

Commencer à développer avec le framework symfony



Par Christopher MANEU 


Date de publication : 2 août 2007

Ce tutoriel vous donnera les bases pour bien comprendre le fonctionnement global du framework PHP symfony. Il s'adresse principalement à des développeurs PHP ayant des notions en programmation orienté-objet.

I - Introduction.....	3
II - Avant de commencer.....	4
II-A - L'installation de symfony.....	4
II-B - Développeurs PHP ? Oubliez vos habitudes !.....	4
II-B-1 - Vive la console !.....	4
II-B-2 - Le design pattern MVC et l'objet.....	5
III - La création d'un projet.....	6
III-A - Projets, Applications et cie.....	6
III-B - La création d'un nouveau projet.....	6
III-C - La création d'une application.....	7
III-D - Finitions.....	7
IV - Les fichiers de configuration.....	8
IV-A - La configuration dans symfony.....	8
IV-B - Syntaxe des fichiers YAML.....	8
IV-C - Les principaux fichiers de configuration.....	9
IV-C-1 - config.php.....	9
IV-C-2 - i18n.yml.....	9
IV-C-3 - logging.yml.....	9
IV-C-4 - routing.yml.....	9
IV-C-5 - security.yml.....	10
IV-C-6 - settings.yml.....	10
IV-C-7 - view.yml.....	10
V - La base de données.....	12
V-A - Relier son projet à une base de données.....	12
V-A-1 - properties.ini.....	12
V-A-2 - databases.yml.....	12
V-A-3 - Propel.ini.....	12
V-B - Le fichier schema.yml.....	13
Créer le fichier.....	13
Avec un éditeur de texte.....	13
A partir d'une base de données existante.....	13
V-C - Préparer symfony pour la base de données.....	14
VI - Les modules.....	15
VI-A - Création d'un module.....	15
VI-A-1 - Création d'un module vierge.....	15
VI-A-2 - Création d'un module CRUD.....	15
VI-B - Structure des modules.....	15
VI-B-1 - Création d'une nouvelle action.....	16
VI-C - Les templates.....	16
VII - Les objets de base.....	17
VII-A - Les liens dans symfony.....	17
VII-B - Passer une variable de l'action au template.....	17
VII-C - Gérer les utilisateurs et les sessions.....	17
VII-C-1 - Gestion des attributs de la session.....	17
VII-C-2 - Connexion de l'utilisateur.....	18
VII-C-3 - Gestion des permissions.....	18
VII-D - La création d'un formulaire.....	18
VII-D-1 - Les champs standards.....	19
VII-D-2 - Les champs en liaison avec la base de données.....	19
VII-D-3 - Les validations.....	19
Les validations basiques.....	19
Les validations avancées.....	20
VIII - La génération du backend d'administration.....	21
IX - Conclusion.....	22
Pour aller plus loin.....	22

I - Introduction

Le web est un environnement est en pleine mutation. De nombreuses technologies et outils font leur apparition et nous proposent tous les jours de plus en plus d'interactivités dans nos pages web (« Atlas », script.aculo.us, Adobe Integrated Runtime...). Toutes ces technologies transforment nos chers navigateurs en véritable plateformes. Cependant, on oublie souvent en voyant toutes ces technologies que l'évolution se situe aussi du côté du serveur. Le langage PHP lui aussi témoigne de ces profonds changements dans notre manière de développer. Le langage PHP dispose depuis quelques temps de bibliothèques de code permettant d'optimiser le développement avec ce langage (tel que PEAR). Cependant, l'avènement de Ruby on rails a démontré les nombreux avantages à posséder une architecture  **MVC**. symfony fait partie des  **frameworks** inspirés du modèle proposé par rails (tels que Prado, CodeIgniter, ou bien CakePHP)

symfony n'est pas développé de zéro. Il intègre d'autres composants déjà approuvés par un grand nombre tel que Propel pour l' **ORM**, une couche d'abstraction de base de données propulsée par Creole, et Mojavi pour l'architecture MVC.

La documentation sur ce framework est très aboutie, cependant, certains concepts peuvent être longs à acquérir. Ce tutoriel a pour objectif de vous donner des bases solides dans la compréhension de ce framework afin que vous puissiez commencer à développer avec, ou continuer votre apprentissage.

II - Avant de commencer

Lorsque l'on est développeur PHP, on a souvent l'habitude d'utiliser des scripts, des bibliothèques déjà existantes dans nos projets (et les includes sont légions en en-tête de nos scripts). A la différence d'autres frameworks PHP - tel que le Zend Framework - ou de librairies tierces, symfony nécessite une installation un peu plus lourde.





symfony est utilisable de deux façons :

- **Traditionnelle:** Le framework est installé directement sur le serveur web dans le répertoire de PHP. Il est partagé entre tous les projets symfony du serveur. L'utilisation des liens symboliques et la gestion des droits unix est nécessaire.
- **Freeze:** Dans ce mode, le dossier de votre projet embarque tous les fichiers nécessaires au framework. On ne travaille généralement pas avec ce mode, par contre, il est tout à fait adapté au déploiement.

Vous avez la possibilité de passer d'une application traditionnelle à une application freeze et vice-versa.

II-A - L'installation de symfony

Il y a de nombreuses façons d'installer symfony, selon vos connaissances, votre plateforme, ... La plus simple reste l'utilisation de PEAR. Voici un ensemble de liens qui vous permettront d'installer symfony quelque soit votre plateforme :

-  **Installer symfony sur Ubuntu Dapper Drake**
-  **Installer symfony via PEAR**
-  **Installer symfony sous MacOSX**
-  **Installer symfony sur WAMP**

Je vous propose également deux versions prêtes à l'emploi :


- Zazou Mini Web Server avec la symfony Sandbox (Windows) (**ftp**) (**http de secours**)
- Une machine virtuelle (vmware) avec PHP, Perl, Python, MySQL et symfony (accessible via un partage réseau) (**ftp**) (**http de secours**)

La première version vous permettra de vous familiariser avec le framework, tandis que la seconde version vous permettra de tester et créer vos applications dans un environnement identique à la production.

