**eBook Gratuit**

# APPRENEZ Laravel

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#laravel

# Table des matières

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: laravel

It is an unofficial and free Laravel ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Laravel.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapitre 1: Commencer avec Laravel

## Remarques

# Communauté Slack de Laravel StackOverflow

*Arrive bientôt*

# Tutoriel vedette

Commencer avec Laravel

# Directives de contribution

*Arrive bientôt*

# Guide de style de contribution

*Arrive bientôt*

# A propos de Laravel

Créé par Taylor Otwell en tant que framework web PHP open source gratuit, Laravel a pour objectif de faciliter et d'accélérer le processus de développement des applications Web avec un goût prononcé pour la simplicité.

Il suit le modèle architectural modèle – vue-contrôleur ( MVC ) ainsi que la norme de codage PSR-2 et la norme de chargement automatique PSR-4 .

Exécuter un développement piloté par les tests ( TDD ) à Laravel est amusant et facile à mettre en œuvre.

Hébergé sur GitHub et disponible à l' adresse https://github.com/laravel/laravel , Laravel se vante d'une architecture de micro-services , ce qui le rend extrêmement extensible et cela, facilement, avec l'utilisation de tiers sur mesure ou existants. paquets.

## Caractéristiques principales

### MVC

Laravel utilise le modèle MVC, il y a donc trois parties du cadre qui fonctionnent ensemble: les modèles, les vues et les contrôleurs. Les contrôleurs sont la partie principale de la majeure partie du travail. Ils se connectent aux modèles pour obtenir, créer ou mettre à jour des données et afficher les résultats sur les vues, qui contiennent la structure HTML réelle de l'application.

### Moteur de modèle de lame

Laravel est livré avec un moteur de template appelé Blade. Blade est assez facile à utiliser, mais puissant. L'une des caractéristiques que le moteur de template Blade ne partage pas avec les autres modèles populaires est sa permissivité; permettant l'utilisation de code PHP simple dans les fichiers de moteur de templates Blade.

Il est important de noter que les fichiers du moteur de `.blade` Blade ont `.blade` ajoutés aux noms de fichiers juste avant l'habituel fichier `.php` qui n'est rien d'autre que l'extension du fichier. En tant que tel, `.blade.php` est l'extension de fichier résultante pour les fichiers de modèles Blade. Les fichiers du moteur de modèle de lame sont stockés dans le répertoire resources / views.

### Routage & Middleware

Vous pouvez définir les URL de votre application à l'aide d'itinéraires. Ces itinéraires peuvent contenir des données variables, se connecter à des contrôleurs ou être intégrés dans des middlewares. Middelware est un mécanisme de filtrage des requêtes HTTP. Ils peuvent être utilisés pour interagir avec les requêtes avant qu'elles n'atteignent les contrôleurs et peuvent ainsi modifier ou rejeter les demandes.

### Artisan

Artisan est l'outil en ligne de commande que vous pouvez utiliser pour contrôler certaines parties de Laravel. De nombreuses commandes sont disponibles pour créer des modèles, des contrôleurs et d'autres ressources nécessaires au développement. Vous pouvez également écrire vos propres commandes pour étendre l'outil de ligne de commande Artisan.

### Eloquent ORM

Pour connecter vos modèles à différents types de bases de données, Laravel propose son propre ORM avec un large éventail de fonctions avec lesquelles travailler. Le framework fournit également la migration et l'amorçage et propose également des restaurations.

### Gestion des événements

Le framework est capable de gérer les événements dans l'application. Vous pouvez créer des écouteurs d'événement et des gestionnaires d'événement similaires à ceux de NodeJs.

## Versions

| Version | Date de sortie |
|---|---|
| 1.0 | 2011-06-09 |
| 2.0 | 2011-11-24 |
| 3.0 | 2012-02-22 |
| 3.1 | 2012-03-27 |
| 3.2 | 2012-05-22 |
| 4.0 | 2013-05-28 |
| 4.1 | 2013-12-12 |
| 4.2 | 2014-06-01 |
| 5.0 | 2015-02-04 |
| 5.1 (LTS) | 2015-06-09 |
| 5.2 | 2015-12-21 |
| 5.3 | 2016-08-24 |
| 5.4 | 2017-01-24 |

# Examples

## Bienvenue dans la documentation de tag Laravel!

**Laravel** est un framework PHP bien connu. Ici, vous apprendrez tout sur Laravel. En commençant par une méthode *aussi simple que la* connaissance de la programmation orientée objet vers la rubrique avancée de développement de paquets Laravel.

Ceci, comme toutes les autres balises de documentation Stackoverflow, est une documentation pilotée par la communauté, donc si vous avez déjà des expériences sur Laravel, partagez vos connaissances en ajoutant vos propres sujets ou exemples! N'oubliez pas de consulter notre **guide de contribution** sur ce sujet pour en savoir plus sur la manière de contribuer et le guide de style que nous avons élaboré pour nous assurer que nous pouvons offrir la meilleure expérience aux personnes qui veulent en savoir plus sur Laravel.

Plus que cela, nous sommes très heureux que vous veniez, espérons que nous pouvons vous voir souvent ici!

## Guide de démarrage

Le guide de démarrage est une navigation personnalisée que nous avons commandée par nous-

mêmes pour faciliter la navigation sur les sujets, en particulier pour les débutants. Cette navigation est ordonnée par niveau de difficulté.

# Commencer

Installation

# Vues Laravel

Lame: introduction

Lame: Variables et structures de contrôle

Ou

**Installation d'ici**

1. Obtenir le compositeur d' ici et l'installer
2. Obtenez Wamp d' ici , installez-le et définissez la variable d'environnement de PHP
3. Obtenir le chemin d'accès à `www` et tapez la commande:

```
composer create-project --prefer-dist laravel/laravel projectname
```

Pour installer une version spécifique de Laravel, accédez à `www` et tapez commande:

```
composer create-project --prefer-dist laravel/laravel=DESIRED_VERSION projectname
```

Ou

**Via Laravel Installer**

Tout d'abord, téléchargez l'installateur Laravel en utilisant Composer:

```
composer global require "laravel/installer"
```

Assurez-vous de placer le `$HOME/.composer/vendor/bin` (ou le répertoire équivalent de votre système d'exploitation) dans votre $ PATH pour que l'exécutable `laravel` puisse être localisé par votre système.

Une fois installée, la commande `laravel new` créera une nouvelle installation Laravel dans le répertoire que vous spécifiez. Par exemple, `laravel new blog` va créer un répertoire nommé `blog` contenant une nouvelle installation de Laravel avec toutes les dépendances de Laravel déjà installées:

```
laravel new blog
```

Lire Commencer avec Laravel en ligne: https://riptutorial.com/fr/laravel/topic/794/commencer-avec-laravel

# Chapitre 2: Artisan

## Syntaxe

- php artisan [commande] [options] [arguments]

## Paramètres

| Commander | La description |
|---|---|
| clair-compilé | Supprimer le fichier de classe compilé |
| vers le bas | Mettre l'application en mode maintenance |
| env | Afficher l'environnement actuel du framework |
| Aidez-moi | Affiche l'aide pour une commande |
| liste | Listes de commandes |
| émigrer | Exécuter les migrations de base de données |
| optimiser | Optimiser le cadre pour de meilleures performances |
| servir | Servir l'application sur le serveur de développement PHP |
| bricoler | Interagir avec votre application |
| en haut | Mettre l'application hors du mode maintenance |
| nom de l'application | Définir l'espace de noms de l'application |
| auth: clear-resets | Rincer les jetons de réinitialisation du mot de passe expiré |
| cache: effacer | Videz le cache de l'application |
| cache: table | Créer une migration pour la table de base de données du cache |
| config: cache | Créez un fichier de cache pour un chargement plus rapide de la configuration |
| config: effacer | Supprimer le fichier de cache de configuration |
| db: graine | Semer la base de données avec des enregistrements |
| événement: générer | Générer les événements et les écouteurs manquants en fonction de l'inscription |

| Commander | La description |
| --- | --- |
| clé: générer | Définir la clé d'application |
| faire: auth | Scaffold basic vues de connexion et d'enregistrement et itinéraires |
| faire: console | Créer une nouvelle commande Artisan |
| faire: contrôleur | Créer une nouvelle classe de contrôleur |
| faire: événement | Créer une nouvelle classe d'événement |
| faire: travail | Créer une nouvelle classe d'emplois |
| faire: auditeur | Créer une nouvelle classe d'écouteur d'événement |
| faire: middleware | Créer un nouveau cours de middleware |
| faire: migration | Créer un nouveau fichier de migration |
| faire: modèle | Créer une nouvelle classe de modèle Eloquent |
| faire: politique | Créer une nouvelle classe de politique |
| faire: fournisseur | Créer une nouvelle classe de fournisseur de services |
| faire une requête | Créer une nouvelle classe de demande de formulaire |
| faire: semoir | Créer une nouvelle classe de semoir |
| faire: test | Créer une nouvelle classe de test |
| migrer: installer | Créer le référentiel de migration |
| migrer: rafraîchir | Réinitialiser et réexécuter toutes les migrations |
| migrer: réinitialiser | Annulation de toutes les migrations de base de données |
| migrer: rollback | Restaurer la dernière migration de base de données |
| migrer: statut | Afficher le statut de chaque migration |
| queue: échoué | Énumérer tous les travaux de file d'attente ayant échoué |
| queue: table échouée | Créer une migration pour la table de base de données des travaux de la file d'attente ayant échoué |
| file d'attente: flush | Videz tous les travaux de la file d'attente ayant échoué |
| file d'attente: oublie | Supprimer un travail de file d'attente ayant échoué |

| Commander | La description |
|---|---|
| queue: écouter | Écouter une file d'attente donnée |
| queue: redémarrer | Redémarrez les démons de travail de file d'attente après leur travail en cours |
| queue: réessayer | Réessayer un travail de file d'attente ayant échoué |
| queue: table | Créer une migration pour la table de la base de données des travaux de file d'attente |
| file d'attente: travail | Traiter le prochain travail dans une file d'attente |
| route: cache | Créer un fichier cache de routage pour un enregistrement plus rapide des itinéraires |
| itinéraire: clair | Supprimer le fichier cache de la route |
| route: liste | Liste de tous les itinéraires enregistrés |
| calendrier: exécuter | Exécuter les commandes planifiées |
| session: table | Créer une migration pour la table de base de données de session |
| vendeur: publier | Publier tous les actifs publiables des packages de fournisseurs |
| vue: effacer | Effacer tous les fichiers de vue compilés |

# Examples

## introduction

Artisan est un utilitaire qui peut vous aider à effectuer des tâches répétitives spécifiques avec des commandes bash. Il couvre de nombreuses tâches, notamment: travailler avec les **migrations** et l' **amorçage des** bases de données, effacer le **cache** , créer les fichiers nécessaires à la configuration de l' **authentification** , **créer de** nouveaux *contrôleurs, modèles, classes d'événements* et bien plus encore.

> Artisan est le nom de l'interface de ligne de commande incluse avec Laravel. Il fournit un certain nombre de commandes utiles pour votre utilisation tout en développant votre application.

Pour afficher une liste de toutes les commandes Artisan disponibles, vous pouvez utiliser la commande list:

```
php artisan list
```

Pour en savoir plus sur les commandes disponibles, il suffit de faire précéder son nom du mot-clé **help** :

```
php artisan help [command-name]
```

### Liste de tous les itinéraires enregistrés filtrés par plusieurs méthodes

```
php artisan route:list --method=GET --method=POST
```

Cela inclura toutes les routes qui acceptent simultanément les méthodes `GET` et `POST` .

### Exécuter les commandes de Laravel Artisan en utilisant le code PHP

Vous pouvez également utiliser les commandes Laravel Artisan de vos itinéraires ou contrôleurs.

Pour exécuter une commande en utilisant le code PHP:

```
Artisan::call('command-name');
```

Par exemple,

```
Artisan::call('db:seed');
```

### Créer et enregistrer une nouvelle commande artisan

Vous pouvez créer de nouvelles commandes via

`php artisan make:command [commandName]`

Cela créera donc la classe de commande [commandName] dans le répertoire `app/Console/Commands` .

A l'intérieur de cette classe, vous trouverez `protected $signature` variables `protected $signature` et `protected $description` , il représente le nom et la `protected $description` de votre commande qui sera utilisée pour décrire votre commande.

Après avoir créé la commande, vous pouvez enregistrer votre commande dans la classe `app/Console/Kernel.php` où vous trouverez la propriété `commands` .

Vous pouvez donc ajouter votre commande dans le tableau $ command comme:

```
protected $commands = [
    Commands\[commandName]::class
];
```

et puis je peux utiliser ma commande via la console.

donc comme exemple j'ai nommé ma commande comme

```
protected $signature = 'test:command';
```

Donc, chaque fois que je vais courir

```
php artisan test:command
```

il appellera la méthode `handle` dans la classe ayant la `test:command` signature `test:command`

Lire Artisan en ligne: https://riptutorial.com/fr/laravel/topic/1140/artisan

# Chapitre 3: Authentification

## Examples

### Multi authentification

Laravel vous permet d'utiliser plusieurs types d'authentification avec des gardes spécifiques.

En laravel 5.3, l'authentification multiple est peu différente de celle de Laravel 5.2

Je vais vous expliquer comment implémenter la fonctionnalité d'authentification multiple dans 5.3

Tout d'abord, vous avez besoin de deux modèles d'utilisateur différents

```
cp App/User.php App/Admin.php
```

changez le nom de la classe en Admin et définissez l'espace de noms si vous utilisez des modèles différents. ça devrait ressembler

**App \ Admin.php**

```
<?php

namespace App;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class Admin extends Authenticatable
{
    use Notifiable;

    protected $fillable = ['name', 'email', 'password'];
    protected $hidden   = ['password', 'remember_token'];

}
```

Vous devez également créer une migration pour admin

```
php artisan make:migration create_admins_table
```

puis modifiez le fichier de migration avec le contenu de la migration utilisateur par défaut. Ressemble à ça

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
class CreateAdminsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('admins', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();

            $table->softDeletes();
        });
    }

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

éditer **config / auth.php**

```
'guards'    => [
        'web'   => [
            'driver'   => 'session',
            'provider' => 'users',
        ],

        'api'   => [
            'driver'   => 'token',
            'provider' => 'users',
        ],
        //Add Admin Guard
        'admin' => [
            'driver'   => 'session',
            'provider' => 'admins',
        ],
    ],
```

et

```
'providers' => [
        'users'  => [
            'driver' => 'eloquent',
            'model'  => App\User::class,
        ],
        //Add Admins Provider
```

```
        'admins' => [
            'driver' => 'eloquent',
            'model'  => App\Admin::class,
        ],
    ],
```

Notez que nous ajoutons deux entrées. une variable de **garde** dans la variable **fournisseurs** .

Et voici comment vous utilisez l'autre garde puis "web"

Mon appli \ Http \ Controllers \ Admin \ LoginController

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\AuthenticatesUsers;
use Illuminate\Support\Facades\Auth;

class AuthController extends Controller
{

    use AuthenticatesUsers;

    protected $guard = 'admin';

    protected $redirectTo = '/admin/';

    public function showLoginForm()
    {
        return view('admin.login');
    }

    protected function guard()
    {
        return Auth::guard($this->guard);
    }

}
```

cela nécessite peu d'explications.

**Auth :: guard ('admin')** vous permettra d'utiliser des méthodes d'authentification (telles que la connexion, la déconnexion, l'enregistrement, etc.) avec votre administrateur.

Par exemple

```
Auth::guard('admin')->login($user)
```

va rechercher $ user dans la table des admins et se connecter avec l'utilisateur

```
Auth::login($user)
```

travaillera normalement avec la table des utilisateurs. La garde par défaut est spécifiée dans

**config / auth.php** avec le tableau par *défaut* . En frais laravel c'est "web".

Dans le contrôleur, vous devez implémenter des méthodes à partir de AuthenticatesUsers pour afficher vos chemins de vue personnalisés. Et vous devez implémenter d'autres fonctions telles que la garde pour utiliser vos nouveaux gardes utilisateurs.

Dans cet exemple, mon login **admin** est **admin / login.blade**

Et en implémentant la fonction guard () pour retourner **Auth :: guard ('admin'),** toutes les méthodes de trait AuthenticatesUsers fonctionnent avec la garde "admin".

Dans les versions antérieures de laravel, ceci est peu différent de 5,3

en 5.2, la fonction getGuard renvoie la variable $ guard de la classe et de la fonction principale (login)

```
Auth::guard($guard)->attempt(...)
```

en 5.3, la fonction de garde retourne Auth :: guard () et la fonction principale l'utilise comme

```
$this->guard()->attempt(...)
```

Lire Authentification en ligne: https://riptutorial.com/fr/laravel/topic/7051/authentification

# Chapitre 4: Autorisation

## Introduction

Laravel fournit un moyen simple d'autoriser les actions des utilisateurs sur des ressources spécifiques. Avec l'autorisation, vous pouvez autoriser de manière sélective les utilisateurs à accéder à certaines ressources tout en refusant l'accès aux autres. Laravel fournit une API simple pour gérer les autorisations utilisateur à l'aide de `Gates` et de `Policies` . `Gates` `AuthServiceProvider` approche simple de l'autorisation basée sur la fermeture à l'aide de `AuthServiceProvider` tandis que les `Policies` vous permettent d'organiser la logique d'autorisation autour des modèles utilisant des classes.

## Examples

### Utiliser les portes

`Gates` sont des fermetures qui déterminent si un utilisateur est autorisé à effectuer une certaine action sur une ressource. `Gates` sont généralement définies dans la méthode d'amorçage d' `AuthServiceProvider` et nommées de manière succincte pour refléter ce qu'elles font. Voici un exemple de portail permettant uniquement aux utilisateurs premium de visualiser certains contenus:

```
Gate::define('view-content', function ($user, $content){
    return $user->isSubscribedTo($content->id);
});
```

Une `Gate` reçoit toujours une instance d'utilisateur en tant que premier argument, vous n'avez pas besoin de la passer lors de l'utilisation de la porte, et peut éventuellement recevoir des arguments supplémentaires tels que le modèle éloquent concerné.

### Autoriser des actions avec des portes

Pour utiliser l'exemple ci-dessus sur un modèle de lame afin de masquer le contenu de l'utilisateur, vous devez généralement procéder comme suit:

```
@can('view-content', $content)
 <! -- content here -->
@endcan
```

Pour empêcher complètement la navigation vers le contenu, vous pouvez effectuer les opérations suivantes dans votre contrôleur:

```
if(Gate::allows('view-content', $content)){
    /* user can view the content */
}
```

```
        OR

if(Gate::denies('view-content', $content)){
    /* user cannot view content */
}
```

Remarque: vous n'êtes pas obligé de transférer l'utilisateur actuellement authentifié à cette méthode, Laravel s'en charge pour vous.

## Les politiques

Les stratégies sont des classes qui vous aident à organiser la logique d'autorisation autour d'une ressource de modèle. En utilisant notre exemple précédent, nous pourrions avoir un `ContentPolicy` qui gère l'accès des utilisateurs au modèle de `Content` .

Pour rendre `ContentPolicy` , laravel fournit une commande artisanale. Il suffit de courir

```
php artisan make:policy ContentPolicy
```

Cela fera une classe de politique vide et placera dans le dossier `app/Policies` . Si le dossier n'existe pas, Laravel le créera et placera la classe à l'intérieur.

Une fois créées, les stratégies doivent être enregistrées pour aider Laravel à savoir quelles politiques utiliser lors de l'autorisation d'actions sur des modèles. `AuthServiceProvider` de Laravel, fourni avec toutes les nouvelles installations Laravel, possède une propriété `AuthServiceProvider` qui `AuthServiceProvider` vos modèles éloquents à leurs règles d'autorisation. Tout ce que vous devez faire ajoute le mappage au tableau.

```
protected $policies = [
    Content::class => ContentPolicy::class,
 ];
```

## Politiques d'écriture

L'écriture de `Policies` suit le même schéma que l'écriture de `Gates` . La porte d'autorisation de contenu peut être réécrite sous la forme d'une politique comme celle-ci:

```
function view($user, $content)
{
    return $user->isSubscribedTo($content->id);
}
```

Les stratégies peuvent contenir plus de méthodes que nécessaire pour prendre en charge tous les cas d'autorisation d'un modèle.

## Autoriser des actions avec des stratégies

### Via le modèle de l'utilisateur

Le modèle Laravel User contient deux méthodes d'aide aux autorisations à l'aide de `Policies` ; `can`

et `can't` . Ces deux peuvent être utilisés pour déterminer si un utilisateur a une autorisation ou non sur un modèle respectivement.

Pour vérifier si un utilisateur peut afficher un contenu ou non, vous pouvez effectuer les opérations suivantes:

```
if($user->can('view', $content)){
    /* user can view content */
}

        OR

if($user->cant('view', $content)){
    /* user cannot view content */
}
```

### Via Middleware

```
Route::get('/contents/{id}, function(Content $content){
    /* user can view content */
})->middleware('can:view,content');
```

### Via les contrôleurs

Laravel fournit une méthode d'assistance, appelée `authorize` qui prend le nom de la stratégie et le modèle associé comme arguments et autorise l'action en fonction de votre logique d'autorisation ou refuse l'action et lance une exception `AuthorizationException` que le gestionnaire Laravel Exception convertit en `403 HTTP response` .

```
pubic function show($id)
{
    $content = Content::find($id);

    $this->authorize('view', $content);

    /* user can view content */
}
```

Lire Autorisation en ligne: https://riptutorial.com/fr/laravel/topic/9360/autorisation

# Chapitre 5: Autorisations pour le stockage

## Introduction

Laravel nécessite que certains dossiers soient accessibles en écriture pour l'utilisateur du serveur Web.

## Examples

### Exemple

Nous devons également définir les autorisations correctes pour `storage` fichiers de `storage` sur le `server` . Donc, nous devons donner une permission d'écriture dans le répertoire de stockage comme suit:

```
$ chmod -R 777 ./storage ./bootstrap
```

ou vous pouvez utiliser

```
$ sudo chmod -R 777 ./storage ./bootstrap
```

Pour les fenêtres

Assurez-vous d'être un administrateur sur cet ordinateur avec un accès en écriture

```
xampp\htdocs\laravel\app\storage needs to be writable
```

La manière NORMALE de définir des autorisations est d'avoir vos fichiers appartenant au serveur Web:

```
sudo chown -R www-data:www-data /path/to/your/root/directory
```

Lire Autorisations pour le stockage en ligne:
https://riptutorial.com/fr/laravel/topic/9797/autorisations-pour-le-stockage

# Chapitre 6: Base de données

## Examples

### Connexions de bases de données multiples

Laravel permet aux utilisateurs de travailler sur plusieurs connexions de bases de données. Si vous devez vous connecter à plusieurs bases de données et les faire fonctionner ensemble, vous devez faire attention à la configuration de la connexion.

Vous pouvez également utiliser différents types de bases de données dans la même application si vous en avez besoin.

**Connexion par défaut** Dans `config/database.php` , vous pouvez voir l'appel de l'élément de configuration:

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Ce nom fait référence au nom des connexions `mysql` ci-dessous:

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' =>  database_path('database.sqlite'),
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', 'localhost'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => false,
        'engine' => null,
    ],
],
```

Si vous n'avez pas mentionné le nom de la connexion à la base de données dans d'autres codes ou commandes, Laravel choisira le nom de connexion de base de données par défaut. cependant, dans les connexions à plusieurs bases de données, même si vous configurez la connexion par défaut, vous avez une meilleure configuration partout où vous avez utilisé la connexion à la base de données.

**Fichier de migration**

Dans le fichier de migration, si une connexion à une base de données unique, vous pouvez utiliser:

```
 Schema::create("table",function(Blueprint $table){
     $table->increments('id');
});
```

En connexion à plusieurs bases de données, vous utiliserez la méthode `connection()` pour indiquer à Laravel quelle connexion de base de données vous utilisez:

```
 Schema::connection("sqlite")->create("table",function(Blueprint $table){
     $table->increments('id');
});
```

**Artisan Migrate**

Si vous utilisez une connexion à une seule base de données, vous exécuterez:

```
php artisan migrate
```

Toutefois, pour une connexion à plusieurs bases de données, il est préférable de savoir quelle connexion de base de données conserve les données de migration. vous exécuterez donc la commande suivante:

```
php artisan migrate:install --database=sqlite
```

Cette commande installe la table de migration dans la base de données cible pour préparer la migration.

```
php artisan migrate --database=sqlite
```

Cette commande exécute la migration et enregistre les données de migration dans la base de données cible

```
php artisan migrate:rollback --database=sqlite
```

Cette commande annulera la migration et enregistrera les données de migration dans la base de données cible.

**Modèle éloquent**

Pour spécifier une connexion à une base de données à l'aide d'Eloquent, vous devez définir la propriété `$connection` :

```
namespace App\Model\Sqlite;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'sqlite';
```

```
}
```

Pour spécifier une autre connexion (seconde) à la base de données en utilisant Eloquent:

```
namespace App\Model\MySql;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'mysql';
}
```

Laravel utilisera la propriété `$connection` définie dans un modèle pour utiliser la connexion spécifiée définie dans `config/database.php` . Si la propriété `$connection` n'est pas définie dans un modèle, la valeur par défaut sera utilisée.

Vous pouvez également spécifier une autre connexion en utilisant la méthode statique `on` :

```
// Using the sqlite connection
Table::on('sqlite')->select(...)->get()
// Using the mysql connection
Table::on('mysql')->select(...)->get()
```

### Base de données / Générateur de requêtes

Vous pouvez également spécifier une autre connexion en utilisant le générateur de requêtes:

```
// Using the sqlite connection
DB::connection('sqlite')->table('table')->select(...)->get()
// Using the mysql connection
DB::connection('mysql')->table('table')->select(...)->get()
```

### Test de l'unité

Laravel fournit `seeInDatabase($table,$fielsArray,$connection)` pour tester le code de connexion à la base de données. Dans le fichier de test unitaire, vous devez faire comme:

```
$this
    ->json(
        'GET',
        'result1/2015-05-08/2015-08-08/a/123'
    )
     ->seeInDatabase("log", ["field"=>"value"], 'sqlite');
```

De cette façon, Laravel saura quelle connexion de base de données tester.

### Transactions de base de données dans le test unitaire

Laravel autorise la base de données à annuler toutes les modifications pendant les tests. Pour tester plusieurs connexions de base de données, vous devez définir `$connectionsToTransact` propriétés `$connectionsToTransact`

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

Lire Base de données en ligne: https://riptutorial.com/fr/laravel/topic/1093/base-de-donnees

# Chapitre 7: Bases Cron

## Introduction

Cron est un démon de planificateur de tâches qui exécute des tâches planifiées à certains intervalles. Cron utilise un fichier de configuration appelé crontab, également appelé table cron, pour gérer le processus de planification.

## Examples

### Créer un travail cron

Crontab contient des tâches cron, chacune liée à une tâche spécifique. Les travaux Cron sont composés de deux parties, l'expression cron et une commande shell à exécuter:

```
* * * * * commande / to / run
```

Chaque champ de l'expression ci-dessus `* * * * *` est une option pour définir la fréquence de programmation. Il est composé de minutes, heure, jour du mois, mois et jour de la semaine dans l'ordre du placement. Le symbole astérisque fait référence à toutes les valeurs possibles pour le champ respectif. En conséquence, le travail cron ci-dessus sera exécuté toutes les minutes de la journée.

La tâche cron suivante est exécutée à **12h30** tous les jours:

```
30 12 * * * commande / to / run
```

Lire Bases Cron en ligne: https://riptutorial.com/fr/laravel/topic/9891/bases-cron

# Chapitre 8: cadre lumen

## Examples

### Démarrer avec Lumen

*L'exemple suivant illustre l'utilisation de `Lumen` dans des environnements `WAMP / MAMP / LAMP` .*

Pour travailler avec `Lumen` vous devez d'abord configurer deux choses.

- Compositeur
- PHPUnit
- git (non requis mais fortement recommandé)

En supposant que vous avez tous ces trois composants installés (au moins vous avez besoin de compositeur), allez d'abord sur la racine du document de vos serveurs Web en utilisant le terminal. MacOSX et Linux sont livrés avec un excellent terminal. Vous pouvez utiliser `git bash` (qui est en fait `mingw32` ou `mingw64` ) dans Windows.

```
$ cd path/to/your/document/root
```

Ensuite, vous devez utiliser composer pour installer et créer le projet `Lumen` . Exécutez la commande suivante.

```
$ composer create-project laravel/lumen=~5.2.0 --prefer-dist lumen-project
$ cd lumen-project
```

`lumen-app` dans le code ci-dessus est le nom du dossier. Vous pouvez le changer comme vous le souhaitez. Vous devez maintenant configurer votre hôte virtuel pour qu'il pointe vers le dossier `path/to/document/root/lumen-project/public` . Disons que vous avez mappé `http://lumen-project.local` à ce dossier. Maintenant, si vous allez sur cette URL, vous devriez voir un message comme suit (en fonction de votre version de `Lumen` installée, dans mon cas c'était 5.4.4) -

```
Lumen (5.4.4) (Laravel Components 5.4.*)
```

Si vous ouvrez le fichier `lumen-project/routers/web.php` , vous devriez voir le suivant

```
$app->get('/', function () use($app) {
    return $app->version();
});
```

Toutes nos félicitations! Vous avez maintenant une installation `Lumen` fonctionnelle. Non, vous pouvez étendre cette application pour écouter vos points de terminaison personnalisés.

Lire cadre lumen en ligne: https://riptutorial.com/fr/laravel/topic/9221/cadre-lumen

# Chapitre 9: Classe CustomException à Laravel

## Introduction

Les exceptions PHP sont émises lorsqu'un événement ou une erreur sans précédent se produit.

En règle générale, une exception ne doit pas être utilisée pour contrôler la logique de l'application, telle que les instructions if, et doit être une sous-classe de la classe Exception.

L'un des principaux avantages de toutes les exceptions interceptées par une seule classe est que nous pouvons créer des gestionnaires d'exceptions personnalisés qui renvoient des messages de réponse différents en fonction de l'exception.

## Examples

### Classe CustomException en laravel

Toutes les erreurs et exceptions, à la fois personnalisées et par défaut, sont gérées par la classe Handler dans app / Exceptions / Handler.php à l'aide de deux méthodes.

- rapport()
- rendre()

```
public function render($request, Exception $e)
{
 //check if exception is an instance of ModelNotFoundException.
 if ($e instanceof ModelNotFoundException)
 {
     // ajax 404 json feedback
     if ($request->ajax())
     {
         return response()->json(['error' => 'Not Found'], 404);
     }
     // normal 404 view page feedback
     return response()->view('errors.missing', [], 404);
  }
   return parent::render($request, $e);
}
```

puis créer une vue liée à une erreur dans le dossier des erreurs nommée 404.blade.php

Utilisateur non trouvé.

Vous avez brisé le solde d'Internet

Lire Classe CustomException à Laravel en ligne: https://riptutorial.com/fr/laravel/topic/9550/classe-

customexception-a-laravel

# Chapitre 10: Classement de base de données

## Examples

### Diriger un semoir

Vous pouvez ajouter votre nouveau Seeder à la classe DatabaseSeeder.

```
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
```

Pour exécuter un segment de base de données, utilisez la commande Artisan

```
php artisan db:seed
```

Cela exécutera la classe DatabaseSeeder. Vous pouvez également choisir d'utiliser l'option `--class=` pour spécifier manuellement le `--class=` à exécuter.

* Remarque: vous devrez peut-être exécuter le composeur dumpautoload si votre classe Seeder est introuvable. Cela se produit généralement si vous créez manuellement une classe de segment au lieu d'utiliser la commande artisan.

### Créer une semence

Les semences de base de données sont stockées dans le répertoire / database / seeds. Vous pouvez créer une graine en utilisant une commande Artisan.

```
php artisan make:seed UserTableSeeder
```

Sinon, vous pouvez créer une nouvelle classe qui étend `Illuminate\Database\Seeder` . La classe doit avoir une fonction publique nommée `run()` .

