TP n°6 - Introduction à Laravel

Site: Université de Reims Champagne-Ardenne
Cours: INFO0303 - Technologies Web 2
Livre: TP n°6 - Introduction à Laravel

Imprimé par: CYRIL RABAT
Date: mercredi 4 octobre 2023, 15:16

Table des matières

1. Pour commencer

Dans ce sujet, toutes les instructions concernant l'installation de *Laravel* et des applications tierces sont basées sur *Windows* avec *WampServer*. Pour les étudiants utilisant un autre système ou d'autres applications, merci de vous référer aux guides correspondants (*Google is your friend!*). Pour ce TP, nous avons besoin des applications suivantes (nous allons décrire leur installation dans la suite) :

- *WampServer* : une distribution contenant un serveur Web, le moteur PHP, un SGBD
- La version 8.2 de PHP (8.2.10 au moment de la rédaction de ce support)
- *composer* : un gestionnaire de dépendances (version 2.6.3 au moment de la rédaction de ce support)

2. Configuration du système

Pour pouvoir utiliser *Laravel*, il faut déjà s'assurer que l'ensemble des éléments nécessaires sont installés et configurés sur votre machine.

2.1. Configuration de PHP

Jusqu'à présent, les scripts PHP étaient exécutés dans le contexte du serveur web, via le navigateur. Pour *Laravel*, nous avons besoin également du CLI PHP (la commande `php` dans l'invite de commandes). Pour cela, il faut que le chemin vers `php.exe` soit renseigné dans la variable d'environnement `Path`. Avec une installation par défaut, il faut ajouter le chemin suivant : `C:\wamp64\bin\php\php8.2.10.` (*WampServer* ne sera pas content, mais tant pis pour lui)

1. Si ce n'est pas déjà fait, modifiez la variable d'environnement `Path` en ajoutant le chemin précédent.

2. Ouvrez l'invite de commandes et vérifiez que la commande `php` est fonctionnelle et qu'elle affiche la bonne version. Pour cela, tapez la commande suivante qui affiche la version de PHP :

```
php -v
```

Pour utiliser *Laravel*, il est nécessaire d'activer des extensions (qui sont normalement activées par défaut sous *WampServer*).

3. Tapez la commande suivante :

```
php -m
```

Vous devriez avoir au moins :

- bz2
- curl
- fileinfo
- gd
- mbstring
- mysqli
- openssl
- pdo_mysql

4. Si ce n'est pas le cas, activez les extensions depuis le menu de *WampServer* (ou dans le fichier « php.ini »).

2.2. Installation de composer

L'installation de *Laravel* utilise le gestionnaire de dépendances *composer* (qui est très utilisé pour les applications PHP) qui est lui-même écrit en PHP.

1. Installez *composer* sur votre machine. Suivez le guide sur la page suivante : https://getcomposer.org/download/ (sous *Windows*, téléchargez le programme « Composer-Setup.exe »).

2. Une fois l'installation terminée, vérifiez que *composer* est bien installé en tapant la commande suivante qui affiche la version :

```
composer -V
```

Si un des éléments présentés (PHP ou *composer*) n'est pas fonctionnel, il est inutile de passer à la suite.

3. Premier projet Laravel

Dans cette section nous allons créer un projet *Laravel*. Nous allons utiliser différentes commande : n'hésitez pas à les noter quelque part...

