eBook Gratuit

APPRENEZ symfony2

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#symfony2

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: symfony2

It is an unofficial and free symfony2 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official symfony2.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec symfony2

Remarques

Cette section fournit une vue d'ensemble de ce qu'est Symfony2 et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tous les grands sujets dans Symfony2 et de les relier aux sujets connexes. La documentation de Symfony2 étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

La dernière version stable au moment de la rédaction est **Symfony 3.1,** qui sera maintenue jusqu'à la fin du mois de juillet 2017.

Symfony a **des** versions de **support à long terme** qui sont maintenues pour un total de 4 ans (3 ans pour les corrections de bogues, 1 année supplémentaire pour les corrections de bogues de sécurité)

Une **version mineure standard** est maintenue pendant huit mois pour les corrections de bogues et pendant quatorze mois pour les problèmes de sécurité.

Versions du support à long terme:

```
Symfony 2.3 – May 2016 end of support for bug fixes
              May 2017 end of support for security fixes(end of life)

Symfony 2.7 – May 2018 end of support for bug fixes
              May 2019 end of support for security fixes(end of life)

Symfony 2.8 – November 2018 end of support for bug fixes
              November 2019 end of support for security fixes(end of life)
```

À partir de la version 3.X, les versions mineures seront limitées à 5 et la dernière version mineure sera LTS.

Syfmony a un mode de maintenance double, libérant des versions mineures six mois une fois en mai et une fois en novembre. Les versions majeures sont publiées tous les deux ans, ce qui signifie qu'il faudra un an pour que la version précédente soit remplacée par la version précédente, donnant aux utilisateurs le choix entre les dernières fonctionnalités de la version standard ou une version LTS prise en charge.

Symfony maintient une compatibilité descendante stricte, tout ce qui casse BC est fait dans la prochaine version majeure. Une implémentation de fonctionnalité qui est une amélioration mais qui brise la BC est conservée avec l'ancienne implémentation qui sera dépréciée

En savoir plus sur les versions et le processus de développement en détail à partir de la

documentation officielle [ici] [1]

[1]: http: //symfony.com/doc/current/contributing/community/releases.html | Version | Date de sortie | | ------- | ---------- | | 2.3.0 | 2013-06-03 | | 2.7.0 | 2015-05-30 | | 2.8.0 | 2015-11-30 |

Examples

Installation ou configuration

Symfony Framework - construit avec des composants symfony, est l'un des principaux frameworks PHP utilisé pour créer des sites Web et des applications Web robustes.

Symfony peut être installé rapidement selon deux méthodes recommandées.

1. La documentation officielle recommande d'installer le framework via le programme d' ***installation de Symfony,*** une application php minuscule installée une fois sur le système local, qui aide à télécharger le framework et à configurer la structure. Le programme d'installation de Symfony nécessite PHP 5.4 ou supérieur. Pour installer sur la version php héritée, utilisez Composer.
2. A travers le gestionnaire de dépendances PHP ***Composer***

Installation via le programme d'installation de Symfony

Sous Linux / Mac OS X, exécutez les commandes suivantes:

```
$ sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

Sous Windows, accédez au répertoire du projet et exécutez la commande suivante:

```
php -r "file_put_contents('symfony', file_get_contents('https://symfony.com/installer'));"
```

Le projet symfony peut être créé en exécutant `symfony new my_project [2.8]` sous Linux / Mac OS X

Sur Windows `php symfony new my_project [2.8]`

ou bien `symfony new my_project lts` utilisera la dernière version de support à long terme de Symfony.

Installation via Composer

- Télécharger le compositeur

- Utilisez la commande `create-project` de Composer pour télécharger Symfony

```
composer create-project symfony/framework-standard-edition my_project_name ["2.8.*"]
```

Excellente documentation officielle détaillée ici

Exécution de l'application Symfony

Pour démarrer le serveur web interne symfony (disponible depuis PHP 5.4), allez dans le répertoire du projet et exécutez cette commande:

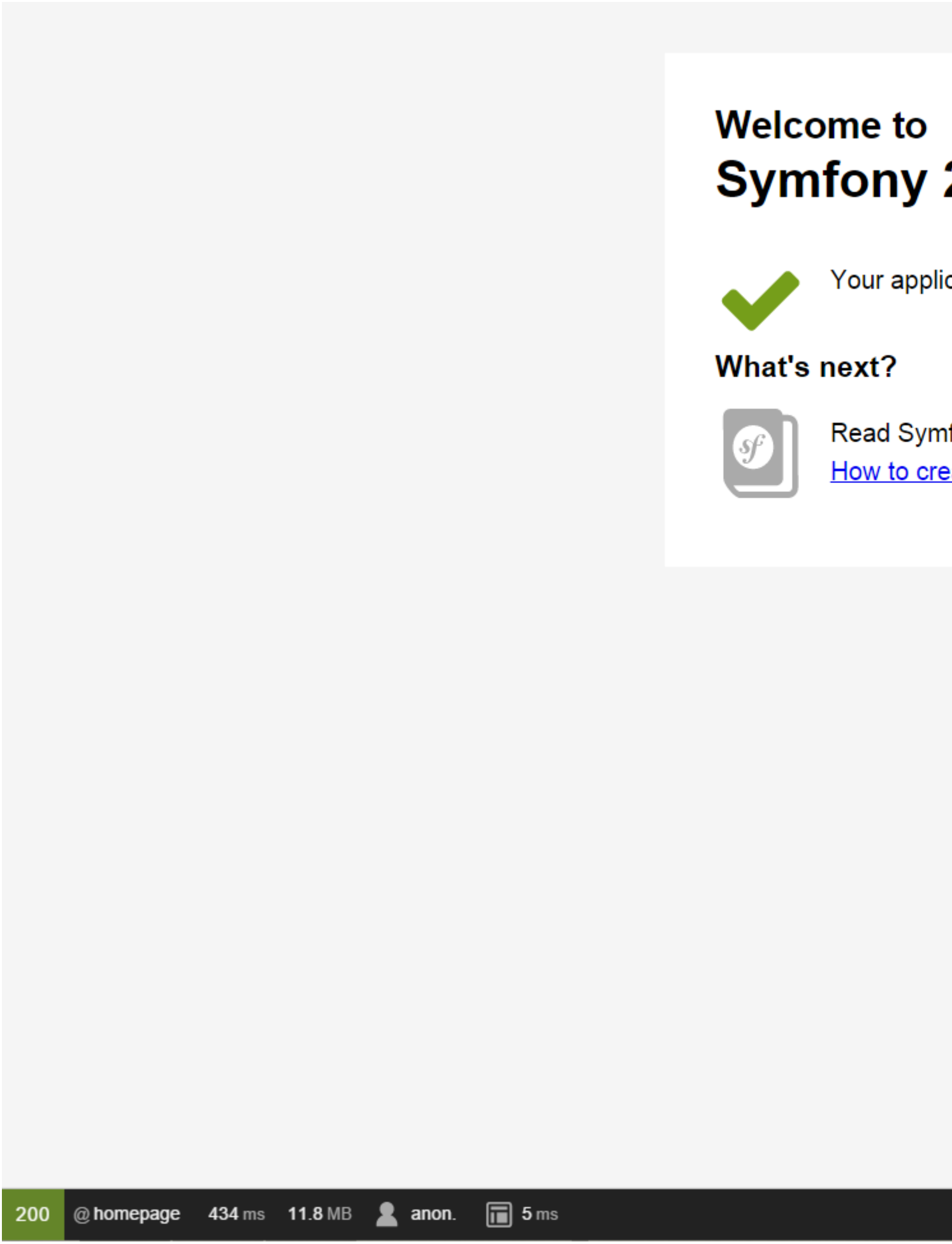
pour symfony <= 2.8

