Chapitre 2

Premiers pas avec Laravel : routes, et contrôleurs

Dans ce chapitre, nous allons découvrir les bases de Laravel en abordant les sujets principaux tels que le routage, l'architecture MVC et les contrôleurs. Ce chapitre sera une première étape pour comprendre les fondamentaux et vous permettra de vous familiariser avec les principales fonctionnalités et outils nécessaires pour le développement d'applications web avancées avec Laravel.

2.1 Le routage

Le routage est une étape cruciale dans le développement d'une application web, car il permet de définir les différentes pages et fonctionnalités de l'application. Le dossier `routes` contient plusieurs fichiers, celui qui nous intéresse ici c'est le fichier `web.php` qui contient initialement deux instructions, la première instruction

```
use Illuminate\Support\Facades\Route;
```

permet d'importer la classe `Route` à utiliser pour le routage, puis une deuxième instruction

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

```
    return view('welcome');
});
```

qui associe l'instruction `return view('welcome')` (qui affiche simplement la page d'accueil appelée `welcome.blade.php`) à l'URL `/` (racine de l'application) lorsque cet URL est visité avec la méthode `get`, ce qui résume les éléments de base d'une instruction de routage qui sont :

— La méthode (get, post, delete, put ... etc.

— L'URL

— l'action à faire.

L'action à associer à chaque URL peut être une fonction définie sur place, mais en générale ces actions sont placées dans des "Contrôleurs" que nous allons expliquer en détails dans la section 2.2. Ces actions peuvent être :

— Recherche d'une information.

— Insérer, modifier ou supprimer des données.

— Afficher une interface utilisateur.

— ... etc.

Pour ajouter un nouveau routage, il faut écrire l'instruction suivante

```
Route::<method>(<url>, <action>);
```

par exemple, pour créer un routage qui affiche une page avec le message "Hello World" lorsque l'utilisateur visite l'URL "/hello" sur un navigateur (avec la méthode "get"), nous écrivons la commande suivante :

```
Route::get("/hello", function() {
    return "Hello World";
});
```

2.1.1 Routage avec Paramètres

Dans Laravel, il est possible d'utiliser des paramètres dans les instructions de routage pour capturer des valeurs dynamiques dans les URL. Les paramètres sont définis dans les routes en utilisant des accolades (`{}`) pour indiquer le nom du paramètre. Par exemple, pour définir une route avec un paramètre `id` nous ecrivons :

```
Route::get('/produits/{id}', function ($id) {
    // Code qui utilise la variable $id
});
```

Dans cet exemple, si l'url visitée est `http://yourhost.com/produit/15`, alors valeur 15 est capturée dans la fonction associé à travers son argument qui porte le même nom (`$id`).

il est possible de rendre les paramètres optionnels dans les routes de Laravel en ajoutant un point d'interrogation (`?`) après le nom du paramètre. Par exemple, pour définir une route avec un paramètre `id` optionnel nous écrivons :

```
Route::get('/produits/{id?}', function ($id = null) {
    if ($id) {
        // code si le paramètre optionnel est passé
    } else {
        // code si le paramètre optionnel n'est pas passé
    }
});
```

2.1.2 Routes avec des noms

Nommer les routes dans Laravel permet d'y faire référence de manière plus facile et pratique. Au lieu d'écrire manuellement l'URL de chaque route dans le code de l'application, vous pouvez utiliser le nom de la route pour générer dynamiquement l'URL correspondante. Cela est particulièrement utile pour les cas où vous devez générer des URL dynamiquement, par exemple pour créer des liens dans les vues ou pour effectuer

des redirections. En nommant les routes, vous pouvez simplifier votre code et rendre votre application plus maintenable.

Pour nommer une route dans Laravel, vous pouvez utiliser la méthode `name()` de l'objet Route. Par exemple :

```
Route::get('/produits/{id}', function ($id) {
    // code pour afficher les informations d'un produit
})->name('produits.afficher');
```

Dans cet exemple, la route est nommée `produits.afficher`. Vous pouvez ensuite générer l'URL correspondante dans votre code en utilisant la méthode `route()` comme le montre l'exemple ci-dessous :

```
<a href="{{route('produits.afficher',['id'=>15])}}">Afficher le produits</a>
```

l'URL générée dans ce cas est `http://yourhost.com/produits/15`. Cela est particulièrement utile si vous devez modifier l'URL de la route ultérieurement, car vous n'aurez pas à rechercher et à modifier manuellement tous les liens correspondants dans votre code.