3.1. Création du projet

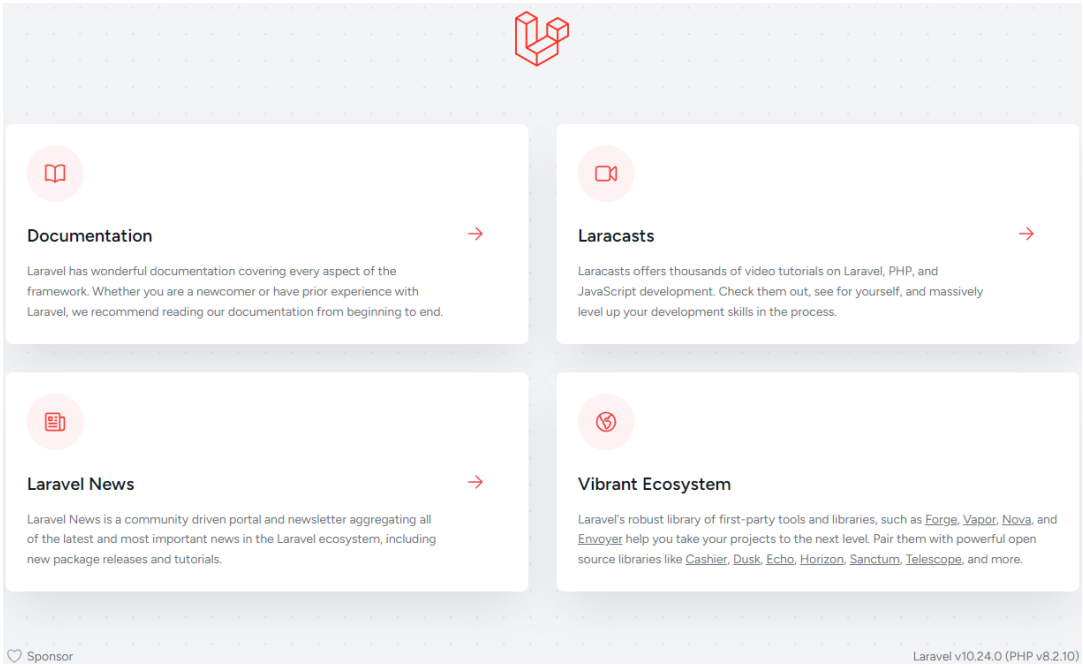
1. Créez un projet *Laravel* dans le répertoire `www` de *WampServer* en tapant la commande suivante :

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

L'installation de *Laravel* démarre, ce qui peut prendre plusieurs minutes.

2. Une fois terminée, vous pouvez accéder à l'application créée en tapant l'URL suivante : http://localhost/exemple/public/ (pour rappel, le routeur est situé dans le répertoire `public` de votre projet).

Vous devriez obtenir la page d'accueil par défaut de *Laravel* (ici, la version 10.24.0 mais qui peut changer rapidement...) :



3.2. Configuration d'un hôte virtuel Apache

> Cette partie explique comment utiliser l'URL `http://exemple/` au lieu de `http://localhost/exemple/public`. Vous pouvez passer cette partie si vous rencontrez des problèmes techniques.

Il est possible de définir des hôtes virtuels sous *Apache* pour simplifier l'accès à certaines applications. C'est le cas par exemple de localhost (le seul hôte virtuel par défaut). Nous ne décrirons par leur fonctionnement dans ce TP, mais ils permettent, dans notre cas, d'accéder directement à l'application via l'URL : http://exemple/.

1. Dans le menu de *WampServer* (clic gauche sur l'icône dans le centre de notification, en bas à droite), cliquez sur « Vos VirtualHosts » puis « Gestion VirtualHost » (par défaut, cela emmène à l'URL http://localhost/add_vhost.php).
2. Pour le nom du Virtual Host, saisissez « exemple » puis spécifiez le chemin complet jusqu'au répertoire « public » de votre application *Laravel* (avec une installation par défaut, cela devrait être « C:\wamp64\www\exemple\public »).
3. Cliquez sur le bouton « Démarrer la création ou la modification du VirtualHost ».
4. Cliquez maintenant avec le bouton droit sur l'icône de *WampServer*. Sélectionnez « Outils » puis « Redémarrage DNS »

Une fois les services redémarrés, l'URL http://exemple/ doit afficher la page principale de votre application *Laravel*.

> Avec un hôte virtuel, vous pouvez stocker votre application *Laravel* dans n'importe quel répertoire de votre ordinateur, même en dehors du répertoire `www` de *WampServer*.

