UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence 2 Informatique
INFO0303 – 2018/2019
***Resp. :* F. Alin**

Travaux Pratiques n° 4

Introduction à Laravel

Dans cette partie nous allons présenter les différentes étapes de la mise en oeuvre d'un site web construit autour du framwork MVC Laravel.

1 Environnement de travail

Il existe de nombreuses solutions pour mettre en place l'environnement nécessaire au développement d'une application Laravel. Nous allons détailler ici une solution simple qui ne nécessite pas de droit particulier sur le poste de travail.

1.1 Composants logiciels nécessaires

Laravel nécessite assez peu de pré requis système. Par ailleurs, vous pouvez éviter toute installation en utilisant la machine virtuelle fournie avec le framework. Dans le cas contraire vous devez disposer d'un système de base de données, et d'un serveur web avec les extensions suivantes :

- PHP >= 7.1.3 ;
- OpenSSL PHP ;
- PDO PHP ;
- Mbstring PHP ;
- Tokenizer PHP ;
- XML PHP ;
- Ctype PHP ;
- JSON PHP ;
- BCMath PHP.

1.2 Les étapes de l'installation :

1. Démarrage de UwAmp ;
2. Ajouter dans le path le répertoire d'accès à l'interpréteur php
   - Rechercher le répertoire dans lequel se trouve le fichier php.exe. Avec UwAmp il se trouve dans le dossier `\UWamp\bin\php\php.X.X.X`. Choisissez de préférence la version `php.7.7.2` ;
   - Dans le panneau de configuration Windows, choisissez "Modifier les variables d'environnement pour mon compte" et ajouter le nouveau chemin dans le path.
3. Lancer un terminal de commande et taper `php -V`. Si tout s'est bien passé vous devriez avoir un message du type suivant :

```
~$ php -v
PHP 7.1.19 (cli) (built: Aug 17 2018 18:03:17) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
```

4. Installer composer :
   — rendez vous sur le site `http::\getcomposer.org`, puis dans l'onglet getting started choisissez installation Windows ;
   — suivez la procédure **Manual Installation** ;
   — ouvrez un nouveau terminal. Tapez la commande composer -V. Si les choses se sont bien passées vous devriez obtenir un affichage du type suivant :

   ```
   ~$ composer -V
   Composer version 1.7.3 2018-11-01 10:05:06
   ```

5. Installation de laravel : en vous plaçant à la racine du répertoire dans lequel vous souhaitez installer votre application Laravel (le répertoire www de votre installation UwAmp par exemple), tapez la commande suivante :

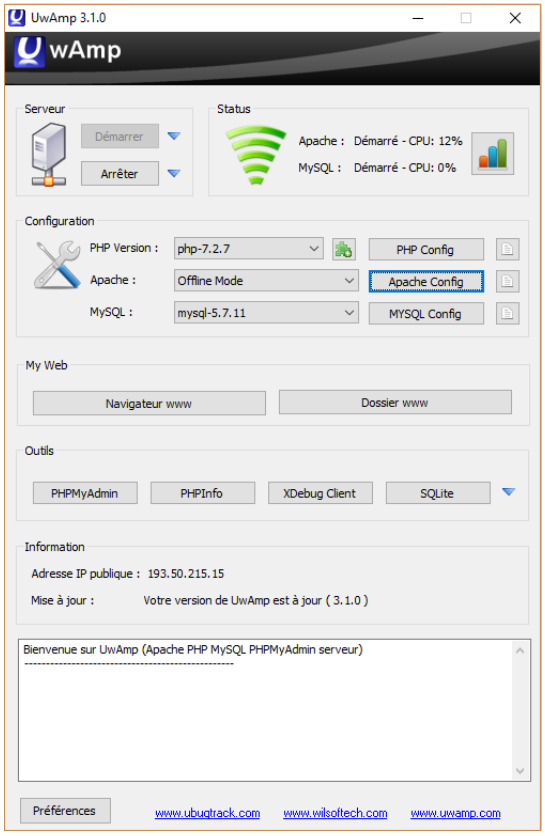
   ```
   composer create-project --prefer-dist laravel/laravel boat
   ```

   Cette commande crée un répertoire boat dans lequel elle va télécharger tous les composants nécessaires à Laravel.

