Sommaire

Le sommaire possède des liens vers les parties concerné

Routing

1) Fichier routing

Il existe plusieurs fichier routing, le fichier routing du projet et celui du bundle ou l'on travail.
Celui du projet se trouve sur :

Symfony/app/config/routing.yml

Ce fichier principal permet de faire la liaison avec les routing du bundle qui se trouve sur :

Symfony/src/ARYGA/ARYGAPlatformBundle/Ressources/config/routing.yml

En cas de création d'un nouveau bundle, ne pas oublier d'indiquer son chemin dans le routing principal.

2) Norme routing

IMPORTANT : ne pas faire de tabulation sur un fichier YML.
Espacer les configurations (path, default,…) de 4 espaces

Le routing principale est sous la forme :

«Nom de bundle»:
resource: @ARYGAPlatformBundle/Resources/config/routing.yml
(@ « Nom du bundle »/ et chemin vers routing)
prefix: / « chemin de base après « /app_dev.php/ » pour toutes les pages du bundle»

Le routing du bundle est sous la forme :

«Nom de bundle»_«Nom du Controller»_«Nom de la fonction (sans Action)»:
Path: « Url après le préfix donné dans le routing principal »
defaults: { _controller: « Nom bundle »:« nom du controller »: « Nom de la fonction (sans Action)»}

Si c'est une requête venant d'un formulaire :

« Idem »:
pattern: « même chose que Path »
defaults: { _controller: « Idem » }
requirements:
_method: POST

Doctrine

1) Créer et manipuler les entités

Attention : Les commande ce font dans le dossier du projet donc ne pas oublier le « cd Symfony ».

La création de d'une entité se fait en ligne de commande :
Taper : php app/console generate:doctrine:entity
Puis suivez le guide sur la console.

Pour ajouter l'entité à la base de données :
Taper : php app/console doctrine:schema:update --dump-sql
Pour voir le code SQL qui modifiera la base de données.

Taper : php app/console doctrine:schema:update --force
Pour exécuter le code SQL.

On peut aussi générer les getter et setter :

Taper : php app/console doctrine:generate:entities ARYGAPlatformBundle:«Nom de l'entité»

Important : après chaque modification du fichier de l'entité, ajouter l'entité a la base de données « cf : au-dessus ».

2) Relations entre les entités

Ajouter une propriété entité_id suivant la relation que l'on veut puis ajouter en commentaire au-dessus :

Pour une relation one to one :

```
/**
 * @ORM\OneToOne(targetEntity="ARYGA\ARYGAPlatformBundle\Entity\"nomEntité", cascade={"persist"})
 * @ORM\JoinColumn(nullable=false)
 */
```

Pour une relation many to one :

```
/**
  * @ORM\ManyToOne(targetEntity="ARYGA\ARYGAPlatformBundle\Entity\"nomEntité"")
```

```
 * @ORM\JoinColumn(nullable=false)
 */
```

Pour une relation many to many :

```
/**
 * @ORM\ManyToMany(targetEntity="ARYGA\ARYGAPlatformBundle\Entity\”nomEntité"", cascade={"persist"})
 */
```

Ne pas oublier de générer le getter et setter puis d’ajouter à la base de données.

Controller

1) utilisation des entitées

Important : Le code qui suit concernant doctrine, utilise le bundle dataFixture de doctrine.

ne pas oublier d’ajouter :

```
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpFoundation\jsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Doctrine\Common\DataFixtures\AbstractFixture; //Bundle qui simplifie le code de Doctrine
use Doctrine\Common\Persistence\ObjectManager;
```

Si on veut ajouter une nouvelle donnée à une entité, ajouter :

```
use ARYGA\ARYGAPlatformBundle\Entity\”nom de l’entité”;
```

Avant d’utiliser une entité dans une fonction créer une variable qui contiendra le manager :

```
$em = $this->getDoctrine()->getManager();
```

Pour accéder à une entité faite :

```
$em->getRepository('ARYGAPlatformBundle:« nom de l’entité »")
```

Pour accéder à une ou plusieurs données :

Ajouter `->find(« variable »)` ,

```
->findBy(array(‘nomPropriete’ => donnéeRecherché)),
->findOneBy(idem), findAll()
```

N’utiliser pas findBy() si vous ne voulez qu’une seul donnée !

