Cours Laravel 6 – les bases – le routage

laravel.sillo.org/cours-laravel-6-les-bases-le-routage/

bestmomo

August 28, 2019

Dans ce chapitre nous allons nous intéresser au devenir d'une requête HTTP qui arrive dans notre application Laravel. Nous allons voir l'intérêt d'utiliser un fichier **.htaccess** pour simplifier les url. Nous verrons aussi le système de routage pour trier les requêtes.

Les requêtes HTTP

Petit rappels

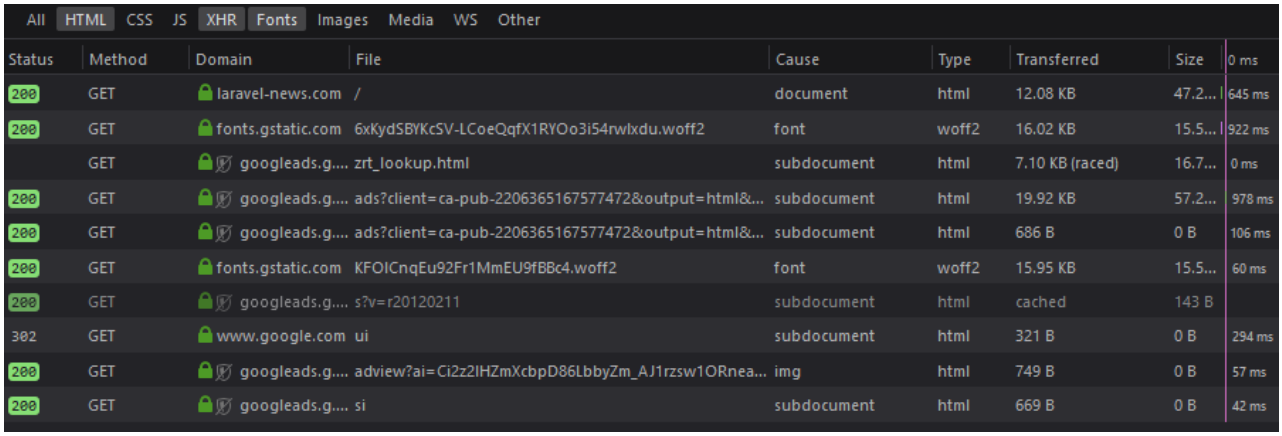
On va commencer par un petit rappel sur ce qu'est une requête HTTP. Voici un schéma illustratif :

Le HTTP (Hypertext Transfer Protocol) est un protocole de communication entre un client et un serveur. Le client demande une ressource au serveur en envoyant une requête et le serveur réagit en envoyant une réponse, en général une page Html.

Quand on surfe sur Internet chacun de nos clics provoque en général cet échange, et plus généralement une rafale d'échanges.

La requête du client comporte un certain nombre d'informations (headers, status code, body...).

Prenons un exemple avec le site Laravel News. Lorsque je clique sur le lien voilà les requêtes HTTP qui se produisent :



En tout 10 requêtes. Regardons d'un peu plus près la première :

On trouve :

- l'url : https://laravel-news.com/
- la méthode : GET
- le code : 200 (donc tout s'est bien passé)
- la version du HTTP : 1.1

Il y a évidemment bien d'autres choses dans les headers (content-type, cookies, encodage...) mais pour le moment on va se contenter de ces informations. Notre navigateur digère tout ça de façon transparente, heureusement pour nous !

Notre application Laravel doit savoir interpréter les informations qui arrivent et les utiliser de façon pertinente pour renvoyer ce que demande le client. Nous allons voir comment cela est réalisé.

Les méthodes

Il est indispensable de connaître les principales méthodes du HTTP :

- **GET** : c'est la plus courante, on demande une ressource qui ne change jamais, on peut mémoriser la requête, on est sûr d'obtenir toujours la même ressource,
- **POST** : elle est aussi très courante, là la requête modifie ou ajoute une ressource, le cas le plus classique est la soumission d'un formulaire (souvent utilisé à tort à la place de PUT),
- **PUT** : on ajoute ou remplace complètement une ressource,
- **PATCH** : on modifie partiellement une ressource (donc à ne pas confondre avec PUT),
- **DELETE** : on supprime une ressource.

