Classe : Mastère professionnel DWFE
Matière : Développement WEB côté serveur et frameworks
Enseignant : Slim NAMOUCHI

Configuration de PHPStorm avec Symfony 3.2 et création du 1er projet

Introduction

Symfony est un framework MVC qui va vous permettre de réaliser des sites **complexes** rapidement, mais de façon **structurée** et avec un code clair et **maintenable**. Par contre nous avons dit que **CodeIgniter** permet de développer des sites **simples**.

Objectifs

Le but de cette séance est la mise en place de l'environnement de Symfony3 et la création d'un premier projet avec l'éditeur PHPStorm.

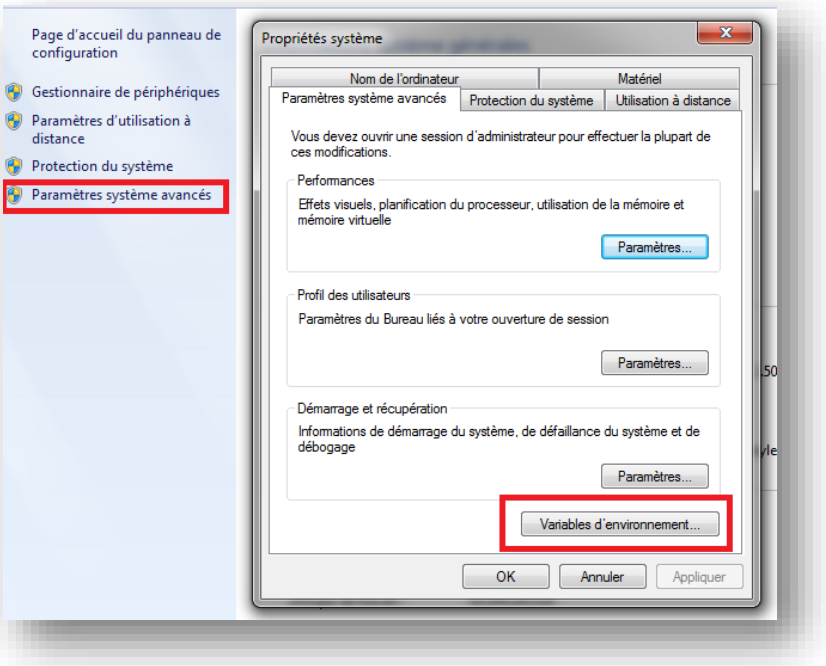
Recommandations

- Utiliser une version de PHP supérieur ou égale à PHP 5.5.X
- Utiliser la version 3.2 de Symfony

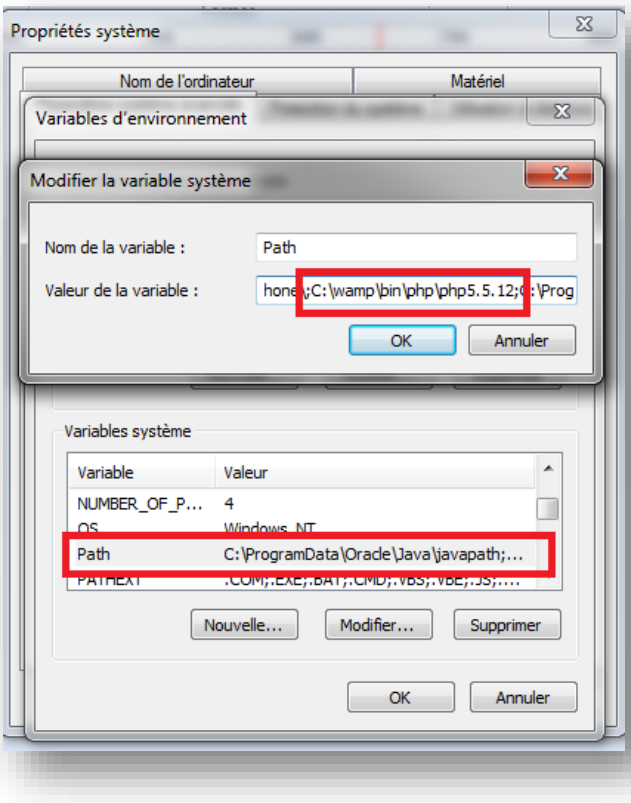
Variable d'environnement

Renseigner la variable d'environnement « php » dans le path si ceci n'est pas déjà fait :

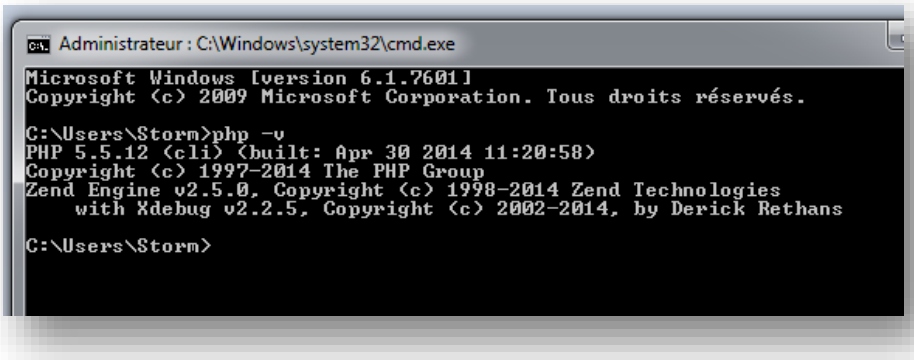
1. Aller sur le répertoire Wamp et copier l'emplacement de votre interpréteur php (exemple :
   a. C:\wamp\bin\php\php5.x)
2. Aller dans **Panneau de configuration** -> **Système** -> **Paramètres système avancés** ->
   a. **Variables d'environnement**.



3. Sélectionner la variable **Path** -> **modifier** -> dans « valeur de la variable » atteindre la fin du textbox et rajouter un point-virgule puis copier l'adresse de votre interpréteur php (C:\wamp\bin\php\php5.x)



4. Tester sous l'application CMD la commande PHP et voir si elle est prise en charge (taper par exemple taper **php –v**).



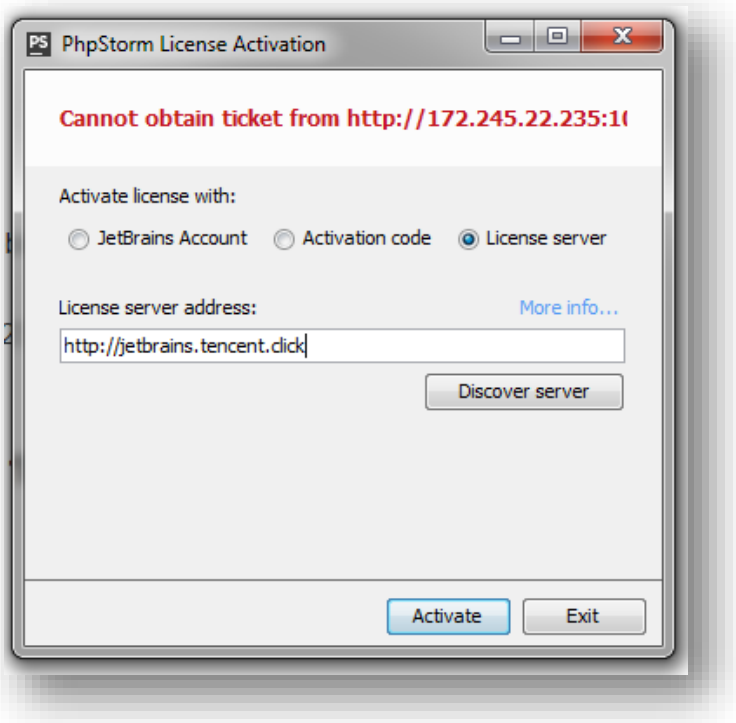
Installer et activer PHPStorm

Après l'installation de PHPStorm, pour l'activer cliquer sur :

1) Serveur de licence (Licence server).