   Création d'un nouveau projet

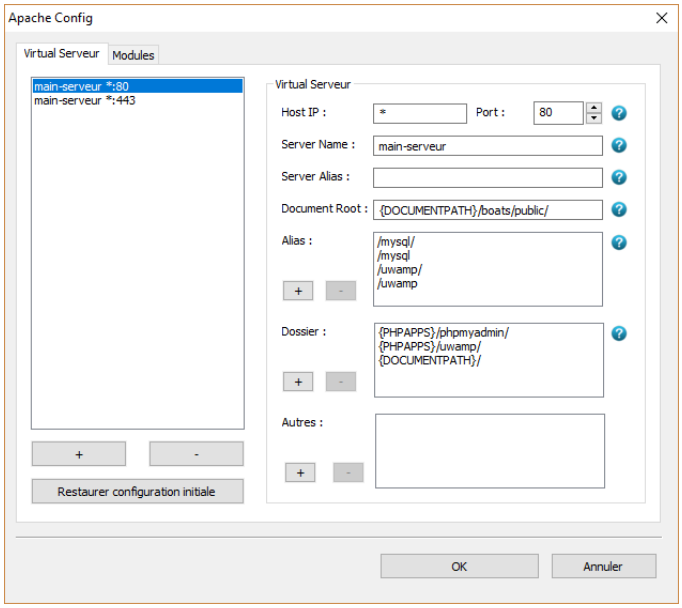
   ```
    composer create-project --prefer-dist laravel/laravel boat
   Installing laravel/laravel (v5.7.13)
     - Installing laravel/laravel (v5.7.13): Loading from cache
   Created project in boat
   > @php -r "file_exists('.env') || copy('.env.example', '.env');"
   Loading composer repositories with package information
   Updating dependencies (including require-dev)
   Package operations: 72 installs, 0 updates, 0 removals
     - Installing vlucas/phpdotenv (v2.5.1): Loading from cache
     - Installing symfony/css-selector (v4.1.7): Loading from cache

   .......

   > @php artisan key:generate --ansi
   Application key set successfully.
   ```

6. Configuration du répertoire root de votre site : Pour configurer le répertoire racine de votre serveur web sélectionnez le bouton de configuration d'Apache dans la fenêtre de contrôle de UwAmp.



Il suffit alors de faire pointer Document Root sur le répertoire public de Laravel :

`{DOCUMENTPATH}/boats/public/`



Si tout s'est passé normalement vous devez en tapant l'adresse `localhost` dans n'importe quel navigateur voir apparaître la page de test Laravel.

7. Installation de quelques extensions utiles de Laravel :
   — Laravel dispose d'un outil de débugage qui n'est pas installé par défaut mais qui va très vite se montrer indispensable, et qui s'installe en utilisant la commande suivante :

   ```
   composer require barryvdh/laravel-debugbar --dev
   ```

   Remarque :

   Avant de taper cette commande vous devez impérativement vous placer dans le répertoire de votre application (le répertoire boat dans notre exemple)

   Après que `composer` ait terminé l'installation des packages nécessaires, il faudra indiquer à Laravel de les prendre en compte en utilisant la commande :

   ```
   php artisan vendor:publish
   ```

8. Vous pouvez maintenant vérifier le bon fonctionnement de votre environnement en tapant dans votre navigateur préféré l'adresse de votre site (définie au moment de la configuration de UwAmp).

Nous pouvons maintenant commencer à construire notre application.

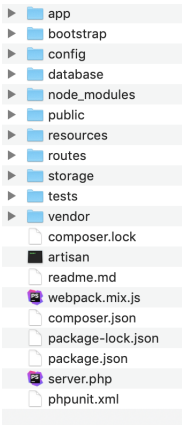
1.3 Exercice

Exercice 1

Mettre en place un environnement de travail Laravel et vérifier son bon fonctionnement.