3.3. Installation de la barre de debug

Par défaut, dès que des erreurs d'exécution sont détectées, plusieurs fenêtres s'affichent dans le navigateur indiquant l'endroit où l'erreur a été générée. Parfois, il peut être compliqué de trouver le chemin exact du script où l'erreur a été générée (parfois, les erreurs entraînent des erreurs dans les scripts de *Laravel*). Il est donc conseillé d'installer la barre de debug dans votre application.

1. Avec l'invite de commandes, allez dans le répertoire de l'application (`www/exemple/`) et tapez la commande suivante :

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

Cela installe une barre de debug. Pour qu'elle soit prise en compte dans *Laravel*, il est parfois nécessaire de taper la commande suivante :

```
php artisan vendor:publish
```

Vous devrez sélectionner le numéro correspondant à la barre de debug (généralement 1).

2. Rechargez la page principale de votre application *Laravel*. Vous devriez voir apparaître la barre ci-dessous :

Nous pourrons tester la barre dans un prochain exercice.

4. Récupérer un projet Laravel depuis un dépôt git

Une application *Laravel* peut être sauvegardée depuis un dépôt *git* pour être ensuite déployée sur une autre machine (le serveur de production, par exemple). Par défaut, la plupart des fichiers ne sont pas stockés dans le dépôt pour une question de taille et pour une question de sécurité (les fichiers de configuration avec les identifiants de la base de données ne sont pas déposés, par exemple). Pour cela, *Laravel* ajoute un fichier « .gitignore » à la racine du projet (éditez-le par curiosité). Tous les répertoires temporaires ou pouvant être regénérés sont ignorés.

> Une mauvaise pratique fréquente est de supprimer le fichier « .gitignore » pour envoyer tous les fichiers sur le dépôt. C'est inutile. Lorsque vous devez synchroniser sur plusieurs machines, l'installation ne doit être effectuée qu'une seule fois ; les prochaines mises-à-jour conserveront les répertoires créés et le fichier de configuration. Ensuite, c'est dangereux en termes de sécurité et coûteux en termes de bande passante et de stockage (pensez à la planète !).

Un dépôt d'exemple a été créé à cet URL : https://gitlab-mi.univ-reims.fr/rabat01/laravel

1. Placez-vous dans le répertoire www de *WampServer* puis tapez la commande suivante (remplacez « toto » par un nom de répertoire quelconque).

```
git clone https://gitlab-mi.univ-reims.fr/rabat01/laravel toto
```

L'application récupérée ne fonctionne pas car il manque des éléments. Nous allons les régénérer.

2. Dans le répertoire « toto », tapez les commandes suivantes :

Pour installer les modules utilisés :

```
composer install
```

Pour créer une copie du fichier de configuration par défaut :

```
copy .env.example .env
```

Pour créer une clé de chiffrement :

```
php artisan key:generate
```

3. Vérifiez que l'application est fonctionnelle.

Vous pouvez supprimer cette nouvelle application histoire de faire un peu de place...

5. Premiers pas avec Laravel

Les différentes sections présentent quelques éléments de base de *Laravel*.

5.1. Premières vues

Pour rappel, les routes sont spécifiées dans le script `web.php` situé dans le répertoire `routes`. Par défaut, le code suivant permet de spécifier la vue affichée par défaut (URL `http://exemple/`) correspondant à la page de Laravel :

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

Nous allons créer une nouvelle vue à la place de *welcome* (celle qui affiche la version de *Laravel*).

1. Créez une vue dans le répertoire `ressources/views/` nommée `index.php`. Dans ce fichier, placez-y la structure d'un document HTML quelconque.

2. Modifiez le script `web.php` pour que votre page s'affiche par défaut (à la place de *welcome*). Vérifiez dans votre navigateur (vous pouvez supprimer la vue *welcome*).

3. Ajouter du code PHP dans `index.php` (n'oubliez pas les balises) et tapez du code incorrect. Vérifiez ce qui est affiché dans votre navigateur et dans la barre de debug.

4. Supprimez le code PHP incorrect.

5. Créez une nouvelle vue (`page1.php`) et modifiez le script `web.php` pour que la route `/page1` emmène vers cette vue.

