Créer rapidement un projet php avec Symfony 2

Pour cet article nous partons du principe que vous avez un environnement Apache, PHP, Mysql fonctionnel. Celui-ci a donc pour but d'installer Symfony, de découvrir ce qu'il contient et de créer notre premier Bundle. L'idée est de créer un petit site collaboratif, dans lequel les utilisateurs pourront ajouter des recettes de cuisine et mettre à disposition les sources du projet sous licence libre sur Github.

Installer symfony

Remarque : l'ensemble des commandes données sont faites pour être exécutées en environnement Unix-Like (Linux ou OS X), pour les utilisateurs de Windows, à vous d'adapter les commandes. Nous travaillons pour l'exemple dans le répertoire /home/programmez-symfony/symfony , pensez à adapter ce chemin. Pour installer Symfony, nous avons choisi d'utiliser la méthode préconisée à savoir utiliser Composer. Nous commençons donc par installer Composer :

```
$ curl -s https://getcomposer.org/installer | php
```

Composer est un fichier exécutable, nous l'utilisons pour télécharger la version standard de Symfony 2

```
$ php composer.phar create-project symfony/framework-standard
-edition /home/programmez-symfony 2.4.4
```

Pour que notre site réponde sur l'adresse : http://www.programmez-symfony.net nous devons créer et renseigner le domaine dans notre fichier /etc/hosts. Ouvrez le fichier et remplacer la ligne :

```
127.0.0.1 localhost
```

par

```
127.0.0.1 localhost www.programmez-symfony.net
```

Puis créer un fichier VirtualHost dans /etc/apache2/sites-available/programmez-symfony.net

```
<VirtualHost *:80>
  ServerName www.programmez-symfony.net
  ServerAlias programmez-symfony.net
  DocumentRoot /home/programmez-symfony/symfony/web/
  DocumentIndex app.php
</VirtualHost>
```

Nous activons le site avec la commande

```
$ a2ensite programmez-symfony.net
```

et redémarrons apache

```
$ /etc/init.d/apache2 restart
```

Nous avons donc un Symfony opérationnel, qui appelle bien la page de base de Symfony qui est app.php via l'url http://www.programmez-symfony.net
Nous allons vérifier que notre serveur est bien configuré en allant sur la page http://www.programmez-symfony.net/config.php
Vous arrivez sur une page de ce type **Fig.1.**
Vous voyez alors deux types de message :

- Major problems : à corriger impérativement, comme par exemple des problèmes de droits sur les répertoires
- Recommandations : des recommandations pour optimiser le fonctionnement de Symfony, à suivre si possible, mais non bloquant.

Une fois nos problèmes résolus (ici des droits d'écriture à donné à Apache et une timezone à définir), nous pouvons maintenant démarrer notre projet, si nous allons à la page de débug : http://www.programmez-symfony.net/app_dev.php, nous obtenons la page suivante : **Fig.2.**
La page par défaut (app.php) affiche une erreur 404, c'est normal, nous n'avons pas encore dit à Symfony ce qu'il devait afficher.

Fig.1

Fig.2

Page d'accueil de Symfony

Découverte du contenu de symfony

L'arborescence de Symfony est composée ainsi :

- app : contient l'intégralité de la configuration de notre site (connexion à la BDD, définition des routes, sécurité, etc.),
- web : la partie publique de notre site, avec par exemple les images et feuilles de styles qui doivent pouvoir être chargées,
- bin : les fichiers nécessaires à l'installation des vendors,
- src : contient tous les fichiers sources, c'est principalement dans ce dossier que nous allons travailler, les sous-dossiers :
  - Application (il peut y avoir plusieurs sous-dossiers)
    - Bundle (un bundle est une fonctionnalité de notre site comme les recettes ou les utilisateurs)
      - Entity : modèle de données
      - Controller : contrôleur contenant notre code métier
      - Repository : Méthodes spécifiques à l'application pour gérer les donnnées
      - Ressources
        - View : les fichiers d'affichage de notre bundle
        - Config : la configuration des routes notamment
      - Tests : pour placer les tests unitaires et fonctionnels de notre bundle
- vendor : les librairies utiles pour notre projet spécifiques à Symfony (Doctrine, Twig, etc).

Création de notre premier Bundle

Pour créer un bundle nous pouvons le faire à la main en créant les fichiers et en les déclarant à Symfony, mais le plus simple et le plus rapide est d'utiliser le générateur de Bundle pour cela nous utilisons la commande :

```
$ php app/console generate:bundle -namespace="ProgrammezSymfony
/RecetteBundle" --bundle-name="ProgrammezSymfonyRecetteBundle"
 --dir="src" --format="annotation"  --structure
```

On demande à Symfony de générer un bundle avec pour namespace ProgrammezSymfony/RecetteBundle et pour nom ProgrammezSymfonyRecetteBundle. Il crée ce bundle dans le répertoire « src » qui utilise le format « annotation » et l'option structure, permet de générer l'intégralité des répertoires mêmes ceux qui sont vides. A la validation de la commande Symfony nous demande de confirmer la création du Bundle et de confirmer deux mises à jour, nous avons choisi yes dans tous les cas. Nous avons donc tout le nécessaire pour démarrer le travail sur notre Bundle.

