Gestion du routage

1. Objectifs

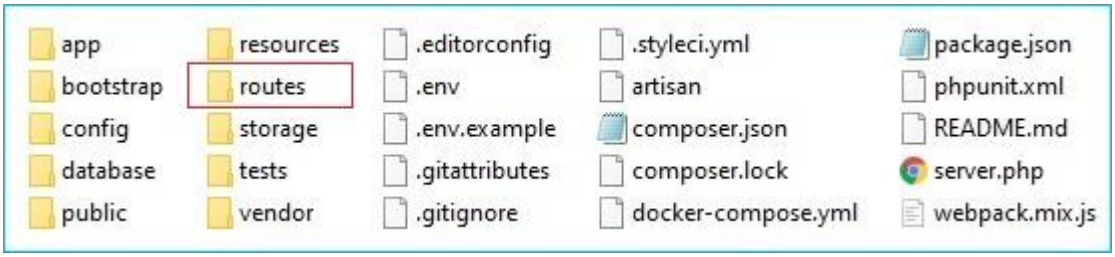
- Connaitre le système de routage Laravel

2. Qu'est-ce que le routage sous Laravel

- Laravel fournit un système de routes simple. Déclarer une route permet de lier une URI (identifiant de ressource uniforme, autrement dit la partie de l'adresse qui suit le nom de domaine) à un code à exécuter.
- Le routage est un moyen de créer une URL de requête pour votre application. La meilleure chose à propos du routage Laravel est que vous êtes libre de définir vos routes comme vous le souhaitez.
- Le routage est l'un des composants clés du framework Laravel, c'est simplement un mécanisme qui effectue le mappage de vos requêtes vers une action de contrôleur spécifique.
- Toutes les routes Laravel sont définies dans le fichier situé sous forme de fichier /routes/web.php , qui est automatiquement chargé par le framework
- Dans Laravel, toutes les demandes sont mappées avec des routes, toutes les routes sont créées dans le dossier racine. Pour votre application Web, vous pouvez définir des itinéraires liés à l'application dans le fichier web.php tandis que tous les itinéraires pour l'API sont définis dans le fichier api.php.

3. Présentation

- Lorsque Laravel est installé et configuré, vous aurez tous les fichiers dont vous avez besoin dans votre dossier pour commencer un projet Laravel.
- Lorsqu'on affiche une page Web Laravel dans le navigateur, l'URL ne se terminera pas par le nom d'un fichier .html ni .php. Plutôt, il s'agira d'une suite de mots ressemblant à des dossiers et sous-dossiers, comme par exemple http://127.0.0.1:8000/inscription.
- Toutes les routes d'application sont enregistrées dans le fichier web.php . Ce fichier indique à Laravel les URI auxquels il doit répondre et le contrôleur associé lui donnera un appel particulier.



- Par défaut, Laravel s'installe avec un dossier de routes, pour gérer divers besoins des itinéraires, dans son répertoire racine, ce dossier contient quatre fichiers, api.php (utilisé pour gérer les routes de l'API), channels.php , console.php et web.php (gère les routes normales).
- Pour créer une route, il faut ainsi appeler la classe Route avec la méthode HTTP souhaitée (get par exemple). Indiquez à cette méthode l'URI concernée et le retour à afficher pour le visiteur comme dans l'exemple ci-dessous.
- Fichier routes/web.php

```
Route::get('/', function () {
    return 'Welcome to Laravel Site.';
});
```

- Dans cet article, nous nous concentrerons principalement sur le web.php, car c'est là que tous nos itinéraires vont vivre.
- Le routage dans Laravel comprend trois catégories:

  Routage de base

  Routes nommés

  Paramètres de Route

4. Routage de base

Route basique

- Une route Laravel très basique est simplement exprimée comme suit:

```
Route::get('/', function () {
 return 'Welcome to Laravel Site.';
});
```

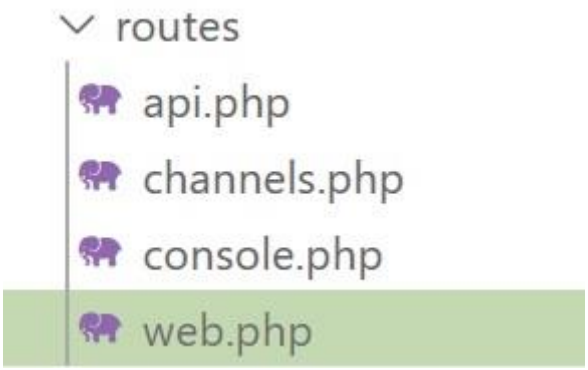
  - Ici, "Route" est une classe qui a une méthode statique "get" qui renvoie une méthode "view" qui présente une page Web.
  - Lorsqu'un utilisateur visite la page d'accueil, il est redirigé vers la page "Welcome" page. Ce "welcome" est en fait un fichier PHP: "welcome.blade.php".