### Insérer des données à l'aide d'un semoir

Vous pouvez référencer des modèles dans un semoir.

```
use DB;
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{
```

```
    public function run(){
        # Remove all existing entrie
        DB::table('users')->delete() ;
        User::create([
            'name' => 'Admin',
            'email' => 'admin@example.com',
            'password' => Hash::make('password')
        ]);

    }
}
```

## Insertion de données avec une fabrique de modèles

Vous voudrez peut-être utiliser les usines modèles dans vos semences. Cela va créer 3 nouveaux utilisateurs.

```
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{

    public function run(){
        factory(User::class)->times(3)->create();
    }
}
```

Vous souhaiterez peut-être également définir des champs spécifiques sur votre semis, comme un mot de passe, par exemple. Cela créera 3 utilisateurs avec le même mot de passe.

```
factory(User::class)->times(3)->create(['password' => '123456']);
```

## Amorcer avec MySQL Dump

Suivez l'exemple précédent de création d'une graine. Cet exemple utilise un vidage MySQL pour générer une table dans la base de données du projet. La table doit être créée avant le semis.

```
<?php

use Illuminate\Database\Seeder;

class UserTableSeeder extends Seeder
{

    /**
     * Run the database seeds.
     *
     * @return void
     */
     public function run()
     {
         $sql = file_get_contents(database_path() . '/seeds/users.sql');

         DB::statement($sql);
      }
```

```
}
```

Notre $ sql va être le contenu de notre vidage users.sql. Le vidage doit avoir une instruction INSERT INTO. Ce sera à vous de décider où vous stockerez vos décharges. Dans l'exemple ci-dessus, il est stocké dans le répertoire du projet `\database\seeds` . Utilisation de la fonction d'aide de laravel `database_path()` et ajout du répertoire et du nom de fichier du dump.

```
INSERT INTO `users` (`id`, `name`, `email`, `password`, `remember_token`, `created_at`,
`updated_at`) VALUES
(1, 'Jane', 'janeDoe@fakemail.com', 'superSecret', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00'),
(2, 'John', 'johnny@fakemail.com', 'sup3rS3cr3t', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00');
```

`DB::statement($sql)` exécutera les insertions une fois le segment exécuté. Comme dans les exemples précédents, vous pouvez placer le `UserTableSeeder` dans la classe `DatabaseSeeder` fournie par laravel:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
}
```

et exécuter à partir de la CLI dans le répertoire de projet `php artisan db:seed` . Ou vous pouvez exécuter le Seeder pour une seule classe en utilisant `php artisan db:seed --class=UsersTableSeeder`

## Utiliser faker et ModelFactories pour générer des semences

### 1) BASIC SIMPLE WAY

Les applications basées sur des bases de données ont souvent besoin de données pré-insérées dans le système à des fins de test et de démonstration.

Pour créer de telles données, créez d'abord la classe de segmentation

**ProductTableSeeder**

```
use Faker\Factory as Faker;
use App\Product;

class ProductTableSeeder extends DatabaseSeeder {
```

```
public function run()
{
    $faker = $this->getFaker();

    for ($i = 0; $i < 10; $i++)
    {
        $name =         $faker->word;
        $image =        $faker->imageUrl;

        Modelname::create([
            'name' => $name,
            'image' => $image,
        ]);
      }
    }
 }
```

Pour appeler une classe capable d'exécuter une classe de segmentation, vous devez l'appeler depuis la classe DatabaseSeeder, simplement en passant le nom du segment que vous souhaitez exécuter:

utiliser Illuminate \ Database \ Seeder;

```
class DatabaseSeeder extends Seeder {

    protected $faker;

    public function getFaker() {
        if (empty($this->faker)) {
            $faker = Faker\Factory::create();
            $faker->addProvider(new Faker\Provider\Base($faker));
            $faker->addProvider(new Faker\Provider\Lorem($faker));
        }
        return $this->faker = $faker;
    }
    public function run() {
        $this->call(ProductTableSeeder::class);
    }
}
```

N'oubliez pas de lancer `$ composer dump-autoload` après avoir créé le Seeder, car ils ne sont pas automatiquement chargés automatiquement par le composeur (sauf si vous avez créé la commande `$ php artisan make:seeder Name` )

Vous êtes maintenant prêt à lancer en exécutant cette commande `php artisan db:seed`

**2) UTILISATION des usines modèles**

Tout d'abord, vous devez définir un ensemble d'attributs par défaut pour chaque modèle dans `App/database/factories/ModelFactory.php`

Prenant un modèle d'utilisateur comme exemple, voici à quoi ressemble un ModelFactory

```
$factory->define(App\User::class, function (Faker\Generator $faker) {
    return [
```

```
        'name' => $faker->name,
        'email' => $faker->email,
        'password' => bcrypt(str_random(10)),
        'remember_token' => str_random(10),
    ];
});
```

Maintenant, créez une table de création `php artisan make:seeder UsersTableSeeder`

Et ajoutez ceci

```
public function run()
{
    factory(App\User::class, 100)->create()
}
```

puis ajoutez ceci à la `DatabaseSeeder`

```
public function run()
{
    $this->call(UsersTableSeeder::class);
}
```

Cela va semer la table avec 100 enregistrements.

Lire Classement de base de données en ligne:
https://riptutorial.com/fr/laravel/topic/1118/classement-de-base-de-donnees

# Chapitre 11: Collections

## Syntaxe

- $ collection = collect (['Value1', 'Value2', 'Value3']); // Les clés sont par défaut à 0, 1, 2, ...,

## Remarques

`Illuminate\Support\Collection` fournit une interface fluide et pratique pour gérer les tableaux de données. Vous les avez peut-être utilisés sans le savoir, par exemple les requêtes de modèle qui récupèrent plusieurs enregistrements renvoient une instance d' `Illuminate\Support\Collection` .

Pour une documentation à jour sur les collections, vous pouvez trouver la documentation officielle ici

## Examples

### Créer des collections

A l'aide de l'assistant `collect()` , vous pouvez facilement créer de nouvelles instances de collection en transmettant un tableau tel que:

```
$fruits = collect(['oranges', 'peaches', 'pears']);
```

Si vous ne souhaitez pas utiliser de fonctions d'assistance, vous pouvez créer une nouvelle collection en utilisant directement la classe:

```
$fruits = new Illuminate\Support\Collection(['oranges', 'peaches', 'pears']);
```

Comme mentionné dans les remarques, les modèles retournent par défaut une instance `Collection` , mais vous êtes libre de créer vos propres collections en fonction des besoins. Si aucun tableau n'est spécifié lors de la création, une collection vide sera créée.

### où()

Vous pouvez sélectionner certains éléments d'une collection en utilisant la méthode `where()` .

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => true],
    ['name' => 'Matt', 'coffee_drinker' => true]
];

$matt = collect($data)->where('name', 'Matt');
```

Ce bit de code sélectionne tous les éléments de la collection dont le nom est «Matt». Dans ce cas,

seul le deuxième élément est renvoyé.

# Nidification

Tout comme la plupart des méthodes de tableau dans Laravel, `where()` prend également en charge la recherche d'éléments imbriqués. Prenons l'exemple ci-dessus en ajoutant un deuxième tableau:

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => ['at_work' => true, 'at_home' => true]],
    ['name' => 'Matt', 'coffee_drinker' => ['at_work' => true, 'at_home' => false]]
];

$coffeeDrinkerAtHome = collect($data)->where('coffee_drinker.at_home', true);
```

Cela ne fera que rendre Taylor, car il boit du café à la maison. Comme vous pouvez le voir, l'imbrication est prise en charge à l'aide de la notation par points.

# Ajouts

Lors de la création d'une collection d'objets à la place de tableaux, ceux-ci peuvent également être filtrés à l'aide de `where()` . La Collection essaiera alors de recevoir toutes les propriétés souhaitées.

5.3

Veuillez noter que depuis Laravel 5.3, la méthode `where()` essaiera de comparer les valeurs par défaut. Cela signifie que lors de la recherche de `(int)1` , toutes les entrées contenant `'1'` seront également renvoyées. Si vous n'aimez pas ce comportement, vous pouvez utiliser la méthode `whereStrict()` .

## Utiliser Get pour rechercher la valeur ou retourner la valeur par défaut

Vous vous trouvez souvent dans une situation où vous devez trouver une variable correspondant à la valeur, et les collections vous ont couvert.

Dans l'exemple ci-dessous, nous avons obtenu trois paramètres régionaux différents dans un tableau avec un code d'appel correspondant. Nous voulons être en mesure de fournir un paramètre régional et obtenir en retour le code d'appel associé. Le deuxième paramètre dans `get` est un paramètre par défaut si le premier paramètre est introuvable.

```
function lookupCallingCode($locale)
{
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, 44);
```

```
}
```

Dans l'exemple ci-dessus, nous pouvons faire ce qui suit

```
lookupCallingCode('de_DE'); // Will return 49
lookupCallingCode('sv_SE'); // Will return 44
```

Vous pouvez même passer un rappel comme valeur par défaut. Le résultat du rappel sera renvoyé si la clé spécifiée n'existe pas:

```
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, function() {
        return 44;
    });
```

## Utilisation de Contient pour vérifier si une collection satisfait à certaines conditions

Un problème commun est d'avoir une collection d'éléments qui doivent tous répondre à certains critères. Dans l'exemple ci-dessous, nous avons collecté deux éléments pour un régime alimentaire et nous voulons vérifier que le régime ne contient aucun aliment malsain.

```
// First we create a collection
$diet = collect([
    ['name' => 'Banana', 'calories' => '89'],
    ['name' => 'Chocolate', 'calories' => '546']
]);

// Then we check the collection for items with more than 100 calories
$isUnhealthy = $diet->contains(function ($i, $snack) {
    return $snack["calories"] >= 100;
});
```

Dans le cas ci-dessus, la variable `$isUnhealthy` sera définie sur `true` car le chocolat remplit la condition et le régime est donc malsain.

## Utiliser Pluck pour extraire certaines valeurs d'une collection

Vous vous retrouverez souvent avec une collection de données où vous êtes uniquement intéressé par des parties des données.

Dans l'exemple ci-dessous, nous avons reçu une liste des participants à un événement et nous voulons fournir un guide avec une simple liste de noms.

```
// First we collect the participants
$participants = collect([
    ['name' => 'John', 'age' => 55],
    ['name' => 'Melissa', 'age' => 18],
```

```
    ['name' => 'Bob', 'age' => 43],
    ['name' => 'Sara', 'age' => 18],
]);

// Then we ask the collection to fetch all the names
$namesList = $partcipants->pluck('name')
// ['John', 'Melissa', 'Bob', 'Sara'];
```

Vous pouvez également utiliser le `pluck` pour des collections d'objets ou des tableaux / objets imbriqués avec une notation par points.

```
$users = User::all(); // Returns Eloquent Collection of all users
$usernames = $users->pluck('username'); // Collection contains only user names

$users->load('profile'); // Load a relationship for all models in collection

// Using dot notation, we can traverse nested properties
$names = $users->pluck('profile.first_name'); // Get all first names from all user profiles
```

## Utiliser Map pour manipuler chaque élément d'une collection

Vous devez souvent modifier la structure d'un ensemble de données et manipuler certaines valeurs.

Dans l'exemple ci-dessous, nous avons une collection de livres avec un montant de remise attaché. Mais nous préférons plutôt une liste de livres avec un prix déjà réduit.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20, 'discount' => 0.5],
    ['title' => 'Continuous Delivery', 'price' => 25, 'discount' => 0.1],
    ['title' => 'The Clean Coder', 'price' => 10, 'discount' => 0.75],
];

$discountedItems =  collect($books)->map(function ($book) {
   return ['title' => $book["title"], 'price' => $book["price"] * $book["discount"]];
});

//[
//    ['title' => 'The Pragmatic Programmer', 'price' => 10],
//    ['title' => 'Continuous Delivery', 'price' => 12.5],
//    ['title' => 'The Clean Coder', 'price' => 5],
//]
```

Cela pourrait également être utilisé pour changer les clés, disons que nous voulions changer le `title` clé pour `name` ceci serait une solution appropriée.

## Utiliser sum, avg, min ou max sur une collection pour des calculs statistiques

Les collections vous offrent également un moyen simple d'effectuer des calculs statistiques simples.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20],
```

```
    ['title' => 'Continuous Delivery', 'price' => 30],
    ['title' => 'The Clean Coder', 'price' => 10],
]

$min = collect($books)->min('price'); // 10
$max = collect($books)->max('price'); // 30
$avg = collect($books)->avg('price'); // 20
$sum = collect($books)->sum('price'); // 60
```

### Trier une collection

Il existe plusieurs manières de trier une collection.

# Trier()

La méthode de `sort` trie la collection:

```
$collection = collect([5, 3, 1, 2, 4]);

$sorted = $collection->sort();

echo $sorted->values()->all();

returns : [1, 2, 3, 4, 5]
```

La méthode de `sort` permet également de transmettre un rappel personnalisé avec votre propre algorithme. Sous le capot, le sort utilise l' `usort` de php.

```
$collection = $collection->sort(function ($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
});
```

# Trier par()

La méthode `sortBy` trie la collection par la clé donnée:

```
$collection = collect([
['name' => 'Desk', 'price' => 200],
['name' => 'Chair', 'price' => 100],
['name' => 'Bookcase', 'price' => 150],
]);

$sorted = $collection->sortBy('price');

echo $sorted->values()->all();

returns:  [
        ['name' => 'Chair', 'price' => 100],
```

```
        ['name' => 'Bookcase', 'price' => 150],
        ['name' => 'Desk', 'price' => 200],
    ]
```

La méthode `sortBy` permet d'utiliser un format de notation par points pour accéder à une clé plus profonde afin de trier un tableau multidimensionnel.

```
$collection = collect([
    ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
    ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
    ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
    ]);

$sorted = $collection->sortBy("product.price")->toArray();

return: [
  ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
  ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
  ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
]
```

# SortByDesc ()

Cette méthode a la même signature que la méthode `sortBy` , mais `sortBy` la collection dans l'ordre inverse.

## En utilisant Reduce ()

La `reduce` procédé réduit la perception à une valeur unique, en passant le résultat de chaque itération dans l'itération suivante. S'il vous plaît voir la méthode de réduction .

La méthode de `reduce` parcourt chaque élément avec une collection et produit un nouveau résultat à la prochaine itération. Chaque résultat de la dernière itération est passé par le premier paramètre (dans les exemples suivants, en tant que `$carry` ).

Cette méthode peut faire beaucoup de traitement sur des ensembles de données volumineux. Par exemple, les exemples suivants, nous allons utiliser les exemples de données d'étudiant suivants:

```
 $student = [
    ['class' => 'Math', 'score' => 60],
    ['class' => 'English', 'score' => 61],
    ['class' => 'Chemistry', 'score' => 50],
    ['class' => 'Physics', 'score' => 49],
];
```

**Somme du score total de l'étudiant**

```
$sum = collect($student)
    ->reduce(function($carry, $item){
        return $carry + $item["score"];
    }, 0);
```

Résultat: `220`

Explication:

- `$carry` est le résultat de la dernière itération.
- Le deuxième paramètre est la valeur par défaut pour le $ carry dans le premier tour d'itération. Ce cas, la valeur par défaut est 0

**Passer un étudiant si toutes ses notes sont> = 50**

```
$isPass = collect($student)
    ->reduce(function($carry, $item){
        return $carry && $item["score"] >= 50;
    }, true);
```

Résultat: `false`

Explication:

- La valeur par défaut de $ carry est true
- Si tous les scores sont supérieurs à 50, le résultat sera vrai. si moins de 50, retournez faux.

**Échouer un élève si un score est <50**

```
$isFail = collect($student)
    ->reduce(function($carry, $item){
        return $carry || $item["score"] < 50;
    }, false);
```

Résultat: `true`

Explique:

- la valeur par défaut de $ carry est fausse
- si un score est inférieur à 50, renvoyer true; si tous les scores sont supérieurs à 50, renvoyer false.

**Renvoyer le sujet avec le score le plus élevé**

```
$highestSubject = collect($student)
    ->reduce(function($carry, $item){
        return $carry === null || $item["score"] > $carry["score"] ? $item : $carry;
    });
```

résultat: `[ "subject" => "English", "score" => 61 ]`

Explique:

- Le second paramètre n'est pas fourni dans ce cas.
- La valeur par défaut de $ carry est null, donc nous vérifions cela dans notre conditionnel.

## Utiliser macro () pour étendre les collections

La fonction `macro()` vous permet d'ajouter de nouvelles fonctionnalités aux objets `Illuminate\Support\Collection`

Usage:

```
Collection::macro("macro_name", function ($parameters) {
    // Your macro
});
```

Par exemple:

```
Collection::macro('uppercase', function () {
    return $this->map(function ($item) {
        return strtoupper($item);
    });
});

collect(["hello", "world"])->uppercase();
```

Résultat: `["HELLO", "WORLD"]`

## Utiliser la syntaxe de tableau

L'objet `Collection` implémente l'interface `ArrayAccess` et `IteratorAggregate` , ce qui lui permet d'être utilisé comme un tableau.

**Élément de collection d'accès:**

```
 $collection = collect([1, 2, 3]);
 $result = $collection[1];
```

Résultat: `2`

**Attribuer un nouvel élément:**

```
$collection = collect([1, 2, 3]);
$collection[] = 4;
```

Résultat: `$collection` est `[1, 2, 3, 4]`

**Collection de boucles:**

```
$collection = collect(["a" => "one", "b" => "two"]);
$result = "";
foreach($collection as $key => $value){
    $result .= "(".$key.": ".$value.") ";
}
```

Résultat: `$result` est `(a: one) (b: two)`

### Conversion de tableau en collection:

Pour convertir une collection en un tableau PHP natif, utilisez:

```
$array = $collection->all();
//or
$array = $collection->toArray()
```

Pour convertir un tableau en collection, utilisez:

```
$collection = collect($array);
```

### Utilisation de collections avec des fonctions de tableau

Sachez que les collections sont des objets normaux qui ne seront pas convertis correctement lorsqu'ils sont utilisés par des fonctions nécessitant explicitement des tableaux, comme `array_map($callback)` .

Assurez-vous de convertir d'abord la collection ou, si disponible, utilisez plutôt la méthode fournie par la classe `Collection` : `$collection->map($callback)`

Lire Collections en ligne: https://riptutorial.com/fr/laravel/topic/2358/collections

# Chapitre 12: Connexions DB multiples à Laravel

## Examples

### Étapes initiales

Plusieurs connexions de base de données, de tout type, peuvent être définies dans le fichier de configuration de la base de données (probablement `app/config/database.php` ). Par exemple, pour extraire des données de 2 bases de données MySQL, définissez-les séparément:

```
<?php
return array(

    'default' => 'mysql',

    'connections' => array(

        # Our primary database connection
        'mysql' => array(
            'driver'    => 'mysql',
            'host'      => 'host1',
            'database'  => 'database1',
            'username'  => 'user1',
            'password'  => 'pass1'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),

        # Our secondary database connection
        'mysql2' => array(
            'driver'    => 'mysql',
            'host'      => 'host2',
            'database'  => 'database2',
            'username'  => 'user2',
            'password'  => 'pass2'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),
    ),
);
```

La connexion par défaut est toujours définie sur `mysql` . Cela signifie que, sauf indication contraire, l'application utilise la connexion `mysql` .

### Utilisation du générateur de schéma

Dans Schema Builder, utilisez la façade Schema avec n'importe quelle connexion. Exécutez la méthode `connection()` pour spécifier la connexion à utiliser:

```
Schema::connection('mysql2')->create('some_table', function($table)
{
    $table->increments('id');
});
```

## Utilisation du générateur de requêtes DB

Semblable à Schema Builder, définissez une connexion sur le générateur de requêtes:

```
$users = DB::connection('mysql2')->select(...);
```

## Utiliser Eloquent

Il existe plusieurs façons de définir la connexion à utiliser dans les modèles Eloquent. Une façon consiste à définir la variable de connexion $ dans le modèle:

```
<?php

class SomeModel extends Eloquent {

    protected $connection = 'mysql2';

}
```

La connexion peut également être définie à l'exécution via la méthode `setConnection` .

```
<?php

class SomeController extends BaseController {

    public function someMethod()
    {
        $someModel = new SomeModel;

        $someModel->setConnection('mysql2');

        $something = $someModel->find(1);

        return $something;
    }
}
```

## De Laravel Documentation

Chaque connexion individuelle est accessible via la méthode de connexion sur la façade de la base de `DB` , même si plusieurs connexions sont définies. Le `name` transmis à la méthode de `connection` doit correspondre à l'une des connexions répertoriées dans le fichier de configuration `config/database.php` :

```
$users = DB::connection('foo')->select(...);
```

L'accès brut est également possible, sous-jacent à l'instance PDO en utilisant la méthode getPdo sur une instance de connexion:

```
$pdo = DB::connection()->getPdo();
```

https://laravel.com/docs/5.4/database#using-multiple-database-connections

Lire Connexions DB multiples à Laravel en ligne:
https://riptutorial.com/fr/laravel/topic/9605/connexions-db-multiples-a-laravel

# Chapitre 13: Contrôleurs

## Introduction

Au lieu de définir toute la logique de gestion des demandes en tant que fermetures dans les fichiers de routage, vous pouvez organiser ce comportement à l'aide de classes de contrôleur. Les contrôleurs peuvent regrouper la logique de traitement des demandes associée en une seule classe. Les contrôleurs sont stockés dans le répertoire `app/Http/Controllers` par défaut.

## Examples

### Contrôleurs de base

```
<?php

namespace App\Http\Controllers;

use App\User;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

Vous pouvez définir une route vers cette action de contrôleur comme suit:

```
Route::get('user/{id}', 'UserController@show');
```

Désormais, lorsqu'une demande correspond à l'URI de la route spécifiée, la méthode `show` de la classe `UserController` sera exécutée. Bien entendu, les paramètres de l'itinéraire seront également transmis à la méthode.

### Contrôleur Middleware

Le middleware peut être assigné aux routes du contrôleur dans vos fichiers de route:

```
Route::get('profile', 'UserController@show')->middleware('auth');
```

Cependant, il est plus pratique de spécifier le middleware dans le constructeur de votre contrôleur. En utilisant la méthode middleware du constructeur de votre contrôleur, vous pouvez facilement

attribuer un middleware à l'action du contrôleur.

```
class UserController extends Controller
{
    /**
     * Instantiate a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');

        $this->middleware('log')->only('index');

        $this->middleware('subscribed')->except('store');
    }
}
```

### Contrôleur de ressources

Le routage des ressources Laravel attribue les routes "CRUD" typiques à un contrôleur avec une seule ligne de code. Par exemple, vous souhaiterez peut-être créer un contrôleur qui gère toutes les requêtes HTTP pour les "photos" stockées par votre application. En utilisant la commande `make:controller` Artisan, nous pouvons rapidement créer un tel contrôleur:

```
php artisan make:controller PhotoController --resource
```

Cette commande va générer un contrôleur sur `app/Http/Controllers/PhotoController.php` . Le contrôleur contiendra une méthode pour chacune des opérations de ressources disponibles.

## Exemple de l'apparence d'un contrôleur de ressources

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PhotoController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
```

```
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
```

```
}
```

L'exemple du contrôleur de ressources partage le nom de la méthode de ceux du tableau ci-dessous.

Ensuite, vous pouvez enregistrer un itinéraire ingénieux vers le contrôleur:

```
Route::resource('photos', 'PhotoController');
```

Cette déclaration de routage unique crée plusieurs itinéraires pour gérer diverses actions sur la ressource. Le contrôleur généré aura déjà des méthodes écrasées pour chacune de ces actions, y compris des notes vous informant des verbes HTTP et des URI qu'ils traitent.

## Actions gérées par le contrôleur de ressources

| Verbe | URI | action | Nom de l'itinéraire |
|---|---|---|---|
| OBTENIR | `/photos` | indice | photos.index |
| OBTENIR | `/photos/create` | créer | photos.créer |
| POSTER | `/photos` | le magasin | photos.store |
| OBTENIR | `/photos/{photo}` | montrer | photos.show |
| OBTENIR | `/photos/{photo}/edit` | modifier | photos.edit |
| PUT / PATCH | `/photos/{photo}` | mettre à jour | photos.update |
| EFFACER | `/photos/{photo}` | détruire | photos.destroy |

Lire Contrôleurs en ligne: https://riptutorial.com/fr/laravel/topic/10604/controleurs

# Chapitre 14: Courrier

## Examples

### Exemple de base

Vous pouvez configurer Mail en ajoutant / modifiant simplement ces lignes dans le fichier **.ENV de** l'application avec les informations de connexion de votre fournisseur de messagerie, par exemple pour l'utiliser avec gmail que vous pouvez utiliser:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=yourEmail@gmail.com
MAIL_PASSWORD=yourPassword
MAIL_ENCRYPTION=tls
```

Ensuite, vous pouvez commencer à envoyer des e-mails via Mail, par exemple:

```
$variable = 'Hello world!'; // A variable which can be use inside email blade template.
Mail::send('your.blade.file', ['variable' => $variable], function ($message) {
            $message->from('john@doe.com');
            $message->sender('john@doe.com');
            $message->to(foo@bar.com);
            $message->subject('Hello World');
        });
```

Lire Courrier en ligne: https://riptutorial.com/fr/laravel/topic/8014/courrier

# Chapitre 15: Demande interdomaine

## Examples

### introduction

Parfois, nous avons besoin de requêtes interdomaines pour nos API en Laravel. Nous devons ajouter des en-têtes appropriés pour mener à bien la requête interdomaine. Nous devons donc nous assurer que les en-têtes que nous ajoutons doivent être exacts, sinon notre API devient vulnérable. Afin d'ajouter des en-têtes, nous devons ajouter un middleware dans laravel qui ajoutera les en-têtes appropriés et transmettra les requêtes.

### CorsHeaders

```
<?php

namespace laravel\Http\Middleware;

class CorsHeaders
{
  /**
   * This must be executed _before_ the controller action since _after_ middleware isn't
executed when exceptions are thrown and caught by global handlers.
   *
   * @param $request
   * @param \Closure $next
   * @param string [$checkWhitelist] true or false Is a string b/c of the way the arguments
are supplied.
   * @return mixed
   */
  public function handle($request, \Closure $next, $checkWhitelist = 'true')
  {
    if ($checkWhitelist == 'true') {
      // Make sure the request origin domain matches one of ours before sending CORS response
headers.
      $origin = $request->header('Origin');
      $matches = [];
      preg_match('/^(https?:\/\/)?([a-zA-Z\d]+\.)*(?<domain>[a-zA-Z\d-\.]+\.[a-z]{2,10})$/',
$origin, $matches);

      if (isset($matches['domain']) && in_array($matches['domain'], ['yoursite.com']) {
        header('Access-Control-Allow-Origin: ' . $origin);
        header('Access-Control-Expose-Headers: Location');
        header('Access-Control-Allow-Credentials: true');

        // If a preflight request comes then add appropriate headers
        if ($request->method() === 'OPTIONS') {
          header('Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH');
          header('Access-Control-Allow-Headers: ' . $request->header('Access-Control-Request-
Headers'));
            // 20 days
          header('Access-Control-Max-Age: 1728000');
        }
      }
```

```
    } else {
      header('Access-Control-Allow-Origin: *');
    }

    return $next($request);
  }
}
```

Lire Demande interdomaine en ligne: https://riptutorial.com/fr/laravel/topic/7425/demande-interdomaine

# Chapitre 16: Demandes

## Examples

### Obtenir des commentaires

La principale méthode pour obtenir des informations serait d'injecter `Illuminate\Http\Request` dans votre contrôleur, après quoi il existe de nombreuses manières d'accéder aux données, dont 4 dans l'exemple ci-dessous.

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    public function store(Request $request)
    {
        // Returns the username value
        $name = $request->input('username');

        // Returns the username value
        $name = $request->username;

        // Returns the username value
        $name = request('username');

        // Returns the username value again
        $name = request()->username;

    }

}
```

Lors de l'utilisation de la fonction d' `input` , il est également possible d'ajouter une valeur par défaut lorsque l'entrée de la demande n'est pas disponible.

```
$name = $request->input('username', 'John Doe');
```

Lire Demandes en ligne: https://riptutorial.com/fr/laravel/topic/3076/demandes

# Chapitre 17: Demandes

## Examples

### Obtenir une instance de requête HTTP

Pour obtenir une instance d'une requête HTTP, la classe `Illuminate\Http\Request` doit être de type indication dans le constructeur ou dans la méthode du contrôleur.

Exemple de code:

```
 <?php

namespace App\Http\Controllers;

/* Here how we illuminate the request class in controller */
use Illuminate\Http\Request;

use Illuminate\Routing\Controller;

class PostController extends Controller
{
 /**
  * Store a new post.
  *
  * @param  Request  $request
  * @return Response
  */
 public function store(Request $request)
 {
     $name = $request->input('post_title');

     /*
     * so typecasting Request class in our method like above avails the
     * HTTP GET/POST/PUT etc method params in the controller to use and
     * manipulate
     */
 }
}
```

### Demande d'instance avec d'autres paramètres provenant d'itinéraires dans la méthode du contrôleur

Parfois, nous devons accepter les paramètres de route et accéder aux paramètres de requête HTTP. Nous pouvons toujours taper le conseil la classe Requests dans le contrôleur laravel et réaliser cela comme expliqué ci-dessous

Par exemple, nous avons un itinéraire qui met à jour un article comme celui-ci

```
Route::put('post/{id}', 'PostController@update');
```

En outre, depuis que l'utilisateur a modifié d'autres champs de formulaire d'édition, il sera disponible dans HTTP Request

Voici comment accéder à la fois à notre méthode

```
    public function update(Request $request,$id){
        //This way we have $id param from route and $request as an HTTP Request object

    }
```

Lire Demandes en ligne: https://riptutorial.com/fr/laravel/topic/4929/demandes

# Chapitre 18: Démarrer avec laravel-5.3

## Remarques

Cette section fournit une vue d'ensemble de ce que laravel-5.3 est et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tout sujet important dans le cadre du programme laravel-5.3 et d'établir un lien avec les sujets connexes. La documentation de laravel-5.3 étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

## Examples

### Installation de Laravel

**Exigences:**
Vous avez besoin de `PHP >= 5.6.4` et `Composer` installés sur votre machine. Vous pouvez vérifier la version des deux en utilisant la commande:
**Pour PHP:**

```
php -v
```

Sortie comme ceci:

```
PHP 7.0.9 (cli) (built: Aug 26 2016 06:17:04) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

**Pour compositeur**
Vous pouvez exécuter la commande sur votre terminal / CMD:

```
composer --version
```

Sortie comme ceci:

```
composer version 1.2.1 2016-09-12 11:27:19
```

Laravel utilise Composer pour gérer ses dépendances. Donc, avant d'utiliser Laravel, assurez-vous que Composer est installé sur votre machine.

## Via Laravel Installer

Tout d'abord, téléchargez l'installateur Laravel en utilisant Composer:

```
composer global require "laravel/installer"
```

Assurez-vous de placer le `$HOME/.composer/vendor/bin` (ou le répertoire équivalent de votre système d'exploitation) dans votre $ PATH pour que l'exécutable `laravel` puisse être localisé par votre système.

Une fois installée, la commande `laravel new` créera une nouvelle installation Laravel dans le répertoire que vous spécifiez. Par exemple, `laravel new blog` va créer un répertoire nommé `blog` contenant une nouvelle installation de Laravel avec toutes les dépendances de Laravel déjà installées:

```
laravel new blog
```

# Via Composer Create-Project

Vous pouvez également installer Laravel en émettant la commande Composer `create-project` dans votre terminal:

```
composer create-project --prefer-dist laravel/laravel blog
```

# Installer

Une fois l'installation de Laravel terminée, vous devrez définir des `permissions` pour les dossiers de stockage et de démarrage.

> **Remarque: la** définition des `permissions` est l'un des processus les plus importants à effectuer lors de l'installation de Laravel.

**Serveur de développement local**

Si vous avez installé PHP localement et que vous souhaitez utiliser le serveur de développement intégré de PHP pour servir votre application, vous pouvez utiliser la commande `serve` Artisan. Cette commande démarre un serveur de développement à l' `http://localhost:8000` :

```
php artisan serve
```

Ouvrez l'URL de demande de votre navigateur `http://localhost:8000`

## Exigences du serveur

Le framework Laravel a quelques exigences système. Bien sûr, toutes les exigences sont satisfaites par la machine virtuelle Laravel Homestead , il est donc fortement recommandé d'utiliser Homestead comme environnement de développement Laravel local.

Toutefois, si vous n'utilisez pas Homestead, vous devez vous assurer que votre serveur répond aux exigences suivantes:

- PHP> = 5.6.4
- Extension PHP OpenSSL
- Extension PHP PDO
- Extension PHP Mbstring
- Extension PHP Tokenizer
- Extension PHP XML

## Serveur de développement local

Si vous avez installé PHP localement et que vous souhaitez utiliser le serveur de développement intégré de PHP pour servir votre application, vous pouvez utiliser la commande `serve` Artisan. Cette commande démarre un serveur de développement à l' `http://localhost:8000` :

```
php artisan serve
```

Bien entendu, des options de développement local plus robustes sont disponibles via Homestead et Valet .