2) Enter cette adresse du serveur http://jetbrains.tencent.click

3) Cliquer sur Activer (Activate).

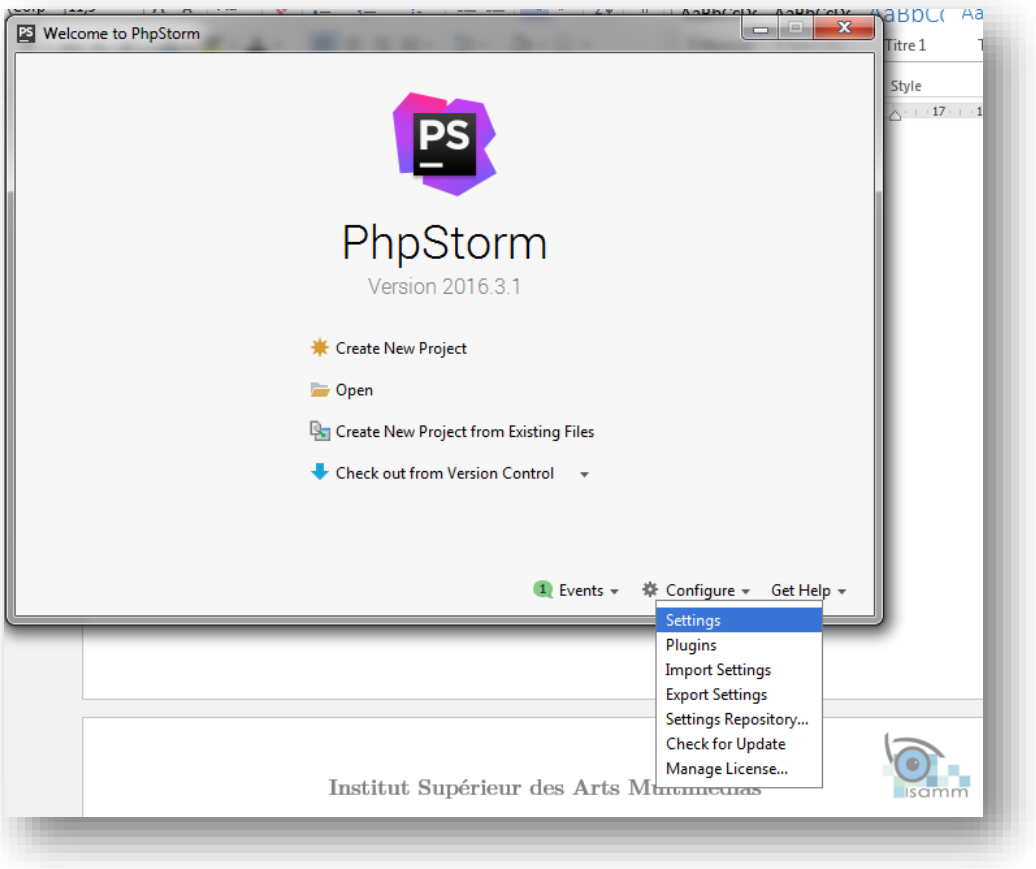


Les étapes de création d'un projet Symfony avec PHPStorm

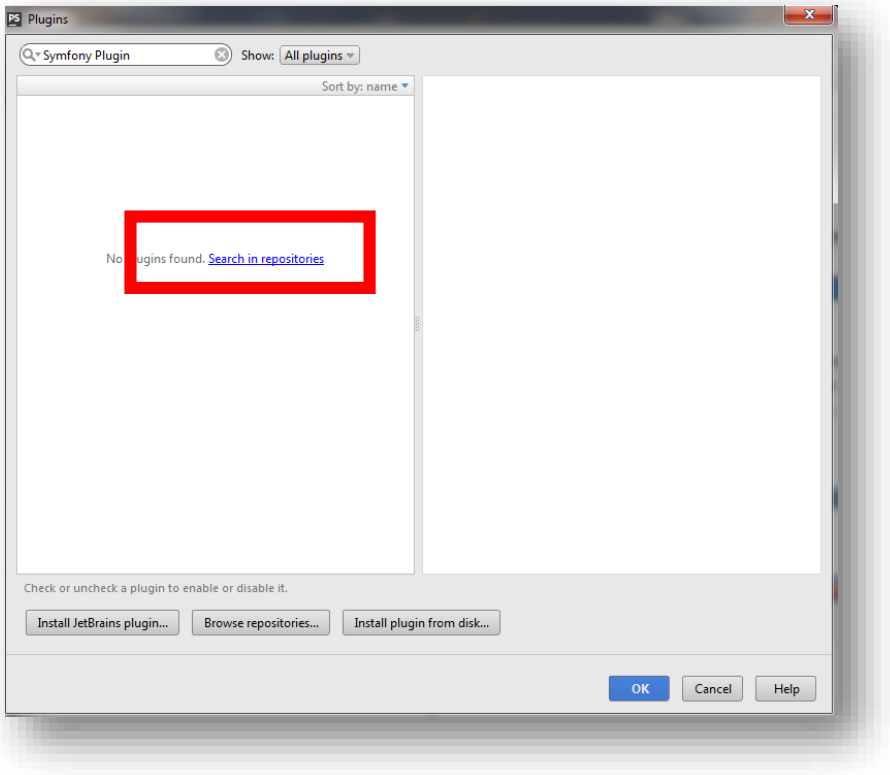
Deux méthodes sont possibles pour créer un projet Symfony avec PHPStorm. La première méthode est recommandée.