2 Premiers pas

La création d'un nouveau projet Laravel produit différents fichiers et répertoires dont l'arborescence est visible sur la figure suivante. Il n'est pas nécessaire d'avoir une connaissance exhaustive de cette architecture pour construire notre première application. Nous ne décrirons donc ici que les éléments strictement nécessaires pour comprendre ce que nous faisons.



app contient l'essentiel du code que nous allons écrire pour construire le site. On y trouve trois répertoires `models`, `controller`, et `middleware` dont le rôle sera précisé dans la suite ;

artisan est un utilitaire en ligne de commande qui permet de générer automatiquement de nombreux fichiers, de connaître la liste des routes de l'application, et qui de manière générale nous évite beaucoup de travail fastidieux.

config est le répertoire dans lequel se trouvent tous les éléments de configuration de votre application Laravel comme les paramètres de connexion à la base de donnée ;

database abrite tous les fichiers de gestion de la base de données de votre application tels que les migrations ou les codes de peuplement de vos tables ;

resources contient les vues du projet ;

public est comme son nom l'indique le répertoire public de votre application. On y trouve en particulier le `.htaccess` ainsi que le fichier `index.php` qui est le contrôleur frontal de votre application ;

route contient le fichier `web.php` dans lequel on code les routes de l'application ;

vendor est le répertoire dans lequel se trouve le code du Framework.

Exercice 2

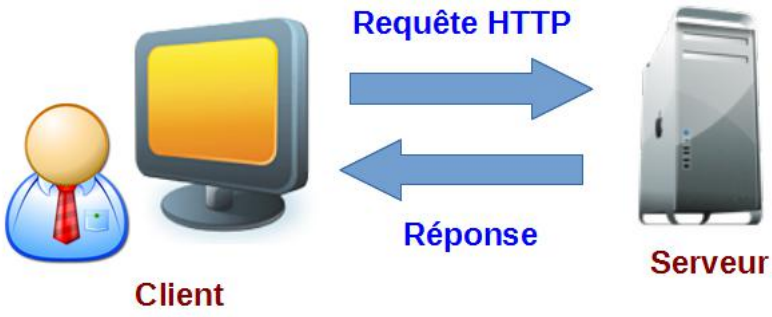
Précisez les répertoires dans lesquelles nous allons créer des fichiers php

2.1 Les routes

2.1.1 Principe de fonctionnement

Commençons par un petit rappel sur ce qu'est une requête HTTP.

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



La requête du client comporte un certain nombre d'informations mais nous allons nous intéresser pour le moment seulement à deux d'entre elles :

— la méthode : get, post, put, delete...

— l'url : c'est l'adresse de la page demandée sur le serveur

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

Lorsque vous avez testé votre installation de Laravel, vous êtes normalement arrivé sur la page d'accueil en tapant dans votre navigateur l'adresse `localhost` . Pour savoir comment nous arrivons à cette page, vous pouvez ouvrir le fichier `web.php`

Regardons ce que l'on y trouve :

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

Il est facile de 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 ;
- la fonction anonyme passée en paramètre à la méthode `get` retourne une vue (view) à partir du fichier "welcome".

On peut modifier cette première route pour créer notre propre affichage :

```php
Route::get('/',function() {return 'Hello World !!!';});
```

Exercice 3

Testez les différents exemples de ce paragraphe. Créez ensuite une route qui permette d'afficher Hello, suivi d'un message personnalisé (comme par exemple votre prénom)

2.1.2 Création de plusieurs routes, routes paramétrées

Il semble difficile de construire un site avec une unique page. Si nous souhaitons créer plusieurs pages, nous allons devoir créer plusieurs routes. Cela est très simple comme le montre l'exemple suivant :

Routes paramétrées

```php
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';});
```

Plutôt que d'écrire comme dans l'exemple précédent une route spécifique pour afficher trois pages quasiment identiques, il est possible de créer des routes paramétrées :

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

Par contre avec ce dernier exemple si l'internaute tape une adresse `localhost\nimportequoi`, laravel interpretera la route comme une route valide et affichera *Je suis la page nimportequoi*. Il faudrait donc pouvoir vérifier la validité du paramètre n. Cela peut se faire grâce à la méthode `where`

