Web - TP Symfony n^o 1
G. Subrenat, R. Zrour

Licence Informatique L3 - 2022-2023
UFR SFA - Université de Poitiers

Installation de Symfony

Table des matières

1 Résumé **2**

2 Cadre de travail **2**

3 Outils **3**

4 Installation de *Symfony* **3**

4.1 Création de l'arborescence Symfony . . . 3
4.2 Configuration du cache . . . 5
4.3 Mise à jour du site . . . 6
4.4 Test du site . . . 6
4.5 Test du cache . . . 7
4.6 Installation des composants/packages . . . 7
4.7 Utilisation d'un IDE . . . 9
4.8 git . . . 9

5 Git : code source **10**

Le code du TP est disponible à : https://gitlab.com/subrenat/l3-web-2022-23
dans la branche *b-TP1* : https://gitlab.com/subrenat/l3-web-2022-23/-/tree/b-TP1

1 Résumé

Dans ce TP nous verrons comment installer et configurer le framework Symfony,

À la fin de ce document, nous aurons un site accessible, sans code personnel et prêt à être modifié.

Vous vous devez de faire des tests sur les différentes commandes pour vérifier votre compréhension. Dans le pire des cas, en cas de mauvaise manipulation, vous réinstallez le site complètement.

Note importante : le site créé dans ce TP sera complété par les autres TP. Autrement dit, voici la liste des sites qui seront créés dans la suite de l'enseignement :

- un seul site pour tous les TP
- un site pour le projet
- vous avez le droit (heureusement) de créer d'autres sites si vous voulez

2 Cadre de travail

Point de cours

Habituellement un développeur web travaille sur plusieurs machines :

- *serveur de production* : le site web ouvert au public est installé sur cette machine, elle est donc particulièrement protégée et la version du site est stable.
- *serveur de développement* : le site en cours de développement est installé dessus et il doit avoir la même configuration que le serveur de production, ou du moins les mêmes versions de PHP, Apache, MySql (ou autre), ... et Symfony.
- *serveur de travail* : il est dédié au développeur pour qu'il puisse faire des tests sans gêner les autres développeurs.
- *machine personnelle* : elle contient tous les outils de développement nécessaires, notamment l'IDE. Il n'y a pas de serveur web installé sur cette machine.
- *serveur de bases de données* : en fait il y en a plusieurs, au moins celui pour le site en production, et un pour le site en développement.

Le code du site est présent à la fois sur les quatre premières machines citées.

Les codes sur la machine personnelle et sur le serveur de travail sont quasiment identiques (certaines données, telle le cache, sont présentes uniquement sur le serveur). Généralement c'est l'IDE qui se charge de synchroniser le code.
Notez que si la majorité des changements sont effectués sur la machine personnelle pour être transférés sur le serveur de travail, il arrive que l'on effectue des modifications sur le serveur de travail qui doivent alors être rapatriées sur la machine personnelle.

Les codes sur les deux autres serveurs sont liés à des versions précises du site. Les versions entre la machine personnelle, le serveur de développement et le serveur de production sont généralement gérées par un outil de versionnement tel *Git*.

Il se trouve sur pour les TP, le schéma est grandement simplifié :

- pas de serveur de production
- pas de serveur de développement
- pas de serveur de base de données (on utilise SQLite)
- le serveur de travail et la machine personnelle sont les mêmes

Bref le code est en un seul exemplaire.

Il est fortement conseiller d'utiliser *Git* pour garder une trace des différentes versions :

- au minimum : un *commit* par TP
- mieux : une branche par TP

Il est aussi recommandé d'avoir un repository distant. Cela vous permettra de travailler sur plusieurs machines et aussi d'avoir une sauvegarde.

Si vous êtes seul à travailler sur le site, la quatuor *pull/add/commit/push* est assez simple à utiliser (i.e. pas de conflit [a] à gérer).

a. sauf si on y met de la mauvaise volonté

À de rares exceptions, les manipulations demandées dans ce TP (et les suivants) sont les mêmes quel que soit le système d'exploitation.

3 Outils

Les outils PHP, SQLite, composer, symfony, git sont supposés installés.