Il est également possible d'utiliser un port personnalisé, quelque chose comme `8080` . Vous pouvez le faire avec l'option `--port` .

```
php artisan serve --port=8080
```

Si vous avez un domaine local dans votre fichier hosts, vous pouvez définir le nom d'hôte. Cela peut être fait par l'option `--host` .

```
php artisan serve --host=example.dev
```

Vous pouvez également exécuter sur un hôte et un port personnalisés, cela peut être fait par la commande suivante.

```
php artisan serve --host=example.dev --port=8080
```

## Hello World Example (Basic) et utiliser une vue

**L'exemple de base**
Ouvrez le fichier `routes/web.php` et collez le code suivant dans le fichier:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

ici ' **helloworld** ' agira comme nom de page **auquel** vous voulez accéder,

et si vous ne voulez pas créer de fichier lame et que vous voulez toujours accéder directement à la page, vous pouvez utiliser le routage laravel de cette façon.

Maintenant, tapez `localhost/helloworld` dans la barre d'adresse du navigateur et vous pouvez accéder à la page affichant Hello World.

**L'étape suivante.**
Vous avez donc appris à créer un Hello World très simple! page en renvoyant une phrase de salut du monde. Mais on peut le rendre un peu mieux!

Étape 1.
Nous allons recommencer à notre fichier `routes/web.php` maintenant, au lieu d'utiliser le code ci-dessus, nous utiliserons le code suivant:

```
Route::get('helloworld', function() {
    return view('helloworld');
});
```

La valeur de retour cette fois-ci n'est pas simplement un simple texte helloworld, mais une vue. Une vue dans Laravel est simplement un nouveau fichier. Ce fichier "helloworld" contient le HTML et peut-être plus tard même du PHP du texte Helloworld.

Étape 2.
Maintenant que nous avons ajusté notre itinéraire pour faire appel à une vue, nous allons faire le point. Laravel travaille avec des fichiers blade.php dans les vues. Donc, dans ce cas, notre itinéraire s'appelle helloworld. Donc notre vue s'appellera `helloworld.blade.php`

Nous allons créer le nouveau fichier dans le répertoire `resources/views` et nous l'appellerons helloworld.blade.php

Nous allons maintenant ouvrir ce nouveau fichier et le modifier en créant notre phrase Hello World. Nous pouvons ajouter plusieurs façons différentes d'obtenir notre phrase comme dans l'exemple ci-dessous.

```
<html>
    <body>
        <h1> Hello World! </h1>

        <?php
            echo "Hello PHP World!";
        ?>

    </body>
</html>
```

allez maintenant dans votre navigateur et tapez à nouveau votre itinéraire comme dans l'exemple de base: `localhost/helloworld` vous verrez votre nouvelle vue créée avec tout le contenu!

## Bonjour Monde Exemple (Basic)

Ouvrir le fichier des itinéraires. Collez le code suivant dans:

```
Route::get('helloworld', function () {
```

```
    return '<h1>Hello World</h1>';
});
```

Après avoir `http://localhost/helloworld` il affiche Hello World.

Le fichier des itinéraires est situé `/routes/web.php`

## Configuration du serveur Web pour les URL Pretty

Si vous avez installé `Laravel` via `Composer or the Laravel installer` , vous aurez besoin de la configuration ci-dessous.

**La configuration d'Apache** Laravel comprend un fichier `public/.htaccess` utilisé pour fournir des URL sans le contrôleur frontal `index.php` dans le chemin. Avant de servir Laravel avec Apache, veillez à activer le module `mod_rewrite` afin que le fichier `.htaccess` soit honoré par le serveur.

Si le fichier `.htaccess` fourni avec Laravel ne fonctionne pas avec votre installation Apache, essayez cette alternative:

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

---

**Configuration pour Nginx** Si vous utilisez Nginx, la directive suivante dans la configuration de votre site dirigera toutes les requêtes vers le contrôleur frontal `index.php` :

```
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

Bien sûr, lorsque vous utilisez Homestead ou Valet , de jolies URL seront automatiquement configurées.

Lire Démarrer avec laravel-5.3 en ligne: https://riptutorial.com/fr/laravel/topic/8602/demarrer-avec-laravel-5-3

# Chapitre 19: Déployer l'application Laravel 5 sur l'hébergement partagé sur un serveur Linux

## Remarques

Pour obtenir plus d'informations sur le déploiement du projet Laravel sur l'hébergement partagé, visitez ce dépôt Github.

## Examples

### Application Laravel 5 sur l'hébergement partagé sur un serveur Linux

Par défaut, le dossier `public` du projet Laravel expose le contenu de l'application qui peut être demandé de n'importe où par n'importe qui, le reste du code de l'application est invisible ou inaccessible à quiconque sans les autorisations appropriées.

Après avoir développé l'application sur votre machine de développement, celle-ci doit être transmise à un serveur de production pour pouvoir y accéder via Internet, où que vous soyez.

Pour la plupart des applications / sites Web, le premier choix consiste à utiliser le package d'hébergement partagé de fournisseurs de services d'hébergement tels que GoDaddy, HostGator, etc., principalement en raison de leur faible coût.

> **note** : vous pouvez demander à votre fournisseur de modifier manuellement **document_root** , il vous suffit donc de télécharger votre application Laravel sur le serveur (via FTP), de demander le remplacement de root par **{app} / public** et vous devriez être bon.

De tels packs d'hébergement partagés ont toutefois des limitations en termes d'accès aux terminaux et d'autorisations de fichiers. Par défaut, il faut télécharger son application / code dans le dossier `public_html` de son compte d'hébergement partagé.

Donc, si vous voulez télécharger un projet Laravel sur un compte d'hébergement partagé, comment le feriez-vous? Devez-vous télécharger l'application entière (dossier) dans le dossier `public_html` sur votre compte d'hébergement partagé? - **Certainement NON**

Parce que tout dans le dossier `public_html` est accessible "publiquement, c'est-à-dire par quiconque", ce qui constituerait un gros risque pour la sécurité.

Étapes pour télécharger un projet sur un compte d'hébergement partagé - la méthode Laravel

**Étape 1**
Créez un dossier appelé laravel (ou tout ce que vous voulez) au même niveau que le dossier

`public_html` .

```
Eg:
/
|--var
    |---www
        |----laravel       //create this folder in your shared hosting account
        |----public_html
        |----log
```

**Étape 2**
Copiez tout sauf le dossier `public` de votre projet laravel (sur la machine de développement) dans le dossier `laravel` (sur l'hôte du serveur - compte d'hébergement partagé).
Vous pouvez utiliser:

- C-panel: qui serait l'option la plus lente
- Client FTP: comme ***FileZilla*** pour vous connecter à votre compte d'hébergement partagé et transférer vos fichiers et dossiers via le téléchargement FTP
- Map Network Drive: vous pouvez également créer un lecteur réseau mappé sur votre machine de développement pour vous connecter au dossier racine de votre compte d'hébergement partagé à l'aide de l'adresse " ftp: // nom-de-votre-domaine ".

**Étape 3**
Ouvrez le dossier `public` de votre projet laravel (sur la machine de développement), copiez tout et collez-le dans le dossier `public_html` (sur l'hôte du serveur - compte d'hébergement partagé).
**Étape 4**
Ouvrez maintenant le fichier `index.php` dans le dossier `public_html` du compte d'hébergement partagé (dans l'éditeur cpanel ou tout autre éditeur connecté) et:

***Changement:***

```
require __DIR__.'/../bootstrap/autoload.php';
```

***À:***

```
require __DIR__.'/../laravel/bootstrap/autoload.php';
```

***Et changer:***

```
$app = require_once __DIR__.'/../bootstrap/app.php';
```

***À:***

```
$app = require_once __DIR__.'/../laravel/bootstrap/app.php';
```

***Sauver et fermer.***

**Étape 5**
Allez maintenant dans le dossier `laravel` (sur le `laravel` hébergement partagé) et ouvrez le fichier

`server.php`

***Changement***

```
require_once __DIR__.'/public/index.php';
```

***À:***

```
require_once __DIR__.'../public_html/index.php';
```

***Sauver et fermer.***

**Étape 6**
Définissez les autorisations de fichiers pour le dossier `laravel/storage` (de manière récursive) et tous les fichiers, sous-dossiers et fichiers qui s'y `laravel/storage` sur le compte d'hébergement partagé - serveur sur `777` .
***Remarque:*** Soyez prudent avec les autorisations de fichier dans Linux, elles sont comme une épée à double tranchant, si elles ne sont pas utilisées correctement, elles peuvent rendre votre application vulnérable aux attaques. Pour comprendre les permissions des fichiers Linux, vous pouvez lire https://www.linux.com/learn/tutorials/309527-understanding-linux-file-permissions

**Étape 7**

Comme le fichier `.env` du serveur local / de développement est ignoré par git et qu'il doit être ignoré car il possède toutes les variables d'environnement, y compris APP_KEY, il ne doit pas être exposé au public en le poussant dans les référentiels. Vous pouvez également voir que le fichier `.gitignore` a `.env` mentionné, il ne sera donc pas téléchargé sur les référentiels.

Après avoir suivi toutes les étapes ci-dessus, créez un fichier `.env` dans le dossier laravel et ajoutez toute la variable d'environnement que vous avez utilisée du fichier `.env` du serveur local / de développement au fichier `.env` du serveur de production.

Même il y a des fichiers de configuration comme `app.php` , `database.php` dans le dossier config de l'application laravel qui définit ces variables par défaut dans le second paramètre de `env()` mais ne codifie pas les valeurs de ces fichiers car cela affectera la les fichiers de configuration des utilisateurs qui récupèrent votre référentiel. Il est donc recommandé de créer `.env` fichier `.env` manuellement!

Aussi laravel donne le fichier `.env-example` que vous pouvez utiliser comme référence.

C'est tout.

Maintenant, lorsque vous visitez l'URL que vous avez configurée en tant que domaine avec votre serveur, votre application laravel doit fonctionner exactement comme sur votre machine de développement localhost, alors que le code de l'application est sécurisé et inaccessible à toute personne sans autorisations de fichier appropriées.

Lire Déployer l'application Laravel 5 sur l'hébergement partagé sur un serveur Linux en ligne: https://riptutorial.com/fr/laravel/topic/2410/deployer-l-application-laravel-5-sur-l-hebergement-

partage-sur-un-serveur-linux

# Chapitre 20: Des aides

## Introduction

Les aides Laravel sont les fonctions accessibles globalement définies par le framework. Il peut être directement appelé et utilisé indépendamment n'importe où dans l'application sans avoir à instancier un objet ou une classe d'importation.

Il existe des aides pour manipuler des *tableaux* , des *chemins* , des *chaînes* , des *URL* , etc.

## Examples

### Méthodes de tableau

#### array_add ()

Cette méthode est utilisée pour ajouter de nouvelles paires de valeurs de clé à un tableau.

```
$array = ['username' => 'testuser'];

$array = array_add($array, 'age', 18);
```

résultat

```
['username' => 'testuser', 'age' => 18]
```

### Méthodes de chaîne

#### affaire de chameau()

Cette méthode change une chaîne en cas de chameau

```
camel_case('hello_world');
```

résultat

```
HelloWorld
```

### Méthodes de parcours

Les méthodes de chemin d'accès facilitent l'accès aux chemins d'accès aux applications depuis n'importe où.

#### chemin_public ()

Cette méthode renvoie le chemin d'accès public complet de l'application. qui est le répertoire public.

```
$path = public_path();
```

## Urls

### url ()

La fonction url génère une URL qualifiée complète pour le chemin donné.

si votre site est `hello.com`

```
echo url('my/dashboard');
```

retournerais

```
hello.com/my/dashboard
```

si rien n'est passé à la méthode url, elle renverrait une instance de `Illuminate\Routing\UrlGenerator` et pourrait être utilisée comme ceci

retournerait l'URL actuelle

```
echo url()->current();
```

retournerait l'URL complète

```
echo url()->full();
```

retournerait l'URL précédente

```
echo url()->previous();
```

Lire Des aides en ligne: https://riptutorial.com/fr/laravel/topic/8827/des-aides

# Chapitre 21: Éloquent

## Introduction

L'Eloquent est un ORM (Object Relational Model) inclus avec le Laravel. Il implémente le modèle d'enregistrement actif et est utilisé pour interagir avec des bases de données relationnelles.

## Remarques

### Nom de table

La convention est d'utiliser «snake_case» pluralisé pour les noms de tables et «StudlyCase» singulier pour les noms de modèles. Par exemple:

- Une table de `cats` aurait un modèle `Cat`
- Une table `jungle_cats` aurait un modèle `JungleCat`
- Une table d' `users` aurait un modèle d' `User`
- Une table de `people` aurait un modèle `Person`

Eloquent essaiera automatiquement de lier votre modèle avec une table qui a le pluriel du nom du modèle, comme indiqué ci-dessus.

Vous pouvez cependant spécifier un nom de table pour remplacer la convention par défaut.

```
class User extends Model
{
    protected $table = 'customers';
}
```

## Examples

### introduction

Eloquent est l' ORM intégré dans le cadre Laravel. Il vous permet d'interagir avec vos tables de base de données de manière orientée objet, en utilisant le modèle ActiveRecord .

Une seule classe de modèle est généralement mappée sur une seule table de base de données et les relations de différents types ( un à un , un à plusieurs , plusieurs à plusieurs , polymorphes) peuvent être définies entre différentes classes de modèles.

Section Créer un modèle décrit la création et la définition des classes de modèle.

Avant de pouvoir utiliser les modèles Eloquent, assurez-vous qu'au moins une connexion à la base de données a été configurée dans votre fichier de configuration `config/database.php` .

Pour comprendre l'utilisation du générateur de requêtes éloquent pendant le développement, vous

pouvez utiliser la commande `php artisan ide-helper:generate` . Voici le lien .

---

# Sous-sujet Navigation

Relation éloquente

### Persister

Outre la lecture des données avec Eloquent, vous pouvez également l'utiliser pour insérer ou mettre à jour des données avec la méthode `save()` . Si vous avez créé une nouvelle instance de modèle, l'enregistrement sera *inséré* . sinon, si vous avez récupéré un modèle de la base de données et défini de nouvelles valeurs, il sera *mis* à *jour* .

Dans cet exemple, nous créons un nouvel enregistrement d' `User` :

```
$user = new User();
$user->first_name = 'John';
$user->last_name = 'Doe';
$user->email = 'john.doe@example.com';
$user->password = bcrypt('my_password');
$user->save();
```

Vous pouvez également utiliser la méthode `create` pour renseigner les champs à l'aide d'un tableau de données:

```
User::create([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

Lorsque vous utilisez la méthode create, vos attributs doivent être déclarés dans le tableau `fillable` de votre modèle:

```
class User extends Model
{
    protected $fillable = [
         'first_name',
         'last_name',
         'email',
         'password',
    ];
}
```

Si vous souhaitez que tous les attributs puissent être assignés en masse, vous pouvez également définir la propriété $ guarded comme un tableau vide:

```
class User extends Model
{
```

```
    /**
    * The attributes that aren't mass assignable.
    *
    * @var array
    */
    protected $guarded = [];
}
```

Mais vous pouvez également créer un enregistrement sans même changer l'attribut `fillable` dans votre modèle en utilisant la méthode `forceCreate` plutôt que de `create` méthode

```
User::forceCreate([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

Voici un exemple de mise à jour d'un modèle d' `User` existant en le chargeant d'abord (en utilisant `find` ), en le modifiant, puis en l'enregistrant:

```
$user = User::find(1);
$user->password = bcrypt('my_new_password');
$user->save();
```

Pour accomplir le même exploit avec un seul appel de fonction, vous pouvez utiliser la méthode de `update` :

```
$user->update([
    'password' => bcrypt('my_new_password'),
]);
```

Les méthodes de `create` et de `update` simplifient le travail avec de grands ensembles de données plutôt que de devoir définir chaque paire clé / valeur individuellement, comme le montrent les exemples suivants:

> Notez l'utilisation de `only` et `except` lors de la collecte des données de demande. Il est important de spécifier les clés exactes que vous souhaitez autoriser / interdire pour la mise à jour, sinon il est possible pour un attaquant d'envoyer des champs supplémentaires avec leur requête et de provoquer des mises à jour involontaires.

```
// Updating a user from specific request data
$data = Request::only(['first_name', 'email']);
$user->find(1);
$user->update($data);

// Create a user from specific request data
$data = Request::except(['_token', 'profile_picture', 'profile_name']);
$user->create($data);
```

## Effacer

Vous pouvez supprimer des données après les avoir écrites dans la base de données. Vous pouvez supprimer une instance de modèle si vous en avez extrait une ou spécifier des conditions pour les enregistrements à supprimer.

Pour supprimer une instance de modèle, récupérez-la et appelez la méthode `delete()` :

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

Vous pouvez également spécifier une clé primaire (ou un tableau de clés primaires) des enregistrements que vous souhaitez supprimer via la méthode `destroy()` :

```
User::destroy(1);
User::destroy([1, 2, 3]);
```

Vous pouvez également combiner l'interrogation avec la suppression:

```
User::where('age', '<', 21)->delete();
```

Cela supprimera tous les utilisateurs correspondant à la condition.

> Remarque: Lors de l' exécution d' une masse delete via Eloquent, la `deleting` et `deleted` les événements du modèle ne seront pas licenciés pour les modèles supprimés. En effet, les modèles ne sont jamais récupérés lors de l'exécution de l'instruction delete.

## Suppression Douce

Parfois, vous ne souhaitez pas supprimer définitivement un enregistrement, mais le conserve pour des besoins d'audit ou de création de rapports. Pour cela, Eloquent fournit *une* fonctionnalité de *suppression en douceur* .

Pour ajouter des fonctionnalités de suppression logicielle à votre modèle, vous devez importer la caractéristique `SoftDeletes` et l'ajouter à votre classe de modèle Eloquent:

```
namespace Illuminate\Database\Eloquent\Model;
namespace Illuminate\Database\Eloquent\SoftDeletes;

class User extends Model
{
     use SoftDeletes;
}
```

Lors de la suppression d'un modèle, il définira un horodatage sur la colonne d'horodatage `deleted_at` de la table pour votre modèle. Veillez donc à créer d' `deleted_at` colonne `deleted_at` dans votre table. Ou dans la migration, vous devez appeler la méthode `softDeletes()` sur votre plan pour ajouter l'horodatage `deleted_at` . Exemple:

```
Schema::table('users', function ($table) {
    $table->softDeletes();
```

```
});
```

Toute requête omettra les enregistrements supprimés. Vous pouvez les afficher de force si vous le souhaitez en utilisant la `withTrashed()` :

```
User::withTrashed()->get();
```

Si vous souhaitez autoriser les utilisateurs à *restaurer* un enregistrement après une suppression en douceur (dans une zone de type corbeille), vous pouvez utiliser la méthode `restore()` :

```
$user = User::find(1);
$user->delete();
$user->restore();
```

Pour supprimer un enregistrement avec force, utilisez la méthode `forceDelete()` qui supprime véritablement l'enregistrement de la base de données:

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

## Modifier la clé primaire et les horodatages

Par défaut, les modèles Eloquent s'attendent à ce que la clé primaire soit nommée `'id'` . Si ce n'est pas votre cas, vous pouvez modifier le nom de votre clé primaire en spécifiant la propriété `$primaryKey` .

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    // ...
}
```

Maintenant, toutes les méthodes Eloquent qui utilisent votre clé primaire (par exemple, `find` ou `findOrFail` ) utiliseront ce nouveau nom.

De plus, Eloquent s'attend à ce que la clé primaire soit un entier à auto-incrémentation. Si votre clé primaire n'est pas un entier auto-incrémenté (par exemple un GUID), vous devez indiquer à Eloquent en mettant à jour la propriété `$incrementing` sur `false` :

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    public $incrementing = false;

    // ...
}
```

Par défaut, Eloquent s'attend à `updated_at` colonnes `created_at` et `updated_at` existent sur vos

tables. Si vous ne souhaitez pas que ces colonnes soient gérées automatiquement par Eloquent, définissez la propriété `$timestamps` de votre modèle sur false:

```
class Citizen extends Model
{
    public $timestamps = false;

    // ...
}
```

Si vous devez personnaliser les noms des colonnes utilisées pour stocker les horodatages, vous pouvez définir les constantes `CREATED_AT` et `UPDATED_AT` dans votre modèle:

```
class Citizen extends Model
{
    const CREATED_AT = 'date_of_creation';
    const UPDATED_AT = 'date_of_last_update';

    // ...
}
```

## Lancer 404 si l'entité n'est pas trouvée

Si vous voulez lancer automatiquement une exception lorsque vous recherchez un enregistrement qui n'a pas été trouvé sur un modal, vous pouvez utiliser soit

```
Vehicle::findOrFail(1);
```

ou

```
Vehicle::where('make', 'ford')->firstOrFail();
```

Si un enregistrement avec la clé primaire de `1` n'est pas trouvé, une `ModelNotFoundException` est levée. Ce qui est essentiellement la même chose que l'écriture ( voir la source ):

```
$vehicle = Vehicle::find($id);

if (!$vehicle) {
    abort(404);
}
```

## Modèles de clonage

Vous devrez peut-être cloner une ligne, peut-être changer quelques attributs, mais vous avez besoin d'un moyen efficace de garder les choses SÈCHES. Laravel fournit une sorte de méthode «cachée» pour vous permettre de faire cette fonctionnalité. Bien qu'il ne soit pas documenté, vous devez chercher dans l'API pour le trouver.

En utilisant `$model->replicate()` vous pouvez facilement cloner un enregistrement

```
$robot = Robot::find(1);
$cloneRobot = $robot->replicate();
// You can add custom attributes here, for example he may want to evolve with an extra arm!
$cloneRobot->arms += 1;
$cloneRobot->save();
```

Le ci-dessus trouverait un robot qui a un identifiant de 1, puis le clone.

Lire Éloquent en ligne: https://riptutorial.com/fr/laravel/topic/865/eloquent

# Chapitre 22: Eloquent: Accessors & Mutators

## Introduction

Les accesseurs et les mutateurs vous permettent de formater des valeurs d'attribut Eloquent lorsque vous les récupérez ou les définissez sur des instances de modèle. Par exemple, vous pouvez utiliser le crypteur Laravel pour crypter une valeur stockée dans la base de données, puis décrypter automatiquement l'attribut lorsque vous y accédez sur un modèle Eloquent. En plus des accesseurs et des mutateurs personnalisés, Eloquent peut également convertir automatiquement des champs de date en instances Carbon ou même convertir des champs de texte en JSON.

## Syntaxe

- définir l'attribut {ATTRIBUTE} (attribut $) // dans le cas camel

## Examples

### Définir un accesseur

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the user's first name.
     *
     * @param  string  $value
     * @return string
     */
    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }
}
```

# Obtenir un accessoire:

Comme vous pouvez le constater, la valeur d'origine de la colonne est transmise à l'accesseur, ce qui vous permet de manipuler et de renvoyer la valeur. Pour accéder à la valeur de l'accesseur, vous pouvez simplement accéder à l'attribut `first_name` sur une instance de modèle:

```
$user = App\User::find(1);
$firstName = $user->first_name;
```

## Définir un mutateur

```
class User extends Model
{
    public function setPasswordAttribute($password)
    {
        $this->attributes['password'] = bcrypt($password);
    }
    ...
}
```

Le code ci-dessus fait "bcrypting" à chaque fois que la propriété password est définie.

```
$user = $users->first();
$user->password = 'white rabbit'; //laravel calls mutator on background
$user->save(); // password is bcrypted and one does not need to call bcrypt('white rabbit')
```

Lire Eloquent: Accessors & Mutators en ligne: https://riptutorial.com/fr/laravel/topic/8305/eloquent--accessors--amp--mutators

# Chapitre 23: Eloquent: Modèle

## Examples

### Faire un modèle

### La création du modèle

Les classes de modèle doivent étendre `Illuminate\Database\Eloquent\Model` . L'emplacement par défaut des modèles est le répertoire `/app` .

Une classe de modèle peut être facilement générée par la commande Artisan :

```
php artisan make:model [ModelName]
```

Cela créera un nouveau fichier PHP dans `app/` par défaut, qui s'appelle `[ModelName].php` , et contiendra toute la langue de base pour votre nouveau modèle, qui inclut la classe, l'espace de nommage et l'utilisation requise pour une configuration de base.

Si vous souhaitez créer un fichier de migration avec votre modèle, utilisez la commande suivante, où `-m` générera également le fichier de migration:

```
php artisan make:model [ModelName] -m
```

Outre la création du modèle, cette opération crée une migration de base de données connectée au modèle. Le fichier PHP de migration de la base de données se trouve par défaut dans `database/migrations/` . Par défaut, cela n'inclut rien d'autre que les colonnes id et `created_at` / `updated_at` . Vous devrez donc modifier le fichier pour fournir des colonnes supplémentaires.

Notez que vous devrez exécuter la migration (une fois que vous avez configuré le fichier de migration) pour que le modèle puisse commencer à utiliser `php artisan migrate` depuis la racine du projet.

De plus, si vous souhaitez ajouter une migration ultérieurement, vous pouvez le faire en exécutant le modèle suivant:

```
php artisan make:migration [migration name]
```

---

Disons par exemple que vous vouliez créer un modèle pour vos chats, vous auriez deux choix, créer avec ou sans migration. Vous auriez choisi de créer sans migration si vous aviez déjà une table de chats ou si vous ne souhaitiez pas en créer une pour le moment.

Pour cet exemple, nous voulons créer une migration car nous n’avons pas déjà de table, donc nous allons exécuter la commande suivante.

```
php artisan make:model Cat -m
```

Cette commande va créer deux fichiers:

1. Dans le dossier App: `app/Cat.php`
2. Dans le dossier de la base de données: `database/migrations/timestamp_creat_cats_table.php`

Le fichier qui nous intéresse est celui-ci car c'est ce fichier que nous pouvons décider de ce que nous voulons que la table ressemble et soit inclus. Pour toute migration prédéfinie, une colonne d'auto-incrémentation automatique et des colonnes d'horodatage sont attribuées.

L'exemple ci-dessous d'un extrait du fichier de migration inclut les colonnes prédéfinies ci-dessus ainsi que l'ajout du nom du chat, de l'âge et de la couleur:

```
public function up()
    {
        Schema::create('cats', function (Blueprint $table) {

            $table->increments('id');     //Predefined ID
            $table->string('name');       //Name
            $table->integer('age');       //Age
            $table->string('colour');     //Colour
            $table->timestamps();         //Predefined Timestamps

        });
    }
```

Comme vous pouvez le constater, il est relativement facile de créer le modèle et la migration pour une table. Ensuite, pour exécuter la migration et la créer dans votre base de données, exécutez la commande suivante:

```
php artisan migrate
```

Qui va migrer toutes les migrations en attente vers votre base de données.

---

## Emplacement du fichier de modèle

Les modèles peuvent être stockés n'importe où grâce au PSR4 .

Par défaut, les modèles sont créés dans le répertoire `app` avec l'espace de noms de l' `App` . Pour les applications plus complexes, il est généralement recommandé de stocker les modèles dans leurs propres dossiers dans une structure adaptée à l'architecture de vos applications.

Par exemple, si vous avez une application qui utilise une série de fruits en tant que modèles, vous pouvez créer un dossier appelé `app/Fruits` et dans ce dossier, vous créez `Banana.php` (en respectant la convention de nommage StudlyCase ). dans l'espace de noms `App\Fruits` :

```
namespace App\Fruits;

use Illuminate\Database\Eloquent\Model;
```

```
class Banana extends Model {
    // Implementation of "Banana" omitted
}
```

## Configuration du modèle

Eloquent suit une approche "convention over configuration". En étendant la classe `Model` base, tous les modèles héritent des propriétés répertoriées ci-dessous. Sauf en cas de substitution, les valeurs par défaut suivantes s'appliquent:

| Propriété | La description | Défaut |
|---|---|---|
| `protected $connection` | Nom de connexion à la base de données | Connexion DB par défaut |
| `protected $table` | Nom de la table | Par défaut, le nom de la classe est converti en `snake_case` et pluralized. Par exemple, `SpecialPerson` devient `special_people` |
| `protected $primaryKey` | Tableau PK | `id` |
| `public $incrementing` | Indique si les ID sont auto-incrémentés | `true` |
| `public $timestamps` | Indique si le modèle doit être horodaté | `true` |
| `const CREATED_AT` | Nom de la colonne d'horodatage de création | `created_at` |
| `const UPDATED_AT` | Nom de la colonne d'horodatage de modification | `updated_at` |
| `protected $dates` | Attributs devant être mutés à DateTime, en plus des attributs d'horodatage | `[]` |
| `protected $dateFormat` | Format dans lequel les attributs de date seront persistés | Par défaut pour le dialecte SQL actuel. |
| `protected $with` | Relations avec impatience avec le modèle | `[]` |
| `protected $hidden` | Attributs omis dans la sérialisation du modèle | `[]` |
| `protected $visible` | Attributs autorisés dans la | `[]` |

| Propriété | La description | Défaut |
|---|---|---|
| | sérialisation du modèle | |
| `protected $appends` | Accesseurs d'attributs ajoutés à la sérialisation du modèle | `[]` |
| `protected $fillable` | Attributs pouvant être assignés en masse | `[]` |
| `protected $guarded` | Attributs classés par liste noire de l'affectation de masse | `[*]` (Tous les attributs) |
| `protected $touches` | Les relations à toucher lors de la sauvegarde | `[]` |
| `protected $perPage` | Le nombre de modèles à retourner pour la pagination. | `15` |

5.0

| Propriété | La description | Défaut |
|---|---|---|
| `protected $casts` | Attributs devant être convertis en types natifs | `[]` |

### Mettre à jour un modèle existant

```
$user = User::find(1);
$user->name = 'abc';
$user->save();
```

Vous pouvez également mettre à jour plusieurs attributs à la fois en utilisant `update` , ce qui ne nécessite pas l'utilisation de `save` after:

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz']);
```

Vous pouvez également mettre à jour un modèle sans l'interroger au préalable:

```
User::where('id', '>', 2)->update(['location' => 'xyz']);
```

Si vous ne voulez pas déclencher une modification de l'horodatage `updated_at` sur le modèle, vous pouvez passer l'option `touch` :

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz'], ['touch' => false]);
```

Lire Eloquent: Modèle en ligne: https://riptutorial.com/fr/laravel/topic/7984/eloquent--modele

# Chapitre 24: Eloquent: Relation

## Examples

### Interroger sur les relations

Eloquent vous permet également d'interroger sur des relations définies, comme indiqué ci-dessous:

```
User::whereHas('articles', function (Builder $query) {
    $query->where('published', '!=', true);
})->get();
```

Cela nécessite que le nom de votre méthode de relation soit des `articles` dans ce cas. L'argument transmis dans la fermeture est le Générateur de requêtes pour le modèle associé. Vous pouvez donc utiliser toutes les requêtes que vous pouvez trouver ailleurs.

**Envie de chargement**

Supposons que le modèle Utilisateur ait une relation avec le modèle Article et que vous souhaitez charger les articles associés. Cela signifie que les articles de l'utilisateur seront chargés lors de la récupération de l'utilisateur.

`articles` est le nom de la relation (méthode) dans le modèle utilisateur.

```
User::with('articles')->get();
```

si vous avez plusieurs relations. par exemple des articles et des articles.

```
User::with('articles','posts')->get();
```

et pour sélectionner des relations imbriquées

```
User::with('posts.comments')->get();
```

Appeler plusieurs relations imbriquées

```
User::with('posts.comments.likes')->get()
```

### Insertion de modèles associés

Supposons que vous ayez un modèle `Post` avec une relation `hasMany` avec `Comment` . Vous pouvez insérer un objet `Comment` associé à une publication en procédant comme suit:

```
$post = Post::find(1);
```

```
$commentToAdd = new Comment(['message' => 'This is a comment.']);

$post->comments()->save($commentToAdd);
```

Vous pouvez enregistrer plusieurs modèles à la fois en utilisant la fonction `saveMany` :

```
$post = Post::find(1);

$post->comments()->saveMany([
    new Comment(['message' => 'This a new comment']),
    new Comment(['message' => 'Me too!']),
    new Comment(['message' => 'Eloquent is awesome!'])
]);
```

Il existe également une méthode `create` qui accepte un tableau PHP simple au lieu d'une instance de modèle Eloquent.

```
$post = Post::find(1);

$post->comments()->create([
    'message' => 'This is a new comment message'
]);
```

### introduction

Les relations éloquentes sont définies comme des fonctions sur vos classes de modèles Eloquent. Étant donné que, tout comme les modèles Eloquent eux-mêmes, les relations servent également de puissants générateurs de requêtes, la définition de relations en tant que fonctions fournit de puissantes fonctions de chaînage et d'interrogation de méthodes. Par exemple, nous pouvons enchaîner des contraintes supplémentaires sur cette relation d'articles:

```
$user->posts()->where('active', 1)->get();
```

Naviguer vers le sujet parent

### Types de relation

## Un à plusieurs

Disons que chaque message peut avoir un ou plusieurs commentaires et chaque commentaire appartient à un seul message.

donc la table de commentaires aura `post_id` . Dans ce cas, les relations seront les suivantes.