```
php app/console server:start
```

et pour symfony> = 3.0

```
php bin/console server:start
```

Cela démarre le serveur Web sur `localhost:8000` en arrière-plan qui sert votre application Symfony. Ensuite, ouvrez votre navigateur et accédez à l'URL `http://localhost:8000/` pour voir la page d'accueil de Symfony:


Welcome to
Symfony 2
Your applic
What's next?
Read Symf
How to cre
200
@ homepage
434 ms
11.8 MB
anon.
5 ms

Exemple le plus simple dans Symfony

1. Installez correctement symfony comme indiqué ci-dessus.
2. Démarrez le serveur symfony si vous n'êtes pas installé dans le répertoire www.
3. Assurez-vous que http: // localhost: 8000 fonctionne si le serveur symfony est utilisé.
4. Maintenant, il est prêt à jouer avec l'exemple le plus simple.
5. Ajoutez le code suivant dans un nouveau fichier **/src/AppBundle/Controller/MyController.php** dans le **répertoire** d'installation de symfony.
6. Testez l'exemple en visitant http: // localhost: 8000 / hello
   (Si vous n'utilisez pas le serveur http intégré de Symfony, rendez- vous sur http: // localhost / (symfony-dir) /web/app_dev.php/hello)
7. C'est tout. Suivant: utilisez twig pour rendre la réponse.

```
<?php
// src/AppBundle/Controller/MyController.php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;

class MyController
{
    /**
     * @Route("/hello")
     */
    public function myHelloAction()
    {
        return new Response(
            '<html><body>
                   I\'m the response for request <b>/hello</b>
             </body></html>'
        );
    }
}
```

REMARQUE: Toutes les classes de contrôleur doivent avoir des extrémités avec le mot « **Controller** » et les méthodes liées aux routages doivent se terminer par le mot « **Action** ». En outre, dans quel contrôleur vos actions sont placées ne sont pas pertinentes jusqu'à ce que vous définissiez un préfixe de routage pour le contrôleur.

Installation et configuration de Symfony

Vérification des exigences

Exécutez `bin/symfony_requirements` pour vérifier les exigences de symfony et les paramètres php cli. Installez tous les paquets nécessaires pour exécuter un projet symfony. Configurer votre php.ini par exemple en définissant le fuseau horaire et short_open_tag. Paramétrer à la fois php.ini pour votre serveur web php (ex: /etc/php/apache2/php.ini) et php cli (ex: /etc/php/cli/php.ini). Ouvrez http: //localhost/config.php pour vérifier les paramètres du serveur Web PHP. Si tout est passé, vous êtes prêt à exécuter votre projet symfony.

Projet en cours

Exécutez le `composer install` pour installer toutes les dépendances. Ensuite, configurez l'autorisation pour `var/cache` , `var/logs` et `var/sessions` .

Documentation officielle détaillée ici

Lire Démarrer avec symfony2 en ligne: https://riptutorial.com/fr/symfony2/topic/909/demarrer-avec-symfony2

Chapitre 2: Configuration pour les bundles

Introduction

Voici une description de la manière dont vous pouvez créer une configuration pour votre propre bundle dans /app/config/config.{yml,xml}

Examples

Créer une configuration dans l'application / config / config.yml

```
amazingservice:
    url: 'http://amazing.com'
    client_id: 'test_client_1'
    client_secret: 'test_secret'
```

Ceci est un exemple de base pour créer une configuration au format yml, pour suivre le format yml, vous pouvez prendre une configuration plus approfondie.

Définir la configuration dans le bundle créé

Par exemple, vous avez un bundle, généré par la console symfony. Dans ce cas, dans DependencyInjection / Configuration.php, vous devez insérer votre représentation de configuration:

```
$treeBuilder = new TreeBuilder();
        $rootNode = $treeBuilder->root('amazingservice');

        $rootNode->children()
            ->scalarNode('url')->end()
            ->scalarNode('client_id')->end()
            ->scalarNode('client_secret')->end()
            ->end()
        ->end();
```

En gros, dans DependencyInjection / AmazingserviceExtension.php, vous verrez les lignes suivantes:

```
$configuration = new Configuration();
$config = $this->processConfiguration($configuration, $configs);
```

Ce n'est pas suffisant pour obtenir la configuration dans les Srevices. Vous devez le prendre dans le conteneur.

```
$container->setParameter(
  'amazingservice.config',
  $config
);
```

Dans ce cas, la configuration dans le conteneur, donc si votre service obtient le conteneur en tant que paramètre constructeur:

```
base.amazingservice:
        class: Base\AmazingBundle\Services\AmazingServices
        arguments: [@service_container]
```

Ensuite, vous pouvez obtenir la configuration dans le service avec le code suivant, où la configuration sera un tableau associatif:

```
private $config;

public function __construct(Container $container){
  $this->config = $container->getParameter('amazingservice.config');
}
```

Lire Configuration pour les bundles en ligne:
https://riptutorial.com/fr/symfony2/topic/8153/configuration-pour-les-bundles

Chapitre 3: Création de services Web avec Symfony 2.8

Examples

Travailler avec l'API RESTFul

Le transfert d'état représentatif (REST) est un style architectural utilisé pour le développement Web, introduit et défini en 2000 par Roy Fielding.

Voir sur le wiki: wiki REST

Il est basé sur le protocole HTTP (HTTP on Wiki), les requêtes HTTP (GET, POST, PATCH, DELETE ...) / les codes réponses (404, 400, 200, 201, 500 ...) et la structure des corps.

C'est un excellent moyen d'exposer vos données à un autre système sur Internet.

Imaginez que vous souhaitiez créer une api RESTFul pour gérer votre StackOverFlower (utilisateur) sur votre base de données locale.

Prenons l'exemple!

Framework Symfony 2.8

1. Serveur Web :

Vous devez installer et configurer un serveur Web sur votre machine locale, voir Wamp ou Lamp ou Mamp : vous devez avoir une version récente de PHP (**!!! Exigences Symfony !!!**)

2. Php cli et compositeur:

Vous devez configurer PHP cli (variable selon notre système), tapez ce "PHP cli [OS-NAME] How-to" dans notre ami Google! Vous devez installer le composeur, voir Composer install

3. Symfony:

Vous devez installer Symfony 2.8 (avec compositeur, c'est mieux), ouvrez un terminal (ou cmd sur Windows) et accédez au chemin de votre serveur Web.

Symfony 2 fonctionne avec l'un des meilleurs types de structure: Bundles. Tous sont des paquets sur Symfony! Nous pouvons le tester ci-dessus.