Pour le vérifier, testez les quatre commandes suivantes :

```
serveur$ php -v
[...]
serveur$ composer -V
[...]
serveur$ symfony -V
[...]
serveur$ git --version
[...]
serveur$ sqlite3 -version # sous Linux
[...]
```

4 Installation de *Symfony*

Les dernières versions stables au 2022/12/01 sont les versions 5.4 et 6.2.
La version 5.4 (LTS) a une durée de vie de 3 ans (4 ans pour les corrections de sécurité) jusqu'en novembre 2024 (novembre 2025 pour la sécurité) :

http://symfony.com/releases/5.4
http://symfony.com/releases/5.4.json

La version 6.2 est maintenue jusqu'en juillet 2023 (durée de 6 mois) :

http://symfony.com/releases/6.2
http://symfony.com/releases/6.2.json

En novembre 2023 est prévue la version 6.4 qui aura également une durée de vie de 3 ans (4 ans pour les corrections de sécurité) jusqu'en novembre 2026 (novembre 2027 pour la sécurité) :

http://symfony.com/releases/6.4
http://symfony.com/releases/6.4.json

Et pour les patients, en novembre 2031 est prévue la version 10.4 :

http://symfony.com/releases/10.4
http://symfony.com/releases/10.4.json

En outre la version 6.2 nécessite au minimum la version 8.1.0 de PHP.

Nous travaillerons impérativement avec la version 6.2 de Symfony.

4.1 Création de l'arborescence Symfony

On utilise l'utilitaire *symfony* précédemment installé sur le serveur.

Pour les utilisateurs d'un serveur Apache : il faut se placer dans un répertoire visible du serveur ; par exemple (sous Linux) un des deux répertoires suivants :

```
serveur$ cd $HOME/public_html/L3
serveur$ cd /var/www/html/L3
```

Si on lance un serveur manuellement (avec *php* ou *symfony*), ce que nous ferons, le choix du répertoire est libre.

Pour vérifier que symfony est utilisable, Commençons par la commande :

```
serveur$ symfony check:requirements
```

Si vous avez plusieurs versions de PHP installées sur votre système [1], la commande suivante peut être utile :

```
serveur$ symfony local:php:list
```

Pour une aide complète pour la commande *new* qui crée une arborescence :

```
serveur$ symfony help new
```

Comme nous allons utiliser *git*, si vous ne l'avez pas encore fait, tapez les deux commandes suivantes (en les adaptant à votre cas) :

```
$ git config --global user.email "gilles@l3.fr"
$ git config --global user.name "Gilles"
```

Pour créer un projet la syntaxe est la suivante :

```
serveur$ symfony [--no-git] [--version=value] new <nom répertoire>
```

L'option *--no-git* est utile si *git* n'est pas installé sur le système. En l'occurrence nous utiliserons *git* et donc cette option ne sera pas spécifiée.
Si on utilise la dernière version de symfony, l'option *--version* n'est pas utile. Dans notre cas nous allons malgré tout l'utiliser à titre d'exemple.

Attention, il ne faut pas utiliser l'option *--webapp* [2] dans le cadre de ce cours, car vous voulons installer un code minimal pour le site. Nous étofferons l'installation au fur et à mesure des besoins.

Par exemple :

```
serveur$ symfony --version=6.2 new tp
```

On insiste sur le fait que si cette commande est lancée au moment de l'enseignement (premier semestre 2023), l'option *--version* est inutile.

Si vous voulez avoir la taille du site mininal déployé avec toute modificiation, voici la commande (sous Linux ou *git-bash* :

```
serveur$ du -hs tp
12M tp
```

Pour information, si on avait utilisé l'option *--webapp* le répertoire aurait une taille d'environ 80 Mo.

Ensuite toutes les commandes se feront à la racine du site.

```
serveur$ cd tp
```

La console (qui se lance toujours à la racine du site) va être très utilisée. Les deux commandes suivantes sont strictement équivalentes :

```
serveur$ symfony console
serveur$ php bin/console
```

Pour avoir la version installée [3] :

```
serveur$ symfony console --version
serveur$ php bin/console --version
```

1. à vos risques et périls
2. auparavant *--full*
3. devrait être la 6.2 puisque nous l'avons explicitement précisée

4.2 Configuration du cache

Cette partie est utile si vous utilisez un serveur apache tournant sous Linux (ce qui sera peut-être la cas sur le serveur de production).

Si vous utilisez le serveur de *php* ou *symfony* vous pouvez passer directement à la section suivante.