**Post modèle**

```
public function comments()
{
   return $this->belongsTo(Post::class);
}
```

Si la clé étrangère est autre que `post_id` , par exemple la clé étrangère est `example_post_id` .

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id');
}
```

et plus, si la clé locale est autre que `id` , par exemple la clé locale est `other_id`

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id', 'other_id');
}
```

**Modèle de commentaire**

définition inverse de un à plusieurs

```
public function post()
{
   return $this->hasMany(Comment::class);
}
```

# Un par un

# Comment associer deux modèles (exemple: modèle `User` et `Phone` )

`App\User`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone record associated with the user.
     */
    public function phone()
    {
        return $this->hasOne('Phone::class', 'foreign_key', 'local_key');
    }
}
```

`App\Phone`

```
<?php

namespace App;
```

```
use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo('User::class', 'foreign_key', 'local_key');
    }
}
```

`foreign_key` : Par défaut, Eloquent supposera que cette valeur est `other_model_name_id` (dans ce cas, `user_id` et `phone_id` ), modifiez-le si ce n'est pas le cas.

`local_key` : Par défaut, Eloquent supposera que cette valeur est `id` (clé primaire du modèle actuel), modifiez-la si ce n'est pas le cas.

> Si votre nom de base de données est déposé conformément au standard Laravel, vous n'avez pas besoin de fournir de déclaration de clé étrangère et de clé locale dans la relation

### Explication

## Plusieurs à plusieurs

Disons qu'il y a des rôles et des autorisations. Chaque rôle peut appartenir à de nombreuses autorisations et chaque autorisation peut appartenir à plusieurs rôles. il y aura donc 3 tables. deux modèles et un tableau pivotant. une table `roles` , `users` et `permission_role` .

**Modèle**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

**Modèle d'autorisation**

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

***Note 1***

*envisagez de suivre tout en utilisant un nom de table différent pour le tableau croisé dynamique.*

Supposons que vous souhaitiez utiliser `role_permission` au lieu de `permission_role` , car eloquent

utilise l'ordre alphabétique pour construire les noms de clés pivot. Vous devrez passer le nom du tableau croisé comme second paramètre comme suit.

**Modèle**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

**Modèle d'autorisation**

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

***Note 2***

*envisagez de suivre tout en utilisant différents noms de clés dans le tableau croisé dynamique.*

Eloquent suppose que si aucune clé n'est passée en troisième et quatrième paramètres, ce seront les noms de table singulier avec `_id` . il suppose donc que le pivot aura les champs `role_id` et `permission_id` . Si des clés autres que celles-ci doivent être utilisées, elles doivent être transmises en troisième et quatrième paramètres.

Disons que `other_role_id` au lieu de `role_id` et `other_permission_id` au lieu de `permission_id` doivent être utilisés. Ce serait donc comme suit.

**Modèle**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

**Modèle d'autorisation**

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

# Polymorphe

Les relations polymorphes permettent à un modèle d'appartenir à plusieurs autres modèles sur une même association. Un bon exemple serait les images, à la fois un utilisateur et un produit peuvent avoir une image. La structure de la table peut ressembler à ceci:

```
user
    id - integer
    name - string
    email - string

product
    id - integer
    title - string
    SKU - string

image
    id - integer
    url - string
    imageable_id - integer
    imageable_type - string
```

Les colonnes importantes à examiner se trouvent dans le tableau des images. La colonne `imageable_id` contiendra la valeur d'ID de l'utilisateur ou du produit, tandis que la colonne `imageable_type` contiendra le nom de la classe du modèle propriétaire. Dans vos modèles, vous configurez les relations comme suit:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Image extends Model
{
    /**
     * Get all of the owning imageable models.
     */
    public function imageable()
    {
        return $this->morphTo();
    }
}

class User extends Model
{
    /**
     * Get all of the user's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}

class Product extends Model
{
    /**
     * Get all of the product's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}
```

Vous pouvez également récupérer le propriétaire d'une relation polymorphe à partir du modèle polymorphe en accédant au nom de la méthode qui effectue l'appel à `morphTo` . Dans notre cas, c'est la méthode `imageable` sur le modèle Image. Nous allons donc accéder à cette méthode en tant que propriété dynamique

```
$image = App\Image::find(1);

$imageable = $image->imageable;
```

Cet `imageable` renverra un utilisateur ou un produit.

## Plusieurs à plusieurs

Disons qu'il y a des rôles et des autorisations. Chaque rôle peut appartenir à de nombreuses autorisations et chaque autorisation peut appartenir à plusieurs rôles. il y aura donc 3 tables. deux modèles et un tableau pivotant. une table `roles` , `users` et `permission_role` .

**Modèle**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

**Modèle d'autorisation**

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

***Note 1***

*envisagez de suivre tout en utilisant un nom de table différent pour le tableau croisé dynamique.*

Supposons que vous souhaitiez utiliser `role_permission` au lieu de `permission_role` , car eloquent utilise l'ordre alphabétique pour construire les noms de clés pivot. Vous devrez passer le nom du tableau croisé comme second paramètre comme suit.

**Modèle**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

**Modèle d'autorisation**

```
public function roles()
{
```

```
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

***Note 2***

*envisagez de suivre tout en utilisant différents noms de clés dans le tableau croisé dynamique.*

Eloquent suppose que si aucune clé n'est passée en troisième et quatrième paramètres, ce seront les noms de table singulier avec `_id` . il suppose donc que le pivot aura les champs `role_id` et `permission_id` . Si des clés autres que celles-ci doivent être utilisées, elles doivent être transmises en troisième et quatrième paramètres.

Disons que `other_role_id` au lieu de `role_id` et `other_permission_id` au lieu de `permission_id` doivent être utilisés. Ce serait donc comme suit.

**Modèle**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

**Modèle d'autorisation**

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

**Accéder à la table intermédiaire avec withPivot ()**

Supposons que vous avez une troisième colonne ' **permission_assigned_date** ' dans le tableau croisé dynamique. Par défaut, seules les clés de modèle seront présentes sur l'objet pivot. Maintenant, pour obtenir cette colonne dans le résultat de la requête, vous devez ajouter le nom avec la fonction withPivot ().

```
   public function permissions()
        {
           return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id')->withPivot('permission_assigned_date');
        }
```

**Fixation / Détachement**

Eloquent propose également quelques méthodes d'aide supplémentaires pour faciliter le travail avec les modèles associés. Par exemple, imaginons qu'un utilisateur puisse avoir plusieurs rôles et qu'un rôle puisse avoir de nombreuses autorisations. Pour associer un rôle à une autorisation en insérant un enregistrement dans la table intermédiaire qui joint les modèles, utilisez la méthode attach:

```
$role= App\Role::find(1);
$role->permissions()->attach($permissionId);
```

Lorsque vous associez une relation à un modèle, vous pouvez également transmettre un tableau de données supplémentaires à insérer dans la table intermédiaire:

```
$rol->roles()->attach($permissionId, ['permission_assigned_date' => $date]);
```

De même, pour supprimer une autorisation spécifique pour un rôle, utilisez la fonction de détachement

```
$role= App\Role::find(1);
//will remove permission 1,2,3 against role 1
$role->permissions()->detach([1, 2, 3]);
```

**Associations de synchronisation**

Vous pouvez également utiliser la méthode sync pour construire des associations plusieurs-à-plusieurs. La méthode sync accepte un tableau d'ID à placer sur la table intermédiaire. Tout identifiant qui ne figure pas dans le tableau donné sera supprimé de la table intermédiaire. Ainsi, une fois cette opération terminée, seuls les ID du tableau donné existeront dans la table intermédiaire:

```
//will keep permission id's 1,2,3 against Role id 1

$role= App\Role::find(1)
$role->permissions()->sync([1, 2, 3]);
```

Lire Eloquent: Relation en ligne: https://riptutorial.com/fr/laravel/topic/7960/eloquent--relation

# Chapitre 25: Ensemencement

## Remarques

L'ensemencement de base de données vous permet d'insérer des données, des données de test générales dans votre base de données. Par défaut, il existe une classe `DatabaseSeeder` sous `database/seeds` .

Les semoirs en cours d'exécution peuvent être faits avec

```
php artisan db:seed
```

Ou si vous voulez seulement traiter une seule classe

```
php artisan db:seed --class=TestSeederClass
```

Comme pour toutes les commandes artisanales, vous avez accès à un large éventail de méthodes disponibles dans la documentation de l' API.

## Examples

### Insérer des données

Il existe plusieurs manières d’insérer des données:

## Utiliser la façade DB

```
public function run()
{
    DB::table('users')
        ->insert([
            'name' => 'Taylor',
            'age'  => 21
        ]);
}
```

## Via l'instanciation d'un modèle

```
public function run()
{
    $user = new User;
    $user->name = 'Taylor';
    $user->save();
}
```

## Utiliser la méthode create

```
public function run()
{
    User::create([
        'name' => 'Taylor',
        'age'  => 21
    ]);
}
```

## En utilisant l'usine

```
public function run()
{
    factory(App\User::class, 10)->create();
}
```

### Amorçage && suppression d'anciennes données et réinitialisation de l'incrémentation automatique

```
public function run()
{
    DB::table('users')->delete();
    DB::unprepared('ALTER TABLE users AUTO_INCREMENT=1;');
    factory(App\User::class, 200)->create();
}
```

Voir l'exemple persistant pour plus d'informations sur l'insertion / mise à jour des données.

### Appeler d'autres semoirs

Dans votre classe `DatabaseSeeder` , vous pouvez appeler d'autres

```
$this->call(TestSeeder::class)
```

Cela vous permet de garder un fichier où vous pouvez facilement trouver vos semoirs. Gardez à l'esprit que vous devez faire attention à l'ordre de vos appels concernant les contraintes de clés étrangères. Vous ne pouvez pas référencer un tableau qui n'existe pas encore.

### Créer un semoir

Pour créer des semoirs, vous pouvez utiliser la commande `make:seeder` Artisan. Tous les semoirs générés seront placés dans le répertoire `database/seeds` .

```
$ php artisan make:seeder MoviesTableSeeder
```

Les semences générées contiendront une méthode: `run` . Vous pouvez insérer des données dans votre base de données avec cette méthode.

```
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class MoviesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        App\Movie::create([
            'name' => 'A New Hope',
            'year' => '1977'
        ]);

        App\Movie::create([
            'name' => 'The Empire Strikes Back',
            'year' => '1980'
        ]);
    }
}
```

Vous voudrez généralement appeler tous vos classeurs dans la classe `DatabaseSeeder` .

Une fois que vous avez fini d'écrire les graisseurs, utilisez la commande `db:seed` . Cela se déroulera `DatabaseSeeder` de `run` fonction.

```
$ php artisan db:seed
```

Vous pouvez également spécifier d'exécuter une classe de `--class` spécifique pour s'exécuter individuellement à l'aide de l'option `--class` .

```
$ php artisan db:seed --class=UserSeeder
```

Si vous souhaitez restaurer et réexécuter toutes les migrations, puis ré-émettre:

```
$ php artisan migrate:refresh --seed
```

> La commande `migrate:refresh --seed` est un raccourci vers ces 3 commandes:
>
> ```
> $ php artisan migrate:reset     # rollback all migrations
> $ php artisan migrate           # run migrations
> $ php artisan db:seed           # run seeders
> ```

## Réensemencement sûr

Vous voudrez peut-être redéposer votre base de données sans affecter vos graines déjà créées. Pour cela, vous pouvez utiliser `firstOrCreate` dans votre semoir:

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
]);
```

Ensuite, vous pouvez amorcer la base de données:

```
php artisan db:seed
```

---

Plus tard, si vous souhaitez ajouter un autre type d'employé, vous pouvez simplement l'ajouter dans le même fichier:

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
]);
EmployeeType::firstOrCreate([
    'type' => 'secretary',
]);
```

Et semer à nouveau votre base de données sans aucun problème:

```
php artisan db:seed
```

Notez dans le premier appel que vous récupérez l'enregistrement mais ne faites rien avec lui.

Lire Ensemencement en ligne: https://riptutorial.com/fr/laravel/topic/3272/ensemencement

# Chapitre 26: Erreur de correspondance de jeton dans AJAX

## Introduction

J'ai analysé ce ratio pour obtenir une erreur TokenMismatch est très élevé. Et cette erreur se produit à cause de quelques erreurs stupides. Il existe de nombreuses raisons pour lesquelles les développeurs font des erreurs. Voici quelques exemples, par exemple No _token sur les en-têtes, No_token sur les données lors de l'utilisation d'Ajax, problème de permission sur le chemin de stockage, chemin de stockage de session non valide.

## Examples

### Configuration du jeton sur l'en-tête

Définissez le jeton sur `<head>` de votre `default.blade.php` .

```
<meta name="csrf-token" content="{{csrf_token()}}">
```

Ajoutez `ajaxSetup` en haut de votre script, qui sera accessible partout. Cela va définir des en-têtes sur chaque appel `ajax`

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

### Mettre le jeton sur marque

Ajoutez la fonction ci-dessous à votre `<form>` . Cette fonction générera un champ masqué nommé `_token` et une valeur remplie avec le jeton.

```
{{csrf_field()}}
```

Ajoutez la fonction `csrf_token ()` à votre `_token` caché dans l'attribut value. Cela ne générera que de la chaîne cryptée.

```
<input type="hidden" name="_token" value="{{csrf_token()}}"/> .
```

### Vérifier le chemin de stockage de la session et l'autorisation

Ici, je suppose que l'URL de l'application du projet est `APP_URL=http://project.dev/ts/toys-store`

1. Définissez l'autorisation en écriture sur `storage_path('framework/sessions')` le dossier.
2. Vérifiez le chemin de votre projet laravel `'path' => '/ts/toys-store',` la racine de votre projet laravel.
3. Changez le nom de votre cookie `'cookie' => 'toys-store',`

```
return [
    'driver' => env('SESSION_DRIVER', 'file'),
    'lifetime' => 120,
    'expire_on_close' => false,
    'encrypt' => false,
    'files' => storage_path('framework/sessions'),
    'connection' => null,
    'table' => 'sessions',
    'lottery' => [2, 100],
    'cookie' => 'toys-store',
    'path' => '/ts/toys-store',
    'domain' => null,
    'secure' => false,
    'http_only' => true,
];
```

## Utilisez le champ _token sur Ajax

Il y a plusieurs façons d'envoyer `_token` sur un appel AJAX

1. Récupère toutes les valeurs du champ de saisie dans la `<form>` utilisant `var formData = new FormData($("#cart-add")[0]);`
2. Utilisez `$("form").serialize();` ou `$("form").serializeArray();`
3. Ajoutez `_token` manuellement aux `data` d'Ajax. en utilisant `$('meta[name="csrf-token"]').attr('content')` ou `$('input[name="_token"]').val()` .
4. Nous pouvons définir comme en-tête un appel Ajax particulier comme ci-dessous le code.

```
$.ajax({
    url: $("#category-add").attr("action"),
    type: "POST",
    data: formData,
    processData: false,
    contentType: false,
    dataType: "json",
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

Lire Erreur de correspondance de jeton dans AJAX en ligne:
https://riptutorial.com/fr/laravel/topic/10656/erreur-de-correspondance-de-jeton-dans-ajax

# Chapitre 27: Essai

## Examples

### introduction

L'écriture de code testable est un élément important de la construction d'un projet robuste, maintenable et agile. La prise en charge du framework de test PHP le plus utilisé, PHPUnit , est intégrée à Laravel. PHPUnit est configuré avec le fichier `phpunit.xml` , qui réside dans le répertoire racine de chaque nouvelle application Laravel.

Le répertoire `tests` , également dans le répertoire racine, contient les fichiers de test individuels contenant la logique permettant de tester chaque partie de votre application. Bien entendu, il est de votre responsabilité en tant que développeur d'écrire ces tests lorsque vous construisez votre application, mais Laravel inclut un exemple de fichier, `ExampleTest.php` , pour vous `ExampleTest.php` à démarrer.

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\DatabaseTransactions;

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $this->visit('/')
             ->see('Laravel 5');
    }
}
```

Dans la méthode `testBasicExample()` , nous visitons la page d'index du site et nous nous assurons que le texte `Laravel 5` quelque part sur cette page. Si le texte n'est pas présent, le test échouera et générera une erreur.

### Test sans middleware et avec une nouvelle base de données

Pour que l'artisan migre une nouvelle base de données avant de lancer des tests, `use DatabaseMigrations` . Aussi, si vous voulez éviter les middleware comme Auth, `use WithoutMiddleware` .

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
```

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
    use DatabaseMigrations, WithoutMiddleware;

    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testExampleIndex()
    {
        $this->visit('/protected-page')
         ->see('All good');
    }
}
```

## Transactions de base de données pour une connexion à plusieurs bases de données

`DatabaseTransactions` trait `DatabaseTransactions` permet aux bases de données d'annuler toutes les modifications pendant les tests. Si vous souhaitez restaurer plusieurs bases de données, vous devez définir `$connectionsToTransact` propriétés `$connectionsToTransact`

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

## Configuration du test, utilisation de la base de données en mémoire

La configuration suivante garantit que le framework de test (PHPUnit) utilise `:memory:` database.

**config / database.php**

```
'connections' => [

    'sqlite_testing' => [
        'driver'   => 'sqlite',
        'database' => ':memory:',
        'prefix'   => '',
    ],
    .
    .
```

```
    .
```

**./phpunit.xml**

```
    .
    .
    .
    </filter>
    <php>
        <env name="APP_ENV" value="testing"/>
        <env name="APP_URL" value="http://example.dev"/>
        <env name="CACHE_DRIVER" value="array"/>
        <env name="SESSION_DRIVER" value="array"/>
        <env name="QUEUE_DRIVER" value="sync"/>
        <env name="DB_CONNECTION" value="sqlite_testing"/>
    </php>
</phpunit>
```

## Configuration

Le fichier phpunit.xml est le fichier de configuration par défaut pour les tests et est déjà configuré pour les tests avec PHPUnit.

L'environnement de test par défaut `APP_ENV` est défini comme `testing` avec `array` comme pilote de cache `CACHE_DRIVER` . Avec cette configuration, aucune donnée (session / cache) ne sera conservée pendant le test.

Pour exécuter des tests sur un environnement spécifique tel que homestead, les valeurs par défaut peuvent être modifiées comme suit:

```
<env name="DB_HOST" value="192.168.10.10"/>
<env name="DB_DATABASE" value="homestead"/>
<env name="DB_USERNAME" value="homestead"/>
<env name="DB_PASSWORD" value="secret"/>
```

Ou pour utiliser une base de données temporaire *en mémoire* :

```
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
```

Une dernière remarque à retenir de la documentation de Laravel :

> Veillez à vider votre cache de configuration en utilisant la commande `config:clear` Artisan avant de lancer vos tests!

Lire Essai en ligne: https://riptutorial.com/fr/laravel/topic/1249/essai

# Chapitre 28: Événements et auditeurs

## Examples

### Utiliser Event et Listeners pour envoyer des emails à un nouvel utilisateur enregistré

Les événements de Laravel permettent de mettre en œuvre le pattern Observer. Cela peut être utilisé pour envoyer un email de bienvenue à un utilisateur chaque fois qu'il s'inscrit sur votre application.

De nouveaux événements et écouteurs peuvent être générés à l'aide de l'utilitaire de ligne de commande artisan après l'enregistrement de l'événement et de son écouteur particulier dans la classe `App\Providers\EventServiceProvider` .

```
protected $listen = [
    'App\Events\NewUserRegistered' => [
        'App\Listeners\SendWelcomeEmail',
    ],
];
```

Notation alternative:

```
protected $listen = [
    \App\Events\NewUserRegistered::class => [
        \App\Listeners\SendWelcomeEmail::class,
    ],
];
```

Maintenant, exécutez `php artisan generate:event` . Cette commande génère tous les événements et les écouteurs correspondants mentionnés ci-dessus dans les répertoires `App\Events` et `App\Listeners` respectivement.

Nous pouvons avoir plusieurs auditeurs pour un seul événement comme

```
protected $listen = [
    'Event' => [
        'Listner1', 'Listener2'
    ],
];
```

`NewUserRegistered` est juste une classe wrapper pour le modèle d'utilisateur nouvellement enregistré:

```
class NewUserRegistered extends Event
{
    use SerializesModels;

    public $user;
```

```
    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }
}
```

Cet `Event` sera géré par le listener `SendWelcomeEmail` :

```
class SendWelcomeEmail
{
    /**
     * Handle the event.
     *
     * @param  NewUserRegistered  $event
     */
    public function handle(NewUserRegistered $event)
    {
        //send the welcome email to the user
        $user = $event->user;
        Mail::send('emails.welcome', ['user' => $user], function ($message) use ($user) {
                $message->from('hi@yourdomain.com', 'John Doe');
                $message->subject('Welcome aboard '.$user->name.'!');
                $message->to($user->email);
        });
    }
}
```

La dernière étape consiste à appeler / déclencher l'événement chaque fois qu'un nouvel utilisateur s'inscrit. Cela peut être fait dans le contrôleur, la commande ou le service, où que vous implémentiez la logique d'enregistrement de l'utilisateur:

```
event(new NewUserRegistered($user));
```

Lire Événements et auditeurs en ligne: https://riptutorial.com/fr/laravel/topic/4687/evenements-et-auditeurs

# Chapitre 29: Fonction d'assistance personnalisée

## Introduction

L'ajout d'aides personnalisées peut vous aider avec votre vitesse de développement. Il y a quelques points à prendre en compte lors de l'écriture de ces fonctions d'aide, d'où ce tutoriel.

## Remarques

Juste quelques conseils:

- Nous avons placé les définitions de fonction dans une vérification ( `function_exists` ) pour empêcher les exceptions lorsque le fournisseur de services est appelé deux fois.
- Une autre méthode consiste à enregistrer le fichier helpers à partir du fichier `composer.json` . Vous pouvez copier la logique du framework Laravel lui-même .

## Examples

### document.php

```
<?php

if (!function_exists('document')) {
    function document($text = '') {
        return $text;
    }
}
```

Créez un fichier helpers.php, supposons pour le moment qu'il se trouve dans `app/Helpers/document.php` . Vous pouvez mettre de nombreux assistants dans un seul fichier (c'est comme cela que fait Laravel) ou vous pouvez les diviser par nom.

### HelpersServiceProvider.php

Maintenant, créons un fournisseur de services. Mettons-le sous `app/Providers` :

```
<?php

namespace App\Providers;

class HelpersServiceProvider extends ServiceProvider
{
    public function register()
    {
        require_once __DIR__ . '/../Helpers/document.php';
```

```
    }
}
```

Le fournisseur de services ci-dessus charge le fichier helpers et enregistre automatiquement votre fonction personnalisée. S'il vous plaît assurez-vous d'enregistrer ce HelpersServiceProvider dans votre `config/app.php` sous `providers` :

```
'providers' => [
     // [..] other providers
     App\Providers\HelpersServiceProvider::class,
]
```

## Utilisation

Vous pouvez maintenant utiliser la fonction `document()` partout dans votre code, par exemple dans les modèles de lame. Cet exemple ne renvoie que la même chaîne qu'il reçoit en argument

```
<?php
Route::get('document/{text}', function($text) {
    return document($text);
});
```

Maintenant, allez dans `/document/foo` dans votre navigateur (utilisez `php artisan serve` or ou `valet` ), qui renverra `foo` .

Lire Fonction d'assistance personnalisée en ligne:
https://riptutorial.com/fr/laravel/topic/8347/fonction-d-assistance-personnalisee

# Chapitre 30: Forfaits Laravel

## Examples

### laravel-ide-helper

Ce package génère un fichier que votre EDI comprend, de sorte qu'il peut fournir une auto-complétion précise. La génération est effectuée en fonction des fichiers de votre projet.

En savoir plus à ce sujet ici

### laravel-datatables

Ce package est créé pour gérer les travaux côté serveur du plug-in DataTables jQuery via l'option AJAX en utilisant Eloquent ORM, Fluent Query Builder ou Collection.

En savoir plus à ce sujet ici ou ici

### Image d'intervention

Intervention Image est une bibliothèque open source de manipulation et de manipulation d'images PHP. Il fournit un moyen plus simple et expressif de créer, éditer et composer des images et supporte actuellement les deux bibliothèques de traitement d'image les plus courantes, GD Library et Imagick.

En savoir plus à ce sujet ici

### Générateur Laravel

Préparez vos API et votre panneau d'administration en quelques minutes.Laravel Generator pour générer des CRUD, des API, des cas de test et de la documentation Swagger

En savoir plus à ce sujet ici

### Socialisme Laravel

Laravel Socialite fournit une interface expressive et fluide à l'authentification OAuth avec Facebook, Twitter, Google, LinkedIn, GitHub et Bitbucket. Il gère presque tout le code d'authentification sociale standard que vous redoutez l'écriture.

En savoir plus à ce sujet ici

### Paquets officiels

## La caissière

Laravel Cashier fournit une interface expressive et fluide aux services de facturation par abonnement de Stripe's et Braintree . Il gère presque tout le code de facturation de l'abonnement que vous écrivez. En plus de la gestion de base des abonnements, Cashier peut gérer les coupons, échanger les abonnements, les «quantités» d'abonnements, les périodes de grâce et même générer des PDF de factures.

Vous trouverez plus d'informations sur ce paquet ici .

## Envoyé

Laravel Envoy fournit une syntaxe propre et minimale pour définir les tâches courantes que vous exécutez sur vos serveurs distants. En utilisant la syntaxe de style Blade, vous pouvez facilement configurer des tâches pour le déploiement, les commandes Artisan, etc. Actuellement, Envoy prend uniquement en charge les systèmes d'exploitation Mac et Linux.

Ce paquet peut être trouvé sur Github .

## Passeport

Laravel facilite déjà l'authentification via les formulaires de connexion traditionnels, mais qu'en est-il des API? Les API utilisent généralement des jetons pour authentifier les utilisateurs et ne maintiennent pas l'état de session entre les requêtes. Laravel simplifie l'authentification des API grâce à Laravel Passport, qui fournit une implémentation complète du serveur OAuth2 pour votre application Laravel en quelques minutes.

Vous trouverez plus d'informations sur ce paquet ici .

## Scout

Laravel Scout fournit une solution simple basée sur un pilote pour ajouter une recherche en texte intégral à vos modèles Eloquent. En utilisant des observateurs de modèles, Scout synchronisera automatiquement vos index de recherche avec vos enregistrements Eloquent.

Actuellement, Scout est livré avec un chauffeur Algolia. Cependant, écrire des pilotes personnalisés est simple et vous êtes libre d'étendre Scout avec vos propres implémentations de recherche.

Vous trouverez plus d'informations sur ce paquet ici .

## Socialite

Laravel Socialite fournit une interface expressive et fluide à l'authentification OAuth avec Facebook, Twitter, Google, LinkedIn, GitHub et Bitbucket. Il gère presque tout le code d'authentification sociale standard que vous redoutez l'écriture.

Ce paquet peut être trouvé sur Github .

Lire Forfaits Laravel en ligne: https://riptutorial.com/fr/laravel/topic/8001/forfaits-laravel

# Chapitre 31: Formulaire de demande (s)

## Introduction

Les demandes personnalisées (ou demandes de formulaire) sont utiles lorsque vous souhaitez **autoriser** et **valider** une demande avant d'appuyer sur la méthode du contrôleur.

On peut penser à deux utilisations pratiques: **créer** et **mettre à jour** un enregistrement alors que chaque action a un ensemble différent de règles de validation (ou d'autorisation).

L'utilisation des demandes de formulaire est triviale, il faut insister sur la classe de requête dans la méthode.

## Syntaxe

- php artisan make: request name_of_request

## Remarques

Les demandes sont utiles pour séparer votre validation de Controller. Il vous permet également de vérifier si la demande est autorisée.

## Examples

### Créer des demandes

```
php artisan make:request StoreUserRequest

php artisan make:request UpdateUserRequest
```

> **Remarque** : Vous pouvez également envisager d'utiliser des noms tels que **StoreUser** ou **UpdateUser** (sans annexe de **demande** ), car vos requêtes FormRequests sont placées dans le dossier `app/Http/Requests/` .

### Utilisation de la demande de formulaire

Disons continuer avec l'exemple d'utilisateur (vous pouvez avoir un contrôleur avec la méthode de magasin et la méthode de mise à jour). Pour utiliser FormRequests, vous utilisez l'indication de type de la demande spécifique.

```
...

public function store(App\Http\Requests\StoreRequest $request, App\User $user) {
    //by type-hinting the request class, Laravel "runs" StoreRequest
    //before actual method store is hit
```

```
    //logic that handles storing new user
    //(both email and password has to be in $fillable property of User model
    $user->create($request->only(['email', 'password']));
    return redirect()->back();
}

...

public function update(App\Http\Requests\UpdateRequest $request, App\User $users, $id) {
    //by type-hinting the request class, Laravel "runs" UpdateRequest
    //before actual method update is hit

    //logic that handles updating a user
    //(both email and password has to be in $fillable property of User model
    $user = $users->findOrFail($id);
    $user->update($request->only(['password']));
    return redirect()->back();
}
```

## Gestion des redirections après validation

Parfois, vous souhaiterez peut-être vous connecter pour déterminer où l'utilisateur sera redirigé après avoir soumis un formulaire. Les demandes de formulaire offrent une variété de moyens.

Par défaut, il y a 3 variables déclarées dans les `$redirect` Request `$redirect` , `$redirectRoute` et `$redirectAction` .

En plus de ces 3 variables, vous pouvez remplacer le gestionnaire de redirection principal `getRedirectUrl()` .

Un exemple de demande est donné ci-dessous pour expliquer ce que vous pouvez faire.

```
<?php namespace App;

use Illuminate\Foundation\Http\FormRequest as Request;

class SampleRequest extends Request {

    // Redirect to the given url
    public $redirect;

    // Redirect to a given route
    public $redirectRoute;

    // Redirect to a given action
    public $redirectAction;


    /**
     * Get the URL to redirect to on a validation error.
     *
     * @return string
     */
    protected function getRedirectUrl()
    {
```

```
        // If no path is given for `url()` it will return a new instance of
`Illuminate\Routing\UrlGenerator`

        // If your form is down the page for example you can redirect to a hash
        return url()->previous() . '#contact';

        //`url()` provides several methods you can chain such as

        // Get the current URL
        return url()->current();

        // Get the full URL of the current request
        return url()->full();

        // Go back
        return url()->previous();

        // Or just redirect back
        return redirect()->back();
    }


    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
}
```

Lire Formulaire de demande (s) en ligne: https://riptutorial.com/fr/laravel/topic/6329/formulaire-de-demande--s-

# Chapitre 32: Guide d'installation

## Remarques

Cette section fournit une vue d'ensemble de ce que laravel-5.4 est et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tout sujet important dans le cadre de laravel-5.4 et d'établir un lien avec les sujets connexes. La documentation de laravel-5.4 étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

## Examples

### Installation

Instructions détaillées sur la mise en place ou l'installation de laravel.

compositeur est nécessaire pour installer laravel facilement.

Il existe 3 méthodes d'installation de laravel dans votre système:

1. **Via Laravel Installer**

   Téléchargez l'installateur Laravel en utilisant `composer`

   ```
   composer global require "laravel/installer"
   ```

   Avant d'utiliser compositeur, nous devons ajouter `~/.composer/vendor/bin` à `PATH` . Une fois l'installation terminée, nous pouvons utiliser la `laravel new` commande `laravel new` pour créer un nouveau projet à `Laravel` .

   Exemple:

   ```
   laravel new {folder name}
   ```

   Cette commande crée un nouveau répertoire nommé `site` et une nouvelle installation `Laravel` avec toutes les autres dépendances est installée dans le répertoire.

2. **Via Composer Create-Project**

   Vous pouvez utiliser la commande du `terminal` pour créer une nouvelle `Laravel app` :

   ```
   composer create-project laravel/laravel {folder name}
   ```

3. **Par téléchargement**

Téléchargez Laravel et décompressez-le.

1. `composer install`
2. Copier `.env.example` à `.env` via `teminal` ou manuellement.