Pour ajouter une donnée :

Créer un objet de l’entité `$entite = new Entite();` ,Ne pas oublier le use.

Puis ajouter la valeur avec le setter : `$entite->setPropriete($valeur);`

Et enfin faire : $em->persist($entite);
$em->flush();

On peut faire plusieurs persist et flush tous en même temps.

Pour la modification ne pas créer d'entité mais récupérer l'enregistrement via un findOneBy ou find puis faire la même chose qu'au-dessus, un setPropriete($valeur) sur l'enregistrement et persist puis flush.

2) Utiliser la fonction d'un autre controller

Pour cela il faut passer le controller en service.
Ouvrir le fichier symfony/app/config/services.yml et ajouter dans la partie services :

```
aryga.controlleraccess.«entite»:
    class: ARYGA\ARYGAPlatformBundle\Controller\«entite»Controller
```

Pour appeler la fonction du controller passer en service :

```
$this->get('aryga.controlleraccess.«entite»)->nomDeLaFonction();
```

3) Retourner une ou plusieurs variables

Un controller retourne obligatoirement quelque chose sa peut être un fichier twig :

```
return $this->render('ARYGAPlatformBundle:cheminDansViews:nomPage.html.twig');
```

ou return new Response(); qui recharge la page où est appeler la fonction.

On peut aussi envoyer des variables à une page twig :

```
return $this->render('ARYGAPlatformBundle:cheminDansViews:nomPage.html.twig',
array('nomUtiliseDansTwig'=>$valeur, '2emeValeur'=> $valeur2));
```

TWIG

1) Utilisation de twig

Attention : on ne peut pas mettre autre chose que du twig dans les balises twig.(ex :{% var variableJs %} est impossible)

Pour utiliser une variable : {{nomDonnéDansleController}}.
On peut l'utiliser comme valeur : var uneVar = {{uneVarTwig}}.

Plusieurs configurations sont disponibles avec {{nomVar|nomConfig}} ex : {{nomVar|length}}.

Pour utiliser des boucles : {% for var in allVar %} et finir par {% endfor %}
{% if condition %} le code {% endif %}

Plus d'info voir doc : http://twig.sensiolabs.org/documentation

2) Routing vers le controller

Pour appeler une fonction du controller utiliser :

{{ path('nom du routing qui amène à la fonction voulu')}}

On peut l'utiliser lors d'un submit du formulaire ou dans l'url d'une requête Ajax.

Bundle

1) Créer un Bundle

Pour Créer un bundle comme ARYGAPlatformBundle il suffit d'utiliser l'invité de commande.

Ce placer sur le dossier du projet (ici Symfony) puis taper :
Php app/console generate:bundle

Mettre le nom du bundle et faite Entrez pour tout le reste.
Puis la structure sera générée.

N'oublier pas d'ajouter le chemin du routing.yml dans le routing principal (voir Routing page 2).

2) Installer un Bundle

Pour installer un bundle, il faut utiliser composer.

Choisir son bundle. Pour cela plusieurs sites sont disponible (ex : www.packagist.org).

Ouvrir le fichier Symfony/composer.json

Dans la partie require déclarer la dépendance puis sa version. Exemple avec assetic :

```
"symfony/assetic-bundle": "~2.3"
```

Ici composer téléchargera la version la plus à jour, à partir de la version 2.3 et en s'arrêtant avant la version 3.0.

Ne pas oublier la virgule entre les bundle.

Puis faite en ligne de commande un « php composer.phar update » pour mettre à jour les dépendances.