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

3 Les vues

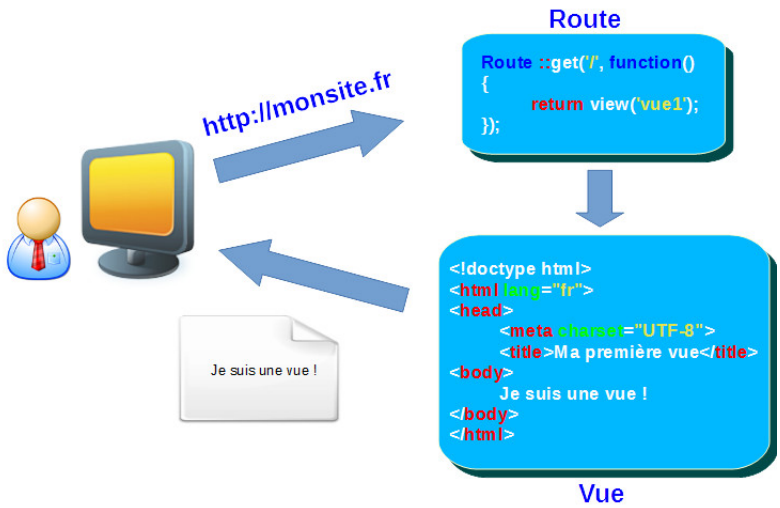
Dans une application réelle vous retournerez rarement la réponse directement à partir d'une route, vous passerez au moins par une vue. Dans sa version la plus simple une vue est un simple fichier avec du code html :

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
        <title>Ma première vue</title>
</head>
<body>
    Je suis une vue !
</body>
</html>
```

Enregistrez cette vue (en lui donnant le nom "vue1" par exemple) dans le dossier `resources/views` avec l'extension `php`. Maintenant pour afficher cette vue, il suffit d'indiquer dans le routeur comment y accéder :

```
<?php
Route::get('/', function()
{
    return view('vue1');
});
```

La figure suivante illustre les différentes étapes suivies par Laravel pour afficher cette vue :



3.1 Vues paramétrées

L'un des objectifs du modèle MVC est de construire des pages dynamiques, et l'exemple précédent ne présente qu'un intérêt limité.Mais comme nous l'avons vu pour les routes, Laravel nous donne également la possibilité de paramétrer les vues.

Supposons que l'on veuille répondre à la requête suivante :

`http://monsite.fr/article/n`

Le paramètre n pouvant prendre une valeur numérique . Cette url peut être analysée ainsi :

— la base de l'url est constante pour le site, quelle que soit la requête,
— la partie fixe ici correspond aux articles,
— la partie variable correspond au numéro de l'article désiré.

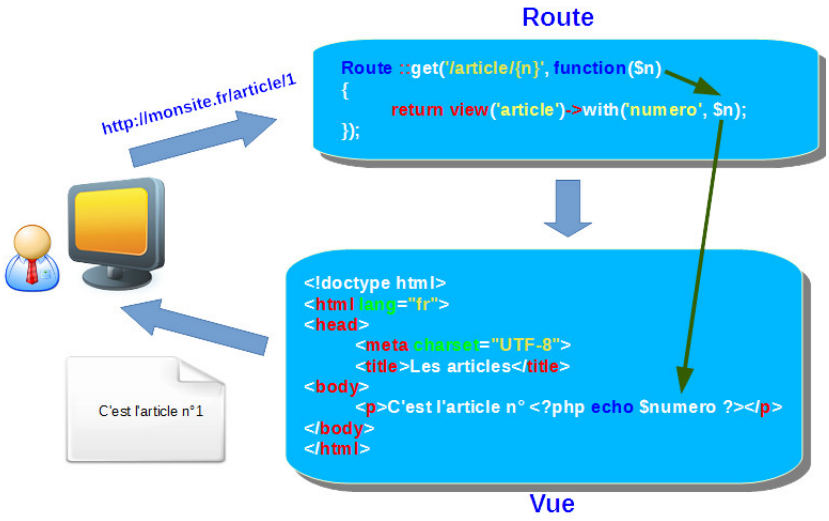
Nous devons alors définir dans le fichier `web.php` la route permettant d'afficher la vue paramétrée ainsi définie :

