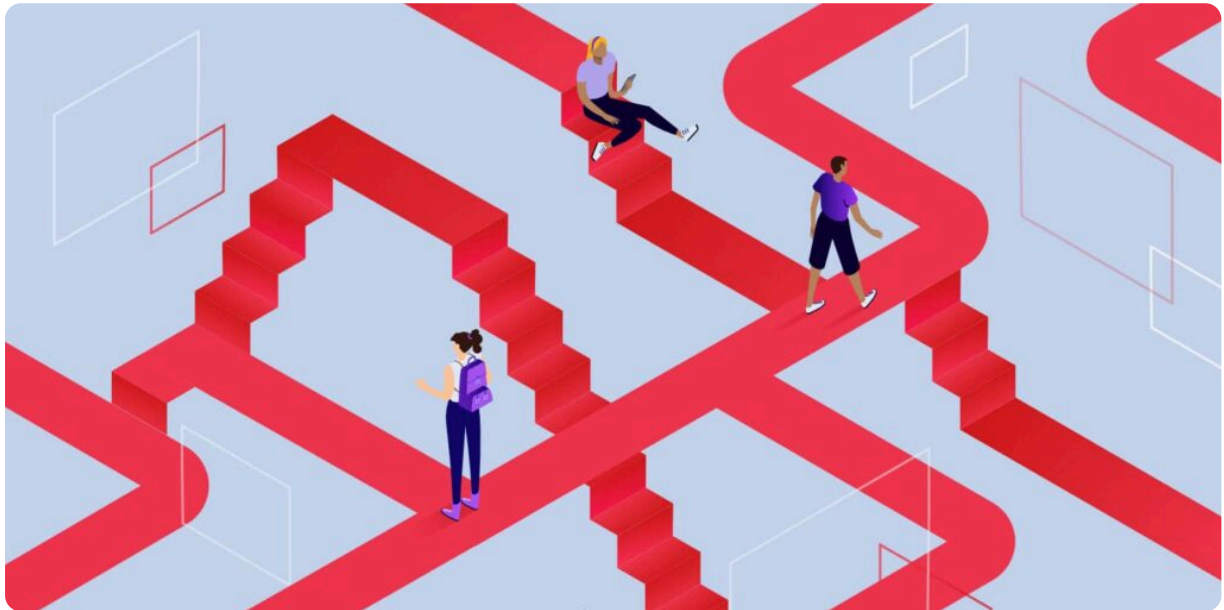


Maîtriser les routes de Laravel

Coman Cosmin | Publié: 16 décembre 2022 | Mis à jour: 20 décembre 2022



Lorsqu'il s'agit du backend, les développeurs rencontrent éventuellement des routes. Les routes peuvent être considérées comme l'épine dorsale du backend puisque chaque requête que le serveur reçoit est redirigée vers un contrôleur par le biais d'une liste de routage qui fait correspondre les demandes aux contrôleurs ou aux actions.

[Laravel](#) nous cache de nombreux détails d'implémentation beaucoup de sucre syntaxique pour aider les développeurs expérimentés, à développer leurs applications web. Voyons de plus près comment gérer les routes dans

Ask us about making the switch to simpler, faster, and more reliable WordPress hosting. Start a live chat now!

Table des matières

[Backend Routing et Cross-Site Scripting dans Laravel](#)

[Routage de base de Laravel](#)

[Paramètres des routes](#)

[Routes nommées](#)

[Groupes de routes](#)

[Mise en cache des routes](#)

Backend Routing et Cross-Site Scripting dans Laravel

Sur un serveur, il existe des routes publiques et privées. Les routes publiques peuvent être une source d'inquiétude en raison de la possibilité de cross-site scripting (XSS), un type d'[attaque par injection](#) qui peut vous rendre, vous et vos utilisateurs, [vulnérables aux acteurs malveillants](#).

Le problème est qu'un utilisateur peut être redirigé d'une route qui ne nécessite pas de jeton de session vers une route qui en nécessite un – et il aura toujours accès sans le jeton.