6. Sur la vue `index.php` ajoutez un lien vers `page1` (si vous avez configuré un *Virtual Host*, vous pouvez spécifier aussi bien `/page1` que `page1` ; préférez la deuxième possibilité pour une question de portabilité). Vérifiez que vous pouvez bien accéder à cette page depuis la page d'accueil.

7. Ajouter un lien vers la page d'accueil sur `page1.php`. Pour cela, vous pouvez utiliser la fonction `url` :

```
<a href="<?php echo url('/'); ?>">Retour à l'accueil</a>
```

La fonction `url` vous permet également de récupérer l'URL courante, l'URL précédente, etc.

Lorsque vous utilisez un hôte virtuel, vous pouvez avoir l'impression que les deux syntaxes suivantes sont équivalentes : `<a href="/">Accueil</a>` et `<a href="<?php echo url('/'); ?>">Accueil</a>`. Ce n'est pas le cas. Sans hôte virtuel, la première retourne vers `http://localhost/` alors que la seconde vers `http://localhost/exemple/public/`. C'est pourquoi vous devez utiliser systématiquement la seconde.

5.2. Interactions avec la barre de debug

Il est possible d'interagir dans votre code avec la barre de *debug* à l'aide des méthodes de la classe `Debugbar`.

1. Dans la vue « page1.php », testez les méthodes `info`, `error` et `warning` en affichant un message quelconque puis en regardant dans l'onglet « Messages » de la barre de *debug*.

2. Testez le code suivant et vérifiez dans la section « Exceptions » de la barre de *debug* :

```
try {
 throw new Exception("un exemple d'exception");
} catch (Exception $e) {
 Debugbar::addThrowable($e);
}
```

3. Testez le code suivant et vérifiez le résultat dans la section « Timeline » de la barre de *debug* :

```
Debugbar::startMeasure('render','Exemple 1');
Debugbar::stopMeasure('render');
Debugbar::addMeasure('maintenant', LARAVEL_START, microtime(true));
Debugbar::measure('Une longue opération', function() {
    $cpt = 0;
    for($i = 0; $i < 1000; $i++)
      for($j = 0; $j < 1000; $j++)
        $cpt += $i + $j;
  });
```

5.3. Vues paramétrées et images

Dans *Laravel*, il est possible de créer des vues paramétrées. Cela permet de passer une valeur dans l'URL et la récupérer dans la vue, tout en assurant le contrôle dans le routeur.

1. Écrivez une vue `item.php`. Avec du PHP, affichez le contenu de la variable `$number`.

2. Ajoutez le code suivant dans `web.php` (l'appel à `name` permet de nommer cette route) :

```
Route::get('item/{n}', function($n) {
        return view('item')->with('number', $n);
    })->where('n', '[0-9]+')->name('item.show');
```

La vue est maintenant affichée lorsqu'on saisit l'URL suivante : `item/X` (où X est un entier).

3. Vérifiez maintenant que vous pouvez accéder à cette vue et que le numéro correspond bien à celui indiqué dans l'URL. Vérifiez également le comportement de *Laravel* si vous saisissez autre chose qu'un entier.

4. Spécifiez maintenant une page d'erreur à l'aide de la route suivante (et créez la vue `error.php` correspondante) :

```
Route::fallback(function () {
        return view('error');
    });
```

Il est souvent important de pouvoir afficher une image de chat dans la page d'erreur (pour apaiser l'utilisateur énervé). Par défaut, les ressources accessibles publiquement doivent être stockées dans le répertoire « storage/app/public/ ».

5. Récupérez une image quelconque à afficher dans la vue d'erreur et placez-la dans le sous-répertoire « images » (vous pouvez supprimer les fichiers `.gitignore` des répertoires « app » et « public » si vous souhaitez que les ressources soient stockées dans votre dépôt *gitlab*).

6. Pour que le répertoire puisse être accessible par le navigateur, tapez la commande suivante dans le répertoire de votre application :

```
php artisan storage:link
```

7. Vérifiez que dans le répertoire « public » (situé à la racine de votre application), il y a bien un lien vers le répertoire « storage » qui a été créé.