II-B - Développeurs PHP ? Oubliez vos habitudes !

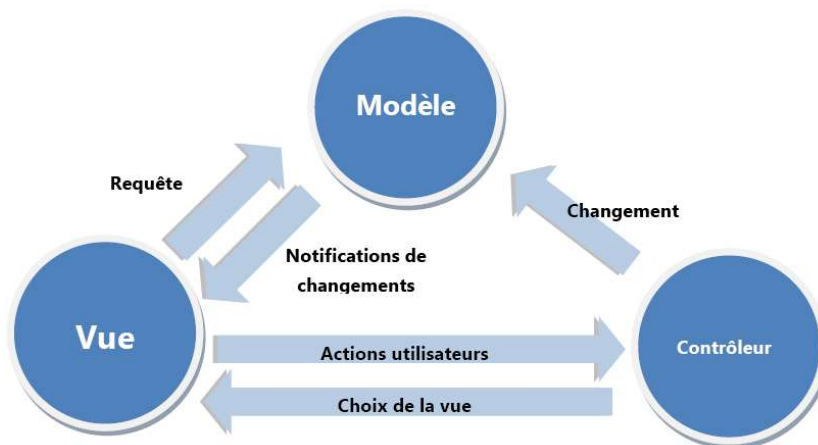
Les détracteurs du langage PHP ont de nombreuses raisons de se plaindre : lenteur, permissivité, pas de séparation entre le code et le design, etc... symfony apporte de nombreuses réponses (que vous découvrirez tout au long de ce tutorial ;-). Cependant, tout a un prix ! Certains développeurs PHP « traditionnels » peuvent être confrontés à des principes ou à l'utilisation d'outils auxquels ils ne sont pas habitués.

II-B-1 - Vive la console !

Développer avec symfony, c'est aussi réveiller le côté vim qui est en vous ! En effet, vous serez amené à exécuter des lignes de commandes de façon très régulière. Si cela fait un moment que vous n'avez pas touché à la ligne de commande, je vous conseille de faire un tour par la **FAQ Linux**. Ces commandes permettent notamment de créer un nouveau projet, une nouvelle application, d'utiliser Propel... Vous trouverez sur le **blog d'Andréia Bohner** une  **fiche récapitulative** des commandes symfony les plus importantes. Avant de faire une commande, assurez-vous d'être à la racine de votre projet !

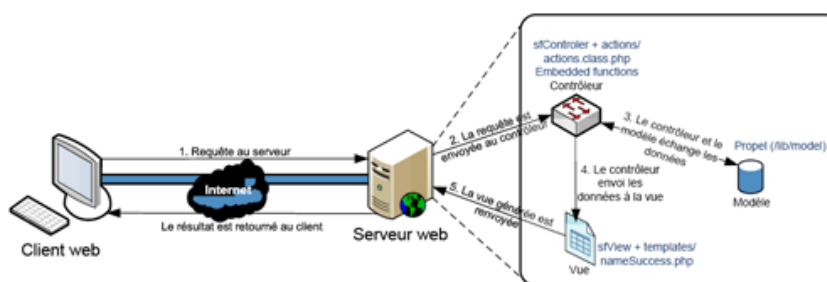
II-B-2 - Le design pattern MVC et l'objet

symfony est entièrement basé sur le design pattern MVC (Modèle-Vue- Contrôleur). Utilisé dans d'autres d'autres technologies (notamment Java), il n'a été largement diffusé dans le monde PHP qu'avec des frameworks tels que symfony. Le MVC est un pattern architectural qui sépare les données (c-à-d le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur).



Le fonctionnement du MVC

Ce modèle de conception impose donc une séparation en 3 couches : - Le modèle : Il représente les données de l'application. Il définit aussi l'interaction avec la base de données et le traitement de ces données. - La vue : Elle représente l'interface utilisateur, ce avec quoi il interagit. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données que lui fournit le modèle. Il peut tout à fait y avoir plusieurs vues qui présentent les données d'un même modèle. - Le contrôleur : Il gère l'interface entre le modèle et le client. Il va interpréter la requête de ce dernier pour lui envoyer la vue correspondante. Il effectue la synchronisation entre le modèle et les vues. symfony implémente donc trois couches répondant à ce pattern. Le schéma ci-dessous illustre le mécanisme du MVC dans un projet symfony lorsque l'utilisateur navigue sur le site.



Le MVC dans symfony

Pour une courte introduction à MVC, je vous conseille la lecture de la première partie du **tutoriel de Baptiste Wicht** (pour une introduction rapide), et la lecture du **cours de Serge Tahé** pour aller plus loin.

La programmation avec le framework symfony est clairement orienté objet. Cependant de nombreux développeurs PHP n'ont pas encore sauté le pas. Si vous êtes dans ce cas, il vous faudra donc passer par l'étape apprentissage de l'objet ! (par exemple en consultant les **tutoriels disponibles** à ce sujet). On peut maintenant entrer dans le vif du sujet : la création de votre premier projet !

III - La création d'un projet

La création d'un projet symfony commence par la création de la structure de base de celui-ci, c'est-à-dire un ensemble de dossiers et de fichiers (voir la [symfony cheat sheet](#)).

