Chapitre 3
Créer des routes

1. Prise en main

Un site web de qualité utilise des adresses qui ont du sens, idéalement lisibles et élégantes. Grâce au système de route proposé par Laravel, des adresses du type `/index.php?page=articles&id=123` sont aisément remplaçables par des adresses plus claires comme `/articles/notre-histoire`.

1.1 Définir une route

Ce chapitre se concentre sur le fichier `routes/web.php` qui contient les routes dédiées à l'application web. Mais une application Laravel possède par défaut plusieurs fichiers dans le dossier routes :

– Le fichier `routes/web.php` pour les requêtes HTTP de la partie application web.

– Le fichier `routes/api.php` pour les requêtes HTTP en tant qu'API : les routes sont *stateless* (le serveur ne créera pas de session pour ces routes) et avec des URI (identifiant de ressource uniforme, autrement dit la partie de l'adresse qui suit le nom de domaine) préfixés par `/api`.

– Le fichier `routes/console.php` pour l'exécution de lignes de commandes.

– Le fichier `routes/channels.php` pour les évènements avec l'usage du protocole *WebSockets*.

Remarque

Les fichiers de route web.php et api.php sont chargés au niveau de la classe ***`App\Providers\RouteServiceProvider`****. C'est dans cette classe que sont définis les traitements à exécuter pour toutes les routes d'un fichier de routes. On y trouve entre autres l'assignation de middlewares comme le gestionnaire de session pour le fichier web.php ou l'application d'un préfixe d'URI* ***`/api`*** *pour le fichier api.php.*

Laravel fournit un système de routes simple. Déclarer une route permet de lier des URI à un code à exécuter.

Définir une route dont l'URI est `/accueil` pour le site `http://bonjour-taxi.org/` revient donc à déterminer le texte à afficher quand un visiteur se rend sur la page `http://bonjour-taxi.org/accueil`.

Pour créer une route, il faut appeler la façade `Illuminate\Support\Facades\Route` avec la méthode HTTP souhaitée (`get` par exemple). Indiquez à cette méthode l'URI concerné et le retour à afficher pour le visiteur comme dans l'exemple ci-dessous.

Fichier routes/web.php

```
Route::get('/accueil', function () {
    return 'Bienvenue sur le site !';
});
```

Il s'agit ici d'une route qui souhaite la bienvenue au visiteur se rendant sur la page d'accueil du site.

1.2 Les méthodes HTTP

En plus des méthodes HTTP que l'on peut retrouver dans n'importe quelle application web (`get`, `post`, `put`, `patch`, `delete`, `options`), la façade `Illuminate\Support\Facades\Route` met à disposition des méthodes supplémentaires afin de simplifier certains cas d'usage comme `redirect` qui redirige vers une nouvelle page avec un statut de réponse `302`.

La partie routing de la documentation de Laravel donne le détail des méthodes disponibles et leurs usages avec des exemples.

Exemple de l'utilisation de la méthode redirect

```
Route::redirect('/ancienne-url', '/nouvelle-url');
```

1.3 Choisir les adresses des routes

Le choix de l'adresse d'une route est libre. Il est ainsi possible de hiérarchiser les pages d'une application ou d'un site web en créant par exemple des espaces de noms séparés par le caractère /, et en séparant les mots par des tirets.

Ci-dessous un exemple de fichier qui contient plusieurs routes correspondant aux différentes pages du site.

Fichier routes/web.php

```
Route::get('accueil', function () {
    return 'Bienvenue chez Bonjour Taxi !';
});

Route::get('a-propos', function () {
    return 'Bonjour Taxi est une startup disruptive';
});

Route::get('tarifs/pros', function() {
    return '10 € par mois pour les pros';
});

Route::get('tarifs/particuliers', function() {
    return 'Gratuit pour les particuliers';
});
```

Plutôt que de définir une route `/accueil` pour la page d'accueil, il est parfois préférable de mettre la page d'accueil directement à la racine du domaine.

