MISSION LARAVEL en PHP : TP découverte

Savoirs et savoir-faire développés :
Utiliser le framework « Laravel », Utiliser l'outil d'ORM "eloqent" en PHP

A4.1.2	Conception ou adaptation de l'interface utilisateur d'une solution applicative
A4.1.2	Conception ou adaptation de l'interface utilisateur d'une solution applicative
A4.1.3	Conception ou adaptation d'une base de données
A4.1.4	Définition des caractéristiques d'une solution applicative
A4.1.5	Prototypage de composants logiciels
A4.1.6	Gestion d'environnements de développement et de test
A4.1.7	Développement, utilisation ou adaptation de composants logiciels

Source de préparation du cours :
https://openclassrooms.com/fr/courses/3613341-decouvrez-le-framework-php-laravel/
https://github.com/c4software/bts-sio/
https://phptherightway.com/
https://laravel.com/
https://laravel.sillo.org/
Livre : Laravel par Raphaël Huchet aux éditions ENI

Pré-requis :
Eclipse pour PHP ou visual studio Code
Wamp server 3(MariaDB 10.1.37 - Apache 2.4.37 - MySQL 5.5.62, 5.6.42, 5.7.24, 8.0.13 - PHP 7.1.23, 7.2.11 - Wampserver 3.1.4)

Gestion de Projet :

Pour ce cours, nous utiliserons un outil de suivi collaboratif SLACK.
Vous pouvez rejoindre le panneau de ce cours : https://btssaintsauveur.slack.com/

ÉTAPE 1 : TÉLÉCHARGEMENT DE L'OUTIL "Composer"

Récupérer l'exécutable de composer sur le NAS (disponible sur **https://getcomposer.org/Composer-Setup.exe)** et l'exécuter. L'installateur vous demandera où se trouve `php.exe` (en général : C:\wamp\bin\php\phpX.X.X). Cela installe **Composer** et met à jour le PATH de façon à ce que l'on puisse taper la commande **composer** en ligne de commande depuis n'importe quel répertoire.

Si erreur "The openssl extension is missing which means that secure https transfer are impossible"
activer la ligne **extension=php_openssl.dll** en visualisant les extensions PHP de votre WAMP.

Composer a besoin d'un fichier `composer.json` associé. Ce fichier contient les instructions pour Composer : les dépendances, les classes à charger automatiquement... Voici un extrait de ce fichier pour Laravel :

```
{
    "name": "laravel/laravel",
    "description": "The Laravel Framework.",
    "keywords": ["framework", "laravel"],
    "license": "MIT",
    "type": "project",
    "require": {
        "php": ">=5.5.9",
        "laravel/framework": "5.2.*"
    },
    ...
}
```

Composer fonctionne en ligne de commande. Pour vérifier la bonne installation de votre composer, vous pouvez taper dans votre invite de commande `Composer`

ÉTAPE 2 : Installation de LARAVEL

Lancer la commande suivante :

```
composer global require "laravel/installer"
```

Vérifier que la commande laravel fonctionne. Pour vérifier que la commande `laravel` fonctionne il suffit de faire dans une console `laravel -h`

```
c:\wamp\www\Laravel>laravel -h
Usage:
  list [options] [--] [<namespace>]

Arguments:
  namespace            The namespace name

Options:
      --raw            To output raw command list
      --format=FORMAT  The output format (txt, xml, json, or md) [default: "txt"]

Help:
  The list command lists all commands:

    php C:\Users\Sophie\AppData\Roaming\Composer\vendor\bin\/../laravel/installer/laravel list

  You can also display the commands for a specific namespace:

    php C:\Users\Sophie\AppData\Roaming\Composer\vendor\bin\/../laravel/installer/laravel list test

  You can also output the information in other formats by using the --format option:

    php C:\Users\Sophie\AppData\Roaming\Composer\vendor\bin\/../laravel/installer/laravel list --format=xml

  It's also possible to get raw list of commands (useful for embedding command runner):

    php C:\Users\Sophie\AppData\Roaming\Composer\vendor\bin\/../laravel/installer/laravel list --raw
```

ÉTAPE 3 : Création du projet LARAVEL

Votre poste est maintenant configuré pour Laravel, vous pouvez donc créer un nouveau projet grâce à la commande ci dessous:

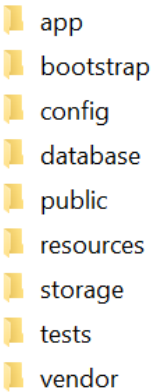
✋ Attention, le projet sera créé dans le dossier courant.