III-A - Projets, Applications et cie

symfony permet à partir d'un seul projet de gérer plusieurs applications, par exemple une pour le frontend et une autre pour le backend. Voici quelques précisions sur les différents niveaux d'un projet :

- **Le projet** : C'est la structure qui englobe tout. C'est à son niveau que l'on spécifie les informations sur la base de données.
- **L'application** : Elle correspond généralement à un site. C'est à ce niveau qu'il y a le plus de configuration à effectuer. Un projet peut comporter plusieurs applications, cependant, dans la plupart du cas, il n'y en aura qu'une.
- **Les modules** : Ce sont des 'unités fonctionnelles' de votre applications. (Par exemple : clients, produits, commandes). C'est à ce niveau que vous allez travailler le plus !
- **Les actions et les templates** : Les modules comportent des couples d'actions-templates. C'est l'application du pattern MVC, et également la séparation entre la partie « traitement » et la partie « affichage ». Toutes les opérations sur la base de données et les autres traitements (web services, ftp, mail, etc...) sont réalisés dans le fichier d'actions (monModule/actions/action.class.php), c'est ensuite le fichier de template (monModule/templates/monActionSuccess.php) qui se charge du rendu graphique.

Le schéma ci-dessous représente les différents niveaux d'un projet symfony (extrait de la conférence symfony-AFUP).



Les différents niveaux dans symfony

III-B - La création d'un nouveau projet

La création d'un nouveau projet passe par l'utilisation de la ligne de commande. Il vous faut créer un répertoire dans lequel placer votre projet. Vous pouvez le faire via l'interface graphique (selon votre serveur) ou via la ligne de commande :

```
mkdir monProjet
cd monProjet
```

Une fois que vous êtes bien dans le répertoire de votre projet, exécuter la commande de création d'un nouveau projet, en remplaçant monProjet par le nom de votre projet :

```
symfony init-project monProjet
```

III-C - La création d'une application

Vous devez ensuite créer une application dans votre projet (vous pourrez par la suite refaire cette étape si vous souhaitez créer plusieurs applications au sein de votre projet). Là aussi, une simple ligne de commande suffit :

Création d'une nouvelle application

```
symfony init-app monProjet monAppli
```

Une nouvelle arborescence va être créée dans monProjet/apps/monAppli. Elle contient un ensemble de répertoires :

- **Config** : Ce répertoire contient tous les fichiers de configuration propre à l'application (cache, routing, sécurité, ...)
- **I18n** : C'est ici que sont stockés les fichiers pour l'internationalisation (régionalisation) de votre projet, notamment les fichiers de traduction (généralement messages.fr.xml).
- **Lib** : Ce dossier contient les bibliothèques personnelles que vous pouvez ajouter. Il contient également un fichier appelé myUser.class.php. Ce fichier ne doit **pas être supprimé ni renommé** (il dérive d'une classe User nécessaire au fonctionnement du site, y compris si votre site ne comporte pas de gestion d'utilisateur !). Vous pouvez surcharger les méthodes afin de personnaliser la gestion des utilisateurs.
- **Modules** : Vide lors de la création de votre application, c'est lui qui contiendra tous les modules de votre application (notamment les fichiers d'actions et les templates).
- **Template** : Contient le template dynamique par défaut (layout.php). C'est lui qui inclut tous les metas et le titre de la page. C'est également lui qui réalise l'inclusion du template de l'action en cours.

III-D - Finitions

Il vous reste une dernière petite chose à effectuer avant de profiter de symfony. Vous devez créer un lien symbolique entre un répertoire de l'installation de symfony et votre projet, afin d'avoir les styles par défaut de symfony et l'accès à la barre web de débogage. Pour se faire, vous devez effectuer la commande suivante (c'est un exemple pour la machine virtuelle, il vous faudra l'adapter à votre version de *nix et votre installation de symfony)

```
ln -s /usr/share/php/data/symfony/web/sf ./web/sf
```

Vous pouvez maintenant passer à la configuration de votre projet !


IV - Les fichiers de configuration

Les fichiers de configuration au sein de symfony sont nombreux. Cette partie va présenter où ils se situent, leur fonctionnement ainsi que leur syntaxe, et enfin quelques informations sur les principaux fichiers de configuration à connaître et à modifier.

IV-A - La configuration dans symfony

La configuration s'effectue à plusieurs niveaux : Le projet, l'application et les modules. On peut tout à fait définir une configuration globale à une application mais surcharger la configuration d'un module particulier en changeant les informations de son fichier de configuration (les noms et le contenu sont identiques entre les fichiers de configuration d'une application et d'un module). Les fichiers de configuration pour une application sont situés dans *monProjet/apps/monAppli/config*. Les fichiers de configuration pour un module sont situés dans *monProjet/apps/monAppli/modules/monModule/config*. Il n'y a pas de documentation exhaustive sur ces fichiers de configuration. Cependant la plupart des informations utiles sont situées directement dans les fichiers de configuration, sous forme de commentaires.