```
cp .env.example .env
```

3. Ouvrez le fichier `.env` et définissez votre base de données, votre email, votre poussoir, etc. (si nécessaire)
4. `php artisan migrate` (si la base de données est configurée)
5. `php artisan key:generate`
6. `php artisan serve`
7. Allez sur localhost: 8000 pour voir le site

Laravel docs

## Bonjour Monde Exemple (Basic)

L'accès aux pages et à la sortie de données est assez facile à Laravel. Toutes les routes de la page sont situées dans `app/routes.php` . Il y a généralement quelques exemples pour vous aider à démarrer, mais nous allons créer une nouvelle route. Ouvrez votre `app/routes.php` et collez le code suivant:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

Cela indique à Laravel que lorsque quelqu'un accède à `http://localhost/helloworld` dans un navigateur, il doit exécuter la fonction et renvoyer la chaîne fournie.

## Hello World Example Avec Views et Controller

En supposant que nous ayons une application Laravel qui fonctionne, disons "mylaravel.com", nous voulons que notre application affiche un message "Hello World" lorsque nous accédons à l'URL `http://mylaravel.com/helloworld` . Cela implique la création de deux fichiers (la vue et le contrôleur) et la modification d'un fichier existant, le routeur.

## La vue

Tout d'abord, nous ouvrons un nouveau fichier de vue lame nommé `helloview.blade.php` avec la chaîne "Hello World". Créez-le dans l'application d'annuaire / ressources / vues

```
<h1>Hello, World</h1>
```

## Le controlle

Nous créons maintenant un contrôleur qui gérera l'affichage de cette vue avec la chaîne "Hello World". Nous utiliserons artisan dans la ligne de commande.

```
$> cd your_laravel_project_root_directory
$> php artisan make:controller HelloController
```

Cela va simplement créer un fichier ( `app/Http/Controllers/HelloController.php` ) contenant la classe qui est notre nouveau contrôleur `HelloController` .

Éditez ce nouveau fichier et écrivez une nouvelle méthode `hello` qui affichera la vue que nous avons créée auparavant.

```
public function hello()
{
    return view('helloview');
}
```

Cet argument 'helloview' dans la fonction view est simplement le nom du fichier de vue sans le ".blade.php" final. Laravel saura le trouver.

Maintenant, lorsque nous appelons la méthode `hello` du contrôleur `HelloController` il affichera le message. Mais comment pouvons-nous lier cela à un appel à `http://mylaravel.com/helloworld` ? Avec l'étape finale, le routage.

## Le routeur

Ouvrez le fichier existant `app/routes/web.php` (dans les anciennes versions de laravel `app/Http/routes.php` ) et ajoutez cette ligne:

```
Route::get('/helloworld', 'HelloController@hello');
```

qui est une commande très explicite à notre application laravel: "Quand quelqu'un utilise le verbe `GET` pour accéder à" / helloworld "dans cette application laravel, renvoyer les résultats de l'appel de la fonction `hello` dans le contrôleur `HelloController` .

Lire Guide d'installation en ligne: https://riptutorial.com/fr/laravel/topic/2187/guide-d-installation

# Chapitre 33: HTML et Form Builder

## Examples

### Installation

HTML and Form Builder n'est pas un composant essentiel depuis Laravel 5, nous devons donc l'installer séparément:

```
composer require laravelcollective/html "~5.0"
```

Enfin, dans `config/app.php` nous devons enregistrer le fournisseur de services et les alias de façades comme ceci:

```
'providers' => [
    // ...
    Collective\Html\HtmlServiceProvider::class,
    // ...
],

'aliases' => [
   // ...
  'Form' => Collective\Html\FormFacade::class,
  'Html' => Collective\Html\HtmlFacade::class,
   // ...
],
```

Les documents complets sont disponibles sur Formulaires et HTML

Lire HTML et Form Builder en ligne: https://riptutorial.com/fr/laravel/topic/3672/html-et-form-builder

# Chapitre 34: Installation

## Examples

### Installation

Les applications Laravel sont installées et gérées avec Composer , un gestionnaire de dépendance PHP populaire. Il existe deux manières de créer une nouvelle application Laravel.

## Via Compositeur

```
$ composer create-project laravel/laravel [foldername]
```

Ou

```
$ composer create-project --prefer-dist laravel/laravel [foldername]
```

Remplacez **[** nom du répertoire **]** par le nom du répertoire dans **lequel** vous voulez installer votre nouvelle application Laravel. Il ne doit pas exister avant l'installation. Vous devrez peut-être également ajouter l'exécutable Composer à votre chemin système.

Si vous souhaitez créer un projet Laravel en utilisant une version spécifique du framework, vous pouvez fournir un modèle de version, sinon votre projet utilisera la dernière version disponible.

Si vous vouliez créer un projet dans Laravel 5.2 par exemple, vous exécuteriez:

```
$ composer create-project --prefer-dist laravel/laravel 5.2.*
```

**Pourquoi --prefer-dist**

Il existe deux manières de télécharger un paquet: `source` et `dist` . Pour les versions stables, Composer utilisera le `dist` par défaut. La `source` est un référentiel de contrôle de version. Si `--prefer-source` est activé, Composer installera le code source s'il y en a un.

`--prefer-dist` est l'inverse de `--prefer-source` , et demande à Composer de l'installer de `dist` si possible. Cela peut accélérer considérablement les installations sur les serveurs de génération et dans d'autres cas d'utilisation où vous n'exécutez généralement pas les mises à jour du fournisseur. Il permet également d'éviter les problèmes avec Git si vous n'avez pas une configuration correcte.

## Via l'installateur Laravel

Laravel fournit un utilitaire de ligne de commande utile pour créer rapidement des applications Laravel. Tout d'abord, installez le programme d'installation:

```
$ composer global require laravel/installer
```

Vous devez vous assurer que le dossier des fichiers binaires Composer se trouve dans votre variable $ PATH pour exécuter le programme d'installation Laravel.

D'abord, regardez s'il est déjà dans votre variable $ PATH

```
echo $PATH
```

Si tout est correct, le résultat devrait contenir quelque chose comme ceci:

```
Users/yourusername/.composer/vendor/bin
```

Si ce n'est pas le cas, modifiez votre `.bashrc` ou, si vous utilisez ZSH, votre `.zshrc` pour qu'il contienne le chemin d'accès au répertoire de votre fournisseur Composer.

Une fois installée, cette commande créera une nouvelle installation Laravel dans le répertoire que vous spécifiez.

```
laravel new [foldername]
```

Vous pouvez également utiliser `.` (un point) à la place de **[nomdossier]** pour créer le projet dans le répertoire de travail en cours sans créer de sous-répertoire.

# Lancer l'application

Laravel est livré avec un serveur Web basé sur PHP qui peut être démarré en cours d'exécution

```
$ php artisan serve
```

Par défaut, le serveur HTTP utilisera le port 8000, mais si le port est déjà utilisé ou si vous souhaitez exécuter plusieurs applications Laravel à la fois, vous pouvez utiliser l'indicateur `--port` pour spécifier un port différent:

```
$ php artisan serve --port=8080
```

Le serveur HTTP utilisera `localhost` comme domaine par défaut pour exécuter l'application, mais vous pouvez utiliser l' `--host` pour spécifier une adresse différente:

```
$ php artisan serve --host=192.168.0.100 --port=8080
```

## Utiliser un autre serveur

Si vous préférez utiliser un logiciel de serveur Web différent, certains fichiers de configuration vous sont fournis dans le répertoire `public` de votre projet. `.htaccess` pour Apache et `web.config` pour ASP.NET. Pour d'autres logiciels tels que NGINX, vous pouvez convertir les configurations Apache à l'aide de divers outils en ligne.

---

L'infrastructure nécessite que l'utilisateur du serveur Web dispose d'autorisations en écriture sur les répertoires suivants:

- `/storage`
- `/bootstrap/cache`

Sur les systèmes d'exploitation * nix, cela peut être réalisé par

```
chown -R www-data:www-data storage bootstrap/cache
chmod -R ug+rwx storage bootstrap/cache
```

(où `www-data` est le nom et le groupe de l'utilisateur du serveur Web)

---

Le serveur Web de votre choix doit être configuré pour servir le contenu du répertoire `/public` de votre projet, ce qui se fait généralement en le définissant comme racine du document. Le reste de votre projet ne devrait pas être accessible via votre serveur Web.

Si vous configurez tout correctement, la navigation vers l'URL de votre site Web doit afficher la page de destination par défaut de Laravel.

## Exigences

Le framework Laravel a les exigences suivantes:

5.3

- PHP> = 5.6.4
- Extension PHP XML
- Extension PHP PDO
- OpenSSL PHP Extension
- Extension PHP Mbstring
- Extension PHP Tokenizer

5.1 (LTS) 5.2

- PHP> = 5.5.9
- Extension PHP PDO
- Laravel 5.1 est la première version de Laravel à supporter PHP 7.0.

5.0

- PHP> = 5.4, PHP <7
- Extension PHP OpenSSL
- Extension PHP Tokenizer
- Extension PHP Mbstring
- Extension PHP JSON (uniquement sur PHP 5.5)

4.2

- PHP> = 5.4
- Extension PHP Mbstring
- Extension PHP JSON (uniquement sur PHP 5.5)

## Hello World Example (Utiliser Controller et View)

1. Créez une application Laravel:

```
$ composer create-project laravel/laravel hello-world
```

2. Accédez au dossier du projet, par exemple

```
$ cd C:\xampp\htdocs\hello-world
```

3. Créez un contrôleur:

```
$ php artisan make:controller HelloController --resource
```

Cela créera le fichier **app / Http / Controllers / HelloController.php** . L'option `--resource` générera des méthodes CRUD pour le contrôleur, par exemple, index, create, show, update.

4. Enregistrez une route vers la méthode d' `index` de HelloController. Ajoutez cette ligne à **app / Http / routes.php** *(version 5.0 à 5.2)* ou à **routes / web.php** *(version 5.3)* :

```
Route::get('hello', 'HelloController@index');
```

Pour voir vos itinéraires nouvellement ajoutés, vous pouvez lancer `$ php artisan route:list`

5. Créez un modèle Blade dans le répertoire `views` :

**resources / views / hello.blade.php:**

```
<h1>Hello world!</h1>
```

6. Maintenant, nous indiquons la méthode index pour afficher le modèle **hello.blade.php** :

**app / Http / Contrôleurs / HelloController.php**

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class HelloController extends Controller
{
```

```
        /**
         * Display a listing of the resource.
         *
         * @return \Illuminate\Http\Response
         */
        public function index()
        {
            return view('hello');
        }

        // ... other resources are listed below the index one above
```

Vous pouvez servir votre application en utilisant la commande PHP Artisan suivante: **`php artisan serve`** ; Il vous indiquera l'adresse à laquelle vous pouvez accéder à votre application *(généralement à l'* *adresse http: // localhost: 8000* *par défaut)* .

Vous pouvez également vous rendre directement à l'emplacement approprié dans votre navigateur. *Si vous utilisez un serveur comme XAMPP (soit: http: // localhost / hello-world / public / hello si vous avez installé votre instance de Laravel,* ***`hello-world`*** *, directement dans votre* ***répertoire xampp / htdocs*** *comme dans: étape 1 de ce Hello Word depuis votre interface de ligne de commande, pointant vers votre* ***répertoire xampp / htdocs*** *)* .

## Bonjour Monde Exemple (Basic)

Ouvrir le fichier des itinéraires. Collez le code suivant dans:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

Après être allé sur la route `localhost/helloworld` il affiche **Hello World** .

Le fichier de routes est situé:

5.3

Pour le Web

```
routes/web.php
```

Pour les API

```
routes/api.php
```

5.2 5.1 (LTS) 5.0

```
app/Http/routes.php
```

4.2

```
app/routes.php
```

### Installation avec LaraDock (Laravel Homestead pour Docker)

LaraDock est un environnement de développement comme Laravel Homestead mais pour Docker au lieu de Vagrant. https://github.com/LaraDock/laradock

# Installation

* Requiert Git et Docker

Cloner le référentiel LaraDock:

A. Si vous avez déjà un projet Laravel, clonez ce dépôt sur votre répertoire racine Laravel:

```
git submodule add https://github.com/LaraDock/laradock.git
```

B. Si vous n'avez pas de projet Laravel et que vous souhaitez installer Laravel depuis Docker, clonez ce dépôt partout sur votre machine:

```
git clone https://github.com/LaraDock/laradock.git
```

# Utilisation de base

1. Run Containers: (Assurez-vous d'être dans le dossier laradock avant d'exécuter les commandes docker-compose).

   Exemple: Exécution de NGINX et MySQL: `docker-compose up -d nginx mysql`

   Vous pouvez sélectionner une liste de conteneurs disponibles pour créer vos propres combinaisons.

   `nginx` , `hhvm` , `php-fpm` , `mysql` , `redis` , `postgres` , `mariadb` , `neo4j` , `mongo` , `apache2` , `caddy` , `memcached` , `beanstalkd` , `beanstalkd-console` , `workspace`

2. Entrez le conteneur Workspace pour exécuter des commandes telles que (Artisan, Composer, PHPUnit, Gulp, ...).

   ```
   docker-compose exec workspace bash
   ```

3. Si vous n'avez pas encore de projet Laravel installé, suivez les étapes pour installer Laravel à partir d'un conteneur Docker.

   une. Entrez le conteneur d'espace de travail.

   b. Installez Laravel. `composer create-project laravel/laravel my-cool-app "5.3.*"`

4. Modifiez les configurations Laravel. Ouvrez le fichier .env de votre Laravel et définissez le DB_HOST sur votre mysql:

```
DB_HOST=mysql
```

5. Ouvrez votre navigateur et visitez votre adresse localhost.

Lire Installation en ligne: https://riptutorial.com/fr/laravel/topic/7961/installation

# Chapitre 35: Intégration de Sparkpost avec Laravel 5.4

## Introduction

Laravel 5.4 est préinstallé avec api lib sparkpost. Sparkpost lib requiert une clé secrète que l'on peut trouver depuis son compte sparkpost.

## Examples

### Données de fichier SAMPLE .env

Pour créer avec succès une configuration d'api de courrier électronique sparkpost, ajoutez les détails ci-dessous au fichier env et votre application sera utile pour commencer à envoyer des e-mails.

MAIL_DRIVER = sparkpost
SPARKPOST_SECRET =

REMARQUE: Les détails ci-dessus ne vous donnent pas le code écrit dans le contrôleur qui a la logique métier pour envoyer des e-mails à l'aide de la fonction laravels Mail :: send.

Lire Intégration de Sparkpost avec Laravel 5.4 en ligne:
https://riptutorial.com/fr/laravel/topic/10136/integration-de-sparkpost-avec-laravel-5-4

# Chapitre 36: Introduction au laravel-5.2

## Introduction

Laravel est un framework MVC avec bundles, migrations et Artisan CLI. Laravel propose un ensemble d'outils robustes et une architecture d'application intégrant les meilleures fonctionnalités des frameworks tels que CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra et autres. Laravel est un framework Open Source. Il possède un ensemble très riche de fonctionnalités qui accélèreront la vitesse du développement Web. Si vous êtes familier avec Core PHP et Advanced PHP, Laravel vous facilitera la tâche. Cela permettra de gagner beaucoup de temps.

## Remarques

Cette section fournit une vue d'ensemble de ce que laravel-5.1 est et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tout sujet important dans le cadre de laravel-5.1 et d'établir un lien avec les sujets connexes. La documentation de laravel-5.1 étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

## Examples

### Installation ou configuration

Instructions sur l'installation de Laravel 5.1 sur une machine Linux / Mac / Unix.

Avant de lancer l'installation, vérifiez si les conditions suivantes sont remplies:

- PHP> = 5.5.9
- Extension PHP OpenSSL
- Extension PHP PDO
- Extension PHP Mbstring
- Extension PHP Tokenizer

Commençons l'installation:

1. Installez le compositeur. Composer Documentation
2. Lancez le `composer create-project laravel/laravel <folder-name> "5.1.*"`
3. Assurez-vous que le dossier de `storage` et le dossier `bootstrap/cache` sont accessibles en écriture.
4. Ouvrez le fichier `.env` et définissez les informations de configuration telles que les informations d'identification de la base de données, l'état du débogage, l'environnement d'application, etc.
5. Exécutez le service `php artisan serve` et dirigez votre navigateur vers `http://localhost:8000` . Si tout va bien, vous devriez obtenir la page

## Installez le framework Laravel 5.1 sur Ubuntu 16.04, 14.04 et LinuxMint

### Étape 1 - Installer la lampe

Pour commencer avec Laravel, nous devons d'abord configurer un serveur LAMP en cours d'exécution. Si vous avez déjà exécuté la pile LAMP, passez cette étape sinon utilisez les commandes suivantes pour configurer la lampe sur le système Ubuntu.

Installer PHP 5.6

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
$ sudo apt-get install -y php5.6 php5.6-mcrypt php5.6-gd
```

Installer Apache2

```
$ apt-get install apache2 libapache2-mod-php5
```

Installer MySQL

```
$ apt-get install mysql-server php5.6-mysql
```

### Étape 2 - Installez Composer

Composer est nécessaire pour installer des dépendances Laravel. Utilisez donc les commandes ci-dessous pour télécharger et utiliser comme commande dans notre système.

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
$ sudo chmod +x /usr/local/bin/composer
```

### Étape 3 - Installer Laravel

Pour télécharger la dernière version de Laravel, utilisez la commande ci-dessous pour cloner le dépôt principal de laravel à partir de github.

```
$ cd /var/www
$ git clone https://github.com/laravel/laravel.git
```

Accédez au répertoire du code Laravel et utilisez le composeur pour installer toutes les dépendances requises pour le framework Laravel.

```
$ cd /var/www/laravel
$ sudo composer install
```

L'installation des dépendances prendra du temps. Après, définissez les autorisations appropriées sur les fichiers.

```
$ chown -R www-data.www-data /var/www/laravel
$ chmod -R 755 /var/www/laravel
$ chmod -R 777 /var/www/laravel/app/storage
```

### Étape 4 - Définir la clé de cryptage

Définissez maintenant la clé de chiffrement à nombre aléatoire de 32 bits, utilisée par le service d'encryptage Illuminate.

```
$ php artisan key:generate

Application key [uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75] set successfully.
```

Maintenant, éditez le fichier de configuration `config/app.php` et mettez à jour la clé d'application générée ci-dessus. Assurez-vous également que le chiffrement est correctement défini.

```
'key' => env('APP_KEY', 'uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75'),

'cipher' => 'AES-256-CBC',
```

### Étape 5 - Créer Apache VirtualHost

Ajoutez maintenant un hôte virtuel dans votre fichier de configuration Apache pour accéder au framework Laravel depuis un navigateur Web. Créez le fichier de configuration Apache dans le répertoire `/etc/apache2/sites-available/` et ajoutez le contenu ci-dessous.

```
$ vim /etc/apache2/sites-available/laravel.example.com.conf
```

Il s'agit de la structure de fichiers de l'hôte virtuel.

```
<VirtualHost *:80>

        ServerName laravel.example.com
        DocumentRoot /var/www/laravel/public

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/laravel>
                AllowOverride All
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Enfin, permet d'activer le site Web et de recharger le service Apache en utilisant la commande ci-dessous.

```
$ a2ensite laravel.example.com
```

```
$ sudo service apache2 reload
```

### Étape 6 - Accès à Laravel

À ce stade, vous avez terminé avec succès le framework PHP Laravel 5 sur votre système. Créez maintenant une entrée de fichier hôte pour accéder à votre application Laravel dans un navigateur Web. Modifiez `127.0.0.1` avec votre serveur ip et `laravel.example.com` avec votre nom de domaine configuré dans Apache.

```
$ sudo echo "127.0.0.1  laravel.example.com" >> /etc/hosts
```

Et accédez à http://laravel.example.com dans votre navigateur Web préféré comme ci-dessous.

Lire Introduction au laravel-5.2 en ligne: https://riptutorial.com/fr/laravel/topic/1987/introduction-au-laravel-5-2

# Chapitre 37: Introduction au laravel-5.3

## Introduction

Nouvelles fonctionnalités, améliorations et modifications de Laravel 5.2 à 5.3

## Examples

### La variable $ loop

Il est connu depuis longtemps que le traitement des boucles dans Blade est limité, à partir de la version 5.3, il existe une variable appelée `$loop` disponible.

```
@foreach($variables as $variable)

    // Within here the `$loop` variable becomes available

    // Current index, 0 based
    $loop->index;

    // Current iteration, 1 based
    $loop->iteration;

    // How many iterations are left for the loop to be complete
    $loop->remaining;

    // Get the amount of items in the loop
    $loop->count;

    // Check to see if it's the first iteration ...
    $loop->first;

    // ... Or last iteration
    $loop->last;

    //Depth of the loop, ie if a loop within a loop the depth would be 2, 1 based counting.
    $loop->depth;

    // Get's the parent `$loop` if the loop is nested, else null
    $loop->parent;

@endforeach
```

Lire Introduction au laravel-5.3 en ligne: https://riptutorial.com/fr/laravel/topic/9231/introduction-au-laravel-5-3

# Chapitre 38: La caissière

## Remarques

Laravel Cashier peut être utilisé pour la facturation par abonnement en fournissant une interface aux services d'abonnement de Braintree et de Stripe. En plus de la gestion de base des abonnements, il peut être utilisé pour gérer des coupons, échanger des abonnements, des quantités, des délais de grâce et la génération de factures PDF.

## Examples

### Configuration de la bande

#### La configuration initiale

Pour utiliser Stripe pour gérer les paiements, nous devons ajouter ce qui suit au `composer.json` puis lancer la `composer update` :

```
"laravel/cashier": "~6.0"
```

La ligne suivante doit ensuite être ajoutée à `config/app.php` , le fournisseur de services:

```
Laravel\Cashier\CashierServiceProvider
```

---

#### Configuration de la base de données

Pour utiliser la caisse, nous devons configurer les bases de données, si une table d'utilisateurs n'existe pas, nous devons en créer une et créer une table d'abonnements. L'exemple suivant modifie une table d' `users` existante. Voir Modèles éloquents pour plus d'informations sur les modèles.

Pour utiliser le caissier, créez une nouvelle migration et ajoutez ce qui suit:

```
// Adjust users table

Schema::table('users', function ($table) {
    $table->string('stripe_id')->nullable();
    $table->string('card_brand')->nullable();
    $table->string('card_last_four')->nullable();
    $table->timestamp('trial_ends_at')->nullable();
});

//Create subscriptions table

Schema::create('subscriptions', function ($table) {
    $table->increments('id');
    $table->integer('user_id');
    $table->string('name');
```

```
    $table->string('stripe_id');
    $table->string('stripe_plan');
    $table->integer('quantity');
    $table->timestamp('trial_ends_at')->nullable();
    $table->timestamp('ends_at')->nullable();
    $table->timestamps();
});
```

Nous devons ensuite exécuter `php artisan migrate` pour mettre à jour notre base de données.

---

### Configuration du modèle

Nous devons ensuite ajouter le trait facturable au modèle d'utilisateur trouvé dans `app/User.php` et le modifier comme suit:

```
use Laravel\Cashier\Billable;

class User extends Authenticatable
{
    use Billable;
}
```

---

### Clés à rayures

Afin de garantir que nous finissons l'argent sur notre propre compte Stripe, nous devons le `config/services.php` dans le fichier `config/services.php` en ajoutant la ligne suivante:

```
'stripe' => [
    'model'  => App\User::class,
    'secret' => env('STRIPE_SECRET'),
],
```

Remplacer le `STRIPE_SECRET` par votre propre clé secrète.

---

Une fois cette opération effectuée, vous pourrez continuer à configurer les abonnements.

Lire La caissière en ligne: https://riptutorial.com/fr/laravel/topic/7474/la-caissiere

# Chapitre 39: La gestion des erreurs

## Remarques

N'oubliez pas de configurer votre application pour l'envoi par courrier électronique en veillant à configurer correctement `config/mail.php`

Vérifiez également que les variables ENV sont correctement définies.

Cet exemple est un guide et est minime. Explorer, modifier et styliser la vue comme vous le souhaitez. Ajustez le code pour répondre à vos besoins. Par exemple, définissez le destinataire dans votre fichier *.env*

## Examples

### Envoyer un rapport d'erreur par e-mail

Les exceptions dans Laravel sont gérées par *App* \ *Exceptions* \ *Handler.php*

Ce fichier contient deux fonctions par défaut. Rapport et rendu. Nous n'utiliserons que le premier

```
public function report(Exception $e)
```

> La méthode de rapport est utilisée pour enregistrer des exceptions ou les envoyer à un service externe tel que BugSnag. Par défaut, la méthode de rapport transmet simplement l'exception à la classe de base où l'exception est consignée. Cependant, vous êtes libre de consigner les exceptions comme vous le souhaitez.

Essentiellement, cette fonction ne fait que renvoyer l'erreur et ne fait rien. Par conséquent, nous pouvons insérer une logique métier pour effectuer des opérations en fonction de l'erreur. Pour cet exemple, nous enverrons un courrier électronique contenant les informations d'erreur.

```
public function report(Exception $e)
{
    if ($e instanceof \Exception) {
        // Fetch the error information we would like to
        // send to the view for emailing
        $error['file']    = $e->getFile();
        $error['code']    = $e->getCode();
        $error['line']    = $e->getLine();
        $error['message'] = $e->getMessage();
        $error['trace']   = $e->getTrace();

        // Only send email reports on production server
        if(ENV('APP_ENV') == "production"){
            #1. Queue email for sending on "exceptions_emails" queue
            #2. Use the emails.exception_notif view shown below
            #3. Pass the error array to the view as variable $e
            Mail::queueOn('exception_emails', 'emails.exception_notif', ["e" => $error],
```

```
function ($m) {
                $m->subject("Laravel Error");
                $m->from(ENV("MAIL_FROM"), ENV("MAIL_NAME"));
                $m->to("webmaster@laravelapp.com", "Webmaster");
            });

        }
    }

    // Pass the error on to continue processing
    return parent::report($e);
}
```

La vue de l'email ("emails.exception_notif") est ci-dessous

```
<?php
$action = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getActionName() : "n/a";
$path = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getPath() : "n/a";
$user = (\Auth::check()) ? \Auth::user()->name : 'no login';
?>

There was an error in your Laravel App<br />

<hr />
<table border="1" width="100%">
    <tr><th >User:</th><td>{{ $user }}</td></tr>
    <tr><th >Message:</th><td>{{ $e['message'] }}</td></tr>
    <tr><th >Action:</th><td>{{ $action }}</td></tr>
    <tr><th >URI:</th><td>{{ $path }}</td></tr>
    <tr><th >Line:</th><td>{{ $e['line'] }}</td></tr>
    <tr><th >Code:</th><td>{{ $e['code'] }}</td></tr>
</table>
```

## Récupération de l'application ModelNotFoundException à l'échelle de l'application

### app \ Exceptions \ Handler.php

```
public function render($request, Exception $exception)
{
    if ($exception instanceof ModelNotFoundException) {
        abort(404);
    }

    return parent::render($request, $exception);
}
```

Vous pouvez attraper / gérer toute exception qui est lancée dans Laravel.

Lire La gestion des erreurs en ligne: https://riptutorial.com/fr/laravel/topic/2858/la-gestion-des-erreurs

# Chapitre 40: Laravel Docker

## Introduction

La cohérence de l'environnement est un défi auquel toutes les équipes de développement et de développement sont confrontées. Laravel est l'un des frameworks PHP les plus populaires aujourd'hui. DDocker, quant à lui, est une méthode de virtualisation qui élimine les problèmes de *«travail sur machine»* lors de la coopération sur du code avec d'autres développeurs. Les deux ensemble créent une fusion d' **utile** et de **puissant** . Bien que les deux fassent des choses très différentes, ils peuvent tous deux être combinés pour créer des produits étonnants.

## Examples

### Utiliser Laradock

**Laradock** est un projet qui fournit un contenant prêt à l'emploi adapté à l'utilisation de Laravel.

Téléchargez ou clonez Laradock dans le dossier racine de votre projet:

```
git clone https://github.com/Laradock/laradock.git
```

Changez de répertoire en Laradock et générez le fichier `.env` nécessaire pour exécuter vos configurations:

```
cd laradock
cp .env-example .env
```

Vous êtes maintenant prêt à lancer docker. La première fois que vous exécuterez le conteneur, tous les paquets nécessaires seront téléchargés sur Internet.

```
docker-compose up -d nginx mysql redis beanstalkd
```

Vous pouvez maintenant ouvrir votre navigateur et afficher votre projet sur `http://localhost` .

Pour la documentation et la configuration complètes de Laradock, cliquez ici .

Lire Laravel Docker en ligne: https://riptutorial.com/fr/laravel/topic/10034/laravel-docker

# Chapitre 41: Le routage

## Examples

### Routage de base

Le routage définit une carte entre les méthodes HTTP et les URI d'un côté et les actions de l'autre. Les routes sont normalement écrites dans le fichier `app/Http/routes.php` .

Dans sa forme la plus simple, une route est définie en appelant la méthode HTTP correspondante sur la façade de la route, en passant en paramètre une chaîne correspondant à l'URI (relative à la racine de l'application) et un rappel.

Par exemple: une route vers l'URI racine du site qui renvoie une vue `home` ressemble à ceci:

```
Route::get('/', function() {
    return view('home');
});
```

Un itinéraire pour une post-demande qui fait simplement écho aux variables post:

```
Route::post('submit', function() {
   return Input::all();
});

//or

Route::post('submit', function(\Illuminate\Http\Request $request) {
   return $request->all();
});
```

# Routes pointant vers une méthode Controller

Au lieu de définir le rappel en ligne, la route peut faire référence à une méthode de contrôleur dans la syntaxe [ControllerClassName @ Method]:

```
Route::get('login', 'LoginController@index');
```

# Un itinéraire pour plusieurs verbes

La méthode de `match` peut être utilisée pour faire correspondre un tableau de méthodes HTTP pour une route donnée:

```
Route::match(['GET', 'POST'], '/', 'LoginController@index');
```

Aussi, vous pouvez utiliser `all` pour faire correspondre n'importe quelle méthode HTTP pour un itinéraire donné:

```
Route::all('login', 'LoginController@index');
```

# Groupes de route

Les routes peuvent être regroupées pour éviter la répétition du code.

Disons que tous les URI avec un préfixe de `/admin` utilisent un certain middleware appelé `admin` et qu'ils vivent tous dans l'espace de noms `App\Http\Controllers\Admin` .

Une manière propre de représenter cela en utilisant des groupes de routage est la suivante:

```
Route::group([
    'namespace' => 'Admin',
    'middleware' => 'admin',
    'prefix' => 'admin'
    ], function () {

        // something.dev/admin
        // 'App\Http\Controllers\Admin\IndexController'
        // Uses admin middleware
        Route::get('/', ['uses' => 'IndexController@index']);

        // something.dev/admin/logs
        // 'App\Http\Controllers\Admin\LogsController'
        // Uses admin middleware
        Route::get('/logs', ['uses' => 'LogsController@index']);
});
```

## Route nommée

Les routes nommées servent à générer une URL ou à rediriger vers un itinéraire spécifique. L'avantage d'utiliser une route nommée est que si nous changeons l'URI d'une route à l'avenir, nous n'aurons pas besoin de changer l'URL ou les redirections pointant vers cette route si nous utilisons une route nommée. Mais si les liens ont été générés en utilisant l'URL [par exemple. `url('/admin/login')` ], il faudrait alors changer partout où il est utilisé.

Les routes nommées sont créées en utilisant `as` clé de tableau

```
Route::get('login', ['as' => 'loginPage', 'uses' => 'LoginController@index']);
```

ou en utilisant le `name` méthode

```
Route::get('login', 'LoginController@index')->name('loginPage');
```

# Générer une URL à l'aide d'une route

# nommée

Pour générer une URL en utilisant le nom de la route

```
$url = route('loginPage');
```

Si vous utilisez le nom de la route pour la redirection

```
$redirect = Redirect::route('loginPage');
```

### Paramètres d'itinéraire

Vous pouvez utiliser les paramètres de route pour obtenir la partie du segment URI. Vous pouvez définir un paramètre / s de route facultatif ou requis lors de la création d'un itinéraire. Les paramètres facultatifs ont un `?` ajouté à la fin du nom du paramètre. Ce nom est entre accolades `{}`

## Paramètre facultatif

```
Route::get('profile/{id?}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Cette route est accessible par `domain.com/profile/23` où 23 est le paramètre id. Dans cet exemple, l' `id` est transmis en tant que méthode de paramètre dans la `view` de `ProfileController` . Comme il s'agit d'un paramètre facultatif, l'accès à `domain.com/profile` fonctionne `domain.com/profile` .