```
cd /your-web-server-path/
composer create-project symfony/framework-standard-edition example "2.8.*"
```

Allez dans l'arborescence voir: Symfony 2.8 est installé sur le répertoire "example".

4. FOSRest (for FriendsOfSymfony) sur JMSSerializer Bundle:

Vous devez installer ces deux bundles:

JMSSerializer (Install):

```
composer require jms/serializer-bundle "~0.13"
```

FosRestBundle (Install):

```
composer require friendsofsymfony/rest-bundle
```

N'oubliez pas de les activer dans AppKernel.php!

5. Configuration de base:

Créez votre propre ensemble "Exemple" et créez la base de données.

```
cd /path/to/your/symfony/
php app/console generate:bundle
php app/console doctrine:generate:database
```

Allez au bas du fichier de configuration de votre application Symfony 2.8 et collez-le:

```
#app/config/config.yml
fos_rest:
    format_listener:
        rules:
            - { path: '^/stackoverflower', priorities: ['xml', 'json'], fallback_format: xml,
prefer_extension: true }
            - { path: '^/', priorities: [ 'text/html', '*/*'], fallback_format: html,
prefer_extension: true }
```

Faites votre répertoire de doctrine ("example / src / ExampleBundle / Entity") et le fichier de ressources ("StackOverFlower.orm.yml"):

```
# src/ExampleBundle/Resources/config/doctrine/StackOverFlower.orm.yml
ExampleBundle\Entity\StackOverFlower:
    type: entity
    table: stackoverflower
    id:
        id:
            type: integer
            generator: { strategy: AUTO }
    fields:
        name:
            type: string
            length: 100
```

Générer un schéma d'entité et de mise à jour:

```
php app/console doctrine:generate:entity StackOverFlower
```

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

Faire un contrôleur par défaut:

```
#src/ExampleBundle/Controller/StackOverFlowerController.php

namespace ExampleBundle\Controller;

use FOS\RestBundle\Controller\FOSRestController;
use Symfony\Component\HttpFoundation\Request;

use FOS\RestBundle\Controller\Annotations\Get;
use FOS\RestBundle\Controller\Annotations\Post;
use FOS\RestBundle\Controller\Annotations\Delete;

use ExampleBundle\Entity\StackOverFlower;

class StackOverFlowerController extends FOSRestController
{
    /**
     * findStackOverFlowerByRequest
     *
     * @param Request $request
     * @return StackOverFlower
     * @throws NotFoundException
     */
    private function findStackOverFlowerByRequest(Request $request) {

        $id = $request->get('id');
        $user = $this->getDoctrine()->getManager()-
>getRepository("ExampleBundle:StackOverFlower")->findOneBy(array('id' => $id));

        return $user;
    }

    /**
     * validateAndPersistEntity
     *
     * @param StackOverFlower $user
     * @param Boolean $delete
     * @return View the view
     */
    private function validateAndPersistEntity(StackOverFlower $user, $delete = false) {

        $template = "ExampleBundle:StackOverFlower:example.html.twig";

        $validator = $this->get('validator');
        $errors_list = $validator->validate($user);

        if (count($errors_list) == 0) {

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

            if ($delete === true) {
                $em->remove($user);
            } else {
                $em->persist($user);
            }

            $em->flush();
```

```
            $view = $this->view($user)
                        ->setTemplateVar('user')
                        ->setTemplate($template);
        } else {

            $errors = "";
            foreach ($errors_list as $error) {
                $errors .= (string) $error->getMessage();
            }

            $view = $this->view($errors)
                        ->setTemplateVar('errors')
                        ->setTemplate($template);

        }

        return $view;
    }

    /**
     * newStackOverFlowerAction
     *
     * @Get("/stackoverflower/new/{name}")
     *
     * @param Request $request
     * @return String
     */
    public function newStackOverFlowerAction(Request $request)
    {
        $user = new StackOverFlower();
        $user->setName($request->get('name'));

        $view = $this->validateAndPersistEntity($user);

        return $this->handleView($view);
    }

    /**
     * editStackOverFlowerAction
     *
     * @Get("/stackoverflower/edit/{id}/{name}")
     *
     * @param Request $request
     * @return type
     */
    public function editStackOverFlowerAction(Request $request) {

        $user = $this->findStackOverFlowerByRequest($request);

        if (! $user) {
            $view = $this->view("No StackOverFlower found for this id:". $request->get('id'),
404);
            return $this->handleView($view);
        }

        $user->setName($request->get('name'));

        $view = $this->validateAndPersistEntity($user);

        return $this->handleView($view);
```

```
    }

    /**
     * deleteStackOverFlowerAction
     *
     * @Get("/stackoverflower/delete/{id}")
     *
     * @param Request $request
     * @return type
     */
    public function deleteStackOverFlowerAction(Request $request) {

        $user = $this->findStackOverFlowerByRequest($request);

        if (! $user) {
            $view = $this->view("No StackOverFlower found for this id:". $request->get('id'),
404);
            return $this->handleView();
        }

        $view = $this->validateAndPersistEntity($user, true);

        return $this->handleView($view);
    }

    /**
     * getStackOverFlowerAction
     *
     * @Get("/stackoverflowers")
     *
     * @param Request $request
     * @return type
     */
    public function getStackOverFlowerAction(Request $request) {

        $template = "ExampleBundle:StackOverFlower:example.html.twig";

        $users = $this->getDoctrine()->getManager()-
>getRepository("ExampleBundle:StackOverFlower")->findAll();

        if (count($users) === 0) {
            $view = $this->view("No StackOverFlower found.", 404);
            return $this->handleView();
        }

        $view = $this->view($users)
                    ->setTemplateVar('users')
                    ->setTemplate($template);

        return $this->handleView($view);
    }
}
```

Faites votre vue Twig par défaut:

```
#src/ExampleBundle/Resources/views/StackOverFlower.html.twig
{% if errors is defined %}
  {{ errors }}
{% else %}
  {% if users is defined %}
```