IV-B - Syntaxe des fichiers YAML

Les fichiers de configuration de symfony sont quasiment tous au format  **YAML** (extension .yaml). Basé sur un fichier texte, ils présentent une syntaxe simple. Les différents niveaux hiérarchiques sont représentés par des espaces avant le texte.

```
prod:      <- Niveau 1: la ligne se termine par deux points (:)
.settings:  <- Niveau 2: la ligne commence par des espaces et se termine par deux points
  no_script_name: off <- Paramètre: la valeur commence au premier caractère (espace non compté) après
  les deux points (:)

dev:        <- Niveau 1
error_reporting: 4095 <- Paramètre
web_debug: on    <- Paramètre
cache: off       <- Paramètre
```



Les fichiers YAML ne supportent pas les tabulations. N'utilisez que des espaces !


```
prod:
  .settings:
    no_script_name:      off

dev:
  .settings:
    # E_ALL | E_STRICT = 4095
    error_reporting:     4095
    web_debug:           on
    cache:               off
    no_script_name:      off
    etag:                off

test:
  .settings:
    # E_ALL | E_STRICT & ~E_NOTICE = 2047
    error_reporting:     2047
    cache:               off
    web_debug:           off
    no_script_name:      off
    etag:                off

#all:
#  .actions:
#    default_module:     default # Default module and action to be called when
#    default_action:     index  # A routing rule doesn't set it
#
#    error_404_module:    default # To be called when a 404 error is raised
#    error_404_action:   error404 # Or when the requested URL doesn't match any route
```

Exemple de fichier YML (avec commentaires)

IV-C - Les principaux fichiers de configuration

Il y a de nombreux fichiers de configuration dans symfony. Voici la liste des principaux fichiers, ainsi que les points importants pour chacun des fichiers.

IV-C-1 - config.php

Ce fichier est généré automatiquement à la création de l'application. C'est un fichier PHP. Il contient les emplacements du framework. Ce fichier ne doit être modifié que si vous changez l'emplacement de l'installation du framework. Il n'est pas surchargeable (au niveau des modules).

IV-C-2 - i18n.yml

Ce fichier contient les paramètres d'internationalisation de l'application. Il permet de paramétrer la langue par défaut, le 'backend' (c'est-à-dire le type de fichiers permettant la traduction. Il y a, entre autres, le format XLIFF, PO et CSV).

IV-C-3 - logging.yml

symfony embarque un système de logs. C'est ce fichier qui permet de le configurer. Vous pouvez notamment choisir le niveau de log, la période, s'il est rotatif ou pas, l'emplacement du fichier de log.

IV-C-4 - routing.yml

Ce fichier permet de gérer la réécriture d'URL. Il est important de le modifier afin que la première page de votre projet ne soit pas la page par défaut de symfony. Pour faire ceci, vous devez créer une action dans un module qui sera exécutée lors de l'ouverture de votre site.

Début du fichier routing.yml

```
# default rules
homepage:
url: /
param: { module: monModule, action: index }
```

IV-C-5 - security.yml

symfony intègre un système de contrôle d'accès, vous permettant de gérer par modules et/ou par actions qui a le droit d'exécuter l'action ou de modifier des données. Le fichier security.yml, qui se retrouve au niveau de l'application (respectivement des modules), vous permet de paramétrer pour chacun des modules (resp. actions) s'il faut être identifié et s'il faut appartenir à un certain groupe d'utilisateurs. Il est préférable d'utiliser le fichier security.yml au niveau des modules et non de l'application. Cependant, il n'existe pas lors de la création du module. Il faut donc le créer manuellement, dans le dossier monProjet/apps/monAppli/modules/monModule/config/. Voici un exemple de fichier :

```
read: # Début de la section pour la lecture (toutes actions)
is_secure: off # Positionné à off, tout le monde peut accéder à l'action en lecture
update: # Début de la section pour la modification
is_secure: on # Positionné à on, il faut être "connecté" pour activer la mise à jour
credentials: admin # Ici, il faut avoir le privilège 'admin' pour réaliser une maj
editArticle: # Début de la section pour l'action editArticle
credentials: admin # Il faut avoir le privilège 'admin' pour pouvoir exécuter l'action
editArticle
publishArticle: # Début de la section pour l'action publishArticle
credentials: publisher # Il faut avoir le privilège 'publisher' pour pouvoir exécuter l'action
publishArticle
```

IV-C-6 - settings.yml

C'est dans le fichier settings.yml que sont définis les principaux paramètres d'une application symfony : internationalisation, cache, désactivation de l'application, utilisation de composants, gestion de l'encodage, ... Il est disponible uniquement au niveau de l'application. Vous avez la possibilité de définir plusieurs environnements dans de fichier. Ainsi, vous pouvez avoir différents paramètres pour le développement (par exemple, toutes les options de débogage activées et pas de cache), pour les tests (cache+débogage) et pour la production (juste le cache).

```
prod:
  .settings:
    web_debug:      off
    cache:          on

dev:
  .settings:
    web_debug:      on
    cache:          off

test:
  .settings:
    cache:          on
    web_debug:      on
```

IV-C-7 - view.yml

Vous commencez votre site sous symfony. En bon développeur, vous commencez à insérer dans votre page les liens vers vos fichiers CSS et Javascript ? Oubliez cela ! C'est le fichier view.yml qui va gérer cela pour vous ! La mise en page est réalisée à partir des éléments suivants :

- Le « cœur » de la page est généré par le fichier de template
- La mise en page globale (l'en-tête, le pied de page, éventuellement les colonnes) proviennent du fichier de layout (layout.php)
- Tout le reste - c'est-à-dire le titre de la page, les fichiers CSS et Javascript à inclure, les métas, les mots clés, la description, le délai du cache... - est configuré dans le fichier view.yml.

Vous avez la possibilité de définir des paramètres globaux dans le fichier view.yml de l'application (monProjet/apps/monAppli/config/view.yml), et personnaliser la vue pour certains modules (en créant un fichier view.yml dans monProjet/apps/monAppli/modules/monModule/config/). Cela peut vous être utile, par exemple, si vous utilisez de l'ajax dans un module spécifique.