1ère méthode

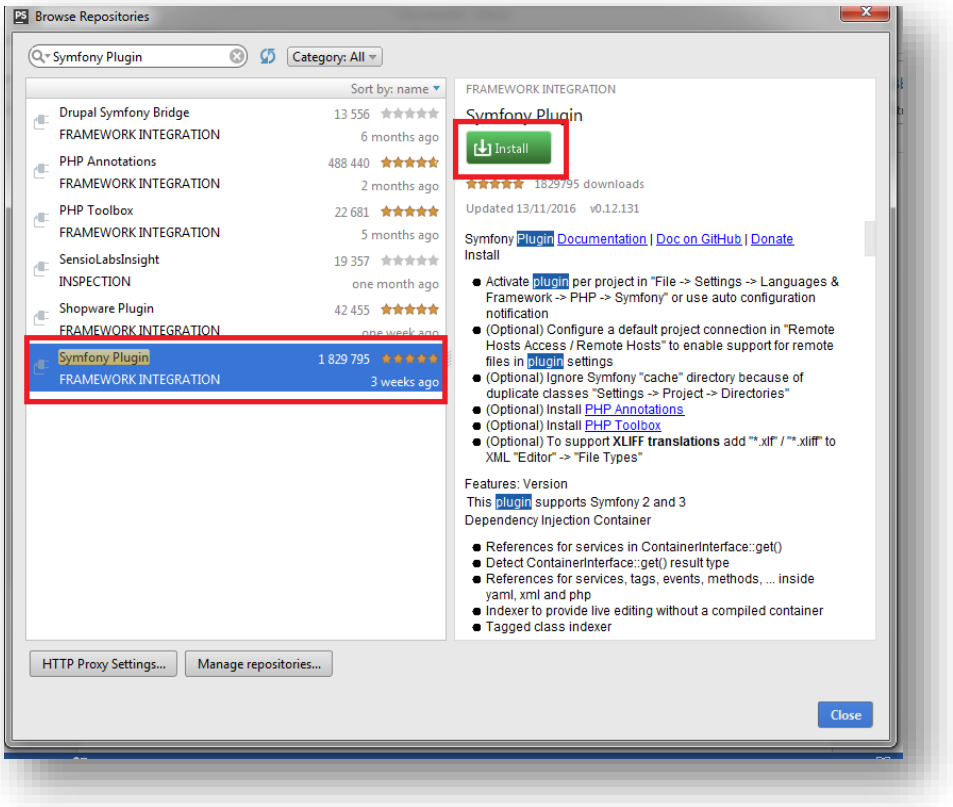
1. Lancer l'IDE PHPStorm, cliquer ensuite sur **Configure** et sélectionner **Plugins**.



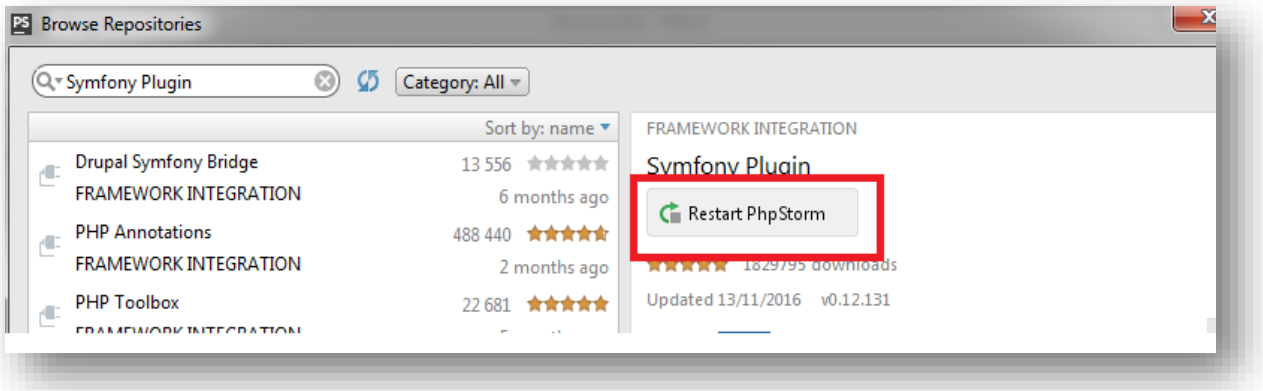
2. Saisir « **Symfony Plugin**» dans la zone de recherche. Si aucun résultat n'apparait, cliquer sur **Search in repositories**.



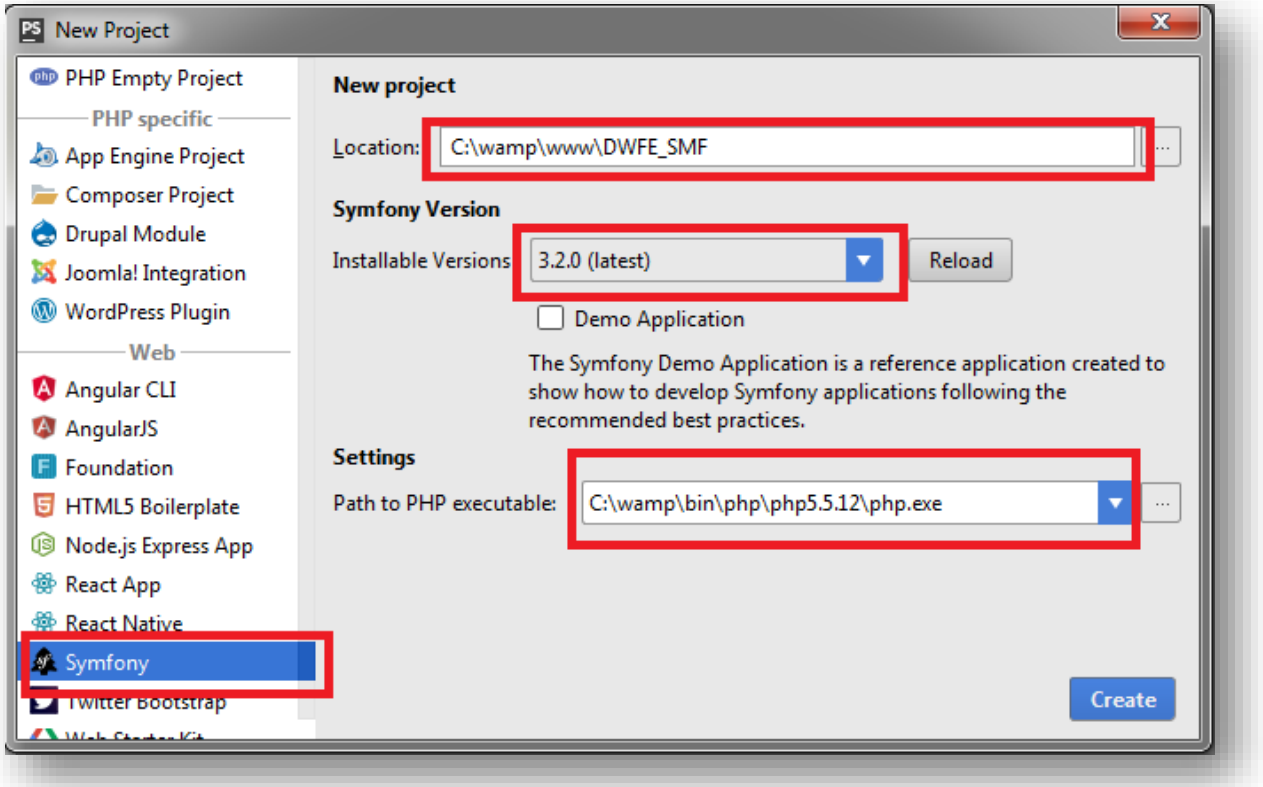
3. Sélectionner « **Symfony Plugin**» puis cliquer sur **Install**.