```
    {{ users | serialize }}
  {% else %}
    {{ user | serialize }}
  {% endif %}
{% endif %}
```

Vous venez de faire votre première API RESTFul!

Vous pouvez le tester sur: http: //votre-nom-serveur/votre-symfony-path/app_dev.php/stackoverflower/new/test .

Comme vous pouvez le voir dans la base de données, un nouvel utilisateur a été créé avec le nom "test".

Vous pouvez obtenir la liste des stackoverflower sur: http: //votre-nom-serveur/votre-symfony-path/app_dev.php/stackoverflowers

Vous avez un exemple complet sur mon compte github de cet exemple: exemple Git Hub , sur la branche "master" de cet exemple, et sur la branche "real-routes" un exemple avec une URL plus appropriée (comme POST et DELETE).

A plus tard pour un exemple avec SOAP!

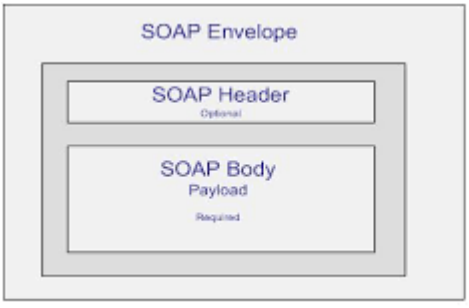
Meilleures salutations,

Mathieu

Travailler avec l'API SOAP

SOAP (Simple Access Object Protocol) est basé sur XML, comme XML-RPC, *est ancêtre* , avec un fichier appelé **WSDL** , qui décrit la méthode à exposer.

Ce protocole est souvent basé sur **SOAP-Enveloppe** , un **SOAP-Body** , ou encore **SOAP-Header** , les données sont enveloppées dans une structure et interprétées de la même manière à partir de différentes langues.



Pour plus d'informations, voir: SOAP sur wiki

Comme décrit ci-dessus, le plus important pour décrire votre service Web est le fichier **WSDL** , voir: explication WSDL sur le wiki

La base du travail sera de définir ce qui est exposé sur votre API SOAP, votre classe et votre

processus métier seront automatiquement gérés par la classe PHP SOAPServer de base. Vous avez toujours besoin du code!

Voyons comment le fichier est construit:

1. Service: Définit l'URI de l'API et ce qui sera associé.
2. Liaison: elle définit les opérations associées au service
3. Opérations: certaines méthodes que vous souhaitez exposer au Web
4. PortTypes: définir des requêtes et des réponses
5. Demandes et réponses: ce que vous attendez d'entrée et de sortie
6. Messages: quelle forme attendez-vous (paramètres) sur chaque IO, ils peuvent être simples (string, integer, float ...) ou complexes (format structuré)

Avec ces informations de base, vous pouvez atteindre toutes les API que vous souhaitez.

Imaginez que vous souhaitiez créer une API SOAP pour gérer votre StackOverFlower (utilisateur) sur votre base de données locale.

Prenons l'exemple!

Installez le serveur Web, Php cli, Composer, Symfony 2.8, créez un nouvel ensemble "ExampleBundle" et créez le schéma comme décrit ci-dessus.

Avant de commencer à construire notre logique métier, nous devions savoir quoi exposer de notre contrôleur. Ce travail est effectué en utilisant le WSDL. Ceci est un exemple d'une bonne syntaxe d'un WSDL:

```
<definitions name="StackOverFlowerService"
   targetNamespace="http://example/soap/stackoverflower.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://example/soap/stackoverflower.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="NewRequest">
      <part name="name" type="xsd:string"/>
   </message>

   <message name="NewResponse">
      <part name="status" type="xsd:string"/>
   </message>

   <message name="getListRequest"></message>

   <message name="getListResponse">
      <part name="list" type="xsd:string"/>
   </message>

   <message name="editRequest">
      <part name="id" type="xsd:string"/>
      <part name="name" type="xsd:string"/>
   </message>

   <message name="editResponse">
      <part name="status" type="xsd:string"/>
```

```
</message>

<message name="deleteRequest">
   <part name="id" type="xsd:string"/>
</message>

<message name="deleteResponse">
   <part name="status" type="xsd:string"/>
</message>

<portType name="StackOverFlower_PortType">
   <operation name="newStack">
      <input message="tns:NewRequest"/>
      <output message="tns:NewResponse"/>
   </operation>
   <operation name="getList">
      <input message="tns:getListRequest"/>
      <output message="tns:getListResponse"/>
   </operation>
   <operation name="edit">
      <input message="tns:editRequest"/>
      <output message="tns:editResponse"/>
   </operation>
   <operation name="delete">
      <input message="tns:deleteRequest"/>
      <output message="tns:deleteResponse"/>
   </operation>
</portType>

<binding name="StackOverFlower_Binding" type="tns:StackOverFlower_PortType">
   <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="newStack">
      <soap:operation soapAction="newStack"/>
      <input>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:example:new"
            use="encoded"/>
      </input>

      <output>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:example:new"
            use="encoded"/>
      </output>
   </operation>

   <operation name="getList">
      <soap:operation soapAction="getList"/>
      <input>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:example:get-list"
            use="encoded"/>
      </input>

      <output>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
                namespace="urn:example:get-list"
                use="encoded"/>
         </output>
      </operation>

      <operation name="edit">
         <soap:operation soapAction="edit"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:example:edit"
               use="encoded"/>
         </input>

         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:example:edit"
               use="encoded"/>
         </output>
      </operation>

      <operation name="delete">
         <soap:operation soapAction="delete"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:example:delete"
               use="encoded"/>
         </input>

         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:example:delete"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="StackOverFlower_Service">
      <documentation>Description File of StackOverFlowerService</documentation>
      <port binding="tns:StackOverFlower_Binding" name="StackOverFlower_Port">
         <soap:address
            location="http://example/stackoverflower/" />
      </port>
   </service>
</definitions>
```

Nous devons prendre cela sur votre répertoire web symfony (dans le sous-répertoire soap et nommez-le "stackoverflower.wsdl").

Vraiment inspiré de WSDl Exemple . Vous pouvez valider cela avec un validateur WSDl en ligne

Après cela, nous pouvons rendre notre service et contrôleur de base inspiré de SOAP Symfony 2.8 Doc .

Service géré par PHP SOAPServer:

```
#src\ExampleBundle\Services\StackOverFlowerService.php
namespace ExampleBundle\Services;

use Doctrine\ORM\EntityManager;
use Symfony\Component\Serializer\Serializer;
use Symfony\Component\Serializer\Encoder\XmlEncoder;
use Symfony\Component\Serializer\Encoder\JsonEncoder;
use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;

use ExampleBundle\Entity\StackOverFlower;

class StackOverFlowerService
{
  private $em;
  private $stackoverflower;

  public function __construct(EntityManager $em)
  {
    $this->em = $em;
  }

  public function newStack($name)
  {
    $stackoverflower = new StackOverFlower();
    $stackoverflower->setName($name);

    $this->em->persist($stackoverflower);
    $this->em->flush();

    return "ok";
  }

  public function getList()
  {
    $stackoverflowers = $this->em->getRepository("ExampleBundle:StackOverFlower")->findAll();

    $encoders = array(new XmlEncoder(), new JsonEncoder());
    $normalizers = array(new ObjectNormalizer());

    $serializer = new Serializer($normalizers, $encoders);

    return $serializer->serialize($stackoverflowers, 'json');
  }

  public function edit($id, $name)
  {
    $stackoverflower = $this->em->getRepository("ExampleBundle:StackOverFlower")-
>findOneById($id);

    $stackoverflower->setName($name);