Pour déclarer une route directement à la racine du domaine, c'est-à-dire une route qui répond au nom de domaine sans adresse supplémentaire, il faut utiliser / dans l'URI de la route.

Une route pour la racine du site

```
Route::get('/', function () {
    return 'Bienvenue chez Bonjour Taxi !';
});
```

1.4 La fonction de rappel

Les réponses retournées par les routes sont construites dans des fonctions de rappel (*callbacks*) appelées également fonctions anonymes.

Remarque

Le chapitre Organiser grâce aux contrôleurs explique comment l'utilisation de contrôleurs permet d'éviter par la suite d'utiliser des fonctions de rappel pour une meilleure organisation du code.

À l'intérieur de ces fonctions de rappel, le développeur est libre de mettre le code PHP qu'il souhaite. Il faut cependant que cette fonction retourne une information au framework pour que ce dernier le renvoie au navigateur. La dernière instruction doit donc être un `return` sur une chaîne de caractères, une réponse JSON ou encore une vue.

Remarque

La notion de vue, permettant d'afficher des templates HTML, sera étudiée en détail au chapitre suivant.

Dans l'exemple suivant, nous affichons sur l'URI `/tarifs/pro` un tarif dépendant de la monnaie enregistrée en session : euro ou dollar. Il s'agit donc d'une fonction de rappel (*callback*) avec plusieurs instructions.

Fichier routes/web.php

```
Route::get('tarifs/pro', function () {

    if (session()->get('currency') == 'euro') {
           $tarif = '10 €';
    } else {
           $tarif = '$15';
    }
    return $tarif . ' par mois pour les pros';

});
```

Avec ces premières notions sur les routes, il est déjà possible de créer un site minimal statique à la manière d'un site en PHP brut tout en évitant la lourdeur de la gestion de son propre système de routes.

La suite de ce chapitre introduit la notion de paramètres des routes : un premier pas vers un site plus dynamique.

2. Paramètres des routes

2.1 Déclarer des paramètres

Il est parfois nécessaire de capturer certaines parties de l'URI d'une route pour ensuite la traiter. Par exemple, dans le cas d'un site de recettes de cuisine, il est nécessaire de capturer l'identifiant de la recette indiquée dans la barre d'adresse par le visiteur pour chercher la recette correspondante dans la base de données et la retourner :

– Pour `/recette/1`, afficher la recette des tomates à la provençale.

– Pour `/recette/2`, afficher la recette de la salade avocat-pamplemousse.

– Pour `/recette/3`, afficher la recette du taboulé express.

– etc.

Il paraît évident que les routes ne doivent pas être créées une par une pour chaque nouvelle recette ajoutée. Il faut donc utiliser des paramètres variables dans les routes.

Pour déclarer un paramètre variable dans l'URI d'une route, il faut utiliser des accolades autour du nom du paramètre. Au lieu d'écrire `recette/1`, `recette/2` et `recette/3` dans l'URI de la route, il faut écrire `recette/{id}`. Les paramètres déclarés doivent uniquement contenir des caractères alphabétiques et des tirets bas `_` .

2.2 Utiliser les paramètres capturés

Pour traiter la variable capturée, il faut la passer en paramètre de la fonction de rappel de la route en lui donnant le même nom. Dans le code ci-dessous, une seule route est déclarée pour traiter toutes les recettes de la base de données.

Paramètre de route, fichier routes/web.php

```
Route::get('recette/{id}', function ($id) {
    return 'Identifiant de la recette :' . $id;
});
```

Le paramètre est donc à la fois déclaré dans la chaîne de caractères qui décrit l'URI et dans la signature de la fonction de rappel associée.

2.3 Utiliser plusieurs paramètres

Tout comme il est possible de créer des routes complexes avec plusieurs termes séparés par des barres obliques /, il est également possible de créer des routes avec plusieurs paramètres.