La façon la plus simple de résoudre ce problème est d'appliquer un nouvel en-tête HTTP, en ajoutant « referrer » à l'itinéraire pour atténuer ce scénario :


```
'main' => [  
    'path' => '/main',  
    'referrer' => 'required,refresh-empty',  
    'target' => ControllerDashboardController::class . '::mainAction'  
]
```

Routage de base de Laravel

Dans Laravel, les routes permettent aux utilisateurs d'acheminer la requête appropriée vers le contrôleur souhaité. La route Laravel la plus basique accepte un Uniform Asset Identifier (votre chemin de route) et une fermeture qui peut être à la fois une fonction ou une classe.

Dans Laravel, les routes sont créées dans les fichiers **web.php** et **api.php**. Laravel est livré avec deux routes par défaut : une pour le WEB et une pour l'API.

Ces routes résident dans le dossier **routes/**, mais elles sont chargées dans le fichier **Providers/RouteServiceProvider.php**.



```
public function boot()  
{  
    $this->configureRateLimiting();  
  
    $this->routes(function () {  
        Route::middleware('api')  
            ->prefix('api')  
            ->group(base_path('routes/api.php'));  
  
        Route::middleware('web')  
            ->group(base_path('routes/web.php'));  
    });  
}
```

– État par défaut du fournisseur de services de routes de Laravel.

Au lieu de procéder ainsi, nous pouvons charger les routes directement dans **RouteServiceProvider.php**, en sautant complètement le dossier **routes/**.

```
// 'as' helps with the naming and matching of routes
// 'prefix' will be applied to all routes in the group
$this->group(["prefix" => "/api/v1", "as" => "api."], function() {

    // all routes under this group will have the authenticate middleware
    $this->group(["middleware" => ['authenticate']], function() {

        // This route will have the name api.country-name
        $this->get('/country-name', GetCountryNameController::class)->name("country-name");
    });
});

$this->group(["prefix" => "/", "as" => "web."], function () {
    $this->get("homepage", GetHomepageController::class)->name("homepage");
});
```

– Chargement des routes Laravel directement dans le fournisseur.

Redirections

Lorsque nous définissons une route, nous souhaitons généralement rediriger l'utilisateur qui y accède, et les raisons en sont très variées. Cela peut être parce qu'il s'agit d'une route obsolète et que nous avons changé le backend ou le serveur, ou parce que nous voulons installer l'authentification à deux facteurs (2FA), etc.

Laravel dispose d'un moyen facile de le faire. Grâce à la simplicité du framework, nous pouvons utiliser la méthode de redirection sur la façade Route, qui accepte la route d'entrée et la route vers laquelle être redirigé.

En option, nous pouvons donner le code d'état pour la redirection comme troisième paramètre. La méthode `permanentRedirect` fera la même chose que la méthode `redirect`, sauf qu'elle renverra toujours un code d'état 301:

```
// Simple redirect
Route::redirect("/class", "/myClass");

// Redirect with custom status
Route::redirect("/home", "/office", 305);

// Route redirect with 301 status code
Route::permanentRedirect("/home", "office");
```

A l'intérieur des routes de redirection, il nous est interdit d'utiliser les mots-clés « destination » et « status » comme paramètres car ils sont réservés par Laravel.

```
// Illegal to use
Route::redirect("/home", "/office/{status}");
```

Vues

Les vues sont les fichiers **.blade.php** que nous utilisons pour rendre le frontend de notre application Laravel. Elles utilisent le moteur de templating blade, et c'est la manière par défaut de construire une application full-stack en utilisant uniquement Laravel.

Si nous voulons que notre route renvoie une vue, nous pouvons simplement utiliser la méthode `view` sur la façade `Route`. Elle accepte un paramètre de route, un nom de vue et un tableau facultatif de valeurs à transmettre à la vue.

```
// When the user accesses my-domain.com/homepage
// the homepage.blade.php file will be rendered
Route::view("/homepage", "homepage");
```

Supposons que notre vue veuille dire « Hello, {name} » en passant un tableau optionnel avec ce paramètre. Nous pouvons le faire avec le code suivant (si le paramètre manquant est requis dans la vue, la requête échouera et lancera une erreur) :

```
Route::view('/homepage', 'homepage', ['name' => "Kinsta"]);
```

Liste de routes

Au fur et à mesure que la taille de votre application augmente, le nombre de requêtes qui doivent être acheminées augmente également. Et avec un grand volume d'informations peut venir une grande confusion.