La différence entre PUT et POST est loin d'être évidente, vous pouvez lire sur le sujet cet excellent article.

.htaccess et index.php

Pour Laravel on veut que toutes les requêtes aboutissent obligatoirement sur le fichier **index.php** situé dans le dossier **public**. Pour y arriver on peut utiliser une URL de ce genre :

http://monsite.fr/index.php/mapage

Mais ce n'est pas très esthétique avec ce **index.php** au milieu. Si vous avez un serveur Apache lorsque la requête du client arrive sur le serveur où se trouve notre application Laravel elle passe en premier par le fichier **.htaccess**, s'il existe, qui fixe des règles pour le serveur. Il y a justement un fichier **.htaccess** dans le dossier **public** de Laravel avec une règle de réécriture de telle sorte qu'on peut avoir une url simplifiée :

http://monsite.fr/mapage

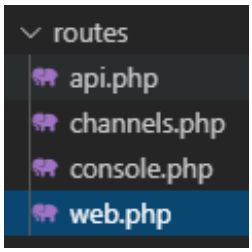
Un petit schéma pour visualiser cette action :

*Pour que ça fonctionne il faut que le serveur Apache ait le module **mod_rewrite** activé.*

Le cycle de la requête

Lorsque la requête atteint le fichier **public/index.php** l'application Laravel est créée et configurée et l'environnement est détecté. Nous reviendrons plus tard plus en détail sur ces étapes. Ensuite le fichier **routes/web.php** est chargé. Voici l'emplacement de ce fichier :

*Les autres fichiers concernent des routes plus spécifiques comme pour les API avec le fichier **api.php** ou les routes pour les actions en ligne de commande avec le fichier **console.php**.*



C'est avec ce fichier que la requête va être analysée et dirigée. Regardons ce qu'on y trouve au départ :

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

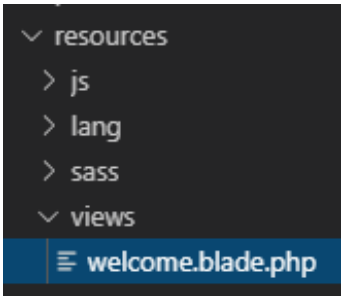
Comme Laravel est explicite vous pouvez déjà deviner à quoi sert ce code :

- **Route** : on utilise le routeur,

- **get** : on regarde si la requête a la méthode « get »,
- **'/'** : on regarde si l'url comporte uniquement le nom de domaine,
- dans la fonction anonyme on retourne (**return**) une vue (**view**) à partir du fichier « welcome ».

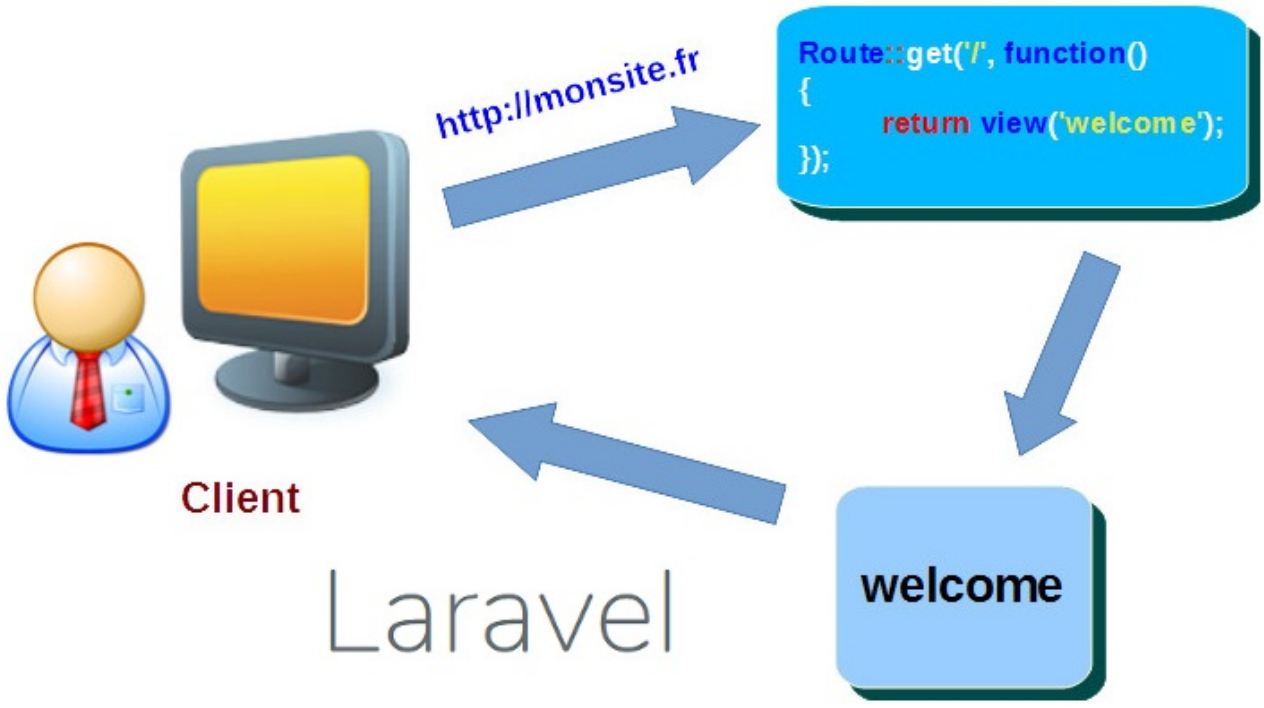
Ce fichier « welcome » se trouve bien rangé dans le dossier des vues :