Par exemple, pour voir le détail d'un commentaire associé à une recette, il pourrait être intéressant d'avoir d'un côté un paramètre pour identifier la recette `recipeId` et d'un autre côté le paramètre pour identifier le commentaire `commentId`.

Ainsi, pour envoyer une réponse à l'adresse `recette/13/commentaire/5`, il faut coder une route qui prend en compte ces deux paramètres comme dans l'exemple ci-dessous.

Route avec plusieurs paramètres, fichier routes/web.php

```
Route::get(
    'recette/{recipeId}/commentaire/{commentId}',
    function ($recipeId, $commentId) {

            $response = "Recette numéro $recipeId";
            $response .= ", commentaire numéro $commentId";
            return $response;
```

```
    }
);
```

Cet exemple affiche le numéro de recette et le numéro de commentaire associé à cette recette au visiteur.

2.4 Paramètres optionnels

L'utilisation d'un point d'interrogation `?` dans l'URI d'une route rend possible la déclaration d'un paramètre optionnel. Un paramètre optionnel sur une route permet au programme de répondre par exemple pour la route `bonjour/{name?}` à la fois à la route bonjour et à la route `bonjour/pierre`.

Un paramètre optionnel nécessite également de donner une valeur par défaut à la variable qui est passée à la fonction de rappel de la route. Cette valeur par défaut peut être `null`, `false` ou toute autre valeur (une chaîne de caractères, un entier, etc.).

Route avec paramètres optionnels, fichier routes/web.php

```
Route::get('bonjour/{name?}', function ($name = null) {

    if ($name === null) {
           return 'Bonjour tout le monde';
    } else {
           return "Bonjour $name";
    }

});
```

Le code ci-dessus permet, à l'aide du paramètre optionnel `name`, de saluer l'utilisateur si un nom est fourni dans l'URL ou de saluer tout le monde dans le cas contraire.

2.5 Usage des expressions régulières

Il est possible de contraindre le format d'un paramètre d'URI en décrivant ses valeurs possibles avec une expression régulière. Si la valeur du paramètre ne passe pas l'évaluation de l'expression régulière, une réponse `404` est automatiquement renvoyée. L'expression régulière s'ajoute en appelant la méthode `where` sur une instance de la classe `Route`.

Route avec paramètres contraint, fichier routes/web.php

```
Route::get('recette/{id}', function ($id) {
    return 'Identifiant de la recette :' . $id;
})->where('id', '[0-9]+');
```

L'ajout de `->where('id', '[0-9]+')` requiert l'utilisation d'un ou plusieurs chiffres pour la valeur du paramètre `id`. Laravel met à disposition quelques méthodes pour les cas d'usage les plus courants en ayant seulement à spécifier le nom du paramètre à évaluer :

- `whereAlpha('name')` évalue l'expression régulière `[a-zA-Z]+`;
- `whereAlphaNumeric('name')` évalue l'expression régulière `[a-zA-Z0-9]+`;
- `whereIn('category', ['movie', 'song', 'painting'])` évalue l'expression régulière `movie|song|painting`;
- `whereNumber('id')` évalue l'expression régulière `[0-9]+`;
- `whereUlid('ulid')` évalue l'expression régulière `[0-7][0-9a-hjkmnp-tv-zA-HJKMNP-TV-Z]{25}`;
- `whereUuid('uuid')` évalue l'expression régulière `[\da-fA-F]{8}-[\da-fA-F]{4}-[\da-fA-F]{4}-[\da-fA-F]{4}-[\da-fA-F]{12}`.

Remarque

Compréhension avancée : pour résoudre une route, Laravel lit l'ensemble des fichiers de route de haut en bas et dans l'ordre de déclaration des fichiers de route dans `App\Providers\RouteServiceProvider`. *Laravel sélectionne alors la route dont l'URI est le plus spécifique. En cas de doublon, ce sera la dernière déclaration qui sera exécutée.*