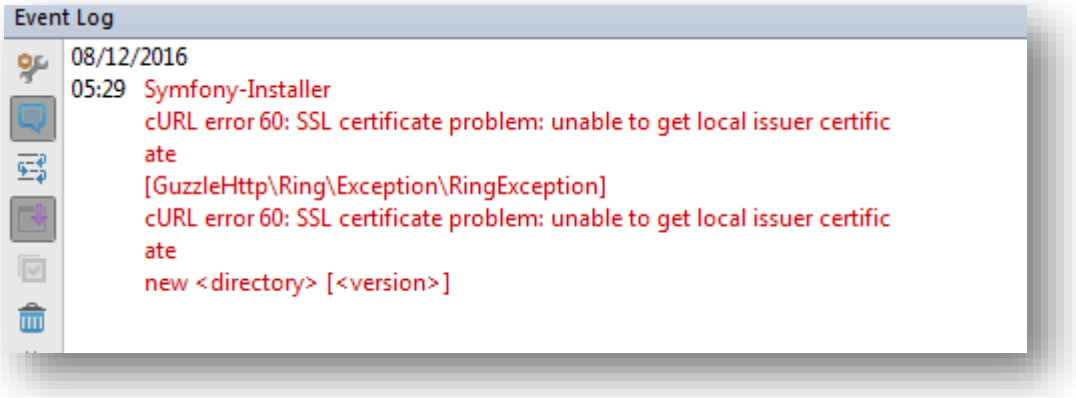
4. Une fois l'installation s'est terminée, cliquer sur « **Restart PhpStorm** ».



5. Maintenant nous pouvons créer un projet Symfony avec la version désirée. Il faut donc choisir la bonne localisation du projet (c:\wamp\www\nomProjet ou c:\xampp\htdocs\ nomProjet), la version de symfony et l'interpréteur PHP.



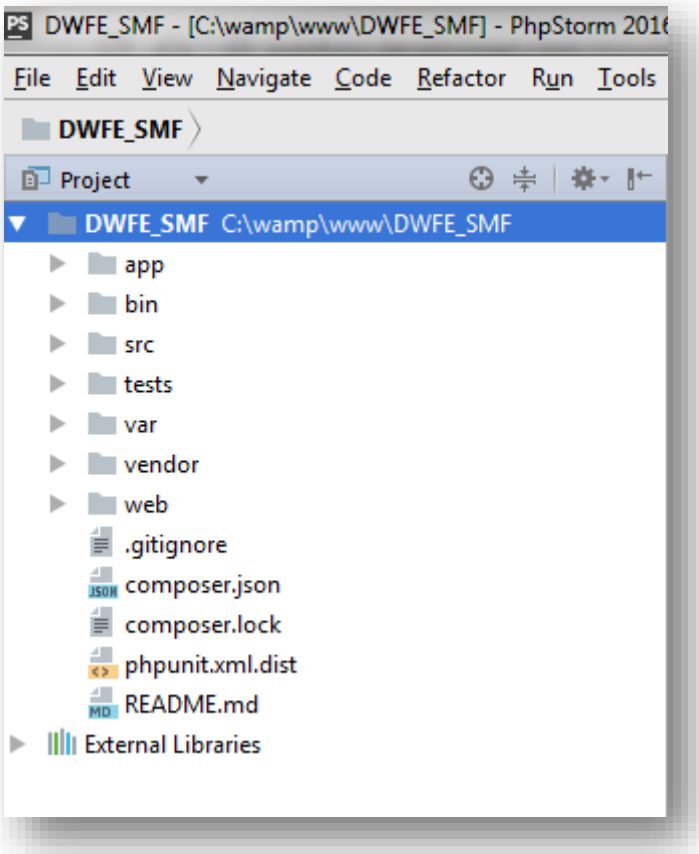
6. En cas de l'apparition de cette erreur : curl: (60) SSL certificate : unable to get local issuer certificate.



a) Il faut télécharger le fichier **cacert.pem** via cet URL https://curl.haxx.se/ca/cacert.pem
b) Ajouter le lien vers ce fichier dans le fichier **php.ini** comme suit :
Ouvrir le fichier **php.ini** ensuite décommenter le paramètre **curl.cainfo** et modifier sa valeur :
curl.cainfo ="C:/wamp /cacert.pem"
c) Redémarrer WAMP ou XAMPP et PHPStorm.

Liste des répertoires

Vous devez avoir l'arborescence suivante du projet créé :



Le répertoire /*app*

Ce dossier contient l'ensemble des éléments compilés de votre application, que ce soit le code **PHP** généré par Symfony ou le code compilé de **Twig**. Les fichiers **de configurations globaux** de l'application sont dans ce dossier (Exemple config BDD).

Aussi, on trouve dans ce dossier un sous-dossier qui s'appelle **Resources** qui contiendra toutes les Templates, toutes les vues, c'est-à-dire tout le HTML de votre projet, alors on n'aura plus de **Twig** dans les sources de notre application. (une bonne pratique).

http://symfony.com/doc/current/best_practices/templates.html

Twig : Twig est le nouveau moteur de **template** créé par **SensioLab**, l'éditeur de Symfony. Ce moteur est conceptuellement très proche du moteur de **Django**.

Le répertoire /*bin*

Ce répertoire contient tous les exécutable « **console** » dont nous allons nous servir pendant le développement. Par exécutable, nous voulons dire des commandes PHP.

Le répertoire */src*

C'est ici que l'on mettra le code source. Dans ce répertoire, nous organiserons notre code en ***bundles.*** Ce répertoire n'est pas vide : il contient en effet quelques fichiers exemples, fournis par Symfony.

Le répertoire /*tests*

Les fichiers de tests unitaires et fonctionnels (Referez-vous à la présentation de **Narimane** pour plus d'informations sur les tests d'un site web).

Le répertoire /*var*

Il contient tout ce que Symfony va écrire durant son process : les logs, le cache, et d'autres fichiers nécessaires à son bon fonctionnement. Nous n'écrirons jamais dedans nous-mêmes.

Le répertoire */vendor*

Ce répertoire contient toutes les bibliothèques externes à notre application. Vous pouvez parcourir ce répertoire ; vous y trouverez des bibliothèques comme **Doctrine**, **Twig**, **SwiftMailer**, etc.

Le répertoire /*web*

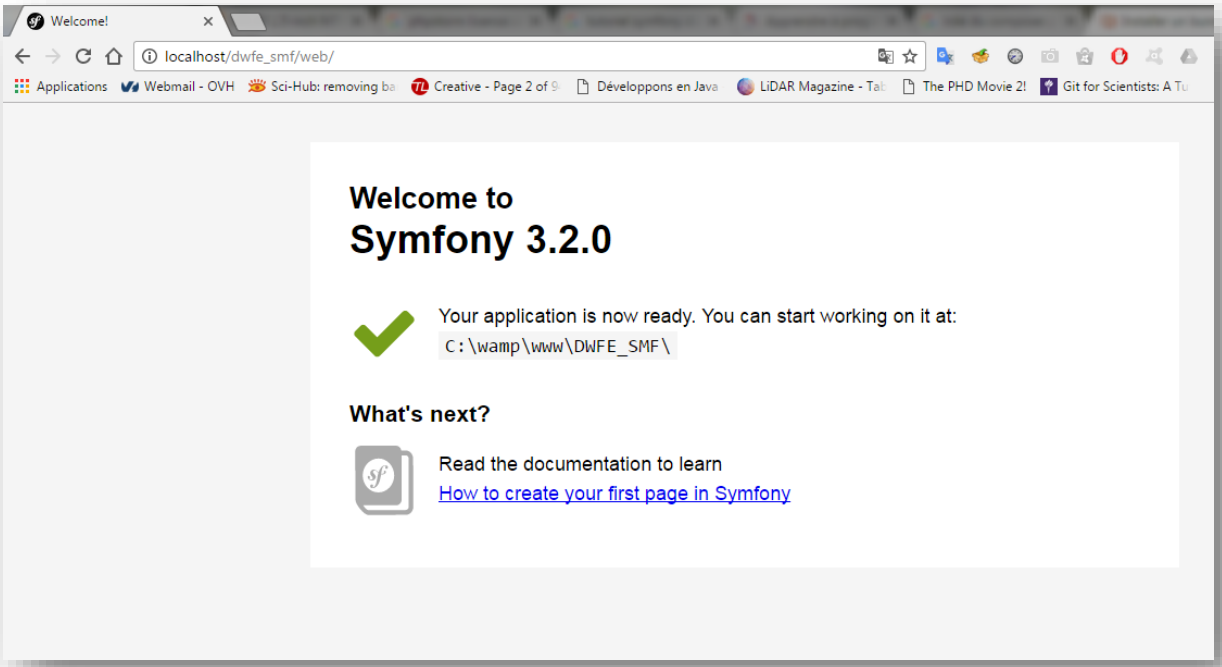
Ce répertoire contient tous les fichiers destinés à vos visiteurs : images, fichiers CSS et JavaScript, etc:

- Le fichier **.htaccess**,
- Le fichier **app.php** (un seul doit être présent en **production**). C'est le fichier source de Symfony. C'est le point d'entrée global de l'application, l'équivalent du index.html pour apache,
- Le fichier **app_dev.php,** équivalent de app.php réservé aux **développements**,
- L'ensemble des ressources externes (js / css / images).

À retenir

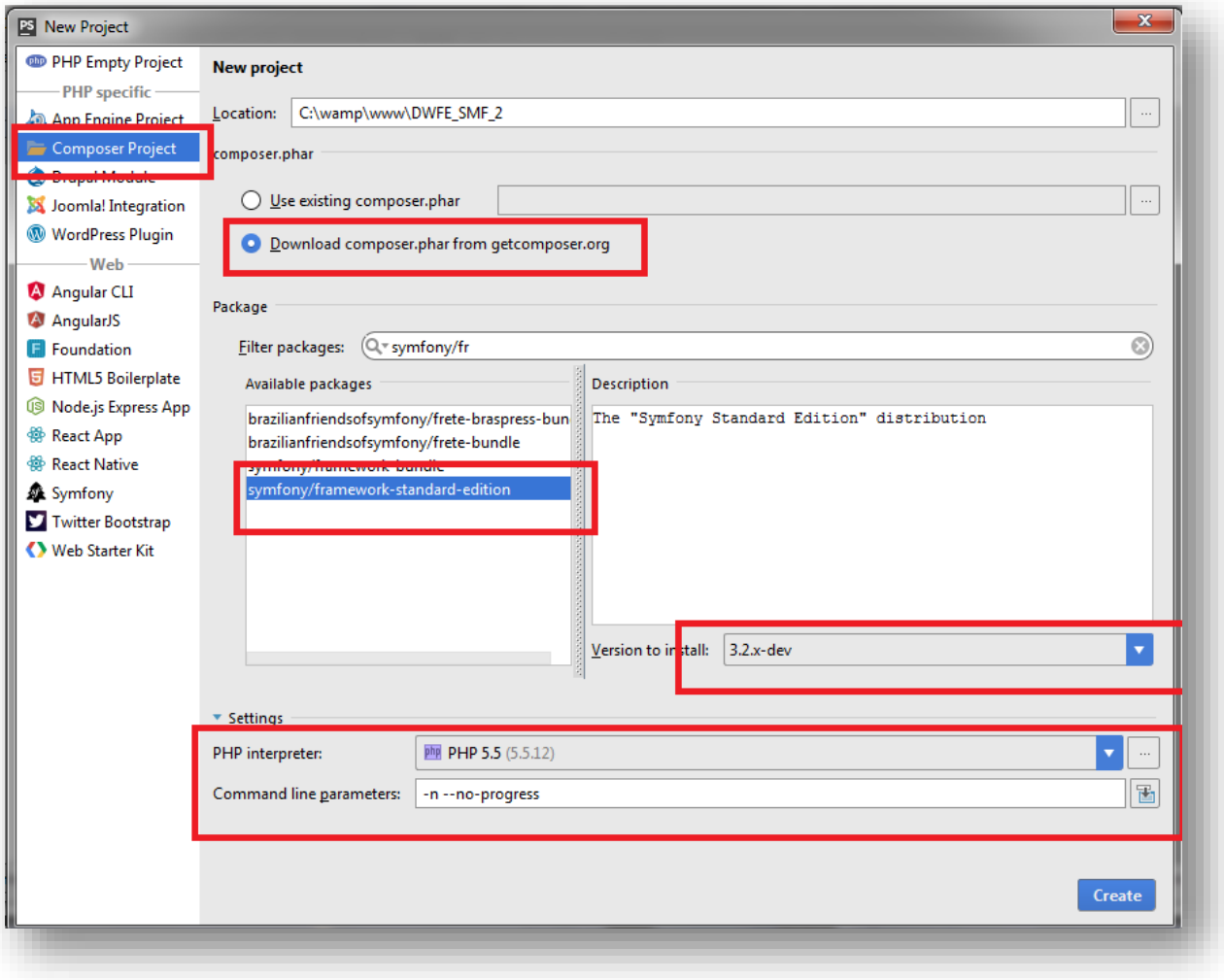
Retenez donc que nous passerons la plupart de notre temps dans le répertoire /**src**, à travailler sur nos bundles. On touchera également pas mal au répertoire /**app** pour configurer notre application. Et lorsque nous installerons des bundles téléchargés, nous le ferons dans le répertoire /**vendor**.

Enfin, pour tester si vous avez bien installé et créé un projet Symfony 3.2, taper l'URL suivante http://localhost/dwfe_smf/web



2ème méthode

1. Créer un nouveau projet sous PHPStorm. Il faut choisir **Composer Project** comme type de projet (File->New Project->Composer Project). Le projet doit être situé sous c:\wamp\www ou c:\xamp\htdocs.
2. Après avoir choisi le type du projet il faut saisir les différentes propriétés pour l'installation de la version symfony 3.2



1. En choisissant cette propriété le **composer.phar** sera installé automatiquement.

Composer, qu'est-ce que c'est ? Un gestionnaire de dépendances

Composer est un outil pour gérer les dépendances en PHP. Les dépendances, dans un projet, ce sont toutes les bibliothèques dont votre projet dépend pour fonctionner. Par exemple, votre projet utilise la bibliothèque SwiftMailer pour envoyer des e-mails, il « dépend » donc de **SwiftMailer**. Autrement dit, SwiftMailer est une dépendance dans votre projet.