## Paramètre requis

```
Route::get('profile/{id}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Notez que le nom du paramètre requis n'a pas de `?` à la fin du nom du paramètre.

## Accéder au paramètre dans le contrôleur

Dans votre contrôleur, votre méthode d'affichage prend un paramètre du **même** nom que celui de `routes.php` et peut être utilisé comme une variable normale. Laravel se charge d'injecter la valeur:

```
public function view($id){
    echo $id;
}
```

### Attraper tous les itinéraires

Si vous souhaitez intercepter tous les itinéraires, vous pouvez utiliser une expression régulière comme indiqué:

```
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

**Important:** Si vous avez d'autres itinéraires et que vous ne voulez pas que le fourre-tout interfère, vous devriez le mettre à la fin. Par exemple:

# Attraper toutes les routes sauf celles déjà définies

```
Route::get('login',  'AuthController@login');
Route::get('logout', 'AuthController@logout');
Route::get('home',   'HomeController@home');

// The catch-all will match anything except the previous defined routes.
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

## Les routes sont appariées dans l'ordre où elles sont déclarées

Ceci est un piège commun avec les routes Laravel. Les routes sont appariées dans l'ordre dans lequel elles sont déclarées. Le premier itinéraire correspondant est celui qui est utilisé.

Cet exemple fonctionnera comme prévu:

```
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
Route::get('/posts/{postId}', 'PostController@show');
```

Une requête get à `/posts/1/comments/1` invoquera `CommentController@show` . Un appel à `/posts/1` invoquera `PostController@show` .

Cependant, cet exemple ne fonctionnera pas de la même manière:

```
Route::get('/posts/{postId}', 'PostController@show');
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
```

Une requête get à `/posts/1/comments/1` invoquera `PostController@show` . Un appel à `/posts/1` invoquera `PostController@show` .

Comme Laravel utilise le premier itinéraire correspondant, la requête vers `/posts/1/comments/1` correspond à `Route::get('/posts/{postId}', 'PostController@show');` et assigne la variable `$postId` à la valeur `1/comments/1` . Cela signifie que `CommentController@show` ne sera jamais invoqué.

## Routes insensibles à la casse

Les routes dans Laravel sont sensibles à la casse. Cela signifie qu'un itinéraire comme

```
Route::get('login', ...);
```

correspondra à une requête GET à `/login` mais ne correspondra pas à une requête GET à `/Login` .

Afin de rendre vos routes insensibles à la casse, vous devez créer une nouvelle classe de validateur correspondant aux URL demandées par rapport aux routes définies. La seule différence entre le nouveau validateur et le nouveau est qu'il ajoute le modificateur **i** à la fin de l'expression régulière pour que l'itinéraire compilé permette l'activation de la correspondance insensible à la casse.

```
<?php namespace Some\Namespace;

use Illuminate\Http\Request;
use Illuminate\Routing\Route;
use Illuminate\Routing\Matching\ValidatorInterface;

class CaseInsensitiveUriValidator implements ValidatorInterface
{
  public function matches(Route $route, Request $request)
  {
    $path = $request->path() == '/' ? '/' : '/'.$request->path();
    return preg_match(preg_replace('/$/','i', $route->getCompiled()->getRegex()),
rawurldecode($path));
  }
}
```

Pour que Laravel utilise votre nouveau validateur, vous devez mettre à jour la liste des comparateurs utilisés pour associer l'URL à un itinéraire et remplacer le UriValidator d'origine par le vôtre.

Pour ce faire, ajoutez ce qui suit en haut de votre fichier routes.php:

```
<?php
use Illuminate\Routing\Route as IlluminateRoute;
use Your\Namespace\CaseInsensitiveUriValidator;
use Illuminate\Routing\Matching\UriValidator;

$validators = IlluminateRoute::getValidators();
$validators[] = new CaseInsensitiveUriValidator;
IlluminateRoute::$validators = array_filter($validators, function($validator) {
  return get_class($validator) != UriValidator::class;
});
```

Cela supprimera le validateur d'origine et ajoutera le vôtre à la liste des validateurs.

Lire Le routage en ligne: https://riptutorial.com/fr/laravel/topic/1284/le-routage

# Chapitre 42: Les constantes

## Examples

### Exemple

Vous devez d'abord créer un fichier constants.php et il est conseillé de créer ce fichier dans app / config / folder. Vous pouvez également ajouter un fichier constants.php dans le fichier compose.json.

Fichier d'exemple:

> app / config / constants.php

Constantes basées sur des tableaux dans le fichier:

```
return [
    'CONSTANT' => 'This is my first constant.'
];
```

Et vous pouvez obtenir cette constante en incluant la façade `Config` :

```
use Illuminate\Support\Facades\Config;
```

Ensuite, obtenez la valeur par le nom de la constante `CONSTANT` comme ci-dessous:

```
echo Config::get('constants.CONSTANT');
```

Et le résultat serait la valeur:

> Ceci est ma première constante.

Lire Les constantes en ligne: https://riptutorial.com/fr/laravel/topic/9192/les-constantes

# Chapitre 43: Les files d'attente

## Introduction

Les files d'attente permettent à votre application de réserver des parties du travail qui nécessitent beaucoup de temps pour être traitées par un processus d'arrière-plan.

## Examples

### Cas d'utilisation

Par exemple, si vous envoyez un e-mail à un client après avoir démarré une tâche, il est préférable de rediriger immédiatement l'utilisateur vers la page suivante tout en mettant en file d'attente l'e-mail à envoyer en arrière-plan. Cela accélérera le temps de chargement de la page suivante, car l'envoi d'un courrier électronique peut parfois prendre plusieurs secondes ou plus.

Un autre exemple serait la mise à jour d'un système d'inventaire après qu'un client ait vérifié sa commande. Plutôt que d'attendre la fin des appels d'API, ce qui peut prendre plusieurs secondes, vous pouvez immédiatement rediriger l'utilisateur vers la page de réussite de l'extraction en mettant en attente les appels d'API en arrière-plan.

### Configuration du pilote de file d'attente

Chacun des pilotes de file d'attente de Laravel est configuré à partir du fichier `config/queue.php` . Un gestionnaire de files d'attente est le gestionnaire permettant de gérer l'exécution d'un travail en file d'attente, d'identifier si les travaux ont réussi ou échoué, et de réessayer le travail s'il est configuré pour le faire.

Laravel prend en charge les pilotes de file d'attente suivants:

#### sync

La synchronisation, ou synchrone, est le pilote de file d'attente par défaut qui exécute un travail en file d'attente dans votre processus existant. Lorsque ce pilote est activé, vous ne disposez d'aucune file d'attente lorsque le travail en file d'attente s'exécute immédiatement. Ceci est utile à des fins locales ou de test, mais n'est clairement pas recommandé pour la production car il supprime les avantages de la configuration de votre file d'attente.

#### database

Ce pilote stocke les travaux en file d'attente dans la base de données. Avant d'activer ce pilote, vous devrez créer des tables de base de données pour stocker vos travaux en attente et échoués:

```
php artisan queue:table
php artisan migrate
```

#### sqs

Ce pilote de file d'attente utilise le service Simple Queue d'Amazon pour gérer les travaux en file d'attente. Avant d'activer ce travail, vous devez installer le package de composition suivant: `aws/aws-sdk-php ~3.0`

Notez également que si vous prévoyez d'utiliser des délais pour les travaux en file d'attente, Amazon SQS ne prend en charge qu'un délai maximum de 15 minutes.

#### iron

Cette file d'attente pilotes utilisent Iron pour gérer les travaux en file d'attente.

#### redis

Ce pilote de file d'attente utilise une instance de Redis pour gérer les travaux en file d'attente. Avant d'utiliser ce pilote de file d'attente, vous devrez configurer une copie de Redis et installer la dépendance de composeur suivante: `predis/predis ~1.0`

#### beanstalkd

Ce pilote de file d'attente utilise une instance de Beanstalk pour gérer les travaux en file d'attente. Avant d'utiliser ce pilote de file d'attente, vous devrez configurer une copie de Beanstalk et installer la dépendance de composeur suivante: `pda/pheanstalk ~3.0`

#### null

Si vous spécifiez null comme pilote de file d'attente, les travaux en attente sont ignorés.

Lire Les files d'attente en ligne: https://riptutorial.com/fr/laravel/topic/2651/les-files-d-attente

# Chapitre 44: Les politiques

## Examples

### Créer des politiques

Étant donné que la définition de toute la logique d'autorisation dans `AuthServiceProvider` peut devenir lourde dans les applications volumineuses, Laravel vous permet de diviser votre logique d'autorisation en classes "Policy". Les stratégies sont des classes PHP simples qui regroupent la logique d'autorisation en fonction de la ressource qu'elles autorisent.

Vous pouvez générer une stratégie à l'aide de la commande `make:policy` artisan. La politique générée sera placée dans le répertoire `app/Policies` :

```
php artisan make:policy PostPolicy
```

Lire Les politiques en ligne: https://riptutorial.com/fr/laravel/topic/7344/les-politiques

# Chapitre 45: Liaison modèle de route

## Examples

### Liaison Implicite

Laravel résout automatiquement les modèles Eloquent définis dans les itinéraires ou les actions de contrôleur dont les noms de variable correspondent à un nom de segment de route. Par exemple:

```
Route::get('api/users/{user}', function (App\User $user) {
    return $user->email;
});
```

Dans cet exemple, comme la variable utilisateur Eloquent $ définie sur la route correspond au segment {utilisateur} dans l'URI de la route, Laravel injectera automatiquement l'instance de modèle dont l'ID correspond à la valeur correspondante de l'URI de la demande. Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse HTTP 404 sera automatiquement générée.

Si le nom de la table du modèle est composé de plusieurs mots, pour que la liaison de modèle implicite fonctionne, la variable d'entrée doit être en minuscules;
Par exemple, si l'utilisateur peut effectuer une *action* quelconque et que nous voulons accéder à cette action, l'itinéraire sera le suivant:

```
Route::get('api/useractions/{useraction}', function (App\UserAction $useraction) {
    return $useraction->description;
});
```

### Reliure Explicite

Pour enregistrer une liaison explicite, utilisez la méthode du modèle du routeur pour spécifier la classe d'un paramètre donné. Vous devez définir vos liaisons de modèle explicites dans la méthode de démarrage de la classe RouteServiceProvider

```
public function boot()
{
   parent::boot();

   Route::model('user', App\User::class);
}
```

Ensuite, nous pouvons définir une route contenant le paramètre {utilisateur}.

```
$router->get('profile/{user}', function(App\User $user) {

});
```

Comme nous avons lié tous `{user}` paramètres `{user}` au modèle `App\User` , une instance User sera injectée dans la route. Ainsi, par exemple, une demande de `profile/1` injectera l'instance d'utilisateur de la base de données dont l' **identifiant est 1** .

Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse **HTTP 404** sera automatiquement générée.

Lire Liaison modèle de route en ligne: https://riptutorial.com/fr/laravel/topic/7098/liaison-modele-de-route

# Chapitre 46: Liens utiles

## Introduction

Dans cette rubrique, vous pouvez trouver des liens utiles pour améliorer vos compétences en Laravel ou approfondir vos connaissances.

## Examples

### Ecosystème Laravel

- Laravel Scout - Laravel Scout fournit une solution simple basée sur un pilote pour ajouter une recherche en texte intégral à vos modèles Eloquent.
- Laravel Passport - Authentification API sans mal de tête. Passport est un serveur OAuth2 prêt en quelques minutes.
- Homestead - L'environnement de développement officiel de Laravel. Propulsé par Vagrant, Homestead met toute votre équipe sur la même page avec les derniers PHP, MySQL, Postgres, Redis, etc.
- Laravel Cashier - Rendre la facturation par abonnement facile avec les intégrations Stripe et Braintree intégrées. Les coupons, les abonnements aux échanges, les annulations et même les factures PDF sont prêts à l'emploi.
- Forge - Fournit et déploie des applications PHP illimitées sur DigitalOcean, Linode et AWS.
- Envoyer - Zero Downtime Repair PHP Deployment.
- Valet - Un environnement de développement Laravel pour les minimalistes Mac. Pas de Vagrant, pas d'Apache, pas de chichi.
- Spark - Puissant échafaudage d'application SaaS. Arrêtez d'écrire à la va-vite et concentrez-vous sur votre application.
- Lumen - Si tout ce dont vous avez besoin est une API et une vitesse fulgurante, essayez Lumen. C'est Laravel super léger.
- Statamic - Un véritable CMS conçu pour rendre les agences rentables, les développeurs heureux et les clients qui vous embrassent.

### Éducation

- Laracasts - Apprenez le développement web moderne et pratique grâce à des screencasts d'experts.
- Laravel News - Restez à jour avec Laravel avec Laravel News.
- Laravel.io - Forum avec code source ouvert.

### Podcasts

- Laravel News Podcasts
- Les Podcasts Laravel

Lire Liens utiles en ligne: https://riptutorial.com/fr/laravel/topic/9957/liens-utiles

# Chapitre 47: Macros dans une relation éloquente

## Introduction

Nous avons de nouvelles fonctionnalités pour Eloquent Relationship dans la version 5.4.8 de Laravel. Nous pouvons récupérer une seule instance d'une relation hasMany (c'est juste un exemple) en la définissant sur place et cela fonctionnera pour toutes les relations

## Examples

### Nous pouvons récupérer une instance de la relation hasMany

Dans notre AppServiceProvider.php

```
public function boot()
{
  HasMany::macro('toHasOne', function() {
      return new HasOne(
          $this->query,
          $this->parent,
          $this->foreignKey,
          $this->localKey
      );
  });
}
```

Supposons que nous ayons une boutique modale et que nous obtenions la liste des produits achetés. Supposons que nous ayons tous des relations achetées pour la boutique modale

```
public function allPurchased()
{
    return $this->hasMany(Purchased::class);
}

public function lastPurchased()
{
    return $this->allPurchased()->latest()->toHasOne();
}
```

Lire Macros dans une relation éloquente en ligne:
https://riptutorial.com/fr/laravel/topic/8998/macros-dans-une-relation-eloquente

# Chapitre 48: Middleware

## Introduction

Les intergiciels sont des classes pouvant être affectées à une ou plusieurs routes et utilisées pour effectuer des actions dans les phases initiales ou finales du cycle de demande. Nous pouvons les considérer comme une série de couches à travers lesquelles une requête HTTP doit transiter pendant son exécution.

## Remarques

Un middleware "Before" sera exécuté avant le code d'action du contrôleur; tandis qu'un middleware "Après" s'exécute après que la requête est gérée par l'application

## Examples

### Définir un middleware

Pour définir un nouveau middleware, nous devons créer la classe de middleware:

```
class AuthenticationMiddleware
{
    //this method will execute when the middleware will be triggered
    public function handle ( $request, Closure $next )
    {
        if ( ! Auth::user() )
        {
            return redirect('login');
        }

        return $next($request);
    }
}
```

Ensuite, nous devons enregistrer le middleware: si le middleware doit être lié à toutes les routes de l'application, nous devons l'ajouter à la propriété middleware de `app/Http/Kernel.php` :

```
protected $middleware = [
        \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
        \App\Http\Middleware\AuthenticationMiddleware::class
];
```

alors que si nous voulons seulement associer le middleware à certaines routes, nous pouvons l'ajouter à `$routeMiddleware`

```
//register the middleware as a 'route middleware' giving it the name of 'custom_auth'
protected $routeMiddleware = [
    'custom_auth' => \App\Http\Middleware\AuthenticationMiddleware::class
```

```
];
```

puis liez-le aux routes simples comme ceci:

```
//bind the middleware to the admin_page route, so that it will be executed for that route
Route::get('admin_page', 'AdminController@index')->middleware('custom_auth');
```

## Avant vs Après Middleware

Un exemple de "avant" middleware serait comme suit:

```
<?php

namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

        return $next($request);
    }
}
```

alors que le middleware "after" ressemblerait à ceci:

```
<?php

namespace App\Http\Middleware;

use Closure;

class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        // Perform action

        return $response;
    }
}
```

La principale différence réside dans la gestion du paramètre `$request` . Si des actions sont exécutées avant `$next($request)` avant que le code du contrôleur ne soit exécuté pendant que l'appel à `$next($request)` entraîne d'abord l'exécution des actions après l'exécution du code du contrôleur.

## Route Middleware

Tout middleware enregistré en tant que `routeMiddleware` dans `app/Http/Kernel.php` peut être affecté à une route.

Il y a plusieurs façons d'affecter un middleware, mais ils font tous la même chose.

```
Route::get('/admin', 'AdminController@index')->middleware('auth', 'admin');
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => 'auth']);
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => ['auth',
'admin']);
```

Dans tous les exemples ci-dessus, vous pouvez également passer des noms de classe complets en tant que middleware, même s'il a été enregistré en tant que middleware de routage.

```
use App\Http\Middleware\CheckAdmin;
Route::get('/admin', 'AdminController@index')->middleware(CheckAdmin::class);
```

Lire Middleware en ligne: https://riptutorial.com/fr/laravel/topic/3419/middleware

# Chapitre 49: Migrations de base de données

## Examples

### Migrations

Pour contrôler votre base de données dans Laravel, utilisez les migrations. Créer une migration avec artisan:

```
php artisan make:migration create_first_table --create=first_table
```

Cela générera la classe CreateFirstTable. A l'intérieur de la méthode up, vous pouvez créer vos colonnes:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFirstTable extends Migration
{
    public function up()
    {
        Schema::create('first_table', function (Blueprint $table) {
            $table->increments('id');
            $table->string('first_string_column_name');
            $table->integer('secont_integer_column_name');
            $table->timestamps();
        });
    }

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

A la fin de l'exécution de toutes vos classes de migration, vous pouvez exécuter la commande artisan:

```
php artisan migrate
```

Cela créera vos tables et vos colonnes dans votre base de données. Les autres commandes de migration utiles sont les suivantes:

- `php artisan migrate:rollback` - Annule la migration de la dernière base de données
- `php artisan migrate:reset` - Restauration de toutes les migrations de bases de données
- `php artisan migrate:refresh` - Réinitialise et réexécute toutes les migrations
- `php artisan migrate:status` - Affiche le statut de chaque migration

### Modification des tables existantes

Parfois, vous devez modifier votre structure de tableau existante, comme `renaming/deleting` colonnes. Ce que vous pouvez accomplir en créant une nouvelle migration.Et Dans la méthode `up` de votre migration.

```
//Renaming Column.

public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->renameColumn('email', 'username');
    });
}
```

Dans l'exemple ci-dessus, vous renommez la `email column` de la `users table` des `users table` en `username` d' `username` . Alors que le code ci-dessous supprime un `username` d' `users` colonne de la table d' `users` .

IMPROTANT: Pour modifier des colonnes, vous devez ajouter la dépendance `doctrine/dbal` au fichier `composer.json` du projet et exécuter la `composer update` à `composer update` pour refléter les modifications.

```
//Droping Column
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('username');
    });
}
```

## Les fichiers de migration

Les migrations dans une application Laravel 5 sont stockées dans le répertoire `database/migrations` . Leurs noms de fichiers sont conformes à un format particulier:

```
<year>_<month>_<day>_<hour><minute><second>_<name>.php
```

Un fichier de migration doit représenter une mise à jour du schéma pour résoudre un problème particulier. Par exemple:

```
2016_07_21_134310_add_last_logged_in_to_users_table.php
```

Les migrations de bases de données sont classées par ordre chronologique afin que Laravel sache dans quel ordre les exécuter. Laravel effectuera toujours des migrations du plus ancien au plus récent.

## Générer des fichiers de migration

Créer un nouveau fichier de migration avec le nom de fichier correct chaque fois que vous devez

modifier votre schéma serait une corvée. Heureusement, la commande `artisan` de Laravel peut générer la migration pour vous:

```
php artisan make:migration add_last_logged_in_to_users_table
```

Vous pouvez également utiliser les drapeaux `--table` et `--create` avec la commande ci-dessus. Celles-ci sont facultatives et ne concernent que la commodité. Elles insèrent le code correspondant dans le fichier de migration.

```
php artisan make:migration add_last_logged_in_to_users_table --table=users

php artisan make:migration create_logs_table --create=logs
```

Vous pouvez spécifier un chemin de sortie personnalisé pour la migration générée à l'aide de l'option `--path` . Le chemin est relatif au chemin de base de l'application.

```
php artisan make:migration --path=app/Modules/User/Migrations
```

### Dans une migration de base de données

Chaque migration doit avoir une méthode `up()` et une méthode `down()` . Le but de la méthode `up()` est d'effectuer les opérations requises pour placer le schéma de base de données dans son nouvel état, et l'objectif de la méthode `down()` est d'inverser toutes les opérations effectuées par la méthode `up()` . Il est essentiel de s'assurer que la méthode `down()` inverse correctement vos opérations pour pouvoir annuler les modifications du schéma de base de données.

Un exemple de fichier de migration peut ressembler à ceci:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class AddLastLoggedInToUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dateTime('last_logged_in')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
```

```
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('last_logged_in');
        });
    }
}
```

Lors de l'exécution de cette migration, Laravel générera le code SQL suivant à exécuter sur votre base de données:

```
ALTER TABLE `users` ADD `last_logged_in` DATETIME NULL
```

## Exécution de migrations

Une fois votre migration écrite, son exécution appliquera les opérations à votre base de données.

```
php artisan migrate
```

Si tout s'est bien passé, vous verrez une sortie similaire à la suivante:

```
Migrated: 2016_07_21_134310_add_last_logged_in_to_users_table
```

Laravel est assez intelligent pour savoir quand vous effectuez des migrations dans l'environnement de production. S'il détecte que vous effectuez une migration destructive (par exemple, une migration qui supprime une colonne d'une table), la commande `php artisan migrate` vous demande une confirmation. Dans les environnements de livraison continue, cela peut ne pas être souhaité. Dans ce cas, utilisez l'indicateur `--force` pour ignorer la confirmation:

```
php artisan migrate --force
```

Si vous ne faites qu'exécuter des migrations, vous risquez de ne pas voir la présence d'une table de `migrations` dans votre base de données. Ce tableau est ce que Laravel utilise pour garder une trace des migrations déjà effectuées. Lors de l'émission de la commande de `migrate` , Laravel déterminera les migrations à exécuter, puis les exécutera dans l'ordre chronologique, puis mettra à jour la table de `migrations` en conséquence.

Vous ne devez jamais modifier manuellement la table des `migrations` , sauf si vous savez absolument ce que vous faites. Il est très facile de laisser par inadvertance votre base de données dans un état défectueux où vos migrations échoueront.

## Faire reculer les migrations

Que se passe-t-il si vous souhaitez annuler la dernière migration, à savoir une opération récente, vous pouvez utiliser la commande awesome `rollback` . Mais rappelez-vous que cette commande annule uniquement la dernière migration, qui peut inclure plusieurs fichiers de migration.

```
php artisan migrate:rollback
```

Si vous êtes intéressé par la restauration de toutes vos migrations d'application, vous pouvez utiliser la commande suivante

```
php artisan migrate:reset
```

De plus, si vous êtes paresseux comme moi et que vous souhaitez annuler et migrer avec une seule commande, vous pouvez utiliser cette commande.

```
php artisan migrate:refresh
php artisan migrate:refresh --seed
```

Vous pouvez également spécifier le nombre d'étapes à effectuer avec l'option d' `step` . Comme cela va annuler 1 étape.

```
php artisan migrate:rollback --step=1
```

Lire Migrations de base de données en ligne:
https://riptutorial.com/fr/laravel/topic/1131/migrations-de-base-de-donnees

# Chapitre 50: Modèles de lames

## Introduction

Laravel soutient le moteur de modélisation Blade. Le moteur de modélisation Blade nous permet de créer des modèles maîtres et des modèles de contenu de chargement à partir de modèles principaux, nous pouvons avoir des variables, des boucles et des instructions conditionnelles dans le fichier lame.

## Examples

### Vues: Introduction

Les vues, dans un modèle MVC, contiennent la logique sur *la* présentation des données à l'utilisateur. Dans une application Web, ils sont généralement utilisés pour générer la sortie HTML renvoyée aux utilisateurs avec chaque réponse. Par défaut, les vues dans Laravel sont stockées dans le répertoire `resources/views` .

Une vue peut être appelée à l'aide de la fonction d'aide de la `view` :

```
view(string $path, array $data = [])
```

Le premier paramètre de l'assistant est le chemin d'accès à un fichier de vue et le second paramètre est un tableau de données facultatif à transmettre à la vue.

Par conséquent, pour appeler les `resources/views/example.php` , vous devez utiliser:

```
view('example');
```

Les fichiers affichés dans des sous-dossiers du répertoire `resources/views` , tels que `resources/views/parts/header/navigation.php` , peuvent être appelés à l'aide de la notation par points: `view('parts.header.navigation');`

Dans un fichier de vue, tel que `resources/views/example.php` , vous êtes libre d'inclure à la fois HTML et PHP:

```
<html>
    <head>
        <title>Hello world!</title>
    </head>
    <body>
        <h1>Welcome!</h1>
        <p>Your name is: <?php echo $name; ?></p>
    </body>
</html>
```

Dans l'exemple précédent (qui n'utilise aucune syntaxe spécifique à Blade), nous générons la

variable `$name` . Pour transmettre cette valeur à notre vue, nous passerions un tableau de valeurs lors de l'appel de l'assistant de vue:

```
view('example', ['name' => $name]);
```

ou alternativement, utilisez l'assistant `compact()` . Dans ce cas, la chaîne transmise à compact () correspond au nom de la variable à transmettre à la vue.

```
view('example', compact('name'));
```

**CONVENTION D'APPEL POUR VARIABLES DE LAME**

Lors de l'envoi de données à l'affichage. Vous pouvez utiliser le `underscore` pour la `variable` multi-mots mais avec `-` laravel donne une erreur.

Comme celui-ci donnera une erreur (remarque `hyphen ( - )` dans l' `user-address` l' `user-address`

```
view('example',['user-address' => 'Some Address']);
```

La **manière correcte** de le faire sera

```
view('example', ['user_address' => 'Some Address']);
```

### Structures de contrôle

Blade fournit une syntaxe pratique pour les structures de contrôle PHP courantes.

Chacune des structures de contrôle commence par `@[structure]` et se termine par `@[endstructure]` . Notez que dans les balises, nous ne faisons que taper du code HTML normal et inclure des variables avec la syntaxe Blade.

# Conditionnels

## "Si" déclarations

```
@if ($i > 10)
    <p>{{ $i }} is large.</p>
@elseif ($i == 10)
    <p>{{ $i }} is ten.</p>
@else
    <p>{{ $i }} is small.</p>
@endif
```

## Déclarations 'à moins'

(Syntaxe courte pour «sinon».)

```
@unless ($user->hasName())
    <p>A user has no name.</p>
@endunless
```

# Boucles

## Boucle 'While'

```
@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

## Boucle 'Foreach'

```
@foreach ($users as $id => $name)
    <p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

## Boucle 'Forelse'

(Identique à la boucle 'foreach', mais ajoute une directive `@empty` spéciale, qui est exécutée lorsque l'expression de tableau itérée est vide, afin d'afficher le contenu par défaut.)

```
@forelse($posts as $post)
    <p>{{ $post }} is the post content.</p>
@empty
    <p>There are no posts.</p>
@endforelse
```

---

Dans les boucles, une variable spéciale `$loop` sera disponible, contenant des informations sur l'état de la boucle:

| Propriété | La description |
|---|---|
| `$loop->index` | L'index de l'itération de la boucle en cours (commence à 0). |
| `$loop->iteration` | L'itération de la boucle en cours (commence à 1). |
| `$loop->remaining` | Les itérations de boucle restantes. |
| `$loop->count` | Le nombre total d'éléments dans le tableau en cours d'itération. |
| `$loop->first` | Si c'est la première itération dans la boucle. |
| `$loop->last` | Si c'est la dernière itération à travers la boucle. |

| Propriété | La description |
|---|---|
| `$loop->depth` | Le niveau d'imbrication de la boucle en cours. |
| `$loop->parent` | Dans une boucle imbriquée, la variable de boucle du parent. |

Exemple:

```
@foreach ($users as $user)
  @foreach ($user->posts as $post)
        @if ($loop->parent->first)
            This is first iteration of the parent loop.
        @endif
    @endforeach
@endforeach
```

---

Depuis Laravel 5.2.22, nous pouvons également utiliser les directives `@continue` et `@break`

| Propriété | La description |
|---|---|
| `@continue` | Arrêtez l'itération en cours et lancez la suivante. |
| `@break` | Arrêtez la boucle en cours. |

Exemple :

```
@foreach ($users as $user)
    @continue ($user->id == 2)
        <p>{{ $user->id }} {{ $user->name }}</p>
    @break ($user->id == 4)
@endforeach
```

Ensuite (en *supposant que plus de 5 utilisateurs sont triés par ID et qu'aucun ID ne soit manquant* ), la page sera rendue

```
1 Dave
3 John
4 William
```

### Faire écho aux expressions PHP

Toute expression PHP entre accolades doubles `{{ $variable }}` sera `echo` après avoir été exécutée via la fonction `e` helper . (Les caractères spéciaux html ( `<` , `>` , `"` , `'` , `&` ) sont donc remplacés en toute sécurité pour les entités HTML correspondantes.) (L'expression PHP doit être évaluée en chaîne, sinon une exception sera levée.)

## Faire écho à une variable

```
{{ $variable }}
```

### Faire écho à un élément dans un tableau

```
{{ $array["key"] }}
```

### Faire écho à une propriété d'objet

```
{{ $object->property }}
```

### Faire écho au résultat d'un appel de fonction

```
{{ strtolower($variable) }}
```

## Vérification de l'existence

Normalement, en PHP, vérifier si une variable est définie et l'imprimer

- Avant PHP 7

```
<?php echo isset($variable) ? $variable : 'Default'; ?>
```

- Après PHP 7 (en utilisant le "Null coalescing operator")

```
<?php echo $variable ?? 'Default'; ?>
```

Opérateur de lame `or` facilite ceci:

```
{{ $variable or 'Default' }}
```

## Échos bruts

Comme mentionné précédemment, la syntaxe des doubles accolades `{{ }}` est filtrée via la fonction `htmlspecialchars` de PHP, pour des `htmlspecialchars` de sécurité (prévention de l'injection malveillante de HTML dans la vue). Si vous souhaitez contourner ce comportement, par exemple si vous essayez de générer un bloc de contenu HTML résultant d'une expression PHP, utilisez la syntaxe suivante:

```
{!! $myHtmlString !!}
```

Notez qu'il est recommandé d'utiliser la syntaxe standard `{{ }}` pour échapper à vos données,

sauf en cas d'absolue nécessité. De plus, lorsque vous faites écho à un contenu non fiable (c.-à-d. Contenu fourni par les utilisateurs de votre site), vous devriez éviter d'utiliser le `{!! !!}` syntaxe.

## Y compris les vues partielles

Avec Blade, vous pouvez également inclure des vues partielles (appelées "partials") directement dans une page comme celle-ci:

```
@include('includes.info', ['title' => 'Information Station'])
```

Le code ci-dessus inclura la vue dans "views / includes / info.blade.php". Il passera également dans une variable `$title` ayant la valeur 'Information Station'.

En général, une page incluse aura accès à toute variable à laquelle la page appelante a accès. Par exemple, si nous avons:

```
{{$user}} // Outputs 'abc123'
@include('includes.info')
```

Et 'includes / info.blade.php' a les caractéristiques suivantes:

```
<p>{{$user}} is the current user.</p>
```

Ensuite, la page affichera:

```
abc123
abc123 is the current user.
```

### Inclure chaque

Parfois, vous souhaiterez combiner une instruction `include` avec une instruction `foreach` et accéder aux variables depuis la boucle foreach de l'inclusion. Dans ce cas, utilisez la directive `@each` de Blade:

```
@each('includes.job', $jobs, 'job')
```

Le premier paramètre est la page à inclure. Le second paramètre est le tableau à parcourir. Le troisième paramètre est la variable affectée aux éléments du tableau. La déclaration ci-dessus est équivalente à:

```
@foreach($jobs as $job)
    @include('includes.job', ['job' => $job])
@endforeach
```

Vous pouvez également transmettre un quatrième argument facultatif à la directive `@each` pour spécifier la vue à afficher lorsque le tableau est vide.

```
@each('includes.job', $jobs, 'job', 'includes.jobsEmpty')
```

## Héritage

Une mise en page est un fichier de vue, étendu par d'autres vues qui injectent des blocs de code dans leur parent. Par exemple:

**parent.blade.php:**

```
<html>
    <head>
        <style type='text/css'>
        @yield('styling')
        </style>
    </head>
    <body>
        <div class='main'>
        @yield('main-content')
        </div>
    </body>
</html>
```

**child.blade.php:**

```
@extends('parent')

@section('styling')
.main {
    color: red;
}
@stop

@section('main-content')
This is child page!
@stop
```

**otherpage.blade.php:**

```
@extends('parent')

@section('styling')
.main {
    color: blue;
}
@stop

@section('main-content')
This is another page!
@stop
```

Vous voyez ici deux exemples de pages enfants, qui étendent chacune le parent. Les pages enfants définissent une `@section` , qui est insérée dans le parent à l'instruction `@yield` appropriée.