2.1.3 Récupérer les données des requêtes

Vous pouvez récupérer les données envoyées via une requête HTTP (souvent via un formulaire) en rajoutant un paramètre de type `Request` à l'action de la route. Pour utiliser les objets `Request` dans une route, vous devez d'abord les importer en haut de votre fichier de route :

```
use Illuminate\Http\Request;
```

Ensuite, vous pouvez utiliser l'objet Request en tant que paramètre dans la fonction de rappel de route, la méthode `input` permet de récupérer les données saisies

```
Route::post('/produits', function (Request $request) {
    // utiliser $request pour accéder aux données de la requête
    // utiliser $request->input('name') pour récuperer le champs 'name'
});
```

2.1.4 Regrouper les routes

Il est possible de regrouper des routes qui ont le même préfixe en utilisant la méthode `prefix` de la classe Route. Par exemple, pour regrouper toutes les routes qui commencent par "/produits", vous pouvez utiliser le code suivant dans votre fichier de routes :

```
Route::prefix('/produits')->group(function () {
    Route::get('/ajouter', function() {});
    Route::get('/supprimer',  function() {});
    Route::get('/modifier',  function() {});
});
```

dans l'exemple précédent, les URL `/produits/ajouter`, `/produits/sipprimer` et `/produits/modifier` sont crées. Le préfixe `/produits` est ajouté automatiquement à chaque route définie dans le groupe précédent.

2.1.5 Utiliser les contrôleurs

Nous avons montré dans les exemples précédents comment associer une fonction à chaque URL, mais en pratique, nous n'allons pas implémenter toutes les actions de notre application dans le fichier de routage. Ces fonctions sont généralement groupées dans des contrôleurs dans le but d'atteindre une meilleure organisation du code, une séparation claire des préoccupations et une meilleure maintenance. Nous montrons comment créer et écrire le code des contrôleurs prochainement dans la section 2.2. Nous allons voir ici simplement comment utiliser des fonction qui sont supposée être déjà déclarées dans un contrôleur.

Vous pouvez définir une route pour appeler la méthode de votre contrôleur en écrivant le nom du contrôleur, suivi par `@` et puis le nom de la fonction comme le montre l'exemple suivant :

```
Route::get('/products', [ProductController::class, 'index'])
```

Cela fait appel à la fonction `index` déclarée dans le contrôleur `ProductController` lorsque

l'URL `/products` est consultée.

2.1.6 Route vers une vue

Dans laravel, il est possible de faire des routes directement vers des vues. Cela peut être utile pour les vues statiques qui ne nécessitent pas de traitement spécifique dans un contrôleur (page d'accueil comme exemple). Pour cela, vous pouvez utiliser la méthode view() dans votre définition de route.

Par exemple, la route initiale qui renvoie vers la vue "welcome.blade.php" peut se réécrire de la façon suivante :

```
Route::view('/', 'welcome');
```

2.1.7 Les middlewares

Les middlewares dans une route permettent de filtrer les requêtes HTTP entrantes avant qu'elles n'atteignent les méthodes de gestion de ces requêtes dans les contrôleurs. Les middlewares peuvent être utilisés pour effectuer une authentification, une validation de formulaire, une autorisation, une gestion de cache, une journalisation, ou toute autre tâche de filtrage nécessaire pour les demandes entrantes.

Dans Laravel, il existe des middlewares intégrés tels que "auth" pour l'authentification des utilisateurs et "verified" pour la vérification de l'adresse e-mail des utilisateurs. Les développeurs peuvent également créer leurs propres middlewares pour répondre aux besoins spécifiques de leurs applications.

Pour utiliser un middleware dans une route, vous pouvez ajouter la méthode `middleware()` à la définition de la route. Cette méthode prend le nom du middleware en tant que paramètre et peut être appelée de deux manières différentes.

