Obtao

Créer une API Rest dans une application Symfony

Posted on **December 10, 2013** by **francois**

Read the English version

Comme nous avions besoin d'une connexion entre une application web Symfony et une appli Android, nous avons dû apprendre et comprendre comment créer une API Rest de manière simple et sécurisée, en nous basant sur nos entités existantes.

Nous avons choisi `WSSE` pour l'accès sécurisé, `FOSRestBundle` pour la restitution de données et `JMSSerializerBundle` pour la sérialisation.

Nous avions également besoin de séparer les logs pour les erreurs d'authentification WSSE. Lisez notre article pour savoir comment créer un fichier de logs séparé avec Monolog (en anglais).

Librairies

Bundles

- FOSRestBundle
- JMSSerializerBundle

Articles

- Blog de Nicholas Paul Masters sur FOSRestBundle (en anglais)
- Blog d'Aurélien sur l'utilisation des events de jmsserializer :cet article vous aidera a appréhender les listeners pour modifier les objets sérialisés

Première étape : Installer FOSRestBundle et JMSSerializerBundle

Vous aurez besoin de ces deux bundles, donc ajoutez-les à votre fichier composer.json :

```
// composer.json

// ...
"friendsofsymfony/rest-bundle": "dev-master",
"jms/serializer-bundle": "dev-master"
```

Lancez la commande `php composer.phar update` pour mettre à jour vos vendors puis enregistrez les bundles dans votre fichier `AppKernel.php` :

```
//app/AppKernel.php

public function registerBundles()
    {
        $bundles = array(
            // ...
            new FOS\RestBundle\FOSRestBundle(),
            new JMS\SerializerBundle\JMSSerializerBundle(),
```

Enfin, configurez le FOSRestBundle:

```
#app/config/config.yml
fos_rest:
    param_fetcher_listener: true
    body_listener: true
    format_listener: true
    view:
        view_response_listener: 'force'
        formats:
            xml: true
            json : true
        templating_formats:
            html: true
        force_redirects:
            html: true
        failed_validation: HTTP_BAD_REQUEST
        default_engine: twig
    routing_loader:
        default_format: json
```

Notre but ici n'est pas de détailler comment choisir la meilleure configuration. Lisez la documentation du FOSRestBundle pour plus d'informations.

Vos bundles sont maintenant prêts à être utilisés.

Deuxième étape : Créer vos fichiers de routing Rest et appeler votre API.

Dans le but de gérer les requêtes Rest, vous devrez définir vos routes. FOSRestBundle vous permet d'utiliser un nouveau type de route : `"rest"`.

Les routes seront générées automatiquement. Suivez les étapes suivantes pour savoir comment.

Importez le fichier routing_rest.yml

Dans votre `app/routing.yml`, déclarez un fichier `routing_rest.yml` qui est chargé avec le type “rest”. Ce type est nécessaire pour la génération automatique des routes.

Nous spécifions également à Symfony que chaque route de `routing_rest.yml` a besoin du préfixe `/api`.

```
#app/routing_yml

# ...

#REST
rest :
  type : rest
  resource : "routing_rest.yml"
  prefix : /api
```

Dans `app/routing_rest.yml`, vous devez déclarer votre (ou vos) fichiers de routing Rest pour chaque bundle :

```
#app/routing_rest.yml
Rest_User :
  type : rest
  resource: "@UserBundle/Resources/config/routing_rest.yml"
```

Dans le fichier de routing Rest du UserBundle, vous devez définir un Contrôleur Rest. La configuration spécifie également que chaque route sera nommée avec le préfixe `api_` pour éviter les conflits :

```
#src/Obtao/UserBundle/Resources/config/routing_rest.yml
users :
  type: rest
  resource:     "UserBundle:UserRest"
  name_prefix:  api_
```

