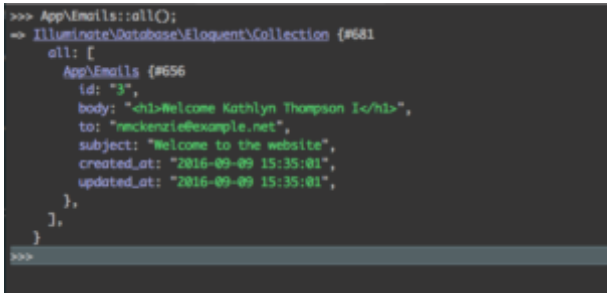
```php
// routes/web.php

Route::get('/', function () {
    $user = User::find(2);

    Mail::send('emails.welcome', ['user' => $user], function ($m) use ($user) {
        $m->to($user->email, $user->name)->subject('Welcome to the website');
    });

    return "done";
})
```

Після переходу по домашньому маршруту ми можемо запустити команду **php artisan tinker**, щоб перевірити записи таблиці електронних листів.



Висновок

У цій статті ми побачили, як розширити систему поштових драйверів для перехоплення листів з метою налагодження. Однією з речей, якими я захоплююся в Laravel, є його безпрецедентна

розширюваність: ви можете змінювати або розширювати практично будь-яку функціональність від маршрутизатора і IOC до Mail і за її межами.

LARAVEL

« ПОПЕРЕДНІЙ

Як написати команду artisan в Laravel, щоб зареєструвати супер адміністратора

ДАЛІ »

Стильна форма пошуку - Search Box

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