```php
<?php
Route::get('article/{n}', function($n) {
    return view('article')->with('numero', $n);
})->where('n', '[0-9]+');
```

Enfin, on sauvegarde dans le répertoire `resources\views` le fichier `article.php` suivant :

```php
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
        <title>Les articles</title>
</head>
<body>
        <p>C'est l'article n° <?php echo $numero ?></p>
</body>
</html>
```

Dans cet exemple, le paramètre est passé à la vue à travers la variable `$numero`. La figure suivante décrit les différentes étapes qui conduisent à l'affichage de cette vue.



3.2 Blade

Laravel possède un moteur de template élégant nommé Blade qui nous permet de faire pas mal de choses. La première est de nous simplifier la syntaxe. Par exemple au lieu de la ligne suivante que nous avons prévue dans la vue précédente :

```
<p>C'est l'article n° <?php echo $numero ?></p>
```

On peut utiliser cette syntaxe avec Blade :

```
<p>C'est l'article n° {{ $numero }}</p>
```

Tout ce qui se trouve entre les doubles accolades est interprété comme du code PHP. Mais pour que ça fonctionne il faut indiquer à Laravel qu'on veut utiliser Blade pour cette vue. Il suffit d'ajouter "blade" avant l'extension "php". Ainsi notre fichier devra-t-il être nommé `article.blade.php`.

3.3 Les template

Une fonction fondamentale de Blade est de permettre de faire du templating, c'est à dire de factoriser du code de présentation. Poursuivons notre exemple en complétant notre application avec une autre route chargée d'intercepter des urls pour des factures ainsi que sa vue :

```
<?php
Route::get('facture/{n}', function($n) {
    return view('facture')->withNumero($n);
})->where('n', '[0-9]+');
```

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Les factures</title>
</head>
<body>
        <p>C'est la facture n° {{ $numero }}</p>
</body>
</html>
```

On se rend compte que cette vue est pratiquement la même que celle pour les articles. Il serait intéressant de placer le code commun dans un fichier. C'est justement le but d'un template d'effectuer cette opération.

Voici alors le template :

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>@yield('titre')</title>
</head>
<body>
        @yield('contenu')
</body>
</html>
```

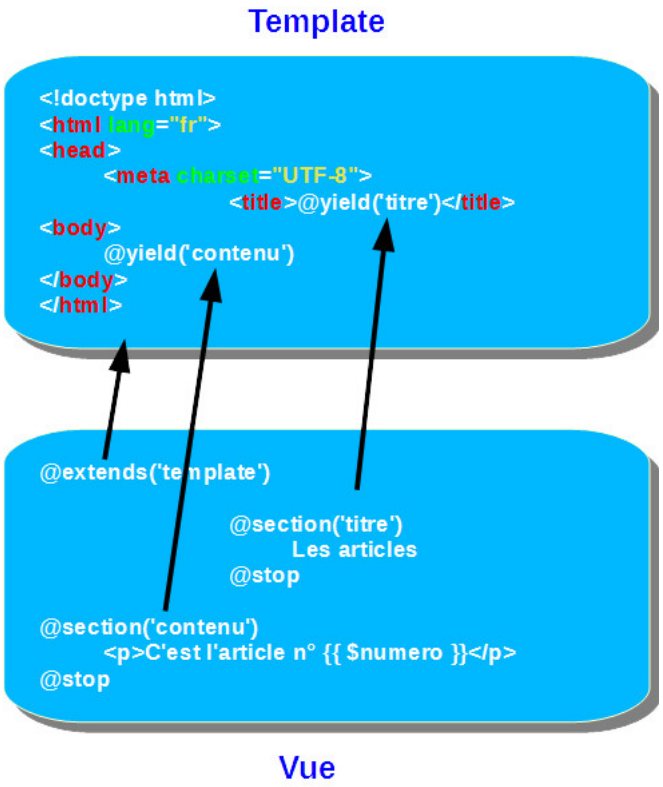
J'ai repris le code commun et prévu deux emplacements repérés par le mot clé @yield et nommés "titre" et "contenu". Il suffit maintenant de modifier les deux vues. Voilà pour les articles :