- Il a été stocké par défaut dans le dossier “views” pendant l’installation de Laravel. Lorsque nous discutons du concept du moteur de modèle “view” et “blade”, vous comprendrez parfaitement ce qui se passe dans la couche de présentation.

- Plusieurs routes et paramètre de route comme suit:
  - À 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 trois pages qui doivent être affichées avec ces URL :
    1. http://ista.ma/1
    2. http://ista.ma/2
    3. http://ista.ma/3

```
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 !';
});
```

- **afficher des pages “view”**



- Si vous ouvrez le dossier Route, vous verrez quatre fichiers.
- Ceux-ci incluent :
  - api.php,
  - channels.php,
  - console.php
  - web.php
- C’est ce fichier web.php que nous voulons examiner maintenant.
- Les routes utilisées sont définies dans le fichier routes\web.php.
- Par défaut, nous avons accès à une route pour la page d’accueil qui est écrite sous cette forme :

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

- Comme Laravel est explicite vous pouvez déjà deviner à quoi sert ce code :
  - `Route` : on utilise le routeur, appel statique à la classe qui gère le système de routage ;
  - `get` : on regarde si la requête a la méthode "**get**" ;
  - '`/`' : on regarde si l'URL comporte uniquement le nom de domaine ; '`return`': retourne une vue (view) à partir du fichier "**welcome**".

Création de nouvelles routes

- Il est possible et même nécessaire d'ajouter d'autres routes pour diriger les demandes vers la bonne méthode d'un contrôleur ou encore la fonction de reppel (callback function), comme c'est le cas dans le code précédent, vers la bonne vue.

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

Verbes de route

- Vous avez peut-être remarqué que nous utilisons `Route :: get` dans nos définitions d'itinéraire.
- Ça signifie nous disons à **Laravel** de ne faire correspondre ces routes que lorsque la requête HTTP utilise la méthode **GET**.
- Mais que se passe-t-il s'il s'agit d'un formulaire **POST**, ou peut-être d'un **JavaScript** envoyant **PUT** ou **DELETE**.
- Il existe quelques autres options pour les méthodes permettant d'appeler une définition d'itinéraire, comme illustré ci-dessous:

```
Route::get('/', function () {
return 'Hello, World!';
});
Route::post('/', function () {});
Route::put('/', function () {});
Route::delete('/', function () {});
Route::any('/', function () {});
```

```
Route::match(['get', 'post'], '/', function () {});
```

- **GET:** Ceci est principalement utilisé pour récupérer les données du serveur, sans modifier les données et les renvoyer à l'utilisateur.
- **POST:** Cela nous permet de créer de nouvelles ressources ou données sur le serveur, bien que cela puisse être utilisé uniquement pour envoyer des données, pour un traitement ultérieur (cela pourrait être la validation des données, comme dans le processus de connexion). Il est considéré comme plus sûr que GET pour l'envoi de données sensibles.
- **PUT:** Put fonctionne comme le POST dans le sens où il vous permet d'envoyer des données au serveur, généralement pour mettre à jour une ressource existante au lieu de la créer. Vous le mettez essentiellement.

5. Redirection

Si vous définissez une route qui redirige vers un autre URI, vous pouvez utiliser la méthode Route::redirect. Cette méthode fournit un raccourci pratique pour que vous n'ayez pas à définir une route ou un contrôleur complet pour effectuer une simple redirection :

```
Route::redirect('/ici', '/là-bas');
```

6. Affichage des routes

Si votre itinéraire ne doit renvoyer qu'une vue, vous pouvez utiliser la méthode `Route::view`. Comme la méthode redirect, cette méthode fournit un raccourci simple pour que vous n'ayez pas à définir une route ou un contrôleur complet. La méthode view accepte un URI comme premier argument et un nom de vue comme second argument. De plus, vous pouvez fournir un tableau de données à transmettre à la vue en tant que troisième argument facultatif :

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

Route::view('/welcome', 'welcome', ['name' => ‘steve']);
```

7. Liste des routes

La commande Artisan `route:list` peut facilement fournir une vue d'ensemble de toutes les routes définies par votre application :