Créez votre première fonction de l'API Rest : getUser

```
namespace Obtao\UserBundle\Controller;

use FOS\RestBundle\Controller\Annotations\View;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\Security\Core\Exception\AccessDeniedException;

class UserRestController extends Controller
{
  public function getUserAction($username){
    $user = $this->getDoctrine()->getRepository('UserBundle:User')->fi
ndOneByUsername($username);
    if(!is_object($user)){
      throw $this->createNotFoundException();
    }
    return $user;
  }
}
```

Vos routes ont été générées, lancez la commande `router:debug` pour vérifier si tout est ok :

```
 $ app/console router:debug | grep api_

fr__RG__api_get_user                        GET      ANY    ANY  /ap
i/users/{username}.{_format}
```

Mais le sérialiser ne sait pas encore comment sérialiser un utilisateur. C'est ce que nous allons voir dans la section suivante.

Configurez la sérialisation de l'entité

Pour le moment, nous avons juste besoin d'envoyer les propriétés "id", "firstname","name" et "usedname". Nous pouvons utiliser les annotation du `JMSSerializerBundle` directement dans nos entités :

```
namespace Obtao\UserBundle\Entity;

use FOS\UserBundle\Entity\User as BaseUser;
use Doctrine\ORM\Mapping as ORM;
use JMS\Serializer\Annotation\ExclusionPolicy;
use JMS\Serializer\Annotation\Expose;
use JMS\Serializer\Annotation\Groups;
use JMS\Serializer\Annotation\VirtualProperty;

/**
 * @ORM\Entity
 * @ORM\Entity(repositoryClass="Obtao\UserBundle\Repository\UserReposi
tory")
 *
 * @ExclusionPolicy("all")
 */
class User extends BaseUser {

    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     * @Expose
     */
    protected $id;

    /**
     * @ORM\Column(type="string", length=150, nullable=true)
     * @Expose
     */
    protected $name;

    /**
     * @ORM\Column(type="string", length=150, nullable=true)
     * @Expose
     */
    protected $firstname;

    // ...

    /**
     * Get the formatted name to display (NAME Firstname or username)
     *
     * @param $separator: the separator between name and firstname (de
fault: ' ')
     * @return String
     * @VirtualProperty
     */
```

```
    public function getUsedName($separator = ' '){
        if($this->getName()!=null && $this->getFirstName()!=null){
            return ucfirst(strtolower($this->getFirstName())).$separat
or.strtoupper($this->getName());
        }
        else{
            return $this->getUsername();
        }
    }
```

Jetons un oeil aà l'exemple précédent plus en détails :

`@ExclusionPolicy("all")` : Chaque propriété de votre entité sera ignoré lors de la sérialisation.

`@Expose` : Cette **propriété** sera sérialisée.

`@VirtualProperty` : Cette **méthode** sera appelée et sérialisée comme propriété virtuelle (used_name dans notre exemple).

Nous y voilà. Appelez `api/users/userName` et vérifiez la réponse.

Pour ceux qui veulent créer une api et étendre des entités de librairies tierces (comme nous le faisons avec le FOSUserBundle), lisez attentivement la section suivante.

Troisième étape (facultative) : Configurer la sérialisation sur les entités de librairies tierces

Vous pouvez ajouter des propriétés privées/publiques sur vos entités, mais vous ne pouvez pas modifier les entités des librairies tierces. Heureusement, il est possible de déclarer des paramètres afin de cacher/afficher les propriétés, comme nous le faisons avec les annotations.

Déclarer une nouvelle méta données pour le JMSSerializerBundle

Ajoutez ceci à votre fichier `config.yml`:

```
//app/config.yml
jms_serializer:
    metadata:
        directories:
            FOSUB:
                namespace_prefix: "FOS\\UserBundle"
                path: "%kernel.root_dir%/serializer/FOSUserBundle"
```

Cela indique à JMS que les méta données des entités du FOSUserBundle seront également déclarées dans le dossier `app/serializer/FOSUserBundle`.