```
@extends('template')

@section('titre')
    Les articles
@endsection

@section('contenu')
    <p>C'est l'article n° {{ $numero }}</p>
@endsection
```

et pour les factures :

```
@extends('template')

@section('titre')
    Les factures
@endsection

@section('contenu')
    <p>C'est la facture n° {{ $numero }}</p>
@endsection
```

Dans un premier temps on dit qu'on veut utiliser le template avec@extends et le nom du template "template". Ensuite on remplit les zones prévues dans le template grâce à la syntaxe@section en précisant le nom de l'emplacement et en fermant avec@endsection (l'ancienne syntaxe@stopest encore fonctionnelle). Voici un schéma pour bien visualiser tout ça avec les articles :



Exercice 4

Testez les différnts exemples proposés dans ce paragraphe. Quel est l'intérêt de l'écriture d'un template ? Comment indique-t-on à Laravel de placer dans un template le contenu d'une variable php ?

4 Les contrôleurs

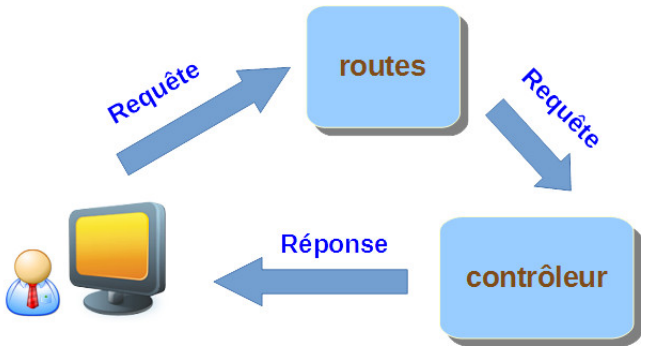
Nous avons vu le cycle d'une requête depuis son arrivée, son traitement par les routes et sa réponse avec des vues qui peuvent être boostées par Blade. Avec tous ces éléments vous pourriez très bien

réaliser un site web complet mais Laravel offre encore bien des outils performants que je vais vous présenter.

Pour bien organiser son code dans une application Laravel il faut bien répartir les tâches. Dans les exemples vus jusqu'à présent j'ai renvoyé une vue à partir d'une route, vous ne ferez jamais cela dans une application réelle (même si personne ne vous empêchera de le faire !). Les routes sont juste un système d'aiguillage pour trier les requêtes qui arrivent. Mais alors qui s'occupe de la suite ? Et bien ce sont les contrôleurs, le sujet de ce chapitre.

4.1 Rôle

La tâche d'un contrôleur est de réceptionner une requête (qui a déjà été triée par une route) et de définir la réponse appropriée, rien de moins et rien de plus. Voici une illustration du processus :



4.2 Constitution

Pour créer un contrôleur nous allons utiliser Artisan, la boîte à outils de Laravel. Dans la console entrez cette commande :

```
php artisan make:controller WelcomeController
```

Si tout se passe bien vous allez trouver le contrôleur `app\http\controllers`. `artisan` a déjà fait une partie du travail pour nous :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class WelcomeController extends Controller
{
    //
}
```

Nous n'avons plus qu'à ajouter nos propres méthodes dans ce contrôleur :

```php
<?php
...
class WelcomeController extends Controller
{
        public function index()
        {
                return view('welcome');
        }
}
```