C'est ce fichier comportant du code Html qui génère le texte d'accueil que vous obtenez au démarrage initial de Laravel :



Laravel propose plusieurs helpers qui simplifient la syntaxe. Il y a par exemple **view** pour la classe **View** comme on l'a vu dans le code ci-dessus.

Visualisons le cycle de la requête :



Sur votre serveur local vous n'avez pas de nom de domaine et vous allez utiliser une url de la forme ***http://localhost/tuto/public*** *en admettant que vous ayez créé Laravel dans un dossier* ***www/tuto****. Mais vous pouvez aussi créer un hôte virtuel pour avoir une situation plus réaliste comme déjà évoqué au précédent chapitre.*

Laravel accepte les verbes suivants : **get**, **post**, **put**, **patch**, **delete**, **options**, **any** (on accepte tous les verbes).

Plusieurs routes et paramètre de route

A l'installation Laravel a une seule route qui correspond à l'url de base composée uniquement du nom de domaine. Voyons maintenant comment créer d'autres routes. Imaginons que nous ayons 3 pages qui doivent être affichées avec ces urls :

1. http://monsite.fr/**1**
2. http://monsite.fr/**2**
3. http://monsite.fr/**3**

J'ai fait apparaître en gras la partie spécifique de l'url pour chaque page. Il est facile de réaliser cela avec ce code :

```
Route::get('1', function() { return 'Je suis la page 1 !'; });
Route::get('2', function() { return 'Je suis la page 2 !'; });
Route::get('3', function() { return 'Je suis la page 3 !'; });
```

Cette fois je n'ai pas créé de vue parce que ce qui nous intéresse est uniquement une mise en évidence du routage, je retourne donc directement la réponse au client. Visualisons cela pour la page 1 :

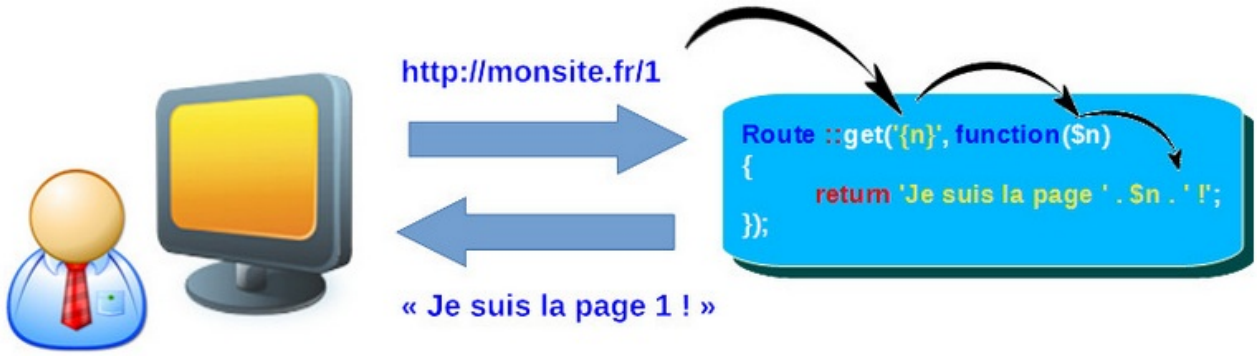
On a besoin du caractère « / » uniquement dans la route de base.