Ajouter les paramètres pour la sérialisation de l'utilisateur

Dans cet exemple, nous voulonssurcharger les méta données de l'entité User. Nous créons donc un nouveau fichier `Model.User.yml` dans le dossier `app/serializer/FOSUserBundle/`

```
//app/serializer/FOSUserBundle/Model.User.yml
FOS\UserBundle\Model\User:
    exclusion_policy: ALL
    properties :
      username :
        expose : true
```

Chaque propriété de l'entité User sera exclue lors de la sérialisation et seule la propriété "username" sera exposée.

Dernière étape : Créer un modèle d'entité pour la sérialisation (afficher/cacher des attributs

selon le contexte)

Dans certains cas, vous aurez besoin de 2 sérialisations différentes pour la même entité. Vous avez donc besoin de la notion de “Groups”.

Voici un exemple :
Vous voulez fournir un service public via votre API qui contient des informations sur l’utilisateur (username et description), mais vous voulez aussi fournir un service sécurisé (avec authentification) qui donne des informations sur le compte des utilisateurs (nom et prénom).
Dans cette partie, nous appelerons “contexte” le fait de sérialiser différemment notre objet User.

Suivez les étapes suivantes pour apprendre comment le faire en utilisant seulement les annotations.

Pour votre entité User, vous avez besoin de 2 méthodes différentes. Une méthode publique pour retrouver les informations publiques de l’utilisateur :

```
server.name/api/users/francois
```

et une autre pour retrouver les informations privées sur l’utilisateur authentifié :

```
server.name/api/me
```

Vous devrez créer ces 2 méthodes dans votre `UserRestController` (pas besoin de créer de route).

Pour la définition du contexte, vous utiliserez les annotations

fournies par FOSRest et basées sur la fonctionnalité “Groups” de JMSSerializer.

```
namespace Obtao\UserBundle\Controller;
use Obtao\ObtaoBundle\Controller\ObtaoController;

use Obtao\TripBundle\Entity\Trip;
use FOS\RestBundle\Controller\Annotations\View;
use Symfony\Component\Security\Core\Exception\AccessDeniedException;

class UserRestController extends ObtaoController
{
  /**
   *
   * @param type $username
   *
   * @View(serializerGroups={"Default","Details"})
   */
  public function getUserAction($username){
    $user = $this->getRepository('UserBundle:User')->findOneByUsernam
e($username);

    if(!is_object($user)){
        throw $this->createNotFoundException();
    }

    return $user;
  }

  /**
   *
   * @View(serializerGroups={"Default","Me","Details"})
   */
  public function getMeAction(){
    $this->forwardIfNotAuthenticated();
    return $this->getUser();
  }

  /**
   * Shortcut to throw a AccessDeniedException($message) if the user i
s not authenticated
   * @param String $message The message to display (default:'warn.use
r.notAuthenticated')
   */
  protected function forwardIfNotAuthenticated($message='warn.user.not
Authenticated'){
```

```
    if (!is_object($this->getUser()))
    {
        throw new AccessDeniedException($message);
    }
  }
}
```

Ce que nous faisons ici est de définir un contexte pour nos méthodes.

- Pour le service public(`getUserAction` – `api/users/{username}`), nous définissons 2 groupes comme contexte : `“Default”` et `“Details”`. Tous les champs de l’entité User marqués comme faisant partie du groupe “Default” (via l’annotation `@Expose`), ou du groupe “Details” (annotations `@Expose` + `@Groups({“Details”})`) seront sérialisés.
- Pour le service sécurisé (`getMe` – `api/me`), nous définissons 3 groupes comme contexte : `“Default”`, `“Details”` (comme pour le service public) mais nous ajoutons également `“Me”`. Tous les champs marqués par l’un de ces 3 groupes seront sérialisés

Le contrôleur est prêt, vous devez maintenant spécifier quels attributs doivent être sérialisés pour chaque contexte.