```
php artisan route:list
```

8. Les paramètres des routes

- Partant de l'exemple précédent on peut se poser une question : est-il vraiment indispensable de créer trois routes alors que la seule différence tient à peu de choses : une valeur qui change ?
- On peut utiliser un paramètre pour une route qui accepte des éléments variables en utilisant des accolades. L'exemple devient alors:

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

- Les paramètres des routes sont toujours entre crochets {} et doivent être composés de caractères alphabétiques. Ne peut pas contenir les caractères "-" , peut utiliser le caractère "_" Substitution de caractères "-" .
- Les paramètres de routage sont injectés dans le contrôleur ou la fonction de rappel en fonction de leur ordre, plutôt que par les paramètres de la fonction de rappel ou du contrôleur.

Paramètres obligatoires

- Par exemple, si vous souhaitez capturer l'ID utilisateur dans l'URL lorsque vous souhaitez obtenir les détails de l'utilisateur, vous pouvez vous référer aux pratiques suivantes:

```
Route::get('user/{id}', function ($id) {
    return 'User '.$id;
});
```

  - Plusieurs paramètres de routage peuvent également être capturés, par exemple:

```
Route::get('posts/{post}/comments/{comment}', function
($postId, $commentId) {
    //
});
```

Paramètres facultatifs

  - Parfois, vous devez spécifier des paramètres de routage, mais les paramètres ne sont pas obligatoires, "?" est utilisé à ce moment. Le caractère est marqué après le nom du paramètre pour garantir que la valeur par défaut est donnée à la variable correspondante de l'itinéraire.

```php
Route::get('user/{name?}', function ($name = null) {
    return $name;
});
Route::get('user/{name?}', function ($name = 'Tom') {
    return $name;
```

9. Routes nommées

- Il est parfois utile de nommer une route, par exemple pour générer une **URL** ou pour effectuer une **redirection**.
- L'avantage d'utiliser une route nommée est que si nous changeons l'**URL** 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.
- Les routes nommées sont créées en utilisant la méthode `name`

La syntaxe (méthode name)

- Laravel fournit un moyen simple de nommer vos routes par lequel nous n'avons pas à coder en dur l'URI dans nos vues Blade.
- Vous pouvez déterminer le nom de l'itinéraire à l'aide de la méthode du nom dans l'exemple d'itinéraire ci-dessous.

```php
Route::view('/home', 'pages.home')->name('home');
```

Générer une URL à l'aide d'une route nommée

- Pour générer une URL en utilisant le nom de la route
  - `<a href="{{route('home') }}">`
- Si vous utilisez le nom de la route pour la redirection

```php
Route::view('/login', 'pages/home')->name('home');

Route::get('/', function () {
    return redirect()->route('home');

});
```

- Si la route nommée définit des paramètres, vous pouvez passer les paramètres comme deuxième argument à la fonction route.

```
Route::get('/user/{id}/profile', function ($id) {
return view('users.profile', [
    'id' => $id
]);
 })->name('profile');
```

Les paramètres donnés seront automatiquement insérés dans l'URL générée dans leurs positions correctes :

```
<a href="{{route('profile', ['id' => 1])}}">home</a>
```

- Si vous transmettez des paramètres supplémentaires dans le tableau :

```
Route::get('/user/{id}/profile', function ($id)
 { //
 })->name('profile');
```

  - Ces paires clé/valeur seront automatiquement ajoutées à la chaîne de requête de l'URL générée :

```
<a href="{{route('profile', ['id' => 1, 'photos' => 'yes'])}}">
My  Profile </a>

// /user/1/profile?photos=yes
```

10. Groupe des routes

• Les routes peuvent être regroupées pour éviter la répétition du code.

• Les groupes de routage vous permettent de partager des attributs de routage, tels que le middleware, sur un grand nombre de routages sans avoir à définir ces attributs sur chaque routage individuel.

• 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::get('/', function () {

  return view('welcome');
});

Route::group([
      'namespace' => 'Admin',
```

```
      'middleware' => 'auth',
      'prefix' => 'admin'
    ], function () {

  Route::view('/home', 'pages.home',['name' => 'steve'])->name('home');
  Route::get('/login/{id?}',function($id=null){
           return view('pages.home', [
               'name' => $id
                ]);
})->name('login');


    });
```

11. Applications

Exercice 01

1. Créer la liste des routes pour répondre aux URLs suivants :
   - http://localhost/laravel/public/
   - http://localhost/laravel/public/login
   - http://localhost/laravel/public/page/1

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

Route::get('login', function()
{
    return 'login';
});

Route::get('page/1', function()
{
    return 'page1';
});
```

2. Ajouter une route /test
3. Ajouter un paramètre qui sera affiché : /test/param
4. Utiliser une vue pour cette route
5. Lister les routes avec la commande artisan

Exercice 02

Soit la route suivante :

```
Route::get('article/{n}', function($n) {
return 'Je suis l\'article numéro ' . $n . ' !';
})->where('n', '[0-9]+');
```

1. Une seule URL ci-dessous est interceptée par cette route, laquelle ?
   - .../article
   - .../article/1254
   - .../11
   - .../article/un