Il faudrait d'ailleurs vérifier, avec les versions récences de Symfony, si les manipulations décrites ci-dessous sont toujours d'actualité.

Point de cours

Lorsqu'un internaute accède à votre site, les scripts PHP ne tournent pas sous le nom du concepteur, mais sous celui de *www-data*.
C'est une sécurité basique : *www-data* n'a, *a priori*, qu'un accès en lecture aux répertoires ce qui peut limiter l'exploitation d'une faille de sécurité.

La première conséquence est que les scripts PHP doivent être accessibles à *www-data*, i.e. aux utilisateurs *other*, soit *r-x* pour les répertoires et *r--* pour les fichiers.

La seconde conséquence est que, dans le cas où les scripts PHP doivent faire des modifications sur le disque (création de fichiers par exemple), les répertoires doivent être accessibles en écriture.

Le cache permet à Symfony d'optimiser les temps d'accès en pré-calculant des données (et du code aussi d'ailleurs). Le répertoire de cache est *var*.
Et donc les scripts PHP (avec les droits de *www-data*) de votre site écrivent dans ce répertoire *var*.

On pourrait imaginer qu'il suffit de donner les droits d'écriture dans ce répertoire et ses fils (par un "`chmod -R o+w var`" par exemple).

Mais le problème est plus complexe. Avec le *chmod* susnommé, Symfony va écrire sans problème dans le cache. Malheureusement il est parfois nécessaire de vider le cache, ce qui se fait en ligne de commandes par le programmeur, et donc avec ses droits.
Si on récapitule :

- *www-data* crée des répertoires et des fichiers dans *var*,
- le programmeur veut les effacer (via une commande Symfony),
- mais il n'a pas les droits d'effacer des fichiers dans les répertoires créés par *www-data*.

On peut imaginer plusieurs cas :

- il n'y pas de solution : heureusement ce n'est pas le cas.
- rajouter le programmeur au groupe *www-data* : ça devrait marcher, mais on aimerait une solution ne nécessitant pas l'intervention de l'administrateur.
- une autre solution.

Nous allons outrepasser les droits habituels d'Unix en utilisant les ACL (Access Control Lists) : nous allons donner les droits d'écriture sur le répertoire *var* aux utilisateurs *www-data* et *programmeur* indépendamment des *rwx* classiques.

Voici la recette [4] [5] et le script, fourni, qui met les bons droits :

set_rights.sh

```
#!/bin/bash

if [ "$0" == "bash" ]
then
    echo ne lancez pas la commande avec \"source\"
    echo lancez la comme une commande externe
    return
fi

if [ ! -d var ]
then
    echo répertoire "var" inexistant
```

4. Nous ne développerons pas ce concept plus loin.
5. pour les utilisateurs de Mac, vous pourrez étudier la commande *chmod +a*

```
13      echo placez-vous àla racine du projet
14      exit 1
15  fi
16
17  setfacl -R -m u:"www-data":rwX -m u:${USER}:rwX var
18  if [ $? -ne 0 ]
19  then
20      echo erreur \"setfacl -R\"
21      exit 1
22  fi
23  setfacl -dR -m u:"www-data":rwX -m u:${USER}:rwX var
24  if [ $? -ne 0 ]
25  then
26      echo erreur \"setfacl -dR\"
27      exit 1
28  fi
29
30  exit 0
```

Le script doit être lancé dans le répertoire racine du projet, i.e. *tp* :

```
serveur$ cd tp
serveur$ pwd
/home/SFASP2MI/gsubrena/public_html/L3/tp
serveur$ ../set_rights.sh
```

Vous ne pouvez lancer ces commandes que sur des répertoires qui vous appartiennent.

4.3 Mise à jour du site

Même si nous venons juste d'installer l'arborescence, tous les packages/bibliothèques liés au framework n'ont peut-être pas les dernières versions.

Attention : lorsqu'un projet est développé à plusieurs, il est fondamental que les développeurs aient tous les mêmes versions des packages. Aussi cette mise à jour ne doit être faite qu'après une concertation.

Le script *composer* que nous avons installé entre en action de la façon suivante, dans le répertoire racine du site :

```
serveur$ pwd
/home/SFASP2MI/gsubrena/public_html/L3/tp
serveur$ composer update
```

Comme nous avons fait une installation minimale, le script s'exécute rapidement.

4.4 Test du site

Normalement tout est opérationnel.