C'est là que le site `artisan route:list` `command` peut nous aider. Il fournit une vue d'ensemble de toutes les routes qui sont définies dans l'application, leurs middlewares et leurs contrôleurs.

```
php artisan route:list
```

Il affichera une liste de toutes les routes sans les middlewares. Pour cela, nous devons utiliser le drapeau `-v` :

```
php artisan route:list -v
```

Dans une situation où vous utilisez une conception pilotée par le domaine où vos routes ont des noms spécifiques dans leurs chemins, vous pouvez utiliser les capacités de filtrage de cette commande comme suit :

```
php artisan route:list -path=api/account
```

Ceci montrera seulement les routes qui commencent par **api/account**.

D'autre part, nous pouvons demander à Laravel d'exclure ou d'inclure des routes définies par des tiers en utilisant les options `-except-vendor` ou `-only-vendor` .

Paramètres des routes

Parfois, vous pouvez avoir besoin de capturer des segments de l'URI avec la route, comme un ID utilisateur ou un jeton. Nous pouvons le faire en définissant un paramètre de route, qui est toujours encadré par des accolades (`{ }`) et ne doit comporter que des caractères alphabétiques.

Si nos routes ont des dépendances dans leurs callbacks, le conteneur de services Laravel les injectera automatiquement :

```
use IlluminateHttpRequest;
use Controllers/DashboardController;
Route::post('/dashboard/{id}', function (Request $request, string $id) {
    return 'User:' . $id;
})
Route::get('/dashboard/{id}', DashboardController.php);
```

Paramètres nécessaires

Les paramètres nécessaires de Laravel sont des paramètres dans les routes que nous ne sommes pas autorisés à ignorer lorsque nous effectuons un appel. Sinon, une erreur sera déclenchée :

```
Route::post("/gdpr/{userId}", GetGdprDataController.php);
```

Maintenant, dans le **GetGdprDataController.php**, nous aurons un accès direct au paramètre **\$userId**.

```
public function __invoke(int $userId) {
    // Use the userId that we received...
```

```
}
```

Une route peut prendre un nombre quelconque de paramètres. Ils sont injectés dans les callbacks/contrôleurs de la route en fonction de l'ordre dans lequel ils sont listés :

```
// api.php
Route::post('/gdpr/{userId}/{userName}/{userAge}', GetGdprDataController
// GetGdprDataController.php
public function __invoke(int $userId, string $userName, int $userAge) {
    // Use the parameters...
}
```



Paramètres facultatifs

Dans une situation où nous voulons faire quelque chose sur une route lorsque seul un paramètre est présent et rien d'autre, le tout sans affecter l'application entière, nous pouvons ajouter un paramètre optionnel. Ces paramètres facultatifs sont signalés par l'adresse `?` qui leur est accolée :

```
Route::get('/user/{age?}', function (int $age = null) {
    if (!$age) Log::info("User doesn't have age set");
    else Log::info("User's age is " . $age);
})
Route::get('/user/{name?}', function (int $name = "John Doe") {
    Log::info("User's name is " . $name);
})
```

Route Wildcard

[Laravel](#) nous fournit un moyen de filtrer ce à quoi doivent ressembler nos paramètres facultatifs ou nécessaires.

Disons que nous voulons une chaîne de caractères d'un ID utilisateur. Nous pouvons le valider ainsi au niveau de la route en utilisant la méthode `where` .

La méthode `where` accepte le nom du paramètre et la règle regex qui sera appliquée à la validation. Par défaut, elle prend le premier paramètre, mais si nous en avons plusieurs, nous pouvons passer un tableau avec le nom du paramètre comme clé et la règle comme valeur, et Laravel les analysera tous pour nous :

```
Route::get('/user/{age}', function (int $age) {
    //
}->where('age', '[0-9]+');
Route::get('/user/{age}', function (int $age) {
    //
}->where('[0-9]+');
Route::get('/user/{age}/{name}', function (int $age, string $name) {
    //
}->where(['age' => '[0-9]+', 'name' => '[a-z][A-z]+');
```

Nous pouvons aller plus loin et appliquer la validation à toutes les routes de notre application en utilisant la méthode `pattern` sur la façade `Route` :

```
Route::pattern('id', '[0-9]+');
```

Cela validera chaque paramètre de `id` avec cette expression regex. Et une fois que nous l'avons définie, elle sera automatiquement appliquée à toutes les routes utilisant ce nom de paramètre.