    $this->em->persist($stackoverflower);
    $this->em->flush();

    return "ok";
  }

  public function delete($id)
  {
    $stackoverflower = $this->em->getRepository("ExampleBundle:StackOverFlower")-
>findOneById($id);
```

```
    $this->em->remove($stackoverflower);
    $this->em->flush();

    return "ok";
  }
}
```

Configurez ce service:

```
#src\ExampleBundle\Resources\config\services.yml
services:
  stackoverflower_service:
    class: ExampleBundle\Services\StackOverFlowerService
    arguments: [@doctrine.orm.entity_manager]
```

Comme vous pouvez le voir, nous injectons la Doctrine Entity Manger comme une dépendance car nous devons l'utiliser pour CRUD StackOverFlower Object.

Contrôleur, qui expose l'objet de service:

```
#src\ExampleBundle\Controller\StackOverFlowerController.php
namespace ExampleBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

class StackOverFlowerController extends Controller
{
  public function indexAction()
  {
    ini_set("soap.wsdl_cache_enabled", "0");

    $options = array(
      'uri' => 'http://example/app_dev.php/soap',
      'cache_wsdl' => WSDL_CACHE_NONE,
      'exceptions' => true
    );

    $server = new \SoapServer(dirname(__FILE__).'/../../../**web/soap/stackoverflower.wsdl**',
$options);
    $server->setObject($this->get('stackoverflower_service'));

    $response = new Response();
    $response->headers->set('Content-Type', 'text/xml; charset=utf-8');

    ob_start();
    $server->handle();
    $response->setContent(ob_get_clean());

    return $response;
  }
}
```

Pour en savoir plus sur les services, voir: Conteneur de services sur le doc Symfony

La route :

```
example_soap:
  path:     /soap
  defaults: { _controller: ExampleBundle:StackOverFlower:index }
```

Le modèle de base de la brindille:

```
#src\ExampleBundle\Resources\views\Soap\default.html.twig
{% if status is defined %}
{{ status }}
{% else %}
{{ list }}
{% endif %}
```

Nous avons créé votre première API SOAP avec Symfony 2.8!

Avant de l'exposer, il faut tester !!

Dans votre StackOverFlowerController, ajoutez ceci:

```
  public function testNewAction(Request $request)
  {
    $service = $this->get('stackoverflower_service');
    $result = $service->newStack($request->query->get('name'));

    return $this->render('ExampleBundle:Soap:default.html.twig', array('status' => $result));
  }

  public function testEditAction(Request $request)
  {
    $service = $this->get('stackoverflower_service');
    $result = $service->edit($request->query->get('id'), $request->query->get('name'));

    return $this->render('ExampleBundle:Soap:default.html.twig', array('status' => $result));
  }

  public function testGetListAction(Request $request)
  {
    $service = $this->get('stackoverflower_service');
    $result = $service->getList();

    return $this->render('ExampleBundle:Soap:default.html.twig', array('list' => $result));
  }

  public function testDeleteAction(Request $request)
  {
    $service = $this->get('stackoverflower_service');
    $result = $service->delete($request->query->get('id'));

    return $this->render('ExampleBundle:Soap:default.html.twig', array('list' => $result));
  }

// To test this from an another server, you can type this :
// $client = new \SoapClient("http://example/app_dev.php/soap?wsdl", array("trace" => 1,
"exception" => 1));
// $result = $client->newStack($request->query->get('name'));
```

```
// print_r($result);
```

Les routes:

```
test_new:
  path:     /stackoverflower/new
  defaults: { _controller: ExampleBundle:StackOverFlower:testNew }

test_edit:
  path:     /stackoverflower/edit
  defaults: { _controller: ExampleBundle:StackOverFlower:testEdit }

test_get_list:
  path:     /stackoverflower/get-list
  defaults: { _controller: ExampleBundle:StackOverFlower:testGetList }

test_delete:
  path:     /stackoverflower/delete
  defaults: { _controller: ExampleBundle:StackOverFlower:testDelete }
```

Vous pouvez taper ceci dans votre navigateur:

1. getList
2. Nouveau
3. modifier
4. effacer

Ceci est un exemple très basique d'une API non sécurisée avec SOAP, je peux faire un exemple d'exemple sécurisé derrière une authentification par clé api ultérieurement.

Que tous les gens ...

Mathieu

Lire Création de services Web avec Symfony 2.8 en ligne:
https://riptutorial.com/fr/symfony2/topic/6355/creation-de-services-web-avec-symfony-2-8

Chapitre 4: Créer des services Web avec Symfony en utilisant Rest

Examples

Utilisation de Symfony REST Edition

Symfony REST Edition est une application **Symfony2** entièrement fonctionnelle que vous pouvez utiliser comme squelette pour vos nouvelles applications.

Disponible sur Github

Il est pré-configuré avec les bundles suivants:

Paquet	La description
FrameworkBundle	Le noyau du framework Symfony.
SensioFrameworkExtraBundle	Ajoute plusieurs améliorations, telles que des capacités d'annotation de modèle et de routage.
DoctrineBundle	Ajoute le support pour l'ORM Doctrine.
TwigBundle	Ajoute le support du moteur de template Twig .
SecurityBundle	Ajout de la sécurité en intégrant le composant de sécurité de Symfony.
SwiftmailerBundle	Ajout du support pour Swiftmailer , une bibliothèque pour l'envoi d'emails.
MonologBundle	Ajout du support pour Monolog , une bibliothèque de journalisation.
AsseticBundle	Ajout du support pour Assetic , une bibliothèque de traitement des actifs.
WebProfilerBundle (dans un environnement de développement / test)	Ajoute une fonctionnalité de profilage et la barre d'outils de débogage Web.
SensioDistributionBundle (dans l'environnement de développement / test)	Ajoute des fonctionnalités pour la configuration et l'utilisation des distributions **Symfony** .
SensioGeneratorBundle (dans un environnement de développement / test)	Ajoute des capacités de génération de code.

Paquet	La description
AcmeDemoBundle (dans un environnement de développement / test)	Un bundle de démonstration avec un exemple de code.
FOSRestBundle	Le FOSRestBundle ajoute une fonctionnalité REST.
FOSHttpCacheBundle	Ce bundle offre des outils pour améliorer la mise en cache HTTP avec **Symfony2** .
NelmioApiDocBundle	Ajoute des fonctionnalités de documentation de l'API.
BazingaHateoasBundle	Ajoute le support HATEOAS.
HautelookTemplatedUriBundle	Ajoute la prise en charge des URIs basés sur des modèles (RFC 6570).
BazingaRestExtraBundle	BazingaRestExtraBundle Fournit des fonctionnalités supplémentaires pour les API REST créées avec Symfony2.

Lire Créer des services Web avec Symfony en utilisant Rest en ligne: https://riptutorial.com/fr/symfony2/topic/6020/creer-des-services-web-avec-symfony-en-utilisant-rest

Chapitre 5: Demande

Remarques

Liens de la documentation API (master):

- Demande
- RequestStack

L'objet Request contient plusieurs données importantes, telles que les paramètres régionaux et le contrôleur correspondant. Vous pouvez les utiliser et les gérer par les événements HttpKernel. Pour une compréhension fiable du cycle live de Request-Responce, lisez cette page de documentation du composant HttpKernel (très utile!).

Examples

Accès à la demande dans un contrôleur