Il y a deux manières d'accéder au site :

- en mode production : c'est l'accès public classique. Les erreurs sont gérées avec un affichage minimaliste[6], et les exécutions des scripts PHP sont efficaces.
- en mode développement : il permet de debugguer. Les erreurs sont très détaillées et le programmeur a accès à des historiques et des statistiques. En revanche les temps de réponses sont dégradés.

Le site est par défaut configuré en mode "dev" et nous resterons dans ce mode.

Le serveur doit être lancé dans le répertoire *public* de l'arborescence :

```
serveur$ cd public
serveur$ symfony server:start
```

ou

6. afin de ne donner aucune indication à un pirate.

```
serveur$ cd public
serveur$ php -S localhost:8000
```

Le serveur piloté par *symfony* est recommandé car plus complet.

L'URL est alors : http://localhost:8000

4.5 Test du cache

Nous verrons qu'il est parfois nécessaire de vider le cache manuellement.
Les caches pour le mode production et le mode développement sont indépendants et doivent être vidés séparément. Le cache du mode développement a rarement besoin d'être vidé.

Nous allons le faire ici pour vérifier que tout se passe bien (et si vous êtes sous Linux et Apache, que les droits du répertoire *var* sont corrects).
La commande pour le mode développement est :

```
serveur$ php bin/console cache:clear
```

ou

```
serveur$ symfony console cache:clear
```

```
serveur$ php bin/console cache:clear

 // Clearing the cache for the dev environment with debug true

 [OK] Cache for the "dev" environment (debug=true) was successfully cleared.

serveur$
```

Et pour le mode production (avec l'autre version de la commande) :

```
serveur$ symfony console cache:clear --env=prod
```

```
serveur$ php bin/console cache:clear --env=prod

 // Clearing the cache for the prod environment with debug false

 [OK] Cache for the "prod" environment (debug=false) was successfully cleared.

serveur$
```

4.6 Installation des composants/packages

Comme nous avons fait une installation minimale, il manque beaucoup de composants soit pour faire fonctionner le site, soit pour le debugguer.

Normalement on n'installe un composant que lorsque le besoin se fait sentir. Ceci dit, pour simplifier les développements futurs, nous allons installer tous les composants utiles dès à présent.

La syntaxe pour installer un composant est la suivante :

```
serveur$ composer require <nom-composant>
```

ou

```
serveur$ composer req <nom-composant>
```

Pour avoir la liste des composants installés :

```
serveur$ composer info -D
```

Précision pour l'installation du premier composant (*symfony/apache-pack*) sous mon Linux et en utilisant apache, où j'ai du faire des manipulations préalables :

- installer le module *rewrite* :

```
serveur$ sudo a2enmod rewrite
```

- dans le fichier */etc/apache2/apache2.conf*, dans la balise <*Directory /var/www*>, j'ai du remplacer la ligne :
  AllowOverride None
  par
  AllowOverride All
- relancer le serveur apache :

```
serveur$ sudo systemctl restart apache2
```

Voici la liste des composants à installer (il est préférable de respecter l'ordre d'installation) :

- *symfony/apache-pack*
  pour avoir de "jolies" URL, il faut répondre "y" pour le recipe
  a priori donc utile uniquement si le site tourne sous Apache (ce qui risque d'être la cas en mode *production*)
- *profiler*
  pour accéder à la toolbar (barre de debuggage intégrée à Symfony)
- *annotations*
  pour les annotations (par exemple des routes dans les controllers)
  Comme nous allons utiliser les attributs de PHP plutôt que les annotations, il est vraisemblable que ce composant n'est plus nécessaire ; mais le doute installez-le.
- *maker*
  pour augmenter la puissance des commandes en mode console
- *symfony/debug-pack*
  pour que les *dump* apparaissent dans la toolbar et non dans la page
- *twig*
  pour utiliser complètement Twig
- *asset*
  pour référencer les ressources publiques
- *orm*
  pour utiliser Doctrine, il faut répondre "y" pour le recipe
- *orm-fixtures*
  pour les fixtures (peuplement de la base) avec Doctrine
- *form*
  pour utiliser les formulaires de Symfony
- *validator*
  pour valider les formulaires
- *security*
  pour l'authentification et la gestion des droits
- *symfony/expression-language*
  pour affiner la gestion des droits
- *translation*
  internationalisation (i18n)

Installez tous ces composants les uns après les autres.