Remarque : le nom de votre bundle doit obligatoirement finir par Bundle pour que Symfony le détecte.

En détail Symfony a fait les actions suivantes :

- Déclarer le bundle à Symfony en éditant le fichier app/appKernel.php et en déclarant le nouveau Bundle dans la fonction registerBundles
- Indiquer à Symfony d'utiliser notre Bundle lorsque nous arrivons sur la page d'accueil du site en modifiant le app/config/routing.yml

Nous avons donc créé notre premier bundle que nous pouvons appeler en utilisant l'adresse (indiquée dans /src/ProgrammezSymfony/RecetteBundle/Controller/DefaultController.php) http://www.programmez-symfony.net/app_dev.php/hello/toto Vous aurez alors une page affichant simplement Hello Toto.

Création et affichage de notre modèle de données avec les entités de Symfony

Pour gérer les données avec une base de données, Symfony s'appuie sur l'ORM Doctrine. C'est lui qui va s'occuper de créer la base, les tables, et de gérer l'insertion, la sélection, la suppression de données. Symfony utilise les entités. Nous allons donc utiliser la génération des entités pour créer notre première entité :

```
$ php app/console doctrine:generate:entity -entity="Programmez
SymfonyRecetteBundle:Recette"   -fields="name:string(255)
tpspreparation:integer tpscuisson:integer ingredients:text
preparation:text"
```

Symfony vous pose une série de questions pour générer notre entité, nous avons choisi de laisser les options par défaut.
Nous demandons donc à doctrine de générer une entité (doctrine:generate:entity), le paramètre –entity permet de donner le nom du bundle, puis après les deux points le nom de l'entité, et l'option –fields permet de définir nos champs avec un nom de champ et un type.
Si nous regardons dans le fichier /src/ProgrammezSymfoy/RecetteBundle/Entity/Recette.php, nous voyons les champs déclarés sous forme d'annotation comme par exemple le champs tpscuisson :

```
/**
* @var integer
*
* @ORM\Column(name="tpscuisson", type="integer")
*/
private $tpscuisson ;
```

Symfony a donc généré la déclaration des variables pour nous ainsi que les getters et setters.
Pour générer notre base et les tables nous utilisons deux commandes :

```
$ php app/console doctrine:database:create
$ php app/console doctrine:schema:create
```

En cas de modification des champs ou d'ajout d'autres entités, vous pouvez demander à doctrine la mise à jour de la BDD avec la commande suivante :

```
$ php app/console doctrine:schema:update -force
```

Pour continuer, nous renseignons manuellement dans la base quelque données, nous aurions pu aussi utiliser Fixtures, pour cela, je vous laisse aller voir la doc officielle :
http://symfony.com/fr/doc/current/bundles/DoctrineFixturesBundle/index.html

Affichage de nos recettes

Pour l'affichage de nos pages, Symfony utilise le moteur de template Twig. Nous allons donc commencer par regarder comment est affichée notre page Hello Toto. Le fichier appelé est dans /src/ProgrammezSymfony/RecetteBundle/Resources/views/Default/index.html.twig
Ce fichier contient simplement :

```
Hello {{name}}
```

Nous allons ajouter un peu de HTML autour pour mettre en forme notre affichage et rendre notre fichier index générique, pour que l'intégralité des autres affichages hérite de l'index.
Puis on crée le fichier qui contiendra la liste de nos recettes /src/ProgrammezSymfony/RecetteBundle/Resources/views/Default/recettes.html.twig qui contient0 le code suivant :

```
{% extends 'ProgrammezSymfonyRecetteBundle:Default:index.html.twig' %}

{% block contenu %}
 {%for recette in Recettes %}
 <div class="fiche-recette">
   <div class="nom"> {{recette.name}}</div>
   <div class="ingredients"> {{recette.ingredients}} </div>
   <div class="preparation"> {{recette.preparation}} </div>
 </div>
 {% else %}<div class="alert">Désolé aucune recette pour
 l'instant</div>
 {% endfor %}
{% endblock %}
```

Nous allons ensuite dire au contrôleur d'utiliser notre template index dans /src/ProgrammezSymfony/RecetteBundle/Controller/DefaultController.php nous remplaçons la ligne

```
@Route('/hello/{name}')
```

par

```
@Route('/')
```

Puis dans la fonction index située dans le même fichier, nous supprimons la variable $name qui n'est plus utile. La fonction devient alors :

```
public function indexAction() {
 return array() ;
}
```

Lorsque nous arrivons sur notre site nous avons alors une page vide qui contient pourtant bien la structure HTML que nous avons créée.
Voyons maintenant comment récupérer les informations depuis la base de données pour permettre leur affichage. Nous allons utiliser les fonctions

fournies par Doctrine avec la méthode findAll() ;.
Pour cela nous modifions notre fonction indexAction comme suit :

```
public function indexAction() {
  $repository = $this->getDoctrine()->getRepository('Programmez
SymfonyRecetteBundle:Recette');
    $variables['Recettes'] = $repository->findAll();

    return $this->render('ProgrammezSymfonyRecetteBundle:
Default:recettes.html.twig', $variables);
}
```

Fig.3.