La première manière consiste à utiliser le nom complet du middleware, en tant que chaîne de caractères, comme suit :

```
Route::get('/users/myprofile', function () {
```

```
        // affichage du profile
    })->middleware('auth');
```

Dans cet exemple, le middleware `auth` est utilisé pour protéger la route `/users/myprofile` et y permettre l'accès uniquement pour les utilisateurs authentifiés.

La deuxième façon consiste à assigner des middlewares à un groupe de routes en utilisant la méthode `middleware` comme le montre l'exemple ci-dessous :

```
Route::middleware(['auth'])->group(function () {
    Route::post('/products', function () {
        // action pour ajouter un produit
    });
    Route::delete('/products/{id}', function ($id) {
        // action pour supprimer un produit
    });
});
```

dans ce cas, les deux routes permettant d'ajouter ou supprimer un produit sont protégées à l'aide du middleware `auth`.

2.1.8 Aller plus loin

Nous avons présenté dans cette section les notions les plus importantes du routage, pour contrainte de temps, certaines autre notions ne seront pas présenté dans ce cours. Il reste donc recommandé d'aller jeter un coup d'oeil sur la documentation officielle [1].

2.2 Contrôleurs

Les contrôleurs sont des classes qui traitent les requêtes HTTP de l'application et effectuent les actions appropriées, tels que récupérer les données à partir de la base de données, les manipuler, les afficher à l'utilisateur, et exécuter des opérations telles que

1. https://laravel.com/docs/10.x/routing

la validation et la gestion des erreurs. Les contrôleurs sont utilisés pour encapsuler la logique de l'application et pour séparer la logique de présentation de la logique de traitement. Dans Laravel, chaque méthode d'un contrôleur correspond généralement à une action spécifique de l'application, telle que l'affichage d'une page d'accueil, la création d'un nouveau compte utilisateur, la récupération d'une liste d'articles de blog ... etc. Les contrôleurs peuvent être appelés à partir des routes de l'application pour gérer les différentes actions associées à chaque URL. Les avantages de l'utilisation de contrôleurs incluent une meilleure organisation du code, une séparation claire des préoccupations et une meilleure maintenance de l'application. Les contrôleurs sont également utiles pour faciliter l'extension et la modification de l'application au fil du temps.

2.2.1 Création d'un contrôleur

Pour créer un contrôleur dans Laravel, vous pouvez utiliser la commande artisan `make:controller` en spécifiant le nom du contrôleur que vous voulez créer :

```
php artisan make:controller AdminController
```

La commande précédente crée un fichier nommé `AdminController` dans le répertoire `App/Http/Controllers`. Initialement, le contenu du fichier est le suivant.

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class AdminController extends Controller
{
    //
}
```

La première instruction définit le namespace dans lequel se trouvent ce contrôleur. Elle permet de regrouper les contrôleurs dans un même espace pour faciliter leur organisation et leur utilisation dans le reste de l'application. Elle est généralement placée en haut des fichiers de contrôleurs pour indiquer que ces derniers appartiennent au namespace

`App\Http\Controllers`. Cela permet également de trouver automatiquement les contrôleurs lorsqu'ils sont utilisés dans des routes ou d'autres parties de l'application.

La deuxième instruction permet d'importer la classe `Request`, qui est utilisée principalement pour récupérer les données envoyées via des formulaire comme nous avons déjà vu dans la section 2.1.3.

Par la suite, une classe appelée `AdminController` qui hérite de la classe `Controller` est déclarée.

Nous ajoutons à ce fichier par la suite les fonctions (méthodes pour être précis) qui correspondent à des actions à effectuer, comme l'authentification, le changement des privilèges ... etc, comme le montre l'exemple ci-dessous :

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class AdminController extends Controller
{
    public function ajouterAdmin(Request $request) {
        // Code pour ajouter un Admin
    }
    public function sAuthentifier() {
        // Code pour s'authentifier
    }
    public function seDeconnecter() {
        // Code pour se déconnecter
    }
    public function modifierPrivileges() {
        // Code pour modifier les privilèges
    }
}
```

Ces action peuvent être associées à des URL dans des routes comme nous l'avons déjà vu dans la section 2.1.5. N'oublier pas que les contrôleurs utilisés dans le fichier `web.php` doivent être importés à l'aide de l'instruction :

```
use App/Http/Controllers/AdminController;
```

2.2.2 Contrôleur invocable

Les contrôleurs invocables sont une fonctionnalité 8 qui permet de créer des contrôleurs qui répondent à une seule action. Cette fonctionnalité n'est disponible qu'à partir de Laravel 8.