Exemple de fichier view.yml au niveau de l'application

```
default:
  http_metas:
    content-type: text/html; charset=utf-8

  metas:
    title:      Bienvenue sur developpez.com
    robots:    index, follow
    description: developpez.com le site no 1 des developpeurs
    keywords:   informatique, developpement, cours, tutorial, programmation
    language:   fr

  stylesheets: [main, develz]

  javascripts: [jquery/src/jquery/jquery.js, libperso/popup.js]

  has_layout: on
  layout:     layout
```

Exemple de fichier view.yml au niveau d'un module


```
articleSuccess:
  metas:
    title: Article - Developpez.com

faqSuccess:
  metas:
    title: Foire aux questions - Developpez.com
  javascripts: [qr.js]
```

Maintenant que les fichiers YAML n'ont plus de secrets pour vous, nous allons passer à la partie modèle, c'est-à-dire la base de données.

V - La base de données

symfony intègre de nombreux composants pour la gestion des bases de données. Il est capable de gérer la connexion à différents moteurs, créer les tables, importer des données... Cette partie va vous permettre d'avoir les bases de la gestion des bases de données sous symfony.

 *symfony effectue une conversion implicite entre la version relationnelle et objet d'une base. Le champ `nom_utilisateur` dans la base de données sera converti en `nomUtilisateur` (Camel style). Il est fortement conseillé de ne pas utiliser ce style directement dans la base de données.*

V-A - Relier son projet à une base de données

Pour relier votre projet à une base de données, vous devez modifier certains fichiers de configuration.

V-A-1 - properties.ini

Ce fichier est simple, mais il est très important. C'est lui qui contient le nom du projet. Il doit être identique dans tous les autres fichiers de configuration (`databases.yml`, `Propel.ini`, `schema.yml`).


```
[symfony]
name=monProjet
```

V-A-2 - databases.yml

Ce fichier contient les paramètres de connexion à la base de données. Il se situe au niveau du projet (`monProjet/config/databases.yml`), mais il est possible de le surcharger pour une application (en le créant dans `monProjet/apps/monAppli/config/databases.yml`)

```
all:
monProjet:
class: sfPropelDatabase
param:
datasource: monProjet
phptype: mysql
hostspec: localhost
database: maBase
username: root
password: toor
```

V-A-3 - Propel.ini

Le fichier `Propel.ini` reprend les mêmes informations que le fichier `databases.yml`, mais il est destiné à la couche  **ORM** (Object-relational mapping) de symfony.

```
Propel.project = monProjet
Propel.database = mysql
Propel.database.createUrl = mysql://localhost/
Propel.database.url = mysql://localhost/monProjet_dev
```

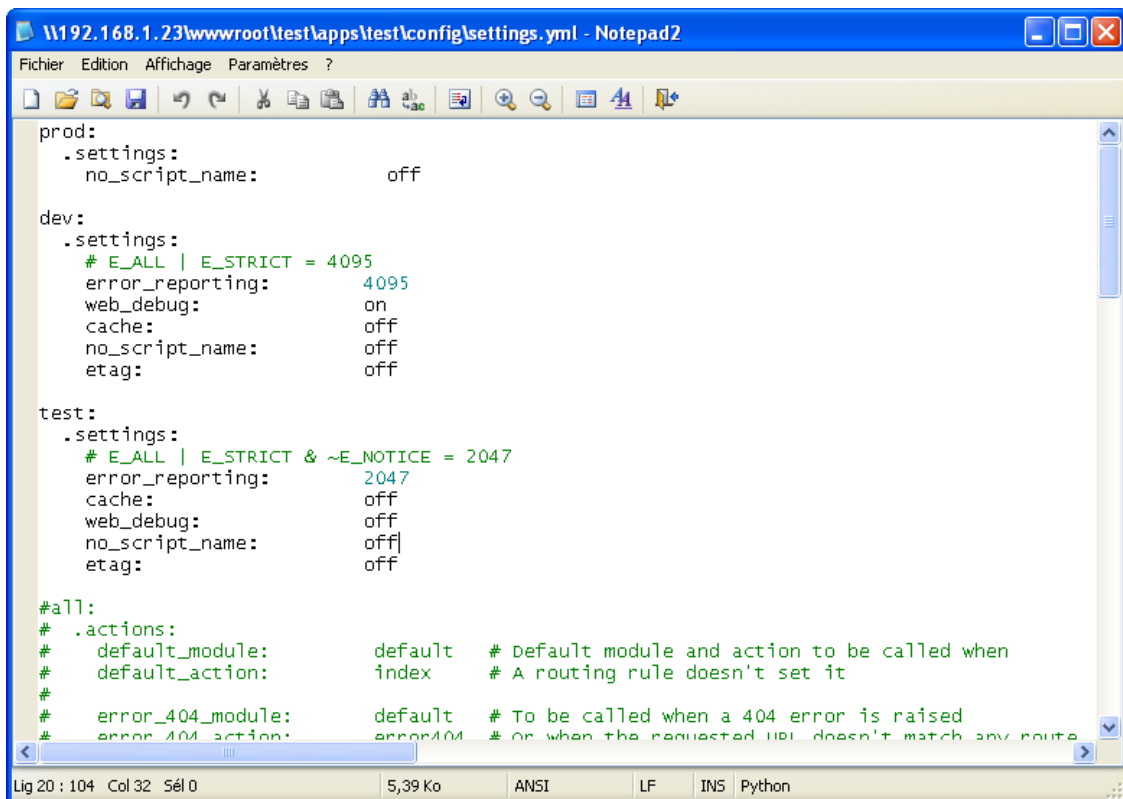
V-B - Le fichier schema.yml

Le fichier schema.yml contient une représentation du modèle relationnel de la base de données. Il est nécessaire pour pouvoir réaliser le mapping objet, ainsi que pour la génération de pages (CRUD et admin)