Dans votre entité, vous pouvez maintenant utiliser l’annotation “Groups” de JMSSerializer pour définir vos attributs :

```
    /**
     * @ORM\Column(type="string", length=150, nullable=true)
     * @Expose
     * @Groups({"Me"})
     */
    protected $name; // Le nom est visible pour le contexte "Me"

    /**
     * @ORM\Column(type="string", length=150, nullable=true)
     * @Expose
```

```
     */
    protected $avatar; // l'avatar est visible pour le contexte "Defau
lt"

    /**
    * @ORM\Column(type="text", nullable=true)
    * @Expose
    * @Groups({"Details"})
    */
    protected $description; // La description est visible pour le cont
exte "Details"
```

`$description` sera sérialisé pour les méthodes du contrôleur qui possèdent l'annotation `@View(serializerGroups={“Details”})`.
`$name` for `@View(serializerGroups={“Me”})`
`$avatar` pour `@View(serializerGroups={“Default”})` (puisqu'aucun groupe n'a été défini)

Configurer les groupes d'une entité d'une librairie tierce (FOSUserBundle)

Pour ceux qui étendent une entité d'une librairie tierce (dans notre cas nous étendons le User du FOSUserBundle), vous devrez également configurer les groupes sur les attributs hérités.

Modifiez simplement votre configuration de modèle de cette manière :

```
//app/serializer/FOSUserBundle/Model.User.yml
FOS\UserBundle\Model\User:
    exclusion_policy: ALL
    properties :
      salt :
        expose : true
        groups : [Details]
      email :
        expose  : true
        groups  : [Me]
      enabled :
```

```
        expose  : true
```

Votre API est maintenant prête à être utilisée.

Le résultat de toute cette configuration sera quelque chose comme :

Pour `http://www.obtao.com/api/users/francois`
-> `getUser($username)` dans UserRestController, contextes “Default” et “Details”) :

```
{"username":"francois", "salt":"mysalt", "enabled" : true, "avatar" :
"myAvatar.png", "description" : "This is my description"}
```

Pour `http://www.obtao.com/api/me` (authentifié en tant que “francois”)
-> `getMe()` dans UserRestController, contextes {“Default”,”Details” et “Me”}

```
{"username":"francois", "salt":"mysalt", "enabled" : true, "avatar" :
"myAvatar.png", "description" : "This is my description", "email" : "m
yemail@obtao.com", "name" : "MyName"}
```

Lisez notre post sur `WSSE` pour savoir comment protéger vos ressources avec un login/mot de passe de façon RESTful

Related Posts:

Create a Rest API in a Symfony application

Créer un type de champ DateTimePicker avec jQuery et

Symfony

Comment bien organiser vos traductions dans Symfony

Exporter des données dans un fichier csv avec Symfony

This entry was posted in **Symfony**, **Tutoriel** and tagged **Rest**, **Symfony**, **WSSE** by **francois**. Bookmark the **permalink [http://obtao.com/blog/2013/12/creer-une-api-rest-dans-une-application-symfony/]** .

18 THOUGHTS ON “CRÉER UNE API REST DANS UNE APPLICATION SYMFONY”

Aurel
on **March 18, 2014 at 8:57 am** said:

Super tuto.

Merci a vous

francois
on **March 18, 2014 at 1:06 pm** said:

Merci !

romain
on **April 18, 2014 at 9:00 pm** said:

Merci super tuto, ça va être pratique pour connecter l'appli Android =)

francois
on **April 19, 2014 at 11:38 am** said:

Bon courage!

Vous avez les 3 articles à votre disposition 😉

marwa
on **May 11, 2014 at 10:10 pm** said:

j'ai une réponse en json : {"hotels ":[{"file":null}]} au lieu d'avoir la liste de l'entité hôtel :'(vous pouvez me dire c'est quoi le problème ?