Donc la vue rendue par `View::make('child')` dira " **This is child page!** " En rouge, alors que `View::make('otherpage')` produira le même html, sauf avec le texte " **Ceci est un autre page!** "en bleu à la place.

Il est courant de séparer les fichiers de vue, par exemple en ayant un dossier de mise en page spécifique aux fichiers de mise en page et un dossier distinct pour les différentes vues individuelles spécifiques.

Les mises en page sont destinées à appliquer du code qui devrait apparaître sur chaque page, par exemple en ajoutant une barre latérale ou un en-tête, sans avoir à écrire tout le code HTML dans chaque vue.

Les vues peuvent être étendues à plusieurs reprises - à savoir **page3** peut `@extend('page2')` , et **page2** peut `@extend('page1')` .

La commande extend utilise la même syntaxe que celle utilisée pour `View::make` et `@include` , de sorte que le fichier `layouts/main/page.blade.php` est accessible en tant que `layouts.main.page` .

### Partage de données à toutes les vues

Parfois, vous devez définir les mêmes données dans plusieurs de vos vues.

# Using View :: share

```
// "View" is the View Facade
View::share('shareddata', $data);
```

Après cela, le contenu de `$data` sera disponible dans toutes les vues sous le nom `$shareddata` .

`View::share` est généralement appelé dans un fournisseur de services, ou peut-être dans le constructeur d'un contrôleur, de sorte que les données seront partagées uniquement dans les vues renvoyées par ce contrôleur.

# Using View :: composer

Les compositeurs de vue sont des méthodes de rappel ou de classe appelées lors du rendu d'une vue. Si vous souhaitez associer des données à une vue chaque fois que cette vue est rendue, un compositeur de vue peut vous aider à organiser cette logique en un seul emplacement. Vous pouvez directement lier variable à une vue spécifique ou à toutes les vues.

## Compositeur basé sur la fermeture

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', function ($view) {
    $view->with('somedata', $data);
});
```

## Compositeur en classe

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', 'App\Http\ViewComposers\SomeComposer');
```

Comme avec `View::share` , il est préférable d'enregistrer les compositeurs dans un fournisseur de services.

Si vous `App/Http/ViewComposers/SomeComposer.php` approche de la classe composer, vous aurez `App/Http/ViewComposers/SomeComposer.php` avec:

```
use Illuminate\Contracts\View\View;

class SomeComposer
{
    public function compose(View $view)
    {
        $view->with('somedata', $data);
    }
}
```

Ces exemples utilisent `'*'` dans l'enregistrement du compositeur. Ce paramètre est une chaîne qui correspond aux noms de vues pour lesquels enregistrer le compositeur ( `*` étant un caractère générique). Vous pouvez également sélectionner une vue unique (par exemple, `'home'` ) d'un groupe de routes sous un sous-dossier (par exemple, `'users.*'` ).

### Exécuter du code PHP arbitraire

Bien que cela puisse ne pas être approprié dans une vue si vous avez l'intention de séparer strictement les préoccupations, la directive `php` Blade permet d'exécuter du code PHP, par exemple pour définir une variable:

```
@php($varName = 'Enter content ')
```

(pareil que:)

```
@php
    $varName = 'Enter content ';
@endphp
```

plus tard:

```
{{ $varName }}
```

Résultat:

Entrer du contenu

Lire Modèles de lames en ligne: https://riptutorial.com/fr/laravel/topic/1407/modeles-de-lames

# Chapitre 51: Modifier le comportement de routage par défaut dans Laravel 5.2.31 +

## Syntaxe

- carte de fonction publique (routeur $ router) // Définit les routes pour l'application.
- function protected mapWebRoutes (Router $ router) // Définit les routes "web" pour l'application.

## Paramètres

| Paramètre | Entête |
|---|---|
| Routeur $ routeur | \ Illuminate \ Routing \ Routeur $ routeur |

## Remarques

Le middleware signifie que chaque appel à un itinéraire passe par le middleware avant que vous n'atteigniez le code spécifique à votre itinéraire. Dans Laravel, le middleware Web est utilisé pour assurer la gestion des sessions ou la vérification du jeton csrf par exemple.

Il y a d'autres middlewares comme auth ou api par défaut. Vous pouvez également créer facilement votre propre middleware.

## Examples

### Ajout d'api-routes avec d'autres logiciels intermédiaires et conservation du middleware Web par défaut

Depuis la version 5.2.31 de Laravel, le middleware Web est appliqué par défaut dans le RouteServiceProvider ( https://github.com/laravel/laravel/commit/5c30c98db96459b4cc878d085490e4677b0b67ed)

Dans app / Providers / RouteServiceProvider.php, vous trouverez les fonctions suivantes qui appliquent le middleware sur chaque route de votre application / Http / routes.php

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
}

// ...

protected function mapWebRoutes(Router $router)
```

```
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}
```

Comme vous pouvez le voir, le web **middleware** est appliqué. Vous pourriez changer cela ici. Cependant, vous pouvez aussi facilement ajouter une autre entrée pour pouvoir mettre vos routes api par exemple dans un autre fichier (par exemple routes-api.php)

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
    $this->mapApiRoutes($router);
}

protected function mapWebRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}

protected function mapApiRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'api',
    ], function ($router) {
        require app_path('Http/routes-api.php');
    });
}
```

Avec cela, vous pouvez facilement séparer les routes api de vos routes d'application sans l'enveloppe de groupe désordonnée dans vos routes.php

Lire Modifier le comportement de routage par défaut dans Laravel 5.2.31 + en ligne: https://riptutorial.com/fr/laravel/topic/4285/modifier-le-comportement-de-routage-par-defaut-dans-laravel-5-2-31-plus

# Chapitre 52: Nommer des fichiers lors du téléchargement avec Laravel sur Windows

## Paramètres

| Param / Fonction | La description |
|---|---|
| téléchargement de fichiers | nom du champ `<input>` du fichier |
| $ sampleName | pourrait également être une chaîne générée dynamiquement ou le nom du fichier téléchargé par l'utilisateur |
| app_path () | est l'aide de Laravel pour fournir le chemin absolu à l'application |
| getCLientOriginalExtension () | Wrapper Laravel pour récupérer l'extension du fichier téléchargé par l'utilisateur tel qu'il était sur la machine de l'utilisateur |

## Examples

### Génération de noms de fichiers horodatés pour les fichiers téléchargés par les utilisateurs.

**Ci-dessous ne fonctionnera pas sur une machine Windows**

```
$file = $request->file('file_upload');
$sampleName = 'UserUpload';
$destination = app_path() . '/myStorage/';
$fileName = $sampleName . '-' . date('Y-m-d-H:i:s') . '.' .
$file->getClientOriginalExtension();
$file->move($destination, $fileName);
```

Il va lancer une erreur comme "Impossible de déplacer le fichier vers / path ..."

**Pourquoi? - Cela fonctionne parfaitement sur un serveur Ubuntu**
La raison en est que, sous le `colon ':'` Windows, `colon ':'` n'est pas autorisé dans un nom de fichier où il est autorisé sur Linux. C'est tellement petit que nous ne le remarquons peut-être pas dès le départ et que nous nous demandons pourquoi un code qui fonctionne bien sur Ubuntu (Linux) ne fonctionne pas.
Notre première intuition serait de vérifier les autorisations de fichiers et des choses comme ça, mais nous ne remarquons peut-être pas que les `colon ':'` sont le coupable ici.
Donc, pour télécharger des fichiers sous Windows, **n'utilisez pas les `colon':'` lors de la génération des noms de fichiers horodatés** , faites plutôt quelque chose comme ci-dessous:

```
$filename = $sampleName . '-' . date('Y-m-d-H_i_s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18-11_25_43.xlsx

                                    OR

$filename = $sampleName . '-' .date('Y-m-d H i s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18 11 25 43.xlsx

                                    OR

$filename = $sampleName . '-'.date('Y-m-d_g-i-A').'.' . $file->getClientOriginalExtension();
//ex output UserUpload-2016-02-18_11-25-AM.xlsx
```

Lire Nommer des fichiers lors du téléchargement avec Laravel sur Windows en ligne: https://riptutorial.com/fr/laravel/topic/2629/nommer-des-fichiers-lors-du-telechargement-avec-laravel-sur-windows

# Chapitre 53: Observateur

## Examples

### Créer un observateur

Les observateurs sont utilisés pour écouter les rappels de cycle de vie d'un certain modèle chez Laravel.
Ces auditeurs peuvent écouter l'une des actions suivantes:

- créer
- créé
- mise à jour
- actualisé
- économie
- enregistré
- effacer
- supprimé
- la restauration
- restauré

Voici un exemple d'observateur.

**UserObserver**

```
<?php

namespace App\Observers;

/**
 * Observes the Users model
 */
class UserObserver
{
    /**
     * Function will be triggerd when a user is updated
     *
     * @param Users $model
     */
     public function updated($model)
     {
         // execute your own code
     }
}
```

Comme le montre l'observateur de l'utilisateur, nous écoutons l'action mise à jour, mais avant que cette classe n'écoute réellement le modèle utilisateur, nous devons d'abord l'enregistrer dans `EventServiceProvider` .

**EventServiceProvider**

```php
<?php

namespace App\Providers;

use Illuminate\Contracts\Events\Dispatcher as DispatcherContract;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;

use App\Models\Users;
use App\Observers\UserObserver;

/**
 * Event service provider class
 */
class EventServiceProvider extends ServiceProvider
{
    /**
     * Boot function
     *
     * @param DispatcherContract $events
     */
    public function boot(DispatcherContract $events)
    {
        parent::boot($events);

        // In this case we have a User model that we want to observe
        // We tell Laravel that the observer for the user model is the UserObserver
        Users::observe(new UserObserver());
    }
}
```

Maintenant que nous avons enregistré notre observateur, la fonction mise à jour sera appelée à chaque fois après avoir enregistré le modèle utilisateur.

Lire Observateur en ligne: https://riptutorial.com/fr/laravel/topic/7128/observateur

# Chapitre 54: Pagination

## Examples

### Pagination à Laravel

Dans d'autres cadres, la pagination est un mal de tête. Laravel le rend facile, il peut générer de la pagination en ajoutant quelques lignes de code dans Controller et View.

**Utilisation de base**

Il existe plusieurs façons de paginer des éléments, mais le plus simple consiste à utiliser la méthode paginate sur le générateur de requêtes ou une requête Eloquent . Laravel out of the box prend en charge la définition des limites et des décalages en fonction de la page en cours d'affichage par l'utilisateur. Par défaut, la page en cours est détectée par la valeur de l'argument de chaîne de requête? Page sur la requête HTTP. Et bien sûr, Laravel détecte automatiquement cette valeur et l'insère dans les liens générés par le paginateur.

Maintenant, disons que nous voulons appeler la méthode `paginate` sur la requête. Dans notre exemple, l'argument passé à paginer est le nombre d'éléments que vous souhaitez afficher "par page". Dans notre cas, disons que nous voulons afficher 10 éléments par page.

```
<?php

namespace App\Http\Controllers;

use DB;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show all of the users for the application.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::table('users')->paginate(10);

        return view('user.index', ['users' => $users]);
    }
}
```

> **Remarque:** Actuellement, les opérations de pagination utilisant une instruction `groupBy` ne peuvent pas être exécutées efficacement par Laravel. Si vous avez besoin d'utiliser un `groupBy` avec un jeu de résultats paginé, il est recommandé d'interroger la base de données et de créer un paginateur manuellement.

**Pagination simple**

Disons que vous voulez simplement afficher les liens Suivant et Précédent sur votre vue de pagination. Laravel vous fournit cette option en utilisant la méthode `simplePaginate` .

```
$users = DB::table('users')->simplePaginate(10);
```

**Affichage des résultats dans une vue**

Maintenant, permet d'afficher la pagination en vue. En fait , lorsque vous appelez la `paginate` ou `simplePaginate` méthodes sur requête Eloquent, vous recevez une instance paginator. Lorsque la méthode paginate est appelée, vous recevez une instance de `Illuminate\Pagination\LengthAwarePaginator` , tandis que lorsque vous appelez la méthode `simplePaginate` , vous recevez une instance de `Illuminate\Pagination\Paginator` . Ces instances / objets sont livrés avec plusieurs méthodes expliquant le jeu de résultats. De plus, en plus de ces méthodes d'aide, les instances de paginateur sont des itérateurs et peuvent être mises en boucle sous forme de tableau.

Une fois que vous avez reçu les résultats, vous pouvez facilement rendre les liens de la page à l'aide de la lame.

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

{{ $users->links() }}
```

La méthode des `links` rendra automatiquement les liens vers les autres pages du jeu de résultats. Chacun de ces liens contiendra le numéro de page spécifique, à savoir la variable de chaîne de requête de `?page` . Le HTML généré par la méthode links est parfaitement compatible avec le framework CSS Bootstrap.

## Modification des vues de pagination

Lorsque vous utilisez la pagination laravel, vous êtes libre d'utiliser vos propres vues personnalisées. Ainsi, lorsque vous appelez la méthode links sur une instance de paginateur, transmettez le nom de la vue comme premier argument à la méthode, comme:

```
{{ $paginator->links('view.name') }}
```

ou

Vous pouvez personnaliser les vues de pagination en les exportant dans votre répertoire `resources/views/vendor` à l'aide de la commande vendor: publish:

```
php artisan vendor:publish --tag=laravel-pagination
```

Cette commande place les vues dans le répertoire `resources/views/vendor/pagination` . Le fichier `default.blade.php` dans ce répertoire correspond à la vue de pagination par défaut. Editez ce

fichier pour modifier le code HTML de la pagination.

Lire Pagination en ligne: https://riptutorial.com/fr/laravel/topic/2359/pagination

# Chapitre 55: Planification des tâches

## Examples

### Créer une tâche

Vous pouvez créer une tâche (commande de la console) dans Laravel en utilisant Artisan. Depuis votre ligne de commande:

```
php artisan make:console MyTaskName
```

Cela crée le fichier dans **app / Console / Commands / MyTaskName.php** . Il ressemblera à ceci:

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

class MyTaskName extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'command:name';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Create a new command instance.
     *
     * @return void
     */
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
        //
    }
}
```

Certaines parties importantes de cette définition sont les suivantes:

- La propriété `$signature` identifie votre commande. Vous pourrez exécuter cette commande ultérieurement via la ligne de commande en utilisant Artisan en exécutant la `php artisan command:name` (Where `command:name` correspond à la `$signature` votre commande)
- La propriété `$description` est l'aide / l'utilisation de Artisan à côté de votre commande lorsqu'elle est disponible.
- La méthode `handle()` est l'endroit où vous écrivez le code pour votre commande.

Finalement, votre tâche sera mise à la disposition de la ligne de commande via Artisan. Le `protected $signature = 'command:name';` propriété sur cette classe est ce que vous utiliseriez pour l'exécuter.

## Rendre une tâche disponible

Vous pouvez rendre une tâche accessible à Artisan et à votre application dans le fichier **app / Console / Kernel.php** .

La classe `Kernel` contient un tableau nommé `$commands` qui rend vos commandes disponibles pour votre application.

Ajoutez votre commande à ce tableau, afin de le rendre disponible pour Artisan et votre application.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class // This line makes MyTaskName available
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {

    }
}
```

Une fois cela fait, vous pouvez maintenant accéder à votre commande via la ligne de commande, en utilisant Artisan. En supposant que votre commande a la propriété `$signature` définie sur `my:task` , vous pouvez exécuter la commande suivante pour exécuter votre tâche:

```
php artisan my:task
```

## Planification de votre tâche

Lorsque votre commande est mise à la disposition de votre application, vous pouvez utiliser Laravel pour planifier son exécution à des intervalles prédéfinis, comme vous le feriez avec un CRON.

Dans le fichier **app / Console / Kernel.php** , vous trouverez une méthode de `schedule` que vous pouvez utiliser pour planifier votre tâche.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
        $schedule->command('my:task')->everyMinute();
        // $schedule->command('my:task')->everyFiveMinutes();
        // $schedule->command('my:task')->daily();
        // $schedule->command('my:task')->monthly();
        // $schedule->command('my:task')->sundays();
    }
}
```

En supposant que la `$signature` votre tâche est `my:task` vous pouvez la planifier comme indiqué ci-dessus, à l'aide de l'objet `Schedule $schedule` . Laravel fournit de nombreuses façons de planifier votre commande, comme indiqué dans les lignes ci-dessus.

### Définition du planificateur à exécuter

Le planificateur peut être exécuté en utilisant la commande:

```
php artisan schedule:run
```

Le planificateur doit être exécuté toutes les minutes pour fonctionner correctement. Vous pouvez configurer cela en créant un travail cron avec la ligne suivante, qui exécute le planificateur toutes les minutes en arrière-plan.

```
* * * * * php /path/to/artisan schedule:run >> /dev/null 2>&1
```

Lire Planification des tâches en ligne: https://riptutorial.com/fr/laravel/topic/4026/planification-des-taches

# Chapitre 56: Prestations de service

## Examples

### introduction

Laravel permet d'accéder à une variété de classes appelées Services. Certains services sont disponibles immédiatement, mais vous pouvez les créer vous-même.

Un service peut être utilisé dans plusieurs fichiers de l'application, comme les contrôleurs. Imaginons un Service `OurService` implémentant une méthode `getNumber()` renvoyant un nombre aléatoire:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        return app(OurService::class)->getNumber();
    }
}
```

Pour créer un service, il suffit de créer une nouvelle classe:

```
# app/Services/OurService/OurService.php

<?php
namespace App\Services\OurService;

class OurService
{
    public function getNumber()
    {
        return rand();
    }
}
```

Pour le moment, vous pouvez déjà utiliser ce service dans un contrôleur, mais vous devrez instancier un nouvel objet chaque fois que vous en aurez besoin:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }

    public function getOtherRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }
```

```
}
```

C'est pourquoi l'étape suivante consiste à enregistrer votre service dans le **conteneur de services** . Lorsque vous enregistrez votre service dans le conteneur de service, vous n'avez pas besoin de créer un nouvel objet chaque fois que vous en avez besoin.

Pour enregistrer un service dans le conteneur de services, vous devez créer un **fournisseur de services** . Ce fournisseur de services peut:

1. Enregistrez votre service dans le conteneur de services avec *la méthode register* )
2. Injection d'autres services dans votre service (dépendances) avec *la méthode de démarrage*

Un **fournisseur de services** est une classe qui étend la classe abstraite `Illuminate\Support\ServiceProvider` . Il doit implémenter la méthode `register()` pour **enregistrer** un service dans le **conteneur de services** :

```
# app/Services/OurService/OurServiceServiceProvider.php

<?php
namespace App\Services\OurService;

use Illuminate\Support\ServiceProvider;

class OurServiceServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('OurService', function($app) {
            return new OurService();
        });
    }
}
```

Tous les **fournisseurs de services** sont enregistrés dans un tableau dans `config/app.php` . Nous devons donc enregistrer notre fournisseur de services dans ce tableau:

```
return [

    ...

    'providers' => [

        ...

        App\Services\OurService\OurServiceServiceProvider::class,

        ...

    ],

    ...

];
```

Nous pouvons maintenant accéder à notre service dans un contrôleur. Trois possibilités:

1. Injection de dépendance:

```
<?php
namespace App\Http\Controllers;

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function __construct(OurService $our_service)
    {
        dd($our_service->getNumber());
    }
}
```

2. Accès via l' `app()` :

```
<?php
namespace App\Http\Controllers;

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return app('OurService')->getNumber();
    }
}
```

Laravel fournit des façades, des classes imaginaires que vous pouvez utiliser dans tous vos projets et refléter un service. Pour accéder plus facilement à votre service, vous pouvez créer une façade:

```
<?php
namespace App\Http\Controllers;

use Randomisator;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return Randomisator::getNumber();
    }

}
```

Pour créer une nouvelle façade, vous devez créer une nouvelle classe étendant `Illuminate\Support\Facades\Facade` . Cette classe doit implémenter la méthode `getFacadeAccessor()` et renvoyer le nom d'un service enregistré par un **fournisseur de services** :

```
# app/Services/OurService/OurServiceFacade.php

<?php
```

```
namespace App\Services\OurService;

use Illuminate\Support\Facades\Facade;

class OurServiceFacade extends Facade
{
    protected static function getFacadeAccessor()
    {
        return 'OurService';
    }
}
```

Vous devez également enregistrer votre façade dans `config/app.php` :

```
return [

    ...

    'aliases' => [

        ....

        'Randomisator' => App\Services\OurService\OurServiceFacade::class,
    ],

];
```

La façade est maintenant accessible partout dans votre projet.

Si vous souhaitez accéder à votre service depuis vos vues, vous pouvez créer une fonction d'assistance. Laravel est livré avec des fonctions d'aide telles que la fonction `auth()` ou la fonction `view()` . Pour créer une fonction d'assistance, créez un nouveau fichier:

```
# app/Services/OurService/helpers.php

if (! function_exists('randomisator')) {
    /**
     * Get the available OurService instance.
     *
     * @return \App\ElMatella\FacebookLaravelSdk
     */
    function randomisator()
    {
        return app('OurService');
    }
}
```

Vous devez également enregistrer ce fichier, mais dans votre fichier `composer.json` :

```
{

    ...

    "autoload": {
        "files": [
            "app/Services/OurService/helpers.php"
```

```
        ],

        ...
    }
}
```

Vous pouvez maintenant utiliser cet assistant dans une vue:

```
<h1>Here is a random number: {{ randomisator()->getNumber() }}</h1>
```

Lire Prestations de service en ligne: https://riptutorial.com/fr/laravel/topic/1907/prestations-de-service

# Chapitre 57: Prestations de service

## Examples

### Relier une interface à la mise en œuvre

Dans une méthode de `register` fournisseur de services, nous pouvons associer une interface à une implémentation:

```
public function register()
{
    App::bind( UserRepositoryInterface::class, EloquentUserRepository::class );
}
```

A partir de maintenant, chaque fois que l'application aura besoin d'une instance de `UserRepositoryInterface` , Laravel injectera automatiquement une nouvelle instance d' `EloquentUserRepository` :

```
//this will get back an instance of EloquentUserRepository
$repo = App::make( UserRepositoryInterface:class );
```

### Relier une instance

Nous pouvons utiliser le conteneur de service en tant que registre en liant une instance d'un objet et le récupérer lorsque nous en aurons besoin:

```
// Create an instance.
$john = new User('John');

// Bind it to the service container.
App::instance('the-user', $john);

// ...somewhere and/or in another class...

// Get back the instance
$john = App::make('the-user');
```

### Liaison d'un singleton au conteneur de services

On peut lier une classe en singleton:

```
public function register()
{
    App::singleton('my-database', function()
    {
        return new Database();
    });
}
```

Ainsi, la première fois qu'une instance de `'my-database'` sera demandée au conteneur de services, une nouvelle instance sera créée. Toutes les requêtes successives de cette classe récupèreront la première instance créée:

```
//a new instance of Database is created
$db = App::make('my-database');

//the same instance created before is returned
$anotherDb = App::make('my-database');
```

## introduction

Le **conteneur de service** est l'objet Application principal. Il peut être utilisé comme conteneur d'injection de dépendances et comme registre pour l'application en définissant des liaisons dans les fournisseurs de services.

**Les fournisseurs de services** sont des classes dans lesquelles nous définissons la manière dont nos classes de service seront créées via l'application, amorcent leur configuration et lient les interfaces aux implémentations.

**Les services** sont des classes qui regroupent une ou plusieurs tâches logiques corrélées

## Utilisation du conteneur de services en tant que conteneur d'injection de dépendances

Nous pouvons utiliser le conteneur de services en tant que conteneur d'injection de dépendances en liant le processus de création d'objets avec leurs dépendances en un point de l'application.

Supposons que la création d'un `PdfCreator` nécessite deux objets en tant que dépendances; chaque fois que nous avons besoin de construire une instance de `PdfCreator` , nous devons transmettre ces dépendances au constructeur. En utilisant le conteneur de service en tant que DIC, nous définissons la création de `PdfCreator` dans la définition de la liaison, en prenant directement la dépendance requise à partir du conteneur de services:

```
App:bind('pdf-creator', function($app) {

    // Get the needed dependencies from the service container.
    $pdfRender = $app->make('pdf-render');
    $templateManager = $app->make('template-manager');

    // Create the instance passing the needed dependencies.
    return new PdfCreator( $pdfRender, $templateManager );
});
```

Ensuite, à chaque `PdfCreator` de notre application, pour obtenir un nouveau `PdfCreator` , nous pouvons simplement faire:

```
$pdfCreator = App::make('pdf-creator');
```

Et le conteneur de service créera une nouvelle instance, avec les dépendances nécessaires pour

nous.

Lire Prestations de service en ligne: https://riptutorial.com/fr/laravel/topic/1908/prestations-de-service

# Chapitre 58: Problèmes courants et solutions rapides

## Introduction

Cette section répertorie les problèmes courants et les correctifs rapides que rencontrent les développeurs (en particulier les débutants).

## Examples

### Exception TokenMisMatch

Vous obtenez cette exception principalement avec les soumissions de formulaire. Laravel protège l'application de `CSRF` et valide chaque demande et s'assure que la demande provient de l'application. Cette validation est effectuée à l'aide d'un `token` . Si ce jeton ne correspond pas à cette exception, il est généré.

**Solution rapide**

Ajoutez ceci dans votre élément de formulaire. Cela envoie `csrf_token` généré par laravel avec d'autres données de formulaire si laravel sait que votre demande est valide

```
<input type="hidden" name="_token" value="{{ csrf_token() }}">
```

Lire Problèmes courants et solutions rapides en ligne:
https://riptutorial.com/fr/laravel/topic/9971/problemes-courants-et-solutions-rapides

# Chapitre 59: Socialite

## Examples

### Installation

```
composer require laravel/socialite
```

Cette installation suppose que vous utilisez Composer pour gérer vos dépendances avec Laravel, ce qui est un excellent moyen de le gérer.

### Configuration

Dans votre `config\services.php` vous pouvez ajouter le code suivant

```
'facebook' => [
    'client_id' => 'your-facebook-app-id',
    'client_secret' => 'your-facebook-app-secret',
    'redirect' => 'http://your-callback-url',
],
```

Vous devrez également ajouter le fournisseur à votre `config\app.php`

Recherchez le tableau `'providers' => []` et, à la fin, ajoutez ce qui suit

```
'providers' => [
      ...

      Laravel\Socialite\SocialiteServiceProvider::class,
]
```

Une façade est également fournie avec le forfait. Si vous souhaitez l'utiliser, assurez-vous que le tableau d' `aliases` (également dans votre `config\app.php` ) a le code suivant

```
'aliases' => [
    ....
    'Socialite' => Laravel\Socialite\Facades\Socialite::class,
]
```

### Utilisation de base - Façade

```
return Socialite::driver('facebook')->redirect();
```

Cela redirigera une demande entrante vers l'URL appropriée à authentifier. Un exemple de base serait dans un contrôleur

```
<?php
```

```
namespace App\Http\Controllers\Auth;

use Socialite;

class AuthenticationController extends Controller {

    /**
     * Redirects the User to the Facebook page to get authorization.
     *
     * @return Response
     */
    public function facebook() {
        return Socialite::driver('facebook')->redirect();
    }

}
```

assurez-vous que votre fichier `app\Http\routes.php` a un itinéraire pour autoriser une demande entrante ici.

```
Route::get('facebook', 'App\Http\Controllers\Auth\AuthenticationController@facebook');
```

## Utilisation de base - Injection de dépendance

```
/**
 * LoginController constructor.
 * @param Socialite $socialite
 */
public function __construct(Socialite $socialite) {
    $this->socialite = $socialite;
}
```

Dans le constructeur de votre contrôleur, vous pouvez désormais injecter la classe `Socialite` qui vous aidera à gérer les connexions avec les réseaux sociaux. Cela remplacera l'utilisation de la façade.

```
/**
 * Redirects the User to the Facebook page to get authorization.
 *
 * @return Response
 */
public function facebook() {
    return $this->socialite->driver('facebook')->redirect();
}
```

## Socialite for API - Stateless

```
public function facebook() {
    return $this->socialite->driver('facebook')->stateless()->redirect()->getTargetUrl();
}
```

Cela renverra l'URL que le consommateur de l'API doit fournir à l'utilisateur final pour obtenir

l'autorisation de Facebook.

Lire Socialite en ligne: https://riptutorial.com/fr/laravel/topic/1312/socialite

# Chapitre 60: Stockage de système de fichiers / cloud

## Examples

### Configuration

Le fichier de configuration du système de fichiers se trouve dans `config/filesystems.php` . Dans ce fichier, vous pouvez configurer tous vos "disques". Chaque disque représente un pilote de stockage et un emplacement de stockage particuliers. Des exemples de configuration pour chaque pilote pris en charge sont inclus dans le fichier de configuration. Donc, modifiez simplement la configuration pour refléter vos préférences de stockage et vos informations d'identification!

Avant d'utiliser les pilotes S3 ou Rackspace, vous devrez installer le package approprié via Composer:

- Amazon S3: `league/flysystem-aws-s3-v2 ~1.0`
- Rackspace: `league/flysystem-rackspace ~1.0`

Bien entendu, vous pouvez configurer autant de disques que vous le souhaitez et même disposer de plusieurs disques utilisant le même pilote.

Lorsque vous utilisez le pilote local, notez que toutes les opérations sur les fichiers sont relatives au répertoire racine défini dans votre fichier de configuration. Par défaut, cette valeur est définie sur le `storage/app directory` . Par conséquent, la méthode suivante stockera un fichier dans `storage/app/file.txt` :

```
Storage::disk('local')->put('file.txt', 'Contents');
```

### Utilisation de base

La façade de `Storage` peut être utilisée pour interagir avec l'un de vos disques configurés. Vous pouvez également indiquer le `Illuminate\Contracts\Filesystem\Factory` sur toute classe résolue via le conteneur de services Laravel.

#### Récupération d'un disque particulier

```
$disk = Storage::disk('s3');

$disk = Storage::disk('local');
```

#### Déterminer si un fichier existe

```
$exists = Storage::disk('s3')->exists('file.jpg');
```

### Méthodes d'appel sur le disque par défaut

```
if (Storage::exists('file.jpg'))
{
    //
}
```

### Récupération du contenu d'un fichier

```
$contents = Storage::get('file.jpg');
```

### Définition du contenu d'un fichier

```
Storage::put('file.jpg', $contents);
```

### Ajouter à un fichier

```
Storage::prepend('file.log', 'Prepended Text');
```

### Ajouter à un fichier

```
Storage::append('file.log', 'Appended Text');
```

### Supprimer un fichier

```
Storage::delete('file.jpg');

Storage::delete(['file1.jpg', 'file2.jpg']);
```

### Copier un fichier dans un nouvel emplacement

```
Storage::copy('old/file1.jpg', 'new/file1.jpg');
```

### Déplacer un fichier vers un nouvel emplacement

```
Storage::move('old/file1.jpg', 'new/file1.jpg');
```

### Obtenir la taille du fichier

```
$size = Storage::size('file1.jpg');
```

### Obtenir l'heure de dernière modification (UNIX)

```
$time = Storage::lastModified('file1.jpg');
```

### Obtenir tous les fichiers dans un répertoire

```
$files = Storage::files($directory);

// Recursive...
$files = Storage::allFiles($directory);
```

### Obtenir tous les répertoires dans un répertoire

```
$directories = Storage::directories($directory);

// Recursive...
$directories = Storage::allDirectories($directory);
```

### Créer un répertoire

```
Storage::makeDirectory($directory);
```

### Supprimer un répertoire

```
Storage::deleteDirectory($directory);
```

## Systèmes de fichiers personnalisés

L'intégration Flysystem de Laravel fournit des drivers pour plusieurs "drivers" prêts à l'emploi. Cependant, Flysystem ne se limite pas à ceux-ci et possède des adaptateurs pour de nombreux autres systèmes de stockage. Vous pouvez créer un pilote personnalisé si vous souhaitez utiliser l'un de ces adaptateurs supplémentaires dans votre application Laravel. Ne vous inquiétez pas, ce n'est pas trop difficile!

Pour configurer le système de fichiers personnalisé, vous devez créer un fournisseur de services tel que `DropboxFilesystemServiceProvider` . Dans la méthode de `boot` du fournisseur, vous pouvez injecter une instance du `Illuminate\Contracts\Filesystem\Factory` et appeler la méthode `extend` de l'instance injectée. Vous pouvez également utiliser la méthode `extend` la façade du `Disk` .

Le premier argument de la méthode `extend` est le nom du pilote et le second, une fermeture qui reçoit les variables `$app` et `$config` . Le résolveur Closure doit renvoyer une instance de `League\Flysystem\Filesystem` .