Comme nous pouvons le voir, Laravel utilise le caractère `/` comme séparateur dans le chemin. Si nous voulons l'utiliser dans le chemin, nous devons l'autoriser explicitement à faire partie de notre espace réservé en utilisant une regex `where` .

```
Route::get('/find/{query}', function ($query) {  
    //  
})->where('query', , '.*');
```

Le seul inconvénient est qu'il ne sera pris en charge que dans le dernier segment du chemin.

Routes nommées

Comme son nom l'indique, nous pouvons nommer les routes, ce qui permet de générer des URL ou des redirections pour des routes spécifiques.

Comment créer des routes nommées

Une façon simple de créer une route nommée est fournie par la méthode `name` enchaînée sur la façade `Route`. Le nom de chaque route doit être unique :

```
Route::get('/', function () {  
})->name("homepage");
```

Groupe de routes

Les groupes de routes vous permettent de partager des attributs de route comme les intergiciels sur un grand nombre de routes sans avoir à les redéfinir sur chaque route.

Middleware

L'attribution d'un intergiciel à toutes les routes que nous avons nous permet de les combiner dans un groupe, d'abord en utilisant la méthode `group`.

Une chose à considérer est que les intergiciels sont exécutés dans l'ordre dans lequel ils sont appliqués au groupe :

```
Route::middleware(['AuthMiddleware', 'SessionMiddleware'])->group(function() {  
    Route::get('/', function() {} );  
    Route::post('/upload-picture', function () {} );  
});
```



Contrôleurs

Lorsqu'un groupe utilise le même contrôleur, nous pouvons utiliser la méthode `controller` pour définir le contrôleur commun à toutes les routes de ce groupe. Nous devons maintenant spécifier la méthode que l'itinéraire appellera.

```
Route::controller(UserController::class)->group(function () {  
    Route::get('/orders/{userId}', 'getOrders');  
    Route::post('/order/{id}', 'postOrder');  
});
```

Routage de sous-domaines

Un nom de sous-domaine est un élément d'information supplémentaire ajouté au début du nom de domaine d'un site web. Cela permet aux sites web de séparer et d'organiser leur contenu pour des fonctions spécifiques, comme les boutiques en ligne, les blogs, les présentations, etc. du reste du site web.

Nos routes peuvent être utilisées pour gérer le routage des sous-domaines. Nous pouvons attraper le domaine et une partie du sous-domaine pour les utiliser dans notre contrôleur et notre route. Avec l'aide de la méthode

`domain` sur la façade `Route` , nous pouvons regrouper nos routes sous un seul domaine :

```
Route::domain('{store}.enterprise.com')->group(function() {  
    Route::get('order/{id}', function (Account $account, string $id) {  
        // Your Code  
    })  
});
```

Préfixes et préfixes de nom

Lorsque nous avons un groupe de routes, au lieu de les modifier une par une, nous pouvons utiliser les utilitaires supplémentaires que Laravel fournit, tels que `prefix` et `name` sur la façade `Route` .

La méthode `prefix` peut être utilisée pour préfixer chaque route du groupe avec un URI donné, et la méthode `name` peut être utilisée pour préfixer chaque nom de route avec une chaîne donnée.

Cela nous permet de créer de nouvelles choses comme des routes d'administration sans avoir à modifier chaque nom ou préfixe pour les identifier :

```
Route::name('admin.')->group(function() {  
    Route::prefix("admin")->group(function() {  
        Route::get('/get')->name('get');  
        Route::put('/put')->name('put');  
        Route::post('/post')->name('post');  
    });  
});
```

Maintenant, les URI de ces routes seront `admin/get` , `admin/put` , `admin/post` , et les noms `admin.get` , `admin.put` , et `admin.post` .

Mise en cache des routes

Lors du déploiement de l'application sur les serveurs de production, un bon [développeur Laravel](#) tirera parti du cache des routes de Laravel.

Qu'est-ce que la mise en cache des routes ?