Déplacer votre invite de commande dans le bon répertoire :

```
CD C:\\wamp\www\
```

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

Après quelques minutes, on obtient l'arborescence suivante :



Avec un peu d'explications …

Dans le répertoire **\app**, on trouve les dossiers suivants.

- **Console/Commands** : toutes les commandes en mode console, il y a au départ une commande `Inspire` qui sert d'exemple,
- **Jobs :** commandes concernant les tâches que doit effectuer l'application.
- **Events** et **Listeners : événements et écouteurs** nécessaires pour l'application,
- **Http** : tout ce qui concerne la communication : contrôleurs, routes, middlewares (il y a quater middlewares de base) et requêtes,
- **Providers** : tous les fournisseurs de services (providers), il y en a déjà 4 au départ. Les providers servent à initialiser les composants.
- **Policies** : permet de gérer facilement les droits d'accès.

Les autres dossiers suivants :

- **bootstrap** : scripts d'initialisation de Laravel pour le chargement automatique des classes, la fixation de l'environnement et des chemins, et pour le démarrage de l'application,
- **public** : tout ce qui doit apparaître dans le dossier public du site : images, CSS, scripts...
- **vendor** : tous les composants de Laravel et de ses dépendances,
- **config** : toutes les configurations : application, authentification, cache, base de données, espaces de noms, emails, systèmes de fichier, session...
- **database** : migrations et les populations,
- **resources** : vues, fichiers de langage et assets (par exemple les fichiers LESS ou Sass),
- **storage** : données temporaires de l'application : vues compilées, caches, clés de session...
- **tests** : fichiers de tests unitaires.

Et à la racine :

- **artisan** : outil en ligne de Laravel pour des tâches de gestion,
- **composer.json** : fichier de référence de Composer,
- **phpunit.xml** : fichier de configuration de phpunit (pour les tests unitaires),
- **.env** : fichier pour spécifier l'environnement d'exécution.

Pour vérifier que tout est fonctionnel, vous pouvez aller visualiser la partie public.
http://localhost/monPojet/public/

Initialisation

Votre nouveau projet contient un fichier `.env` ouvrez le, et ajouter en début de fichier
`APP_NAME=monProjet`

QUESTIONS :

Ou se trouve le fichier des variables d'environnement de l'application ?

A quoi sert la variable `APP_DEBUG` ?

A quoi sert la variable `APP_KEY` ?

A quoi sert le fichier .htacess du projet ?

Lancer le projet d'exemple

Laravel intègre un serveur de test permettant de valider son développement avec rien d'autre que PHP sur sa machine. Pour ça dans le dossier du projet (`cd monProjet`) vous pouvez faire la commande suivante :

```
CD c:\\wamp\www\monProjet
php artisan serve
```

Rendez-vous maintenant dans votre navigateur pour voir le site de démonstration fourni par Laravel.
http://localhost:8000/

ÉTAPE 4 : Modifier le template par défaut

Passer en mode éditeur Eclipse, visual Studio Code, notePad ++ ou sublim Text (au pire !).

Éditer le fichier `resources/views/welcome.blade.php`, ajouter la variable `$titre`. La syntaxe « blade » est la suivante `{{ $titre }}`. À la ligne 84 ajouter après Laravel « `{{ $titre }}` ». Vous avez défini votre première variable c'est bien ! Mais pour l'instant rien ne se passe… Pour que quelques choses s'affiche :

Éditer le fichier `routes/web.php`, transformer :

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

en

```
return view('welcome', ['titre' => 'Mon premier exemple.']);
```

❗ Vous pouvez également appeler des fonctions dans les templates, exemple `{{ time() }}`. Tester cette fonction en ajoutant :

```
<p>Le Timestamp est {{ time() }}</p>
```

QUESTIONS :

À votre avis est-il possible d'appeler d'autre fonctions ?

ÉTAPE 4 : Créer des routes

Laravel permet que des adresses du type `/index.php ?page=articles&id=123` soit remplacées par des adresses plus expressives comme `/articles/monArticle`

Laravel utilise 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.