> Remarque: La variable $ config contiendra déjà les valeurs définies dans `config/filesystems.php` pour le disque spécifié. Exemple de Dropbox

```
<?php namespace App\Providers;

use Storage;
use League\Flysystem\Filesystem;
use Dropbox\Client as DropboxClient;
use League\Flysystem\Dropbox\DropboxAdapter;
use Illuminate\Support\ServiceProvider;

class DropboxFilesystemServiceProvider extends ServiceProvider {

    public function boot()
```

```
    {
        Storage::extend('dropbox', function($app, $config)
        {
            $client = new DropboxClient($config['accessToken'], $config['clientIdentifier']);

            return new Filesystem(new DropboxAdapter($client));
        });
    }

    public function register()
    {
        //
    }

}
```

## Création d'un lien symbolique sur un serveur Web à l'aide de SSH

Dans la documentation de Laravel, un lien symbolique (lien symbolique ou lien logiciel) entre public / stockage et stockage / application / public doit être créé pour rendre les fichiers accessibles depuis le Web.

*(CETTE PROCÉDURE CRÉERA UN LIEN SYMBOLIQUE AU SEIN DU RÉPERTOIRE DU PROJET LARAVEL)*

Voici les étapes à suivre pour créer un lien symbolique sur votre serveur Web Linux à l'aide du client SSH:

1. Connectez-vous et connectez-vous à votre serveur Web à l'aide du client SSH (par exemple, PUTTY).

2. Lier le **stockage / app / public** à **public / storage en** utilisant la syntaxe

   ```
   ln -s target_path link_path
   ```

   Exemple (dans le répertoire de fichiers CPanel)

   ```
   ln -s /home/cpanel_username/project_name/storage/app/public
   /home/cpanel_sername/project_name/public/storage
   ```

*(Un dossier nommé **storage** sera créé pour lier le chemin avec un indicateur **>>>** sur l'icône du dossier.)*

Lire Stockage de système de fichiers / cloud en ligne:
https://riptutorial.com/fr/laravel/topic/3040/stockage-de-systeme-de-fichiers---cloud

# Chapitre 61: Structure du répertoire

## Examples

### Modifier le répertoire d'application par défaut

Il y a des cas d'utilisation où vous pourriez vouloir renommer votre répertoire d'application à autre chose. Dans Laravel4, vous pouvez simplement modifier une entrée de configuration, voici une façon de le faire dans Laravel5.

Dans cet exemple, nous allons renommer le répertoire de l' `app` en `src` .

## Remplacer la classe d'application

L' `app` nom de répertoire est codée en dur dans la classe principale de l'application, elle doit donc être remplacée. Créez un nouveau fichier `Application.php` . Je préfère garder le mien dans le répertoire `src` (celui que nous remplacerons app with), mais vous pouvez le placer ailleurs.

Voici comment la classe surchargée devrait ressembler. Si vous voulez un nom différent, remplacez simplement la chaîne `src` par autre chose.

```
namespace App;

class Application extends \Illuminate\Foundation\Application
{
    /**
     * @inheritdoc
     */
    public function path($path = '')
    {
        return $this->basePath . DIRECTORY_SEPARATOR . 'src' . ($path ? DIRECTORY_SEPARATOR .
$path : $path);
    }
}
```

Enregistrez le fichier. Nous en avons fini avec ça.

## Appeler la nouvelle classe

Ouvrez `bootstrap/app.php` et localisez

```
$app = new Illuminate\Foundation\Application(
    realpath(__DIR__.'/../')
);
```

Nous allons le remplacer par ceci

```
$app = new App\Application(
```

```
    realpath(__DIR__.'/../')
);
```

## Compositeur

Ouvrez votre fichier `composer.json` et modifiez le chargement automatique en fonction de votre nouvel emplacement

```
"psr-4": {
    "App\\": "src/"
}
```

Et enfin, dans la ligne de commande, exécutez `composer dump-autoload` et votre application devrait être diffusée depuis le répertoire `src` .

### Changer le répertoire des contrôleurs

si nous voulons changer le répertoire des `Controllers` , nous avons besoin de:

1. Déplacez et / ou renommez le répertoire des `Controllers` par défaut où vous le souhaitez. Par exemple de l' `app/Http/Controllers` à l' `app/Controllers`
2. Mettez à jour tous les espaces de noms des fichiers dans le dossier `Controllers` , en les faisant adhérer au nouveau chemin, en respectant les spécificités du PSR-4.
3. Modifiez l'espace de noms appliqué au fichier `routes.php` en modifiant `app\Providers\RouteServiceProvider.php` et modifiez-le:

```
protected $namespace = 'App\Http\Controllers';
```

pour ça:

```
protected $namespace = 'App\Controllers';
```

Lire Structure du répertoire en ligne: https://riptutorial.com/fr/laravel/topic/3153/structure-du-repertoire

# Chapitre 62: Supprimer public de l'URL dans laravel

## Introduction

Comment supprimer `public` d'URL dans Laravel, il y a beaucoup de réponses sur internet mais le plus simple est décrit ci-dessous

## Examples

### Comment faire ça?

Suivez ces étapes pour supprimer `public` de l'URL

1. Copiez le fichier `.htaccess` depuis `/public` répertoire `/public` vers le dossier racine `Laravel/project` .
2. Renommez le `server.php` dans le dossier racine `Laravel/project` en `index.php` .

Bravo, vous serez bien maintenant.

S'il vous plaît noter: Il est testé sur *Laravel 4.2* , *Laravel 5.1* , *Laravel 5.2* , *Laravel 5.3* .

Je pense que c'est le moyen le plus simple de supprimer `public` de l'URL.

### Supprimer le public de l'URL

1. Renommer le `server.php` en `index.php`
2. Copiez le `.htaccess` du dossier `public` vers `root` dossier `root`
3. Changer un peu `.htaccess` comme suit pour la statique:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)/$ /$1 [L,R=301]

RewriteCond %{REQUEST_URI} !(\.css|\.js|\.png|\.jpg|\.gif|robots\.txt)$ [NC]
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !^/public/
RewriteRule ^(css|js|images)/(.*)$ public/$1/$2 [L,NC]
```

Parfois, j'utilise cette méthode pour supprimer `public` url de forme `public` .

Lire Supprimer public de l'URL dans laravel en ligne:

https://riptutorial.com/fr/laravel/topic/9786/supprimer-public-de-l-url-dans-laravel

# Chapitre 63: utiliser les alias de champs dans Eloquent

Lire utiliser les alias de champs dans Eloquent en ligne:
https://riptutorial.com/fr/laravel/topic/7927/utiliser-les-alias-de-champs-dans-eloquent

# Chapitre 64: Valet

## Introduction

Valet est un environnement de développement sur mesure pour macOS. Cela élimine le besoin de machines virtuelles, Homestead ou Vagrant. Plus besoin de mettre à jour constamment votre fichier `/etc/hosts` . Vous pouvez même partager vos sites publiquement en utilisant des tunnels locaux.

Laravel Valet rend tous les sites disponibles sur un domaine `*.dev` en liant les noms de dossier aux noms de domaine.

## Syntaxe

- commande valet [options] [arguments]

## Paramètres

| Paramètre | Ensemble de valeurs |
|---|---|
| commander | domaine , fetch-share-url, oublier, aide, installer, lien , liens , liste, journaux, on-latest-version, ouvrir, parc , chemins, redémarrer, sécuriser, démarrer, arrêter, désinstaller, dissocier, non sécurisé, qui |
| options | -h, --help, -q, --quiet, -V, --version, --ansi, --no-ansi, -n, --no-interaction, -v, -vv, -vvv, - -verbeux |
| arguments | *(optionnel)* |

## Remarques

Comme Valet pour Linux et Windows ne sont pas officiels, il n'y aura pas de support en dehors de leurs référentiels Github respectifs.

## Examples

### Lien de valet

Cette commande est utile si vous souhaitez diffuser un seul site dans un répertoire et non l'intégralité du répertoire.

```
cd ~/Projects/my-blog/
valet link awesome-blog
```

Valet créera un lien symbolique dans `~/.valet/Sites` qui pointe vers votre répertoire de travail actuel.
Après avoir exécuté la commande link, vous pouvez accéder au site dans votre navigateur à l' `http://awesome-blog.dev` .

Pour afficher une liste de tous vos répertoires liés, exécutez la commande valet links . Vous pouvez utiliser `valet unlink awesome-blog` pour détruire le lien symbolique.

## Parc de valet

```
cd ~/Projects
valet park
```

Cette commande enregistre votre répertoire de travail actuel en tant que chemin que Valet doit rechercher pour les sites. Désormais, tout projet Laravel que vous créez dans votre répertoire "parqué" sera automatiquement servi à l'aide de la convention `http://folder-name.dev` .

## Liens de valet

Cette commande affiche tous les liens Valet enregistrés que vous avez créés et leurs chemins d'accès correspondants sur votre ordinateur.

**Commander:**

```
valet links
```

**Échantillon sortie:**

```
...
site1 -> /path/to/site/one
site2 -> /path/to/site/two
...
```

**Remarque 1:** Vous pouvez exécuter cette commande de n’importe où, et pas seulement depuis un dossier lié.

**Note 2: Les** sites seront listés sans la terminaison **.dev** mais vous utiliserez toujours **site1.dev** pour accéder à votre application depuis le navigateur.

## Installation

***IMPORTANT!!* Valet est un outil conçu uniquement pour macOS.**

**Conditions préalables**

- Valet utilise le port HTTP de votre machine locale (port 80). Vous ne pourrez donc pas l'utiliser si *Apache* ou *Nginx* sont installés et s'exécutent sur le même ordinateur.
- Le gestionnaire de paquets non officiel de macOS, Homebrew, est requis pour utiliser

correctement Valet.
- Assurez-vous que Homebrew est mis à jour vers la dernière version en exécutant une `brew update` à `brew update` dans le terminal.

**Installation**

- Installez PHP 7.1 en utilisant Homebrew via `brew install homebrew/php/php71` .
- Installer Valet avec Composer via un `composer global require laravel/valet` .
- Ajoutez le `~/.composer/vendor/bin` au "PATH" de votre système s'il n'y est pas déjà.
- Exécutez la commande d' `valet install` .

**Post-installation** Pendant le processus d'installation, Valet a installé *DnsMasq* . Il a également enregistré le démon de Valet pour se lancer automatiquement au démarrage de votre système, vous n'avez donc pas besoin de lancer `valet start` ou `valet install` chaque redémarrage de votre machine.

## Domaine de valet

Cette commande vous permet de modifier ou d'afficher le TLD *(domaine de premier niveau)* utilisé pour lier des domaines à votre ordinateur local.

**Obtenir le TLD actuel**

```
$ valet domain
> dev
```

**Définir le TLD**

```
$ valet domain local
> Your Valet domain has been updated to [local].
```

## Installation (Linux)

***IMPORTANT!! Valet est un outil conçu pour macOS, la version ci-dessous est portée pour Linux.***

**Conditions préalables**

- **N'installez pas** valet en tant que `root` ou en utilisant la commande `sudo` .
- Valet utilise le port HTTP de votre machine locale (port 80). Vous ne pourrez donc pas l'utiliser si Apache ou Nginx sont installés et s'exécutent sur le même ordinateur.
- Une version à jour du `composer` est requise pour installer et exécuter Valet.

**Installation**

- Exécutez le `composer global require cpriego/valet-linux` installe globalement Valet.
- Exécutez la commande `valet install` pour terminer l'installation.

**Post Installer**

Au cours du processus d'installation, Valet a installé DnsMasq. Il a également enregistré le démon de Valet pour se lancer automatiquement au démarrage de votre système, vous n'avez donc pas besoin de lancer `valet start` ou `valet install` chaque redémarrage de votre machine.

La documentation officielle est disponible ici .

Lire Valet en ligne: https://riptutorial.com/fr/laravel/topic/1906/valet

# Chapitre 65: Validation

## Paramètres

| Paramètre | Détails |
|---|---|
| Champs obligatoires | Ce champ est requis |
| parfois | Exécuter des contrôles de validation sur un champ uniquement si ce champ est présent dans le tableau d'entrée |
| email | L'entrée est un email valide |
| Valeur max | La valeur d'entrée doit être inférieure à la valeur maximale |
| unique: nom_table_base | La valeur d'entrée doit être unique dans le nom de la table de base de données fournie |
| accepté | Oui / Oui / 1 vrai, utile pour vérifier les TOS |
| active_url | Doit être une URL valide selon checkdnsrr |
| après *: date* | Le champ en cours de validation doit fournir une valeur après la date donnée |
| alpha | Le champ en cours de validation doit être entièrement alphabétique. |
| alpha_dash | Le champ en cours de validation peut comporter des caractères alphanumériques, ainsi que des tirets et des traits de soulignement. |
| alpha_num | Le champ en cours de validation doit être entièrement composé de caractères alphanumériques. |
| tableau | Doit être un tableau PHP |
| avant *: date* | Le champ doit être une valeur sous la date donnée |
| entre: min, max | La valeur d'entrée doit être comprise entre les valeurs minimum (min) et maximum (max) |
| booléen | Le champ en cours de validation doit pouvoir être converti en booléen. Les entrées acceptées sont `true` , `false` , `1` , `0` , `"1"` et `"0"` . |
| confirmé | Le champ en cours de validation doit avoir un champ correspondant de `foo_confirmation` . Par exemple, si le champ en cours de validation est un `password` , un champ `password_confirmation` correspondant doit être présent dans l'entrée. |

| Paramètre | Détails |
| --- | --- |
| rendez-vous amoureux | Le champ en cours de validation doit être une date valide selon la fonction PHP strtotime . |
| entier | Le champ en cours de validation doit être un entier |
| chaîne | Le champ en cours de validation doit être un type de chaîne . |

# Examples

## Exemple de base

Vous pouvez valider les données de demande à l'aide de la méthode `validate` (disponible dans le contrôleur de base, fournie par le trait `ValidatesRequests` ).

Si les règles passent, votre code continuera à s'exécuter normalement; cependant, si la validation échoue, une réponse d'erreur contenant les erreurs de validation sera automatiquement renvoyée:

- pour les demandes de formulaire HTML typiques, l'utilisateur sera redirigé vers la page précédente, le formulaire conservant les valeurs soumises
- pour les requêtes qui attendent une réponse JSON, une réponse HTTP avec le code 422 sera générée

Par exemple, dans votre `UserController` , vous pouvez enregistrer un nouvel utilisateur dans la méthode `store` , ce qui nécessite une validation avant l'enregistrement.

```
/**
 * @param  Request  $request
 * @return Response
 */
public function store(Request $request) {
    $this->validate($request, [
        'name' => 'required',
        'email' => 'email|unique:users|max:255'
    ],
    // second array of validation messages can be passed here
    [
        'name.required' => 'Please provide a valid name!',
        'email.required' => 'Please provide a valid email!',
    ]);

    // The validation passed
}
```

Dans l'exemple ci-dessus, nous validons que le champ de `name` existe avec une valeur non vide. Deuxièmement, nous vérifions que le champ de `email` a un format de courrier électronique valide, est unique dans la table de base de données "users" et a une longueur maximale de 255 caractères.

> Le `|` Le caractère (pipe) combine différentes règles de validation pour un champ.

Parfois, vous souhaiterez peut-être arrêter l'exécution des règles de validation sur un attribut après le premier échec de validation. Pour ce faire, assignez la règle de `bail` à l'attribut:

```
$this->validate($request, [
    'name' => 'bail|required',
    'email' => 'email|unique:users|max:255'
]);
```

La liste complète des règles de validation disponibles se trouve dans la section des paramètres ci-dessous .

## Validation de tableau

La validation des champs d'entrée de formulaire de tableau est très simple.

Supposons que vous deviez valider chaque nom, email et nom de père dans un tableau donné. Vous pouvez faire ce qui suit:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required'
]);

if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

Laravel affiche les messages par défaut pour la validation. Toutefois, si vous souhaitez des messages personnalisés pour les champs basés sur des tableaux, vous pouvez ajouter le code suivant:

```
[
    'name.*' => [
        'required' => 'Name field is required',
    ],
    'email.*' => [
        'unique'   => 'Unique Email is required',
    ],
    'fatherName.*' => [
        'required' => 'Father Name required',
    ]
]
```

Votre code final ressemblera à ceci:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required',
], [
    'name.*'       => 'Name Required',
    'email.*'      => 'Unique Email is required',
```

```
    'fatherName.*' => 'Father Name required',
]);

if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

## Autres approches de validation

### 1) Validation de la demande de formulaire

Vous pouvez créer une "demande de formulaire" pouvant contenir la logique d'autorisation, les règles de validation et les messages d'erreur pour une demande particulière dans votre application.

La commande CLI de `make:request` Artisan génère la classe et la place dans le répertoire `app/Http/Requests` :

```
php artisan make:request StoreBlogPostRequest
```

La méthode `authorize` peut être remplacée par la logique d'autorisation pour cette requête:

```
public function authorize()
{
    return $this->user()->can('post');
}
```

La méthode `rules` peut être remplacée par les règles spécifiques à cette requête:

```
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

La méthode des `messages` peut être remplacée par les messages spécifiques à cette requête:

```
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'title.unique' => 'There is another post with the same title',
        'title.max' => 'The title may not exceed :max characters',
        'body.required' => 'A message is required',
    ];
}
```

Pour valider la requête, il suffit d'indiquer la classe de requête spécifique sur la méthode de contrôleur correspondante. Si la validation échoue, une réponse d'erreur sera renvoyée.

```
public function store(StoreBlogPostRequest $request)
{
    // validation passed
}
```

---

### 2) Création manuelle de validateurs

Pour plus de flexibilité, vous pouvez créer un validateur manuellement et gérer directement la validation échouée:

```
<?php
namespace App\Http\Controllers;

use Validator;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
        ]);

        if ($validator->fails()) {
            return redirect('post/create')
                    ->withErrors($validator)
                    ->withInput();
        }

        // Store the blog post...
    }
}
```

### 2) Création fluide de règles

De temps en temps, vous pourriez avoir besoin de créer des règles uniques à la volée, travailler avec la méthode `boot()` dans un fournisseur de services peut être exagéré, à partir de Laravel 5.4, vous pouvez créer de nouvelles règles avec la classe `Rule` .

Par exemple, nous allons travailler avec `UserRequest` pour savoir quand vous voulez insérer ou mettre à jour un utilisateur. Pour l'instant, nous voulons un nom et l'adresse e-mail doit être unique. Le problème avec l’utilisation de la règle `unique` est que si vous modifiez un utilisateur, il se peut qu’il conserve le même courrier électronique. Vous devez donc exclure l’utilisateur actuel de la règle. L'exemple suivant montre comment vous pouvez facilement le faire en utilisant la nouvelle classe `Rule` .

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;
```

```
class UserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(Request $request)
    {
        $id = $request->route()->getParameter('user');

        return [
            'name'           =>  'required',

            // Notice the value is an array and not a string like usual
            'email'         =>  [
                'required',
                Rule::unique('users')->ignore($id)
            ]
        ];
    }
}
```

## Classe de demande de formulaire unique pour POST, PUT, PATCH

Suivant l'exemple de 'Validation de demande de formulaire' , la même classe de demande peut être utilisée pour `POST` , `PUT` , `PATCH` ce qui vous `PATCH` de créer une autre classe en utilisant les mêmes validations / similaires. Cela est pratique si vous avez des attributs uniques dans votre table.

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules() {
    switch($this->method()) {
        case 'GET':
        case 'DELETE':
            return [];
        case 'POST':
            return [
                'name'     => 'required|max:75|unique',
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        case 'PUT':
```

```
        case 'PATCH':
            return [
                'name'     => 'required|max:75|unique:product,name,' . $this->product,
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        default:break;
    }
}
```

En partant du haut, notre instruction switch va examiner le type de méthode de la requête ( `GET` , `DELETE` , `POST` , `PUT` , `PATCH` ).

Selon la méthode, renverra le tableau de règles défini. Si vous avez un champ unique, tel que le champ `name` de l'exemple, vous devez spécifier un identifiant particulier pour que la validation soit ignorée.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore`
```

Si vous avez une clé primaire étiquetée autrement que `id` , vous spécifiez la colonne clé primaire comme quatrième paramètre.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore . ',primary_key_column'
```

Dans cet exemple, nous utilisons `PUT` et transmettons à la route ( `admin/products/{product}` ) la valeur de l'ID du produit. Donc, `$this->product` sera égal à l' `id` à ignorer.

Maintenant, vos règles de validation `PUT|PATCH` et `POST` ne doivent pas nécessairement être identiques. Définissez votre logique en fonction de vos besoins. Cette technique vous permet de réutiliser les messages personnalisés que vous avez peut-être définis dans la classe de demande de formulaire personnalisée.

### Messages d'erreur

## Personnalisation des messages d'erreur

Les fichiers `/resources/lang/[lang]/validation.php` contiennent les messages d'erreur à utiliser par le validateur. Vous pouvez les modifier selon vos besoins.

La plupart d'entre eux ont des espaces réservés qui seront automatiquement remplacés lors de la génération du message d'erreur.

Par exemple, dans `'required' => 'The :attribute field is required.'` , l'espace réservé `:attribute` sera remplacé par le nom du champ (vous pouvez également personnaliser la valeur d'affichage de chaque champ du tableau d' `attributes` du même fichier).

*Exemple*

configuration du message:

```
'required' => 'Please inform your :attribute.',
//...
'attributes => [
    'email' => 'E-Mail address'
]
```

règles:

```
`email' => `required`
```

message d'erreur résultant:

> "S'il vous plaît informer votre adresse e-mail."

---

## Personnalisation des messages d'erreur dans une classe Request

La classe Request a accès à une méthode `messages()` qui devrait renvoyer un tableau. Elle peut être utilisée pour remplacer les messages sans avoir à entrer dans les fichiers lang. Par exemple, si nous avons une validation personnalisée `file_exists` vous pouvez envoyer des messages comme ci-dessous.

```
class SampleRequest extends Request {

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'image' =>  'required|file_exists'
        ];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    public function messages()
    {
        return [
            'image.file_exists' =>  'That file no longer exists or is invalid'
        ];
    }
```

```
}
```

# Affichage des messages d'erreur

Les erreurs de validation sont flashées sur la session et sont également disponibles dans la variable `$errors` , qui est automatiquement partagée avec toutes les vues.

Exemple d'affichage des erreurs dans une vue Blade:

```
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

## Règles de validation personnalisées

Si vous souhaitez créer une règle de validation personnalisée, vous pouvez le faire par exemple dans la méthode de `boot` d'un fournisseur de services, via la façade Validator.

```
<?php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Validator;

class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        Validator::extend('starts_with', function($attribute, $value, $parameters, $validator)
{
            return \Illuminate\Support\Str::startsWith($value, $parameters[0]);
        });

        Validator::replacer('starts_with', function($message, $attribute, $rule, $parameters)
{
            return str_replace(':needle', $parameters[0], $message);
        });
    }
}
```

La méthode `extend` prend une chaîne qui sera le nom de la règle et une fonction qui à son tour recevra le nom de l'attribut, la valeur en cours de validation, un tableau des paramètres de la règle et l'instance du validateur, et devrait indiquer si la validation passe. Dans cet exemple, nous vérifions si la chaîne de valeur commence par une sous-chaîne donnée.

Le message d'erreur pour cette règle personnalisée peut être défini comme d'habitude dans le

fichier `/resources/lang/[lang]/validation.php` et peut contenir des espaces réservés, par exemple, pour les valeurs de paramètres:

```
'starts_with' => 'The :attribute must start with :needle.'
```

La méthode `replacer` prend une chaîne qui est le nom de la règle et une fonction qui à son tour recevra le message d'origine (avant remplacement), le nom de l'attribut, le nom de la règle et un tableau des paramètres de la règle. et devrait retourner le message après avoir remplacé les espaces réservés selon les besoins.

Utilisez cette règle comme toute autre:

```
$this->validate($request, [
    'phone_number' => 'required|starts_with:+'
]);
```

Lire Validation en ligne: https://riptutorial.com/fr/laravel/topic/1310/validation

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Commencer avec Laravel | alepeino, Alphonsus, boroboris, Colin Herzog, Community, Ed Rands, Evgeniy Maynagashev, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Kovah, Lance Pioch, Marek Skiba, Martin Bean, Misa Lazovic, nyedidikeke, Oliver Adria, Prakash, rap-2-h, Ru Chern Chong, SeinopSys, Tatranskymedved, Tim |
| 2 | Artisan | Alessandro Bassi, Gaurav, Harshal Limaye, Himanshu Raval, Imam Assidiqqi, Kaspars, Laurel, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, SeinopSys |
| 3 | Authentification | Aykut CAN, Imam Assidiqqi |
| 4 | Autorisation | Daniel Verem |
| 5 | Autorisations pour le stockage | A. Raza |
| 6 | Base de données | A. Raza, adam, caoglish, Ian, Iftikhar uddin, Imam Assidiqqi, liamja, Panagiotis Koursaris, RamenChef, Rubens Mariuzzo, Sanzeeb Aryal, Vucko |
| 7 | Bases Cron | A. Raza |
| 8 | cadre lumen | maksbd19 |
| 9 | Classe CustomException à Laravel | ashish bansal |
| 10 | Classement de base de données | Achraf Khouadja, Andrew Nolan, Dan Johnson, Isma, Kyslik, Marco Aurélio Deleu |
| 11 | Collections | A. Raza, Alessandro Bassi, Alex Harris, bhill77, caoglish, Dummy Code, Gras Double, Ian, Imam Assidiqqi, Josh Rumbut, Karim Geiger, matiaslauriti, Nicklas Kevin Frank, Ozzy, rap-2-h, simonhamp, Vucko |
| 12 | Connexions DB multiples à Laravel | 4444, A. Raza, Rana Ghosh |
| 13 | Contrôleurs | Ru Chern Chong |
| 14 | Courrier | Yohanan Baruchel |

| | | |
|---|---|---|
| 15 | Demande interdomaine | Imam Assidiqqi, Suraj |
| 16 | Demandes | Ian, Jerodev, RamenChef, Rubens Mariuzzo |
| 17 | Démarrer avec laravel-5.3 | A. Raza, Advaith, Community, davejal, Deathstorm, Manish, Matthew Beckman, Robin Dirksen, Shital Jachak |
| 18 | Déployer l'application Laravel 5 sur l'hébergement partagé sur un serveur Linux | Donkarnash, Gayan, Imam Assidiqqi, Kyslik, PassionInfinite, Pete Houston, rap-2-h, Ru Chern Chong, Stojan Kukrika, ultrasamad |
| 19 | Des aides | aimme |
| 20 | Éloquent | aimme, alepeino, Alessandro Bassi, Alex Harris, Alfa, Alphonsus, andretzermias, andrewtweber, Andrey Lutskevich, aynber, Buckwheat, Casper Spruit, Dancia, Dipesh Poudel, Ian, Imam Assidiqqi, James, James, jedrzej.kurylo, John Slegers, Josh Rumbut, Kaspars, Ketan Akbari, KuKeC, littleswany, Lykegenes, Maantje, Mahmood, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, Martin Bean, matiaslauriti, MM2, Nicklas Kevin Frank, Niklas Modess, Nyan Lynn Htut, patricus, Pete Houston, Phroggyy, Prisoner Raju, RamenChef, rap-2-h, Rubens Mariuzzo, Sagar Naliyapara, Samsquanch, Sergio Guillen Mantilla, Tim, tkausl, whoan, Yasin Patel |
| 21 | Eloquent: Accessors & Mutators | Diego Souza, Kyslik |
| 22 | Eloquent: Modèle | Aeolingamenfel, alepeino, Alex Harris, Imam Assidiqqi, John Slegers, Kaspars, littleswany, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, matiaslauriti, Nicklas Kevin Frank, Samsquanch, Tim |
| 23 | Eloquent: Relation | Advaith, aimme, Alex Harris, Alphonsus, bhill77, Imam Assidiqqi , Ketan Akbari, Phroggyy, rap-2-h, Ru Chern Chong, Zulfiqar Tariq |
| 24 | Ensemencement | A. Raza, Alphonsus, Ian, Imam Assidiqqi, Kyslik, SupFrost, whoan |
| 25 | Erreur de correspondance de jeton dans AJAX | Pankaj Makwana |
| 26 | Essai | Alessandro Bassi, Brayniverse, caoglish, Julian Minde, Kyslik, rap-2-h, Sven |

| | | |
|---|---|---|
| 27 | Événements et auditeurs | Bharat Geleda, matiaslauriti, Nauman Zafar |
| 28 | Fonction d'assistance personnalisée | Ian, Luceos, rap-2-h, Raunak Gupta |
| 29 | Forfaits Laravel | Casper Spruit, Imam Assidiqqi, Ketan Akbari, rap-2-h, Ru Chern Chong, Tosho Trajanov |
| 30 | Formulaire de demande (s) | Bookeater, Ian, John Roca, Kyslik, RamenChef |
| 31 | Guide d'installation | Advaith, Amarnasan, aynber, Community, davejal, Dov Benyomin Sohacheski, Imam Assidiqqi, PaladiN, rap-2-h, Ru Chern Chong |
| 32 | HTML et Form Builder | alepeino, Casper Spruit, Himanshu Raval, Prakash |
| 33 | Installation | A. Raza, alepeino, Alphonsus, Black, boroboris, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Lance Pioch, Marek Skiba, Martin Bean, nyedidikeke, PaladiN, Prakash, rap-2-h, Ru Chern Chong, Sagar Naliyapara, SeinopSys, Tim |
| 34 | Intégration de Sparkpost avec Laravel 5.4 | Alvin Chettiar |
| 35 | Introduction au laravel-5.2 | A. Raza, ashish bansal, Community, Edward Palen, Ivanka Todorova, Shubhamoy |
| 36 | Introduction au laravel-5.3 | Ian |
| 37 | La caissière | littleswany, RamenChef |
| 38 | La gestion des erreurs | Isma, Kyslik, RamenChef, Rubens Mariuzzo |
| 39 | Laravel Docker | Dov Benyomin Sohacheski |
| 40 | Le routage | A. Raza, alepeino, Alessandro Bassi, Alex Juchem, beznez, Dwight, Ilker Mutlu, Imam Assidiqqi, jedrzej.kurylo, Kyslik, Milan Maharjan, Rubens Mariuzzo, SeinopSys, Vucko |
| 41 | Les constantes | Mubashar Iqbal, Oscar David, Zakaria Acharki |
| 42 | Les files d'attente | Alessandro Bassi, Kyslik |

| | | |
|---|---|---|
| 43 | Les politiques | Tosho Trajanov |
| 44 | Liaison modèle de route | A. Raza, GiuServ, Vikash |
| 45 | Liens utiles | Jakub Kratina |
| 46 | Macros dans une relation éloquente | Alex Casajuana, Vikash |
| 47 | Middleware | Alex Harris, Kaspars, Kyslik, Moppo, Pistachio |
| 48 | Migrations de base de données | Chris, Chris White, Hovsep, hschin, Iftikhar uddin, Imam Assidiqqi, Kaspars, liamja, littleswany, mnoronha, Nauman Zafar, Panagiotis Koursaris, Paulo Freitas, Vucko |
| 49 | Modèles de lames | A. Raza, agleis, Akshay Khale, alepeino, Alessandro Bassi, Benubird, cbaconnier, Christophvh, Imam Assidiqqi, matiaslauriti, Nauman Zafar, rap-2-h, Safoor Safdar, Tosho Trajanov, yogesh |
| 50 | Modifier le comportement de routage par défaut dans Laravel 5.2.31 + | Frank Provost |
| 51 | Nommer des fichiers lors du téléchargement avec Laravel sur Windows | Donkarnash, RamenChef |
| 52 | Observateur | matiaslauriti, Szenis |
| 53 | Pagination | Himanshu Raval, Iftikhar uddin |
| 54 | Planification des tâches | Jonathon |
| 55 | Prestations de service | A. Raza, El_Matella |
| 56 | Problèmes courants et solutions rapides | Nauman Zafar |
| 57 | Socialite | Jonathon, Marco Aurélio Deleu |
| 58 | Stockage de système de fichiers / cloud | Imam Assidiqqi, Nitish Kumar, Paulo Laxamana |

| | | |
|---|---|---|
| 59 | Structure du répertoire | Kaspars, Moppo, RamenChef |
| 60 | Supprimer public de l'URL dans laravel | A. Raza, Rana Ghosh, ultrasamad |
| 61 | utiliser les alias de champs dans Eloquent | MM2 |
| 62 | Valet | David Lartey, Dov Benyomin Sohacheski, Imam Assidiqqi, Misa Lazovic, Ru Chern Chong, Shog9 |
| 63 | Validation | A. Raza, alepeino, Alessandro Bassi, Alex Harris, Andrew Nolan , happyhardik, Himanshu Raval, Ian, Iftikhar uddin, John Slegers, Marco Aurélio Deleu, matiaslauriti, rap-2-h, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, Stephen Leppik, sun, Vucko |