Voici une approximation de la place prise par chaque composant :

composant	surplus		total	
	avec var&vendor	sans var&vendor	avec var&vendor	sans var&vendor
(initial)			11.58 Mo	0.52 Mo
apache-pack	0.02 Mo	0.00 Mo	11.61 Mo	0.52 Mo
profiler	4.94 Mo	0.04 Mo	16.55 Mo	0.56 Mo
annotations	1.69 Mo	0.01 Mo	18.23 Mo	0.57 Mo
maker	3.69 Mo	0.01 Mo	21.93 Mo	0.58 Mo
debug-pack	3.42 Mo	0.02 Mo	25.35 Mo	0.61 Mo
twig	0.20 Mo	0.00 Mo	25.55 Mo	0.61 Mo
asset	1.78 Mo	0.00 Mo	27.33 Mo	0.61 Mo
orm	13.58 Mo	0.08 Mo	40.91 Mo	0.69 Mo
orm-fixtures	0.63 Mo	0.02 Mo	41.54 Mo	0.71 Mo
form	5.41 Mo	0.02 Mo	46.95 Mo	0.72 Mo
validator	4.24 Mo	0.01 Mo	51.19 Mo	0.73 Mo
security	5.47 Mo	0.02 Mo	56.66 Mo	0.75 Mo
expression-language	2.96 Mo	0.00 Mo	59.62 Mo	0.75 Mo
translation	3.45 Mo	0.01 Mo	63.07 Mo	0.76 Mo

Le site, en fonctionnement, pèse maintenant plus de 60 Mo contre les 12 Mo initiaux. Il faut aussi noter que le répertoire *var* va évoluer au fur et à mesure de l'utilisation du site.

Quant au code du site, c'est à dire la partie versionnée sous *git*, elle pèse moins de 1 Mo.
C'est l'occasion d'insister sur le fait que les répertoires *var* et *vendor* ne sont pas versionnés (cf. fichier *.gitignore*) : lorsqu'on récupère un site pour la première fois, il faut regénérer ces deux répertoires avec la commande *"composer install"*[7].
Pour information le répertoire *var* (cache) fait 24 Mo, et le répertoire *vendor* (code source du site) fait 38 Mo.

En rechargeant le site sur le navigateur, une barre d'outils apparaît en bas de la fenêtre.

4.7 Utilisation d'un IDE

Utiliser un IDE avec ce type de framework est quasiment une obligation dans le monde professionnel si l'on veut développer efficacement.

Il en existe plusieurs. Nous vous proposons d'utiliser PHPStorm qui est dédié au développement web en PHP, et à Symfony en particulier.

Ces IDE peuvent stocker leurs configurations directement dans l'arborescence de Symfony.

L'inconvénient est alors que la configuration est versionnée par *git* et donc rappatriée par les autres développeurs :

- Soit un autre développeur utilise un autre IDE et cela n'aura pas d'incidence si ce n'est lui aussi versionnera un répertoire de configuration.
- Soit un autre développeur utilise le même IDE et alors les deux développeurs s'écraseront à tour de rôle leurs configurations.

La solution la plus simple est de rajouter le répertoire de configuration dans le *.gitignore* pour qu'il ne soit pas versionné.
Par exemple pour PHPStorm, il faut rajouter le répertoire *.idea* dans le fichier *.gitignore* (celui à la racine de l'arborescence).
Cf. https://gitlab.com/subrenat/l3-web-2022-23/-/blob/TP1/.gitignore

4.8 git

On peut créer, pour clôturer le TP, une nouvelle version sous *git* (en étant dans le répertoire racine de l'arborescence) :

```
serveur$ git status
[...]
```

7. et non pas *"composer update"*

```
serveur$ git add .
serveur$ git status
[...]
serveur$ git commit -m "squelette avec composants ; ignore répertoire .idea"
[...]
# s'il y a un repository distant
serveur$ git push;
[...]
serveur$ git status
[...]
serveur$ git log --oneline --decorate --branches --graph --all
[...]
```

5 Git : code source

Le code source à l'issue de ce TP peut-être consulté à :
https://gitlab.com/subrenat/l3-web-2022-23/-/blob/TP1

Les différentes étapes du TP sont dans la branche *b-TP1*.

Voici un screenshot fait avec git-kraken (https://www.gitkraken.com) :