Nous avons donc vu l'affichage de nos recettes, mais il n'est pas très pratique de passer directement par des requêtes SQL ou phpMyAdmin pour insérer des données. Nous allons donc voir comment gérer des formulaires d'ajout sur notre site.

Création d'un formulaire d'ajout de recette

Afin de nous faciliter la vie, Doctrine a (encore) prévu un générateur de formulaire. Pour cela nous allons utiliser la commande :

```
$php app/console generate:doctrine:form ProgrammezSymfony
RecetteBundle:Recette
```

Cette commande génère un formulaire à partir des informations de l'entité, n'hésitez pas à aller voir le code généré dans \src\ProgrammezSymfony\RecetteBundle\Form\RecetteType.php
Dans ce code vous trouverez différentes fonctions :

- buildForm, permet de déclarer les champs du formulaire
- setDefaultOptions : définit les options par défaut des champs. Par défaut le générateur reproduira les valeurs par défaut de l'entité
- getName : définit le nom du formulaire.

Ce code s'il n'est pas parfait, permet d'avoir un formulaire fonctionnel.
Nous allons donner le chemin de notre formulaire à Symfony, pour cela nous avons rajouté les lignes suivantes au routing.yml :

```
admin_recette_add:
  path:      /ajouter_recette
  defaults:  { _controller: ProgrammezSymfonyRecetteBundle:
Default:ajouter }
```

Puis nous allons définir dans notre contrôleur (DefaultController.php) l'action ajouter appelé par notre routeur :

```
    public function ajouterAction()
{
  $recette = new Recette;
  $form = $this->createForm(new RecetteType, $recette);
  $request = $this->get('request');
  if ($request->getMethod() == 'POST') {
    $form->bind($request);
    if ($form->isValid()) {
      $em = $this->getDoctrine()->getManager();
      $em->persist($recette);
      $em->flush();
       return new Response('Recette ajoutée');
    }
  }
  return $this->render('ProgrammezSymfonyRecetteBundle:Default:
ajout-recette.html.twig', array(
    'form' => $form->createView(),
  ));
}
```

Sans oublier dans ce même fichier de lui donner les classes à utiliser avec (en haut du fichier avec les autres appels) :

```
use Symfony\Component\HttpFoundation\Response;
use ProgrammezSymfony\RecetteBundle\Entity\Recette;
use ProgrammezSymfony\RecetteBundle\Form\RecetteType;
```

Il nous reste que l'affichage à gérer.

Affichage de notre formulaire

Une fois le formulaire généré, il nous reste à créer la vue associée. Pour cela nous créons un fichier ajout-recette.html.twig dans le répertoire /src/ProgrammezSymfony/RecetteBundle/Resources/views/Default/.
Ce fichier contient le code suivant :

```
{% extends 'ProgrammezSymfonyRecetteBundle:Default:index.
html.twig' %}
{% block title %}Administration | Ajout d'une recette{% endblock %}

{% block contenu %}
<h1>Ajout d'une nouvelle recette</h1>
<form action="{{ path('admin_recette_add') }}" method="post"
 {{ form_enctype(form) }}>
    {{ form_widget(form) }}
    <input type="submit" />
</form>
{% endblock %}
```

Lorsque nous affichons notre formulaire, voici la page que nous obtenons : Fig.4.

Cet article avait pour but de vous donner de quoi démarrer rapidement un projet Symfony. Il ne vous reste plus qu'à coder, encore et encore avec Symfony pour connaître ce puissant framework.

Vanessa **Kovalsky David**
Lead développeuse Drupal / PHP chez Webnet - http://vanessakovalsky.net
Auteur du livre 'Drupal 7 en mode avancé' aux éditions Eyrolles
Membre actif de DrupalFr, et de la promotion du logiciel libre avec l'ALDIL
Lead dev sur le projet Esecouristes : https://github.com/vanessakovalsky/esecouristes

Fig.4

Fig.3

Crepes
Oeufs Farine Sucre Lait Pincée de sel
Mélanger la farine avec les oeufs, le sucre et le sel. Ajouter le lait en remuant petit à petit. Votre pate à crêpes est prête!

Affichage de notre première recette