Pour créer un contrôleur invocable, il est possible d'ajouter l'option `--invokable` à la commande `artisan` de création de contrôleur comme le montre l'exemple suivant :

```
php artisan make:controller HomeController --invokable
```

Le code initial dans un contrôleur invocable après sa création est le suivant :

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class HomeController extends Controller
{
    /**
     * Handle the incoming request.
     */
    public function __invoke(Request $request)
    {
        //
    }
}
```

A la différence d'un contrôleur classique, un contrôleur invocable contient la méthode

`__invoke` qui est la seule action à effectuer dans ce contrôleur. Pour faire appelle à un contrôleur invocable dans une route, le nom de la méthode n'a pas besoin d'être spécifié, il faut seulement donner le nom du contrôleur.

```
Route::get('/home', HomeController::class);
```

2.2.3 Les ressources

Dans Laravel, une ressource est une représentation des opérations CRUD (Create, Read, Update, Delete) d'un modèle de données (à voir dans le chapitre 4). Elle permet de définir des routes pour gérer les opérations CRUD d'un modèle de manière standardisée et cohérente.

Pour créer un contrôleur de ressource, il est possible d'ajouter l'option `--resource` à la commande `artisan` permettant de créer le contrôleur.

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

Cette commande créera un contrôleur de ressource avec les méthodes correspondantes aux opérations CRUD, ainsi que la logique nécessaire pour gérer ces opérations. À sa création, un contrôleur de ressource contient par défaut les méthode suivantes :

- **index :** pour afficher le contenu de cette ressources.
- **show :** pour afficher un élément spécifique.
- **create :** pour afficher le formulaire de création d'un nouvel élément.
- **store :** pour enregistrer le nouvel élément (dans une base de données).
- **edit :** pour afficher le formulaire de modification d'un élément existant.
- **update :** pour mettre à jour les informations d'un élément existant.
- **destroy :** pour supprimer un élément existant.

Initiallemnt, un contrôleur d'une ressource contient le code suivant :

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

```
use Illuminate\Http\Request;
class ProductController extends Controller {
    public function index() {
    }
    public function create() {
    }
    public function store(Request $request) {
    }
    public function show(string $id) {
    }
    public function edit(string $id) {
    }
    public function update(Request $request, string $id) {
    }
    public function destroy(string $id) {
    }
}
```

Vous pouvez ensuite définir les routes correspondantes dans votre fichier de routes en utilisant la méthode resource :

```
use App\Http\Controllers\PhotoController;

Route::resource('/products', ProductsController::class);
```

la commande `Route::resource` génère une route pour chaque fonction dans le contrôleur. Pour vérifier les routes générées avec leurs noms et actions, vous pouver utiliser la commande artisan `route:list`

```
php artisan route:list
```

Cette commande va afficher toutes les routes générées avec leurs URL, actions et noms.

Routage partiel de ressources

Vous pouvez utiliser les méthodes `only` et `except` sur les contrôleurs de ressources pour spécifier les méthodes de contrôleurs qui doivent être incluses ou exclues de la ressource. La méthode `only` est utilisée pour inclure les méthodes spécifiées dans la ressource. Par exemple, si vous souhaitez inclure uniquement les méthodes `index` et `show` dans une ressource `Product`, vous pouvez le faire comme ceci :

```
Route::resource('/products', ProductController::class)->only(['index', 'show']);
```

Cela ne créera que les routes pour les méthodes `index` et `show` du contrôleur `ProductController`.

La méthode `except` est utilisée pour exclure les méthodes spécifiées de la ressource. Par exemple, si vous souhaitez exclure les méthodes `create` et `edit` d'une ressource `Product`, vous pouvez le faire comme ceci :

```
Route::resource('/products', ProductController::class)->except(['create', 'edit']);
```

Cela créera toutes les routes pour le contrôleur `ProductController`, à l'exception des méthodes `create` et `edit`. Pour plus de détails sur la création et la manipulation des contrôleur, il est possible de visiter cette page [2] de la documentation.

2. https://laravel.com/docs/10.x/controllers