On peut maintenant se poser une question : est-il vraiment indispensable de créer 3 routes alors que la seule différence tient à peu de chose : une valeur qui change ?

On peut utiliser un paramètre pour une route qui accepte des éléments variables en utilisant des accolades. Regardez ce code :

```
Route::get('{n}', function($n) {
    return 'Je suis la page ' . $n . ' !';
});
```

Et une visualisation du fonctionnement :



On dit que la route est paramétrée parce qu'elle possède un paramètre qui peut prendre n'importe quelle valeur.

On peut rendre un paramètre optionnel en lui ajoutant un point d'interrogation mais il ne doit pas être suivi par un paramètre obligatoire. Dans ce cas pour éviter une erreur d'exécution il faut prévoir une valeur par défaut pour le paramètre, par exemple :

```
Route::get('{n?}', function($n = 1) {
```

Le paramètre **n** est devenu optionnel et par défaut sa valeur est 1.

Erreur d'exécution et contrainte de route

Dans mon double exemple précédent lorsque je dis que le résultat est le même je mens un peu. Que se passe-t-il dans les deux cas pour cette url ?

http://monsite.fr/4

Dans le cas des trois routes vous tombez sur une erreur :Par contre dans la version avec le paramètre vous obtenez une réponse valide ! Ce qui est logique parce qu'une route est trouvée. Le paramètre accepte n'importe quelle valeur et pas seulement des nombres. Par exemple avec cette url :

http://monsite.fr/nimportequoi

Vous obtenez :

Je suis la page nimportequoi !

Ce qui vous l'avouerez n'est pas très heureux !

Pour éviter ce genre de désagrément il faut contraindre le paramètre à n'accepter que certaines valeurs. On réalise cela à l'aide d'une expression régulière :

```
Route::get('{n}', function($n) {
    return 'Je suis la page ' . $n . ' !';
})->where('n', '[1-3]');
```

Maintenant je peux affirmer que les comportements sont identiques ! Mais il nous faudra régler le problème des routes non prévues.

Route nommée

Il est parfois utile de nommer une route, par exemple pour générer une url ou pour effectuer une redirection. La syntaxe pour nommer une route est celle-ci :

```
Route::get('/', function() {
  return 'Je suis la page d\'accueil !';
})->name('home');
```

Par exemple pour générer l'url qui correspond à cette route on peut utiliser l'helper route :

```
route('home')
```

Ce qui va générer l'url de base du site dans ce cas : **http://monsite**.

Un avantage à utiliser des routes nommées est qu'on peut réorganiser les urls d'un site sans avoir à modifier beaucoup de code.

Nous verrons des cas d'utilisation de routes nommées dans les prochains chapitres.

Ordre des routes

Une chose importante à connaître est l'ordre des routes !

Lisez bien ceci pour vous éviter des heure de recherches et de prises de tête . La règle est :

Les routes sont analysée dans leur ordre dans le fichier des routes.

Regardez ces deux routes :

```
Route::get('{n}', function($n) {
    return 'Je suis la page ' . $n . ' !';
});

Route::get('contact', function() {
    return "C'est moi le contact.";
});
```

Que pensez-vous qu'il va se passer avec **http://monsite/contact** ?

Je vous laisse deviner et tester et surtout bien retenir ce fonctionnement !

On peut aussi grouper des routes pour simplifier la syntaxe mais nous verrons ça plus tard...

En résumé

- Laravel possède un fichier **.htaccess** pour simplifier l'écriture des url.
- Le système de routage est simple et explicite.
- On peut prévoir des paramètres dans les routes.
- On peut contraindre un paramètre à correspondre à une expression régulière.
- On peut nommer une route pour faciliter la génération des url et les redirections.