Créer le fichier

Avec un éditeur de texte

Le fichier schema.yml étant un fichier YAML, vous en connaissez déjà la syntaxe. Le niveau 1 correspond à une table et le niveau 2 à un champ. Le contenu du niveau 2 est entre accolades et il regroupe toutes les caractéristiques du champ (type, taille, ...). L'exemple ci-dessous vous explique comment structurer votre fichier pour une table.




```
prod:
  .settings:
    no_script_name:      off

dev:
  .settings:
    # E_ALL | E_STRICT = 4095
    error_reporting:     4095
    web_debug:          on
    cache:               off
    no_script_name:     off
    etag:                off

test:
  .settings:
    # E_ALL | E_STRICT & ~E_NOTICE = 2047
    error_reporting:     2047
    cache:               off
    web_debug:          off
    no_script_name:     off
    etag:                off

#all:
#  .actions:
#    default_module:     default # Default module and action to be called when
#    default_action:     index  # A routing rule doesn't set it
#
#    error_404_module:   default # To be called when a 404 error is raised
#    error_404_action:   error404 # Or when the requested URL doesn't match any route
```

La syntaxe du fichier schema.yml

Le fichier schema.yml peut être remplacé par un fichier au format XML (appelé schema.xml). Il est beaucoup plus compliqué d'éditer un tel fichier, cependant, vous pouvez utiliser le logiciel  **DbDesigner** et convertir le fichier XML de ce logiciel en schema.xml sur [ce site](#).

A partir d'une base de données existante

symfony offre également la possibilité de créer ce fichier (au format YAML ou XML) à partir d'une base de données existante. Il faut tout d'abord paramétrer la connexion à la base de données (fichier properties.ini, databases.yml et Propel.ini), puis exécuter la commande suivante :

Création du fichier schema au format YAML

```
symfony propel-build-model
```

Création du fichier schema au format XML

```
symfony propel-build-model xml
```

V-C - Préparer symfony pour la base de données

symfony sait maintenant comment est structurée votre base de données. Il faut maintenant faire la passerelle entre la base (relationnelle) et le php (en objet). C'est la génération du modèle. Cette étape va créer les classes correspondantes aux tables indiquées dans le fichier schema.yml. Les fichiers sont générés dans le répertoire `monProjet/lib/model/om/`. Cette génération est réalisée avec la ligne de commande suivante : `symfony propel-build-model`. Vous avez la possibilité de surcharger toutes les méthodes créées par symfony dans le fichier `maTable.php` situé dans `monProjet/lib/model/om/`. Il ne vous reste plus qu'à créer les tables dans votre base de données. Vous avez deux solutions pour cela, en ligne de commande : Créer un fichier SQL et l'exécuter vous-même :

Création du fichier SQL

```
symfony propel-build-sql
```

Laisser symfony se connecter à la base de données et créer les tables :

symfony crée la base de données

```
symfony propel-build-db
```

VI - Les modules

VI-A - Création d'un module

La création d'un module s'effectue en ligne de commande. Vous pouvez créer un module vierge (ce sera alors à vous de créer toutes les actions), ou bien de créer un module permettant d'effectuer les opérations de bases sur une table.

VI-A-1 - Création d'un module vierge

La création d'un module vierge se fait via la ligne de commande :

```
symfony init-module monProjet monModule
```

Il est conseillé de ne pas utiliser de majuscules ni de caractères spéciaux ou d'espaces dans le nom du module. Ce module nouvellement créé ne contient que le fichier actions.class.php et l'arborescence d'un module. Si votre module n'utilise pas ou très peu la base de données, c'est un module vierge qu'il faut créer. Cependant, si vous créez un module qui utilise la base de données, vous avez intérêt à utiliser un module CRUD.

VI-A-2 - Création d'un module CRUD

L'un des grands intérêts de symfony est de pouvoir créer des modules vous permettant d'effectuer toutes les opérations courantes sur votre base de données, à savoir :

- Voir la liste des enregistrements
- Voir un enregistrement
- Modifier un enregistrement
- En créer un nouveau
- En supprimer

C'est la génération CRUD (Create-Retrieve-Update-Delete). La génération s'effectue par la ligne de commande suivante :

```
symfony propel-generate-crud monAppli monModule Maclasse
```

Le paramètre monModule correspond au nom du module que vous souhaitez créer. Maclasse correspond au nom de la classe de la base de données. Les deux paramètres sont généralement identiques, mais le second commence toujours par une majuscule. Cette génération va créer toutes les actions et les templates requis pour effectuer ces opérations de base. Vous pouvez bien sûr modifier ce module comme le module vierge.

VI-B - Structure des modules

Lors de la création d'un module, un nouveau répertoire est créé dans /monProjet/apps/monAppli/modules. Il contient 5 dossiers :

- Actions : Ce dossier contient le fichier actions.class.php. C'est lui qui contient toutes les méthodes-actions du module
- Config : il permet de surcharger la configuration de l'application (il est vide par défaut). La configuration est principalement surchargée pour la sécurité (security.yml) et pour l'affichage (view.yml)
- Lib : Ce dossier permet d'insérer des classes personnalisées accessibles uniquement dans ce module.
- Templates : Contient tous les templates associés aux actions du module

- **Validate** : Ce dossier contient les fichiers de configuration permettant la validation automatique des formulaires.

VI-B-1 - Création d'une nouvelle action

Toutes les actions pour un module sont regroupées dans un fichier : /monProjet/apps/monAppli/modules/monModule/actions/actions.class.php. Tout d'abord, il faut créer dans ce fichier une nouvelle méthode (public function). Le nom de la méthode doit obligatoirement commencer par execute et être suivie du nom de votre action (la première lettre en majuscule) :

```
public function executeNouveauclient()  
{  
    ...  
}
```

Ce fichier va vous permettre d'effectuer vos opérations, mais pas l'affichage. Pour cela, vous devez passer par les templates.