```
<?php

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class TestController extends Controller
{
   //Inject Request HTTP Component in your function then able to exploit it
   public function myFunctionAction(Request $request)
   {
      //BASICS

      //retrieve $_POST variables from request
      $postRequest = $request->request->get('my_data');
      //retrieve $_GET variables from request
      $getRequest = $request->query->get('my_data');
      //get current locale
      $locale = $request->getLocale();
   }
}
```

Notez que l'objet Request injecté s'applique à la demande en cours (il peut ou non correspondre à la demande principale).

Accès à la requête dans un modèle Twig ou PHP.

Dans le modèle Twig, l'objet Request est disponible à

```
{{ app.request }}
```

Lorsque vous souhaitez afficher la méthode de requête dans Twig, essayez ceci:

```
<p>Request method: {{ app.request.method }}</p>
```

Dans le template PHP

```
<p>Request method: <?php echo $app->getRequest()->getMethod() ?></p>
```

Lire Demande en ligne: https://riptutorial.com/fr/symfony2/topic/4870/demande

Chapitre 6: Déploiement de Symfony2

Examples

Étapes pour déplacer le projet Symfony 2 vers l'hébergement manuellement

Cela dépend du type d'hébergement que vous avez:

1. Si vous avez une console SSH, vous pouvez le faire sur l'hébergement après l'étape 2, si vous ne l'avez pas encore fait localement: lancez la commande

   ```
   php app/console cache:clear --env=prod'.
   ```

2. Supposons que vous ayez sur vous des répertoires d'hébergement de votre `youdomain/public_html` , de sorte que tous les fichiers Web doivent se trouver dans `public_html` . Vous devez donc tout télécharger depuis le projet Symfony (dossiers: `app` , `src` , `vendors` , `bin` ; fichiers: `deps` , `deps.lock`), sauf pour le dossier `web` dans le dossier `youdomain` . Tout depuis le dossier `web` télécharger dans le dossier `public_html` .

3. Vérifiez CHMOD pour les dossiers `app/cache` et `app/logs` , il devrait y avoir un accès en écriture.

4. S'il n'y a pas de fichier .htaccess dans public_html, créez-le et ajoutez-y un tel code: https://raw.github.com/symfony/symfony-standard/master/web/.htaccess

5. Vous devez maintenant utiliser `youdomain.com/index` au lieu de `youdomain.com/app_dev.php/index` , que vous utilisez localement. Si un site ne fonctionne toujours pas, vous pouvez ouvrir le fichier `web/config.php` et trouver un code où une vérification des performances IP est effectuée, vous y trouverez uniquement l'adresse IP `127.0.0.1` . Ajoutez votre adresse IP actuelle à cette liste et téléchargez une nouvelle configuration sur le serveur. Ensuite, vous pouvez ouvrir le chemin `yourdomain/config.php` et vérifier ce qui ne va pas. Si `config.php` montre que tout va bien, mais que cela ne fonctionne toujours pas, vous pouvez activer `app_dev.php` pour le déboguer: ouvrez `app/app_dev.php` et votre adresse IP de la même manière que dans `config.php` . Vous pouvez maintenant exécuter des scripts en local en utilisant `app_dev.php` .

Lire Déploiement de Symfony2 en ligne:
https://riptutorial.com/fr/symfony2/topic/6039/deploiement-de-symfony2

Chapitre 7: Envoi d'options à une classe de formulaire

Syntaxe

- $ form = $ this-> createForm (HouseholdType :: class, $ household, $ formOptions);

Paramètres

Paramètre	Définition
HouseholdType :: class	classe de formulaire personnalisée pour l'entité Household
$ ménage	une instance de l'entité Household (généralement créée par `$household = new Household();`)
$ formOptions	un tableau d'options définies par l'utilisateur à transmettre à la classe de formulaire, par exemple, `$formOptions = array('foo' => 'bar');`

Remarques

Lorsque vous créez une classe de formulaire, les champs de formulaire sont ajoutés dans la `public function buildForm(FormBuilderInterface $builder, array $options) {...}` . Le paramètre `$options` inclut un ensemble d'options par défaut telles que `attr` et `label` . Pour que vos options personnalisées soient disponibles dans la classe de formulaire, les options doivent être initialisées dans `configureOptions(OptionsResolver $resolver)`

Donc, pour notre exemple concret:

```
public function configureOptions(OptionsResolver $resolver)
{
    $resolver->setDefaults(array(
        'data_class' => 'AppBundle\Entity\Household',
        'disabledOptions' => [],
    ));
}
```

Examples

Un exemple concret d'un contrôleur de ménage

Contexte: L'entité Household comprend un ensemble d'options, chacune d'entre elles étant une entité gérée dans un backend admin. Chaque option a un booléen `enabled` drapeau. Si une option

précédemment activée est définie sur désactivée, elle devra être conservée dans les modifications ultérieures du foyer, mais ne pourra pas être modifiée. Pour ce faire, la définition de champ dans la classe de formulaire affichera le champ en tant que champ de sélection désactivé si l'option a été `enabled = false` (mais est persistée car le bouton de soumission déclenche un script JavaScript qui supprime l'attribut `disabled`). d'être affiché

La classe de formulaire doit alors connaître, pour une entité Household donnée, laquelle de ses options a été désactivée. Un service a été défini qui renvoie un tableau des noms des entités optionnelles désactivées. Ce tableau est `$disabledOptions` .

```
    $formOptions = [
        'disabledOptions' => $disabledOptions,
        ];
    $form = $this->createForm(HouseholdType::class, $household, $formOptions);
```

Comment les options personnalisées sont utilisées dans la classe de formulaire

```
        ->add('housing', EntityType::class,
            array(
            'class' => 'AppBundle:Housing',
            'choice_label' => 'housing',
            'placeholder' => '',
            'attr' => (in_array('Housing', $options['disabledOptions']) ? ['disabled' =>
'disabled'] : []),
            'label' => 'Housing: ',
            'query_builder' => function (EntityRepository $er) use ($options) {
                if (false === in_array('Housing', $options['disabledOptions'])) {
                    return $er->createQueryBuilder('h')
                        ->orderBy('h.housing', 'ASC')
                        ->where('h.enabled=1');
                } else {
                    return $er->createQueryBuilder('h')
                        ->orderBy('h.housing', 'ASC');
                }
            },
        ))
```

Entité de logement

```
/**
 * Housing.
 *
 * @ORM\Table(name="housing")
 * @ORM\Entity
 */
class Housing
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
```

```
     */
    protected $id;

    /**
     * @var bool
     *
     * @ORM\Column(name="housing", type="string", nullable=false)
     * @Assert\NotBlank(message="Housing may not be blank")
     */
    protected $housing;