8. Ajoutez la balise HTML dans votre vue d'erreur :

```
<img src="<?php echo asset('storage/images/chat.jpg'); ?>" title="Une image de chat"/><br/>
```

La fonction `asset` vous permet de créer un lien vers une ressource de votre application.

La gestion des URL peut être complexe. C'est pourquoi la fonction `url` est importante. Si on veut accéder à l'article numéro 1 depuis l'accueil, on peut utiliser l'instruction suivante :

```
url('/item/1')
```

Une autre solution consiste à utiliser le nom des routes. Lors de la déclaration de la route paramétrée, nous lui avons donné le nom `item.show` à l'aide de l'appel à la méthode `name`.

9. Dans la page d'accueil, ajoutez un lien vers l'article numéro 1 à l'aide de l'instruction suivante :

```
<a href="<?php echo route('item.show', ['n' => 1]); ?>">Article n°1</a>
```

10. Dans la page de l'article, ajoutez un lien vers l'article précédent (seulement si le numéro courant est supérieur à 1) et l'article suivant.

5.4. Introduction à Blade

Pour rappel, *Laravel* propose un moteur de *template* nommé *Blade*. Il permet de créer des *templates* qui peuvent ensuite être utilisés dans des vues.

1. Renommez la vue `item.php` en `item.blade.php` (pour que le code soit interprété par *Blade*). Modifiez la vue et affichez le contenu de la variable `number` en utilisant la notation {{ }} (les balises PHP et l'appel à `echo` ne sont plus utiles).

2. Créez maintenant un *template* nommé `template.blade.php`. Définissez 2 sections : `title` (le titre de la page) et `content` (le contenu de la page). Pour cela, il suffit d'ajouter le code suivant (sans balise PHP) dans le code HTML à l'endroit où l'on souhaite ajouter les données (remplacez `nomSection` par `title` et `content`):

```
@yield('nomSection')
```

3. Modifiez `item.blade.php`. La vue doit hériter de ce *template* (avec `@extends('template')`). Vous ne conservez que le titre de la page et le contenu. Ils doivent être définis comme suit :

```
@section('nomSection')
Le contenu de la section
@endsection
```

4. Vérifiez que la page s'affiche comme avant.

5. Modifiez les 3 autres vues de la même manière.

6. Modifiez la vue `item.blade.php` et utilisez la syntaxe de *Blade* pour écrire la conditionnelle (l'affichage du lien vers l'article précédent si l'identifiant est supérieur à 1).

7. De même, modifiez toutes les vues pour utiliser la syntaxe {{ et }} : vous ne devriez plus avoir de balise PHP, ni d'appel à la fonction `echo`.

5.5. Les contrôleurs

Comme vu en CM et en TD, il est préférable de créer des contrôleurs pour structurer votre application et simplifier le code du routeur.

1. Exécutez la commande suivante (dans le répertoire de l'application):

```
php artisan make:controller ItemController
```

Vous devriez avoir un fichier `ItemController.php` dans le répertoire `app/Http/Controllers`. Par défaut, il n'y a aucune méthode dans la classe.

2. Ajoutez la méthode index comme suit :

```
public function index() {
    return view('index');
}
```

3. Dans le script `web.php`, modifiez la route principale comme suit :

```
Route::get('/', [App\Http\Controllers\ItemController::class, 'index']);
```

Pour rappel, si vous ajoutez l'instruction `use App\Http\Controllers\ItemController;` dans le script `web.php`, vous pouvez écrire simplement `ItemController` au lieu du nom complet (avec les espaces de nom).

4. Dans le contrôleur, ajoutez la méthode `show` qui prend en paramètre un entier `n`. Elle appelle la vue paramétrée `item.blade.php` comme vu précédemment.

5. Modifiez la route comme suit (notez l'expression régulière) :

```
Route::get('item/{n}',
        [App\Http\Controllers\ItemController::class, 'show']
    )->where('n', '[0-9]+')->name('item.show');
```

5.6. Pour finir

Consultez la documentation *Laravel* pour voir les autres possibilités offertes par *Blade* :

https://laravel.com/docs/10.x/blade