VI-C - Les templates

Comme vous avez pu le remarquer, il existe un dossier templates dans votre module. Par défaut, lorsque l'action monAction est exécutée, c'est le fichier monActionSuccess.php dans ce dossier qui est appelé. Vous devez donc créer ce fichier dans le dossier templates. Vous avez également la possibilité de modifier le fichier template qui sera appelé dans le code de votre action. Cela vous permet de modifier dynamiquement le template qui sera appelé par une action donnée.

```
$this->setTemplate('monTemplate');
```

symfony offre également la possibilité de se passer de fichier template et de préparer la sortie directement dans le fichier actions. Cette méthode n'est pas conseillée et vous devez la réserver pour des cas bien spécifiques (comme le renvoi de JSON lors d'un appel ajax).

```
public function executeIndex()  
{  
    echo "<html><body>Bonjour à tous</body></html>";  
    return sfView::NONE;  
}  
Ou  
public function executeIndex()  
{  
    return $this->renderText("<html><body>Hello, World!</body></html>");  
}
```


VII - Les objets de base

VII-A - Les liens dans symfony

symfony utilise la réécriture d'URL. C'est pourquoi il ne faut pas faire les liens directement en HTML (en utilisant la balise `<a link>`). Les liens pointent vers une action (login) d'un module (utilisateur) :

```
<?php echo link_to("utilisateur/login") ?>
```

Pour passer des paramètres à la page appelée, vous pouvez utiliser la syntaxe habituelle :

```
<?php echo link_to("utilisateur/login?email=chris@bulles.org") ?>
```

VII-B - Passer une variable de l'action au template

Tous les résultats des traitements que vous réalisez dans votre fichier d'actions doivent être passés au fichier template pour être affichés. Il faut tout d'abord déclarer cette variable dans l'action. Cette variable doit être de la forme : `$this->maVar` Pour récupérer cette variable dans le template, il suffit de faire : `$maVar` Ce « changement » de variables entre les actions et les templates peut être déroutant au début. Il est également valable pour « les objets de la base de données » :

dans le fichier actions.class.php

```
$this->utilisateur = UtilisateurPeer ::doSelectOne($cr) ;
```

dans le fichier de template

```
<p>Bienvenue <?php echo $utilisateur->getNom() ?>, c'est votre <?php echo $utilisateur->getNbConnexion() ?>eme connexion.</p>
```

VII-C - Gérer les utilisateurs et les sessions

symfony intègre un système complet de gestion des sessions et des utilisateurs. Voici les bases de cette gestion, de très nombreuses fonctionnalités ne sont pas abordées ici. La documentation vous permettra d'approfondir ces concepts. Gestion des

VII-C-1 - Gestion des attributs de la session

Vous avez la possibilité de stocker des attributs dans l'objet session, comme par exemple la liste des articles du panier d'achat. Deux fonctions vous permettent d'écrire et de lire ces attributs :

Dans les actions

```
$this->getUser()->setAttribute('nbArticles', 15) ;  
$this->getUser()->getAttribute('nbArticles') ;
```

Dans les templates

```
$sf_user->setAttribute('nbArticles', 15) ;  
$sf_user->getAttribute('nbArticles') ;
```

VII-C-2 - Connexion de l'utilisateur

Vous pouvez stocker l'état de l'utilisateur : connecté ou déconnecté.

Dans les actions

```
$this->getUser()->setAuthenticated(true) ; // Connecte  
$this->getUser()->setAuthenticated(false) ; // Deconnecte  
$this->getUser()->isAuthenticated() ; // Vrai si connecté
```

Dans les templates

```
$sf_user->setAuthenticated(true) ; // Connecte  
$sf_user->setAuthenticated(false) ; // Deconnecte  
$sf_user->isAuthenticated() ; // Vrai si connecté
```

VII-C-3 - Gestion des permissions

Même si de nombreux sites se contenteraient d'un état « Administrateur », d'autres ont besoin d'une gestion beaucoup plus fine (un blog peut avoir par exemple un auteur, des utilisateurs pouvant poster des commentaires, et des modérateurs). Voici comment utiliser ces permissions.

Ajouter une permission :

```
$this->getUser()->addCredential('auteur') ;
```

Tester si l'utilisateur a une permission particulière :

```
$this->getUser()->hasCredential('modérateur') ;
```

Supprimer une permission :

```
$this->getUser()->removeCredential('auteur') ;
```

Supprime toutes les permissions :

```
$this->getUser()->clearCredentials() ;
```


VII-D - La création d'un formulaire

Vous allez faire votre entrée dans le monde des helpers ! Les helpers sont des méthodes php qui génèrent du code Javascript. Elles permettent de générer du code propre, respectueux des standards et tout ceci simplement. Les helpers sont présents dans de nombreux domaines : Javascript, AJAX, Flash, formatage de données, régionalisation et les formulaires ! La première étape pour créer un formulaire est de débiter par la balise `<form>`. Le code suivant va créer l'entête du formulaire

```
<?php echo form_tag('user/create') ?>
```

Il faut ensuite créer tous les champs de votre formulaire. Il existe deux grands types d'helpers : les standards (commençant par `input`) et ceux reliés à la base de données (object helper - commençant par `object`). Il faut ensuite fermer vous-même le formulaire (en fermant la balise `</form>`).