    /**
     * @var bool
     *
     * @ORM\Column(name="enabled", type="boolean", nullable=false)
     */
    protected $enabled;

    /**
     * Get id.
     *
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set housing.
     *
     * @param int $housing
     *
     * @return housing
     */
    public function setHousing($housing)
    {
        $this->housing = $housing;

        return $this;
    }

    /**
     * Get housing.
     *
     * @return int
     */
    public function getHousing()
    {
        return $this->housing;
    }

    /**
     * Set enabled.
     *
     * @param int $enabled
     *
     * @return enabled
     */
    public function setEnabled($enabled)
    {
```

```
        $this->enabled = $enabled;

        return $this;
    }

    /**
     * Get enabled.
     *
     * @return int
     */
    public function getEnabled()
    {
        return $this->enabled;
    }

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\OneToMany(targetEntity="Household", mappedBy="housing")
     */
    protected $households;

    public function addHousehold(Household $household)
    {
        $this->households[] = $household;
    }

    public function getHouseholds()
    {
        return $this->households;
    }
}
```

Lire Envoi d'options à une classe de formulaire en ligne:
https://riptutorial.com/fr/symfony2/topic/7237/envoi-d-options-a-une-classe-de-formulaire

Chapitre 8: Exemple de Symfony Twig Extenstion

Paramètres

Paramètres	La description
$ doctrine	objet de doctrine que nous passons du service.

Examples

Exemple de base de l'extension Symfony Twig

Dans cet exemple, je définis deux fonctions personnalisées. 1 - la fonction countryFilter obtient le code abrégé du pays en entrée et renvoie le nom complet du pays. 2 - _countPrinterTasks est utilisé pour calculer le nombre de tâches assignées à un utilisateur particulier.

```
<?php

namespace DashboardBundle\Twig\Extension;

use Symfony\Bridge\Doctrine\RegistryInterface;
use Symfony\Component\HttpKernel\HttpKernelInterface;
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\Security\Core\SecurityContext;

/**
 * Class DashboardExtension
 * @package DashboardBundle\Twig\Extension
 */
class DashboardExtension extends \Twig_Extension
{
    protected $doctrine;
    private $context;


    /**
     * DashboardExtension constructor.
     * @param RegistryInterface $doctrine
     * @param SecurityContext $context
     */
    public function __construct(RegistryInterface $doctrine,SecurityContext $context)
    {
        $this->doctrine = $doctrine;
        $this->context = $context;
    }

    /**
     * @return mixed
     */
    public function getUser()
```

```
    {
        return $this->context->getToken()->getUser();
    }

    /**
     * @return array
     */
    public function getFilters()
    {
        return array(
            new \Twig_SimpleFilter('country', array($this, 'countryFilter')),
            new \Twig_SimpleFilter('count_printer_tasks', array($this, '_countPrinterTasks')),
        );
    }


 /**
     * @param $countryCode
     * @param string $locale
     * @return mixed
     */
    public function countryFilter($countryCode,$locale = "en")
    {
        $c = \Symfony\Component\Intl\Intl::getRegionBundle()->getCountryNames($locale);

        return array_key_exists($countryCode, $c)
            ? $c[$countryCode]
            : $countryCode;
    }



    /**
     * Returns total count of printer's tasks.
     * @return mixed
     */
    public function _countPrinterTasks(){
        $count = $this->doctrine->getRepository('DashboardBundle:Task')-
>countPrinterTasks($this->getUser());
        return $count;
    }




    /**
     * {@inheritdoc}
     */
    public function getName()
    {
        return 'app_extension';
    }
}
```

Pour l'appeler depuis la brindille, nous devons juste utiliser comme ci-dessous;

```
{% set printer_tasks = 0|count_printer_tasks() %}

 <tr>
    <td>Nationality</td>
```

```
    <td>
        {{ user.getnationality|country|ucwords }}
    </td>
</tr>
```

Déclarez cette extension en tant que service dans votre fichier `bundle/resource/config/service.yml` .

```
services:

    app.twig_extension:
        class: DashboardBundle\Twig\Extension\DashboardExtension
        arguments: ["@doctrine", @security.context]
        tags:
            - { name: twig.extension }
```

Lire Exemple de Symfony Twig Extenstion en ligne: https://riptutorial.com/fr/symfony2/topic/5891/exemple-de-symfony-twig-extenstion

Chapitre 9: Extensions Symfony Twig

Examples

Une simple extension de brindille - Symfony 2.8

Avant de créer une extension, vérifiez toujours si elle a déjà **été implémentée** .

La première chose à faire est de définir la classe d'extension qui hébergera les filtres et / ou fonctions des ramifications.

```
<?php

namespace AppBundle\Twig;

class DemoExtension extends \Twig_Extension {
    /**
     * A unique identifier for your application
     *
     * @return  string
     */
    public function getName()
    {
        return 'demo';
    }

    /**
     * This is where one defines the filters one would to use in their twig
     * templates
     *
     * @return  Array
     */
    public function getFilters()
    {
        return array (
            new \Twig_SimpleFilter (
                'price',                        // The name of the twig filter
                array($this, 'priceFilter')
            ),
        );
    }

    public function priceFilter($number, $decimals = 0, $decPoint = '.', $thousandsSep = ',')
    {
        return '$' . number_format($number, $decimals, $decPoint, $thousandsSep);
    }

    /**
     * Define the functions one would like availed in their twig template
     *
     * @return  Array
     */
    public function getFunctions() {
        return array (
            new \Twig_SimpleFunction (
                'lipsum',                        // The name of the twig function
```

```
                array($this, 'loremIpsum')
            )
        );
    }

    public function loremIpsum($length=30) {
        $string = array ();
        $words = array (
            'lorem',        'ipsum',       'dolor',        'sit',
            'amet',         'consectetur', 'adipiscing',   'elit',
            'a',            'ac',          'accumsan',     'ad',
            'aenean',       'aliquam',     'aliquet',      'ante',
            'aptent',       'arcu',        'at',           'auctor',
            'augue',        'bibendum',    'blandit',      'class',
            'commodo',      'condimentum', 'congue',       'consequat',
            'conubia',      'convallis',   'cras',         'cubilia',
            'cum',          'curabitur',   'curae',        'cursus',
            'dapibus',      'diam',        'dictum',       'dictumst',
            'dignissim',    'dis',         'donec',        'dui',
            'duis',         'egestas',     'eget',         'eleifend',
            'elementum',    'enim',        'erat',         'eros',
            'est',          'et',          'etiam',        'eu',
            'euismod',      'facilisi',    'facilisis',    'fames',
            'faucibus',     'felis',       'fermentum',    'feugiat',
            'fringilla',    'fusce',       'gravida',      'habitant',
            'habitasse',    'hac',         'hendrerit',    'himenaeos',
            'iaculis',      'id',          'imperdiet',    'in',
            'inceptos',     'integer',     'interdum',     'justo',
            'lacinia',      'lacus',       'laoreet',      'lectus',
            'leo',          'libero',      'ligula',       'litora',
            'lobortis',     'luctus',      'maecenas',     'magna',
            'magnis',       'malesuada',   'massa',        'mattis',
            'mauris',       'metus',       'mi',           'molestie'
        );

        for ( $i=0; $i<$length; $i++ )
            $string[] = $words[rand(0, 99)];

        return implode(" ", $string);
    }
}
```

On alerte alors le conteneur de service de l'extension de rameau nouvellement créée.