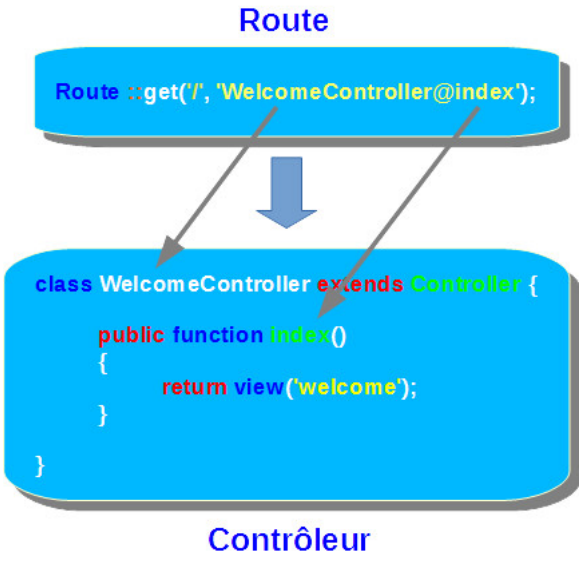
— on trouve en premier l'espace de nom ;
— le contrôleur hérite de la classe `Controller` qui se trouve dans le même dossier et qui permet de factoriser des actions communes à tous les contrôleurs ;
— on trouve enfin une méthode `index` qui renvoie quelque chose que maintenant vous connaissez : une vue, en l'occurrence "welcome" dont nous avons déjà parlé. Donc si j'appelle cette méthode je retourne la vue "welcome" au client.

4.3 Liaison avec les routes

Maintenant la question qu'on peut se poser est : comment s'effectue la liaison entre les routes et les contrôleurs ? Ouvrez le fichier des routes et entrez ce code :

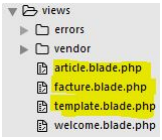
```php
<?php
Route::get('/', 'WelcomeController@index');
```

Voici une visualisation de la liaison entre la route et le contrôleur :



4.4 Utilisation d'un contrôleur

Voyons maintenant un exemple pratique de mise enœuvre d'un contrôleur. On va conserver notre exemple avec les articles mais maintenant traité avec un contrôleur. On conserve le même template et les mêmes vues :



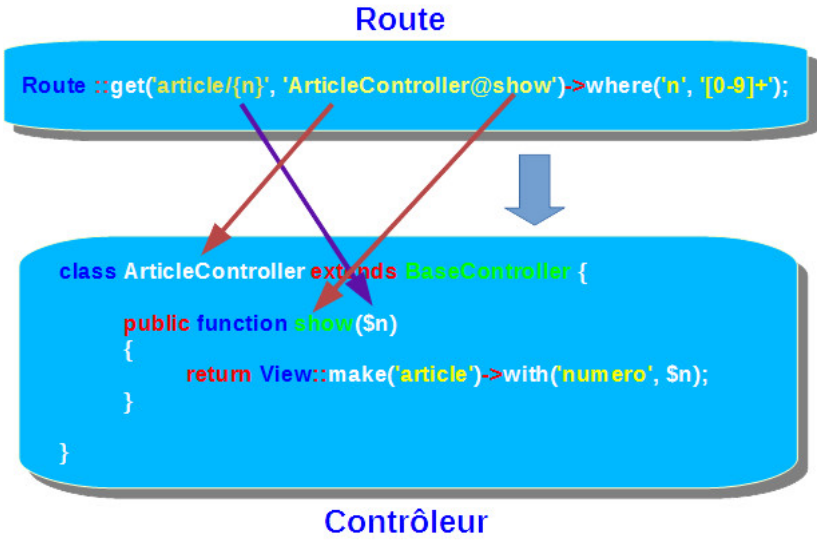
On va créer un contrôleur (entraînez-vous à utiliser Artisan) pour les articles :

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

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class ArticleController extends Controller
{

    public function show($n)
        {
                return view('article')->with('numero', $n);
        }
}
```

Dans ce contrôleur on a une méthode show chargée de générer la vue. Il ne nous reste plus qu'à créer la route :

```php
<?php
Route::get('article/{n}', 'ArticleController@show')->where('n', '[0-9]+');
```

Voici une illustration du fonctionnement avec un contrôleur :



Exercice 5

Après avoir testé les différents exemples proposés dans ce paragraphe, expliquez le rôle et le fonctionnement d'un contrôleur Laravel