La mise en cache des routes diminue le temps nécessaire à l'enregistrement de toutes les routes de l'application.

En exécutant `php artisan route:cache` une instance de `Illuminate/Routing/RouteCollection` est générée, et après avoir été encodée, la sortie sérialisée est écrite dans `bootstrap/cache/routes.php`.

Désormais, toute autre requête [chargera ce fichier de cache](#) s'il existe. Par conséquent, notre application ne doit plus analyser et convertir les entrées du fichier de route en objets `Illuminate/Routing/Route` dans `Illuminate/Routing/RouteCollection`.

Pourquoi il est important d'utiliser la mise en cache des routes

En n'utilisant pas la fonction de mise en cache des routes fournie par Laravel, votre application risque de fonctionner plus lentement qu'elle ne le pourrait, ce qui pourrait à son tour diminuer les ventes, la rétention des utilisateurs et la [confiance dans votre marque](#).

En fonction de l'échelle de votre projet et du nombre de routes, l'exécution d'une simple commande de mise en cache des routes peut [accélérer votre application](#) de 130 % à 500 % - un gain considérable pour un effort quasi-nul.

Résumé

Le routage est l'épine dorsale du développement backend. Le framework Laravel excelle dans ce domaine en fournissant une manière verbeuse de définir et de gérer les routes.

Le développement peut en effet être accessible à tous et contribuer à [accélérer une application](#) du simple fait qu'elle est construite en Laravel.

Quels autres trucs et astuces avez-vous rencontrés concernant les routes Laravel ? Faites-nous en part dans la section des commentaires !

Obtenez toutes vos [applications](#), [bases de données](#) et [Sites WordPress](#) en ligne et sous un même toit. Notre plateforme cloud haute performance et pleine de fonctionnalités comprend :

- Configuration et gestion faciles dans le tableau de bord MyKinsta
- Support expert 24/7
- Le meilleur matériel et réseau de la plateforme Google Cloud, propulsé par Kubernetes pour une évolutivité maximale
- Une intégration Cloudflare au niveau de l'entreprise pour la vitesse et la sécurité
- Une audience mondiale avec jusqu'à 37 centres de données et 260 PoP dans le monde

Obtenez un essai gratuit de notre [Hébergement d'application](#) ou [Hébergement de base de données](#). Explorez nos [plans](#) ou [contactez nous](#) pour trouver ce qui vous convient le mieux.



Coman Cosmin

Cosmin Coman is a technology writer and developer with over 3 years of experience. Apart from writing for Kinsta, he has assisted in research at nuclear physics facilities and universities. Tech-savvy and integrated into the community, he always comes up with innovative solutions.