Et pour finir activer le bundle dans le Kernel, Pour cela ouvrir le fichier Symfony/app/AppKernel.php
et déclarer dans array() le bundle. Exemple avec assetic :

```
new Symfony\Bundle\AsseticBundle\AsseticBundle(),
```

Assetic

Assetic permet de réunir tous les fichiers javascript ou css en seul. Un seul fichier sera donc généré lors du chargement d'une page, pour améliorer sa vitesse de chargement.

Tous les fichiers sont générés en Dev pour faciliter le débogage.

1) Configuration

La configuration d'assetic se fait dans le fichier Symfony/app/config/config.yml.

Dans la partie assetic plusieurs config sont disponible comme le debug, write_to qui est le chemin où va être généré le fichier, ou le filters où l'on peut utiliser des filtres comme cssrewrite ou un compresseur javascript ou css.

2) Utilisation

Pour utiliser assetic avec twig il faut utiliser un block javascript ou stylesheet pour cela faite, dans la vue Twig.

Pour Javascript :

```
{% javascripts    'https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js'
                  'un autre fichier js '
                  output='js/main.js' // donne un nom au fichier généré
%}
     <script src="{{ asset_url }}"></script>
{% endjavascripts %}
```

Pour le css :

```
{% stylesheets filter='cssrewrite' //Le cssrewrite permet de réécrire les chemins dans le css si le fichier est déplacer
        'http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css'
        'un autre css'
        output='css/main.css'
%}
        <link rel="stylesheet" href="{{ asset_url }}" type="text/css" />
{% endstylesheets %}
```

Généré ensuite les fichiers avec un dump en ligne de commande.

Php app/console assetic:dump en dev ou ajouter --env=prod en prod.

Ou faite un php app/console assetic:watch pour les générés à chaque modifications

Par la suite vider le cache de symfony en Dev ou en Prod selon l'utilisation.

Php app/console cache:clear pour le dev,

Php app/console cache:clear --env=prod pour la prod.

En cas de problème, essayer de supprimer le cache avant et après le dump.

Utilisateurs

La configuration du bundle fosUserBundle se fait dans le fichier Symfony/app/config/config.yml.

1) Ajouter un utilisateur

Il y deux solutions pour ajouter un utilisateur :

Utiliser le formulaire que j'ai créé pour l'administrateur, si le formulaire bug ou que vous avez supprimé le compte administrateur, faite le en ligne de commande :

Php app/console fos:user:create nomUtilisateur email motDePasse

Pour lui donner un rôle :

Php app/console fos:user:promote nomUtilisateur nomDuRole

2) Rôle de l'utilisateur

Pour ajouter un Rôle, ouvrir le fichier symfony/app/config/security.yml.
Puis dans role_hierarchy : ajouter le nom du rôle et si c'est un regroupement de rôle, ajouter les rôles qu'il constitue, exemple :

ROLE_ECRITURE :
ROLE_LECTURE :
// le role aryga possède les rôles écriture et lecture.
ROLE_ARYGA : [ROLE_ ECRITURE, ROLE_LECTURE]

Les utilisateurs sans rôle auront de base le rôle ROLE_USER.

Pour mettre à jour le formulaire de création d'utilisateurs ouvrir
symfony/vendor/friendsofsymfony/user-bundle/Form/Type/RegistrationFormType.php

Puis dans la fonction buildForm ajouter dans choices le nouveau rôle.

3) Accès aux pages

Pour gérer les accès des rôles, ouvrir le fichier : symfony/app/config/security.yml.

Et dans la partie access_control :
Ajouter le path et le rôle qui aura accès a ce chemin, exemple :

- { path: ^/platform/Groupe/add, roles: ROLE_ECRITURE }
  Ici, la page ayant l'url « /platform/groupe/add » est réservé à ROLE_ECRITURE.
  Comme ROLE_ECRITURE fait partie de ROLE_ARYGA, aryga aura aussi accès à la page.

- { path: ^/platform/admin/*, roles: ROLE_ADMIN }
  Ici toute les pages commençant par « ^/platform/admin/ » seront réservé à l'admin.