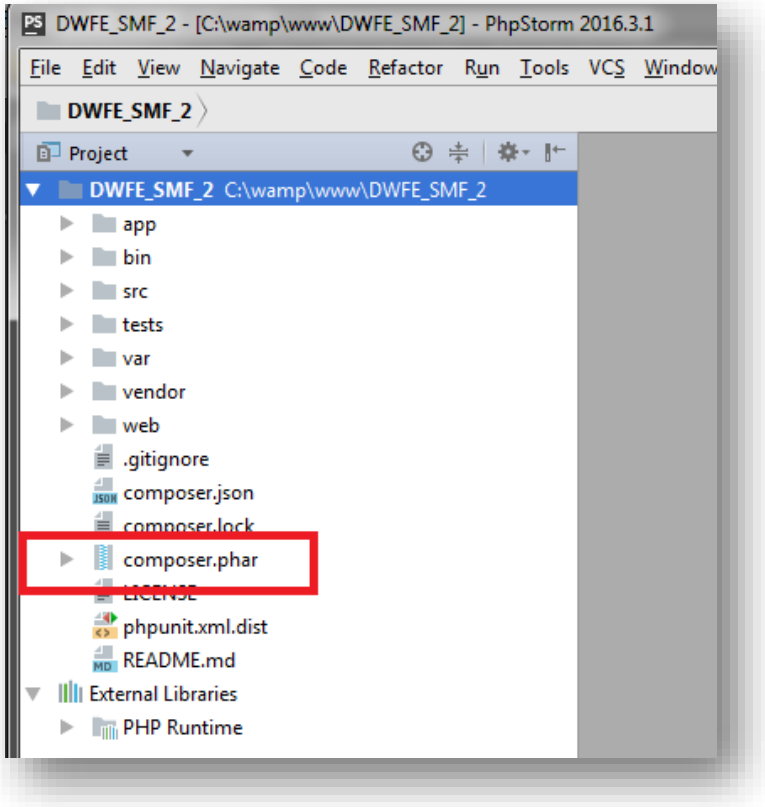
Composer a donc pour objectif de vous aider à gérer toutes vos dépendances. En effet, il y a plusieurs problématiques lorsqu'on utilise des bibliothèques externes :

- Ces bibliothèques sont mises à jour. Il vous faut donc les mettre à jour une à une pour vous assurer de corriger les bogues de chacune d'entre elles.
- Ces bibliothèques peuvent elles-mêmes dépendre d'autres bibliothèques. En effet, si une de vos bibliothèques dépend d'autres bibliothèques, cela vous oblige à gérer l'ensemble de ces dépendances (installation, mises à jour, etc.).
- Ces bibliothèques ont chacune leur paramètres **d'autoload,** et vous devez gérer leur **autoload** pour chacune d'entre elles.

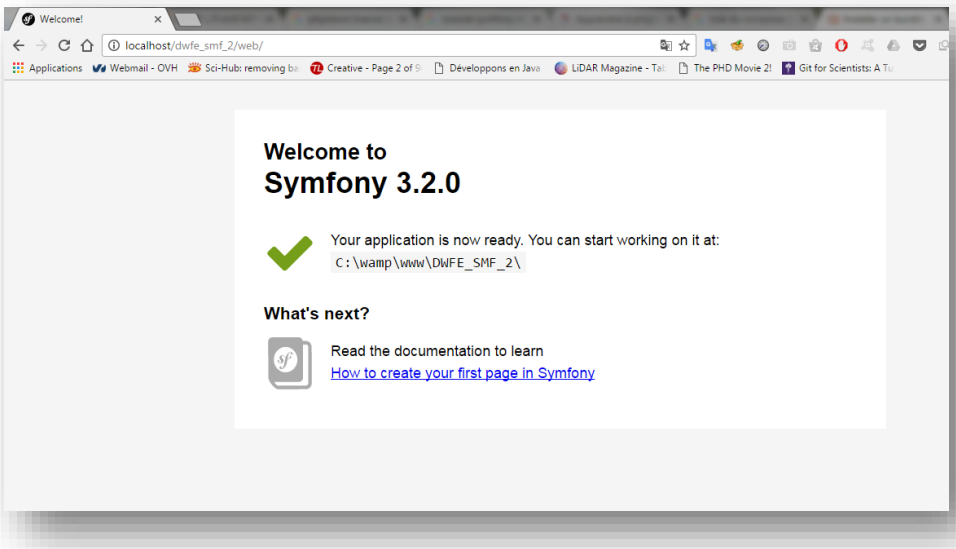
Composer va nous aider dans chacune de ces tâches.

Après avoir saisi ces différents paramètres il faut cliquer sur **create**.

Vous devez avoir l'arborescence suivante du projet créé :



Au niveau de votre navigateur

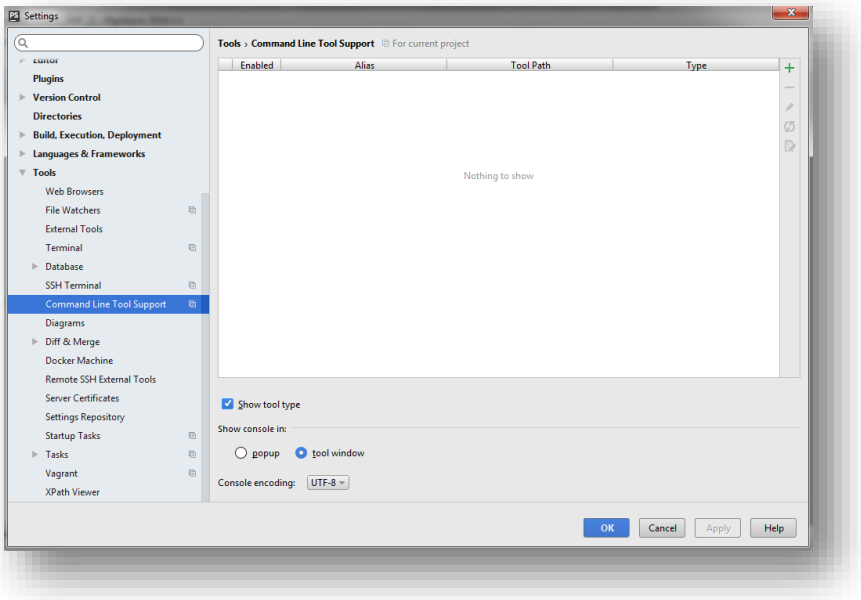


Ajout des commandes de Composer et Symfony

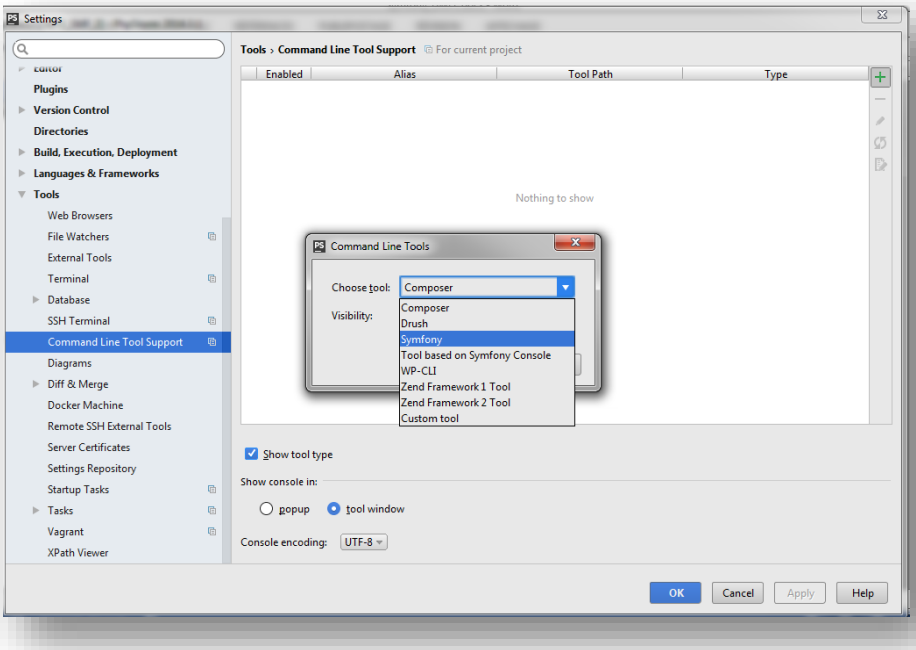
Dans cette partie nous allons voir comment utiliser la console de PHPStorm que nous allons s'en servir dans nos projets Symfony.