Gregquat
on **May 12, 2014 at 10:53 am** said:

Bonjour, dans le tuto, nous excluons par défaut toutes les propriétés lors de la sérialisation grâce à l'annotation `@ExclusionPolicy("all")`.
En conséquence, vous devez spécifiez quelles propriétés doivent être retournées lors de la sérialisation en utilisant l'annotation `@Expose` (comme nous le faisons pour les propriétés "name" et "firstname" par exemple).
Si ce n'est pas ça, contactez-nous par email via le site pour nous donner plus de code afin que l'on puisse identifier ce qui ne va pas.
Bon courage

marwa
on **May 12, 2014 at 11:45 pm** said:

Merciii pour votre réponse mais c la même chose 😟 jvais vous contacter par mail pour plus de détail

et merci d'avance 😀

marwa

on **May 13, 2014 at 4:51 pm** said:

Merciiiii ça marche très bien (y)

EDIT Gregquat : le problème était que le contrôleur doit retourner directement un objet ou une liste d'objets, et non un objet Response.

cyrine
on **May 14, 2014 at 9:12 pm** said:

bjr il m'affiche toujours cette erreur "Fatal error: Class 'FOS\RestBundle\FOSRestBundle' not found in C:\xampp\htdocs\Tunisie-business\app\AppKernel.php on line 21
Done." c'est quoi le problem exactement?

merci de me repondre

francois
on **May 15, 2014 at 8:51 am** said:

Bonjour,

Il semble que le bundle FOSRest ne soit pas correctement installé (et donc que la classe ne soit pas trouvée) :
- Vérifiez le composer.json (et la présence du `"friendsofsymfony/rest-bundle": "dev-master"`php composer.phar update et assurez vous que tout se termine sans erreur

François

cyrine
on **May 16, 2014 at 1:12 am** said:

c'est resolu !! j'ai installé git du site "msysgit.github.io" puis

j'ai ajouté le chemin C:\Program Files\Git\cmd dans la variable d'environnement et ca marche tres bien

merci a tous

Aurélien
on **May 21, 2014 at 12:41 pm** said:

Bonjour, merci pour ce super article qui reprend de bonnes bases et des bonnes pratiques pour la création d'une API sur une application Symfony2.
Ayant rencontrée une problématique il y a peu, liée à la manipulation des attributs d'une entité lors de la serialisation (l'idée en deux mots : transformer un File::$filename en File::$resourceUrl).
Je pense que la notion abordée dans cet article (les Event Subscriber avec JMSSerializerBundle) est pertinente dans la continuité du votre, n'hésitez pas à me contacter par email pour me demander le lien, je vous inviterai à le renseigner sur cette page si vous le voulez bien, bien sûr.
Bien à vous.

francois
on **May 22, 2014 at 12:30 pm** said:

Bonjour,
Merci pour le lien ! On devait s'occuper d'écrire un article la dessus.
Mais celui la est très bien !

http://aubm.net/blog/utiliser-les-event-subscriber-avec-fosrestbundle-et-jmsserializerbundle
Je l'ai rajouté dans la liste des articles.

Francois

Aurélien
on **May 22, 2014 at 12:44 pm** said:

Bonjour,

Merci beaucoup et avec plaisir

Aurélien

FEKI
on **May 26, 2014 at 10:40 am** said:

Bonjour,
Je vous remerci de ce super tuto. J'ai une petitte question: comment faire pour avoir plusieurs controlleurs qui gère le web service SVP ? J'ai tester de le faire mais en vain. Exemple: Deux controlleurs ArticleRestController et CommentaireRestController, ayant chacun une action : getArticleAction() et getCommentaireAction(). Puisque le bundle crée les route lui même les route, il écrase toujours la première (ws_get).

FEKI
on **May 26, 2014 at 10:53 am** said:

C'est bon, j'ai trouvé la solution. Il suffit de définir un name_prefix différent pour chacun d'eux.

cyrine
on **June 7, 2014 at 12:25 pm** said:

bjr j'ai tjr cette erreur [RuntimeException] You need to disable the view annotations in SensioFrameworkExtraBundle when using the FOSRestBundle View Response listener.

HELP PLZ

francois
on **June 10, 2014 at 9:47 am** said:

Bonjour,
Vous trouverez plus d'infos sur la doc de FOSRest :
https://github.com/FriendsOfSymfony/FOSRestBundle/blob/master/Resources/doc/3-listener-support.md#view-response-listener

sensio_framework_extra:
view: { annotations: false }
router: { annotations: true }

Cela devrait résoudre votre problème,
François