La liste des routes se trouve dans le fichier `routes/web.php` d'un projet Laravel. Il faut alimenter ce fichier au fur et à mesure de l'ajout de nouvelles pages sur le site.

Pour tester le fonctionnement nous allons ajouter une nouvelle `Route` dans le projet de démonstration. Nous allons donc ajouter dans le fichier `routes/web.php` :

```
Route::get('/ping', function () {
    return "pong";
});
```

Tester la modification en accédant à votre site

QUESTIONS :

Quelle URL avez-vous utilisé ?

Pensez qu'il est possible d'obtenir des arborescences plus complexes ?

ÉTAPE 5 : Ajouter une nouvelle vue

Maintenant que nous avons déclaré une nouvelle route, nous allons revoir légèrement les templates pour :

- Déclarer un template principal (aussi appelé : layout).
- Modifier le welcome.blade.php pour y faire référence.
- Utiliser le layout pour répondre `pong`.

QUESTIONS :

A votre avis, pourquoi un tel découpage ?

Créer le layout

Créer un nouveau fichier `resources/views/layouts/base.blade.php` avec le contenu suivant :

```
<!doctype html>
<html lang="{{ app()->getLocale() }}">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <title>Laravel - @yield('title')</title>

        <!-- Fonts -->
        <link href="https://fonts.googleapis.com/css?family=Raleway:100,600"
rel="stylesheet" type="text/css">

        <!-- Styles -->
        <style>
```

```
            html, body {
                background-color: #fff;
                color: #636b6f;
                font-family: 'Raleway', sans-serif;
                font-weight: 100;
                height: 100vh;
                margin: 0;
            }
            .full-height {
                height: 100vh;
            }
            .flex-center {
                align-items: center;
                display: flex;
                justify-content: center;
            }
            .position-ref {
                position: relative;
            }
            .top-right {
                position: absolute;
                right: 10px;
                top: 18px;
            }
            .content {
                text-align: center;
            }
            .title {
                font-size: 84px;
            }
            .links > a {
                color: #636b6f;
                padding: 0 25px;
                font-size: 12px;
                font-weight: 600;
                letter-spacing: .1rem;
                text-decoration: none;
                text-transform: uppercase;
            }
            .m-b-md {
                margin-bottom: 30px;
            }
        </style>
    </head>
    <body>
        <div class="flex-center position-ref full-height">
            @if (Route::has('login'))
                <div class="top-right links">
                    @auth
                        <a href="{{ url('/home') }}">Home</a>
                    @else
                        <a href="{{ route('login') }}">Login</a>
                        <a href="{{ route('register') }}">Register</a>
                    @endauth
                </div>
            @endif

            <div class="content">
                @yield('content')
            </div>
        </div>
    </body>
</html>
```

✋ Mais d'où vient ce contenu ?
C'est tout simplement un découpage en « layout » du template de base de démonstration.

QUESTIONS :

A votre avis, à quoi sert le mot clef @yield?

Utiliser le layout dans welcome.blade.php

Documentation sur les templates : https://laravel.com/docs/5.7/blade

Maintenant que nous avons notre template de base nous allons l'utiliser dans le template « Welcome ». Remplacer le contenu de `resources/views/welcome.blade.php` par :

```
@extends('layouts.base')

@section('title', 'Bienvenue')

@section('content')
   <div class="title m-b-md">
        Laravel
    </div>

    <div class="links">
        <a href="https://laravel.com/docs">Documentation</a>
        <a href="https://laracasts.com">Laracasts</a>
        <a href="https://laravel-news.com">News</a>
        <a href="https://forge.laravel.com">Forge</a>
        <a href="https://github.com/laravel/laravel">GitHub</a>
    </div>
@endsection
```

➔ Vérifier votre travail !

Utiliser le layout dans la route Pong

Bon, maintenant que nous avons déclaré un layout utilisons-le dans la 2nd route (/ping) que nous avons créé tout à l'heure. Pour cette dernière action je ne vous donne pas de code, mais uniquement les étapes :

- Créez une Vue par exemple `ping.blade.php` (Dans le dossiers `views`)
- Utilisez `@extends('base')` pour « hériter » de votre layout principal.
- Modifiez `web.php` pour répondre avec la fonction `view` comme dans l'autre route.

Avec ces quelques explications vous allez pouvoir atteindre l'objectif. Bon courage.