Symfony intègre un utilitaire qui permet de taper des lignes de commande que nous allons utiliser dans tous les ateliers Symfony. Dans PHPStorm il faut configurer **Commande Line Tools Console** (qui s'affiche en appuyant sur **Ctrl-Shift-X**) à travers du quel nous allons exécuter les commandes Symfony.

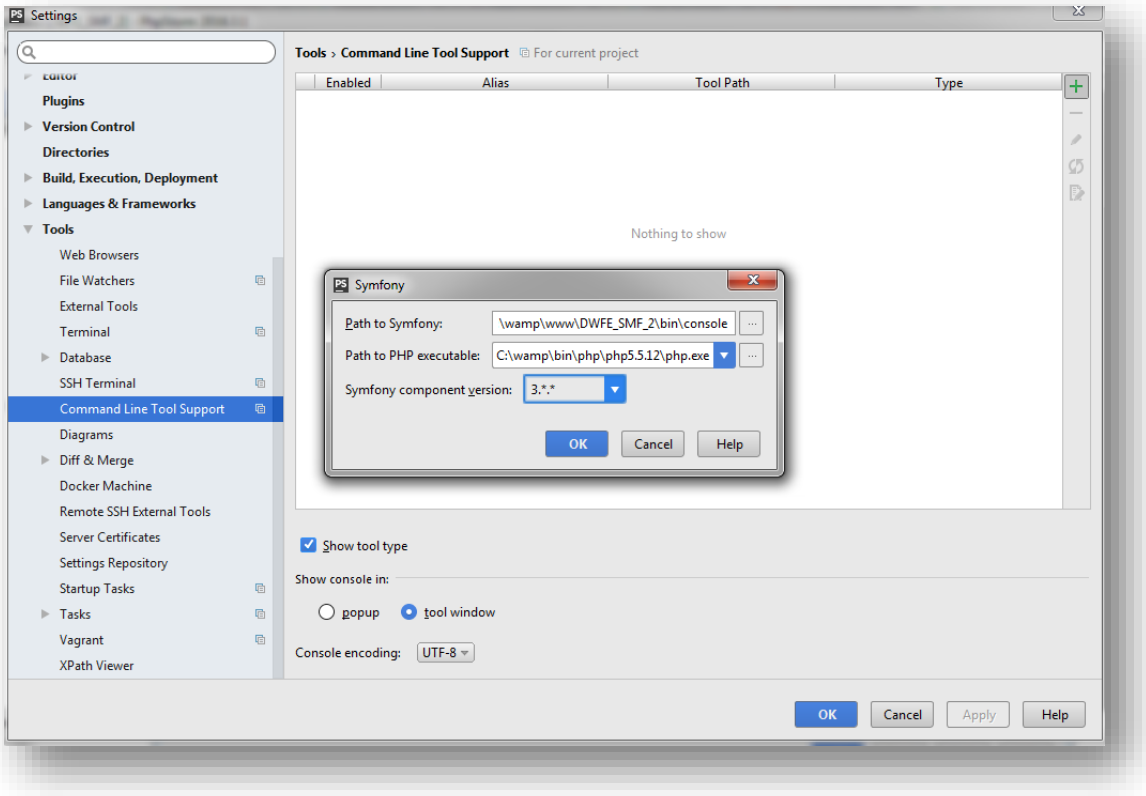
1. Pour configurer cette console accéder à File->Settings->Tools->Commande Line Tool



2. Cliquer sur + puis choisir **Symfony** au lieu de **Composer.**



a) Il faut indiquer ensuite où se situe le terminal de commande fournit par symfony (dans notre cas C:\wamp\www\DWFE_SMF_2\bin\console) ainsi que l'interpréteur PHP.



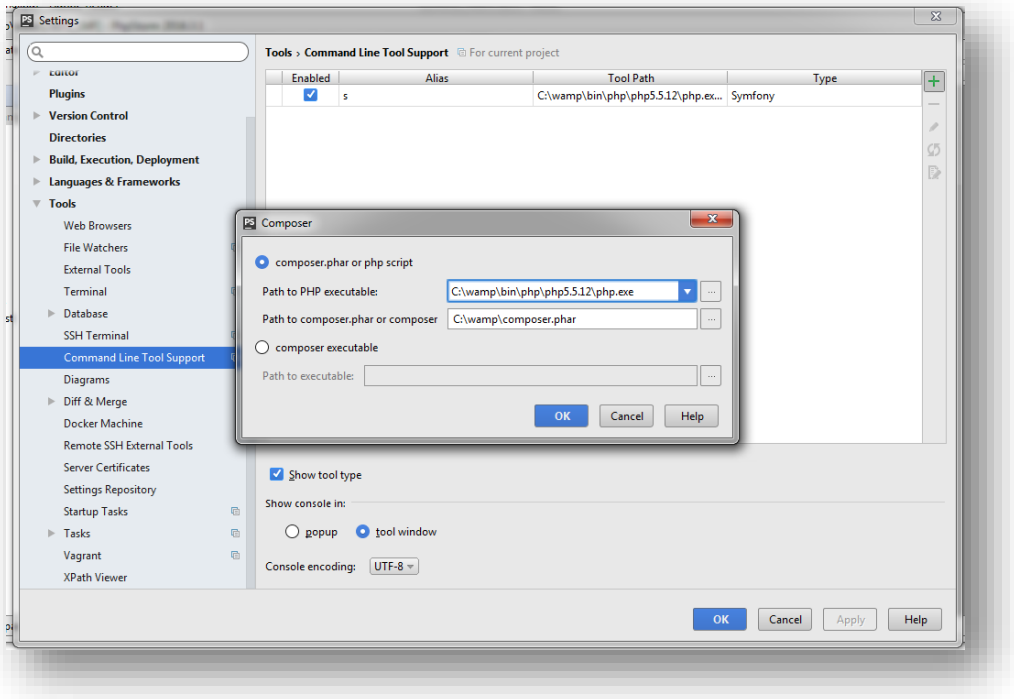
b) En fin, cliquer sur **OK**, **Apply** puis **OK**. Et maintenant les commandes Symfony sont bien installées.