VII-D-1 - Les champs standards

Les extraits de code suivants regroupent tous les helpers standards. C'est une traduction (partielle) du  **manuel de symfony**.

```
// Champ de saisie (input)
<?php echo input_tag('nom', 'valeur par défaut') ?>
// Tous les helpers supportent des paramètres additionnels.
// Ils vous permettent de spécifier des attributs au code généré
<?php echo input_tag('nom', 'valeur par défaut', 'maxlength=20') ?>
// Champ texte multiligne (text area)
<?php echo textarea_tag('nom', 'valeur par défaut', 'size=10x20') ?>
// Case à cocher (Check box)
<?php echo checkbox_tag('coche moi', 1, true) ?>
// Sélecteur (Radio button)
<?php echo radiobutton_tag('status[]', 'value1', true) ?>
<?php echo radiobutton_tag('status[]', 'value2', false) ?>
// Liste déroulante (Combo)
<?php echo select_tag('paiement',
options_for_select(array('Visa', 'Eurocard', 'Mastercard')
)
)?>
// Envoi de fichier
<?php echo input_file_tag('name') ?>
// Champ mot de passe
<?php echo input_password_tag('name', 'value') ?>
// Champ caché
<?php echo input_hidden_tag('name', 'value') ?>
// Bouton validation (texte)
<?php echo submit_tag('Save') ?>
// Bouton validation (image)
<?php echo submit_image_tag('submit_img') ?>
```

VII-D-2 - Les champs en liaison avec la base de données

Ces champs sont automatiquement remplis avec la valeur de la base de données. Vous devez pour cela préciser la variable qui contient l'objet enregistrement ainsi que l'accesseur :

```
<?php echo object_input_tag($client, 'getTelephone') ?>
```

Voici la liste des helpers objets :

```
<?php echo object_input_tag($object, $method, $options) ?>
<?php echo object_input_date_tag($object, $method, $options) ?>
<?php echo object_input_hidden_tag($object, $method, $options) ?>
<?php echo object_textarea_tag($object, $method, $options) ?>
<?php echo object_checkbox_tag($object, $method, $options) ?>
<?php echo object_select_tag($object, $method, $options) ?>
<?php echo object_select_country_tag($object, $method, $options) ?>
<?php echo object_select_language_tag($object, $method, $options) ?>
```

Les helpers objets sont massivement utilisés dans la generation de modules CRUD et des modules d'administration.

VII-D-3 - Les validations

Les validations basiques

Neuf validateurs standards sont intégrés à symfony. Ils s'utilisent à partir de fichiers YAML.

- Chaînes
- Nombres
- Email
- URL
- Expression régulière
- Comparaison
- Doublet dans la base de données
- Fichier
- Méthode personnelle

Pour créer un validateur sur l'action utilisateur/inscription, il faut créer un fichier inscription.yml dans le dossier monProjet/apps/monAppli/modules/utilisateur/validate. Voici un exemple de fichier de validation :

```
methods:
  post: [nom, email, tel, adresse, cp, ville]
names:
  nom:
    required: true
    required_msg: Veuillez renseigner votre nom.
    validators: stringValidator
  email:
    required: true
    required_msg: L'adresse e-mail est requise.
    validators: emailValidator
  tel:
    required: true
    required_msg: Vous devez entrer un num&eacute;ro de t&eacute;l&eacute;phone.
    validators: [phoneValidator, phoneLengthValidator]
  cp:
    required: true
    required_msg: Le code postal est requis.
    validators: [codeValidator, codeIntValidator]

stringValidator:
  class: sfStringValidator
  param:
    min: 3
    min_error: Le champ doit comporter au moins 3 caract&egrave;res.
codeValidator:
  class: sfStringValidator
  param:
    min: 5
    min_error: Le code postal doit comporter au minimum 5 chiffres.
```

Les validations avancées

Pour des cas complexes, vous pouvez réaliser des validations avancées. Ces validations n'utilisent pas de fichiers YML. Elles vous permettent d'exécuter du code PHP afin de personnaliser la validation. Si vous voulez valider l'action update, créez dans le fichier action.class.php une méthode valideInscription. symfony l'exécutera automatiquement avant executeInscription. Si cette action renvoie true, l'exécution se poursuit. Dans le cas contraire, la page d'erreur (ici inscriptionError.php) sera affichée.


VIII - La génération du backend d'administration

On a déjà vu que symfony était capable de générer des modules avec les principales fonctionnalités (CRUD). Vous avez également la possibilité de créer une interface administration permettant de modifier la base de données. Pour commencer, créez une nouvelle application au sein de votre projet :

```
symfony init-app adminapp
```

Il vous faudra ensuite créer, pour chaque classe Propel, le module d'administration correspondant (avec une syntaxe similaire à la génération d'un module CRUD).


```
symfony propel-init-admin adminapp utilisateur Utilisateur
```





Cette syntaxe vous permettra de créer ces modules avec la configuration par défaut. Ils sont entièrement personnalisables. La documentation officielle y consacre une section entière ( [http://www.symfony-project.com/book/trunk/14- Generators#Administration](http://www.symfony-project.com/book/trunk/14-Generators#Administration)).

IX - Conclusion

Vous êtes arrivés au bout de ce tutorial, félicitations ! Il ne vous reste plus qu'à essayer par vous-même:)

Pour aller plus loin

Tout d'abord, la  **documentation** sur symfony est riche, voici les principaux points de repère :

- « The definitive guide to symfony », c'est le livre officiel sur ce framework. Vous pouvez l'acheter, ou le consulter gratuitement en ligne  **en anglais** ou  **en français**
-  **La documentation de l'API**, elle n'est pas facile à comprendre, mais elle permet de se plonger au coeur du framework.
-  **Le projet Askeet** : l'idée est de créer un projet avec symfony en 24 jours (à 1 heure par jour). C'est un bon moyen de continuer l'apprentissage après ce tutorial

Si vous avez des questions, la  **communauté symfony** vous aidera sûrement. Et n'oubliez pas le  **forum de developpez sur le sujet !**