Articles similaires et sujets

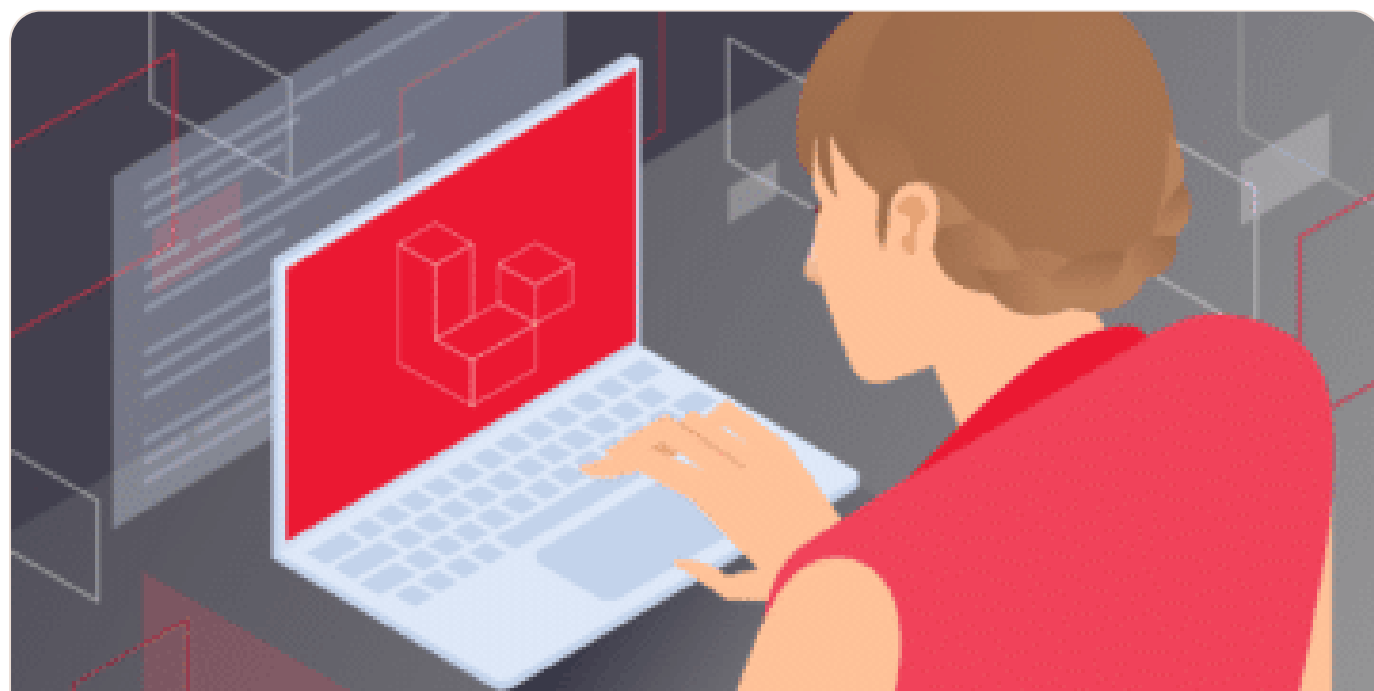


Hébergement WordPress infogéré puissant

Débloquez un hébergement WordPress infogéré de qualité supérieure avec Kinsta. Des fonctionnalités de premier plan pour des performances de site in...

12 min de lecture · 5 juin 2024 · Page · Vidéo

[En savoir plus](#)



Vous voulez devenir un développeur Laravel ? Voici tout ce que vous devez savoir

Vous souhaitez devenir un développeur Laravel ? Voici un guide qui vous montre par où commencer et ce que vous devez accomplir.

20 min de lecture · 15 septembre 2023 · Blog · Laravel

PHP

Développement web

Base de données

Commentaires

Laissez un commentaire

Laisser un commentaire

Politique des commentaires : nous aimons les commentaires et apprécions le temps que les lecteurs passent pour partager des idées et donner des commentaires. Cependant, tous les commentaires sont modérés manuellement et ceux réputés pour être du spam ou uniquement promotionnels seront effacés.

Commentaire

Nom

Email

En envoyant ce formulaire : Vous acceptez le traitement des données personnelles soumises conformément à la [Politique de Confidentialité](#) de Kinsta, y compris le transfert de données vers les États-Unis.

- ☒ Vous acceptez également de recevoir des informations de Kinsta relatives à nos services, événements et promotions. Vous pouvez vous désabonner à tout moment en suivant les instructions figurant dans les communications reçues.

Laisser un commentaire

Produits

Hébergement WordPress infogéré

Hébergement d'application web

Hébergement de base de données infogéré

Hébergement de site statique

Tarifs

Faits marquants

Intégration Cloudflare

API Kinsta

Support expert

Migrations gratuites

Outil APM

DevKinsta

Cache Edge

Plans de modules

Cas d'utilisation

Entreprises

Agences

Boutiques WooCommerce

Petites entreprises

Organisations caritatives

Sites uniques critiques

Études de cas

Ressources

Documentation

Journal des changements

Blog

Newsletter

Base de connaissances

Outils de développement

Kinsta et la concurrence

Annuaire d'agences

État du système

Toutes les ressources

Entreprise

À propos de nous

Pourquoi choisir Kinsta

Carrières

Partenaires

Programme d'affiliation

Presse

Sécurité et confiance

Nous contacter

Nous prenons la sécurité et la protection de la vie privée au sérieux

En savoir plus sur [la sécurité et la conformité chez Kinsta](#)

SOC 2
Type II

GDPR

CCPA



Français

© 2013 - 2024 Kinsta Inc. Tous droits réservés. Kinsta[®] et WordPress[®] sont des marques déposées. [Mentions légales.](#)