3. Installer **Composer.phar** via https://getcomposer.org/download . En accédant à cette page vous devez suivre la figure ci-dessous et cliquer sur la version 1.2.1 pour le téléchargement de la dernière version du **composer.phar**. Il faut placer ce fichier sous **c:\wamp**

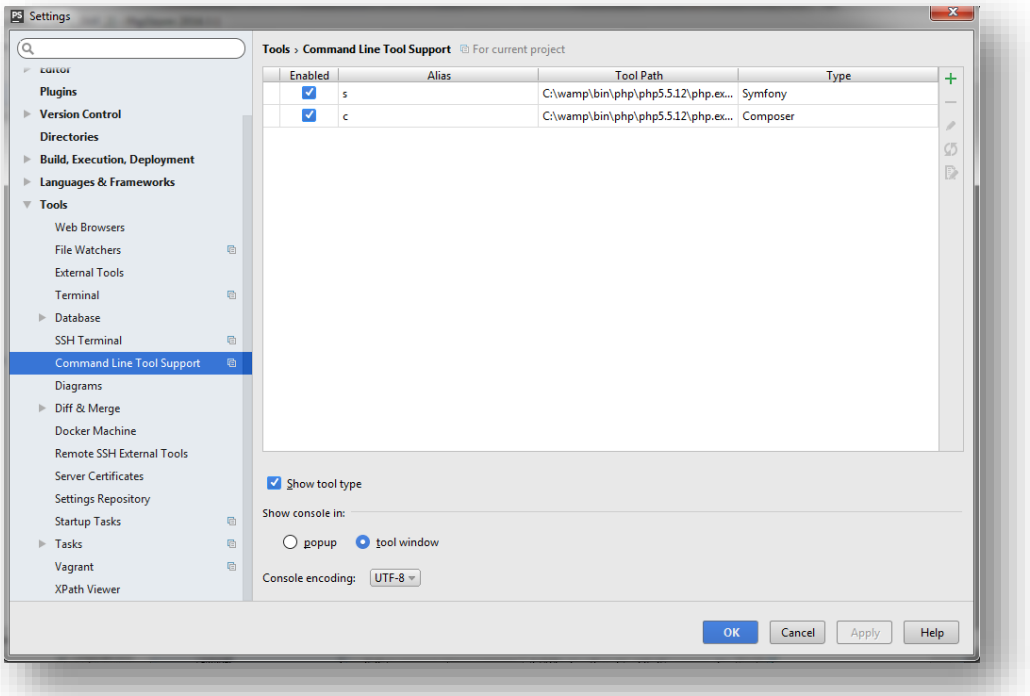
4. Cliquer sur + pour ajouter un outil composer

a) Ensuite il faut indiquer l'interpréteur PHP (qui se trouve sous c:\xamp\php\php.exe ou c:\wamp\bin\php\php5.6.25\php.exe)

b) Ensuite, il faut mettre dans **Path to composer** le chemin vers le **composer.phar** (c:\wamp\composer.phar).



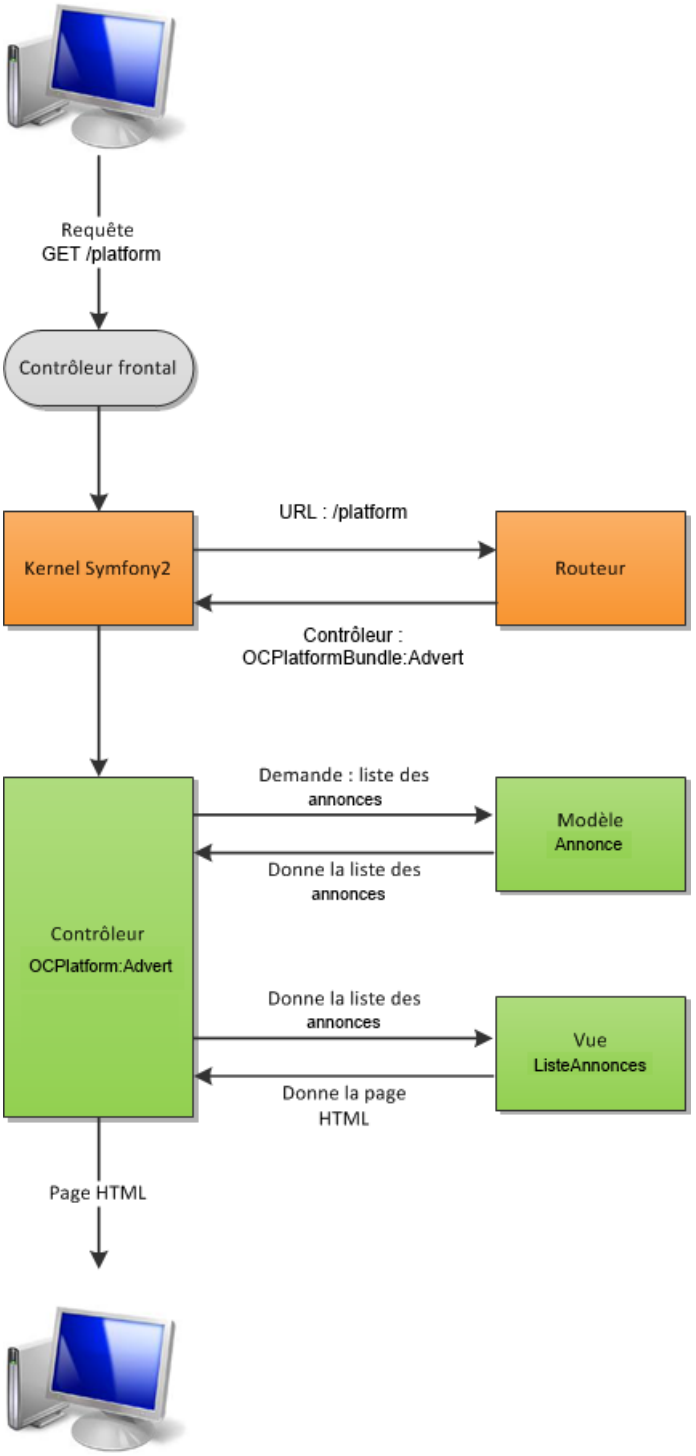
c) En fin, cliquer sur **OK**. Dans ce cas il rajoutera des commandes qu'on utilisera via la console.



Symfony et MVC

Sachez que Symfony respecte bien entendu cette architecture MVC. Je ne vais pas entrer dans ses détails, car nous avons vu avec CodeIgniter, comment le patron MVC fonctionne.

Afin de bien visualiser tous les acteurs que nous avons vus jusqu'à présent, voici un schéma du parcours complet d'une requête dans Symfony :



En le parcourant avec des mots, voici ce que cela donne :

1. Le visiteur demande la page /***platform*** ;
2. Le contrôleur frontal reçoit la requête, charge le ***Kernel*** et la lui transmet ;
3. Le Kernel demande au Routeur quel contrôleur exécuter pour l'URL /***platform***. Ce Routeur est un composant Symfony qui fait la correspondance entre URL et contrôleurs. Le Routeur fait donc son travail, et dit au Kernel qu'il faut exécuter le contrôleur ***OCPlatform:Advert*** ;
4. Le Kernel exécute donc ce contrôleur. Le contrôleur demande au modèle ***Annonce*** la liste des annonces, puis la donne à la vue ***ListeAnnonces*** pour qu'elle construise la page HTML et la lui retourne. Une fois cela fini, le contrôleur envoie au visiteur la page HTML complète.