```
# app/config/services.yml
services:
    app.twig.demo_extension:
        class: AppBundle\Twig\DemoExtension
        tags:
            - { name: twig.extension }
```

Avec cela, vous avez tout ce dont vous avez besoin pour pouvoir utiliser votre filtre ou fonction Twig nouvellement créé dans vos modèles de brindilles.

```
<p>Price Filter test {{ '5500' | price }}</p>
<p>{{ lipsum(25) }}</p>
```

Faites un nombre court, par exemple 1 000 -> 1 k, 1 000 000 -> 1 M etc.

Symfony 2.8

```
# AppBundle\Twig\AppExtension.php
<?php
namespace AppBundle\Twig;

class AppExtension extends \Twig_Extension
{
    /**
     * This is where one defines the filters one would to use in their twig
     * templates
     *
     * @return  Array
     */
    public function getFilters()
    {
        return array(
            new \Twig_SimpleFilter('shortNumber', array($this, 'shortNumber')),
        );
    }

    /**
     * Shorten the number
     *
     * @param integer
     * @return string
     */
    public function shortNumber($number)
    {
        $k   = pow(10,3);
        $mil = pow(10,6);
        $bil = pow(10,9);

        if ($number >= $bil)
            return number_format((float)$number / $bil, 1, '.', '').'Billion';
        else if ($number >= $mil)
            return number_format((float)$number / $mil, 1, '.', '').'M';
        else if ($number >= $k)
            return number_format((float)$number / $k, 1, '.', '').'K';
        else
            return (int) $number;
    }

    /**
     * Get name
     */
    public function getName()
    {
        return 'app_extension';
    }
}
```

Ajoutez votre extension à services.yml

```
# app/config/services.yml
services:
```

```
    app.twig_extension:
        class: AppBundle\Twig\AppExtension
        public: false
        tags:
            - { name: twig.extension }
```

Utilisez-le dans TWIG

```
<span>{{ number|shortNumber }}</span>
e.g.
<span>{{ 1234|shortNumber }}</span> -> <span>1.2k</span>
```

Lire Extensions Symfony Twig en ligne: https://riptutorial.com/fr/symfony2/topic/6000/extensions-symfony-twig

Chapitre 10: Gestion des pare-feu et de la sécurité Symfony

Examples

Gestion de la sécurité

La sécurité faisait partie du côté obscur de la documentation de symfony, elle comportait un composant dédié nommé **Security Component** .

Ce composant est configuré dans le fichier **security.yml** du projet d'application principal.

La configuration par défaut est comme celle-ci:

```
# app/config/security.yml
security:
    providers:
        in_memory:
            memory: ~

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        default:
            anonymous: ~
```

Vous pouvez définir des **pare-feu** spécifiques pour restreindre l'accès à certaines URL à des **rôles** spécifiques en fonction d'une hiérarchie définie pour vos **utilisateurs** par un **fournisseur** et des **encodeurs** qui gèrent la sécurité du mot de passe.

Par exemple, si vous souhaitez créer un **fournisseur** personnalisé à partir de votre moteur de base de données, vous pouvez définir comme suit **security.yml** :

```
providers:
    your_db_provider:
        entity:
            class: AppBundle:User
            property: apiKey
```

Ceci est détaillé dans la documentation de symfony: Comment définir un UserProvider personnalisé et à partir de la base de données ou contre LDAP par exemple.

Après cela, vous pouvez définir un **pare-feu** pour restreindre explicitement certaines URL basées sur votre fournisseur utilisateur personnalisé (security.yml):

```
firewalls:
    secured_area:
```

```
        pattern: ^/admin
```

Ou avec **contrôle d'accès** :

```
access_control:
 - { path: ^/admin/users, roles: ROLE_SUPER_ADMIN }
 - { path: ^/admin, roles: ROLE_ADMIN }
```

Voir plus de documentation détaillée ici .

La meilleure façon de gérer les utilisateurs est d'utiliser FosUserBundle qui étend certaines fonctionnalités du framework.

Lire Gestion des pare-feu et de la sécurité Symfony en ligne:
https://riptutorial.com/fr/symfony2/topic/7486/gestion-des-pare-feu-et-de-la-securite-symfony

Chapitre 11: Installez Symfony2 sur localhost

Examples

En utilisant l'invite de commande

La meilleure façon d'installer et de configurer un projet Symfony2 est décrite dans la documentation officielle comme suit:

Mac OS X / Linux

```
$ sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

les fenêtres

```
c:\> php -r "file_put_contents('symfony',
file_get_contents('https://symfony.com/installer'));"
```

Vous pouvez ensuite utiliser le binaire Symfony pour créer le bon échafaudage:

```
$ symfony new my_project
```

Utilisation du composeur sur la console

Étant donné que vous avez déjà installé Compositeur et qu'il est accessible partout dans le monde, vous pouvez simplement créer de nouveaux projets Symfony comme indiqué dans la documentation officielle .

Vous pouvez maintenant créer un nouveau projet Symfony avec le compositeur:

```
composer create-project symfony/framework-standard-edition my_project_name
```

Cela créera le projet dans le répertoire actuel dans lequel vous vous trouvez.

Si vous souhaitez créer le projet avec une version spécifique de Symfony, vous pouvez ajouter un paramètre avec le numéro de version:

```
composer create-project symfony/framework-standard-edition my_project_name "2.8.*"
```

Conseil: Si vous pensez que le compositeur ne fera rien, ajoutez l'indicateur `-vvv` à la commande. De cette façon, vous recevrez des informations détaillées sur ce que fait compositeur en ce moment.

Lire Installez Symfony2 sur localhost en ligne:
https://riptutorial.com/fr/symfony2/topic/6340/installez-symfony2-sur-localhost

Chapitre 12: Le routage

Examples

Renvoie une réponse 404

404 réponses sont renvoyées lorsqu'une ressource est introuvable sur le serveur. Dans Symfony, ce statut peut être créé en lançant une exception `NotFoundHttpException` . Pour éviter une instruction d' `use` supplémentaire dans un contrôleur, utilisez le `createNotFoundException()` fourni par la classe `Controller`

```
<?php

namespace Bundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class TestController extends Controller
{
    /**
     * @Route("/{id}", name="test")
     * Recommended to avoid template() as it has a lot of background processing.
     * Query database for 'test' record with 'id' using param converters.
     */
    public function testAction(Test $test)
    {
        if (!$test) {
            throw $this->createNotFoundException('Test record not found.');
        }
        return $this->render('::Test/test.html.twig', array('test' => $test));
    }

}
```

Routes multiples

Dans Symfony, il est possible de définir plusieurs routes pour une action. Cela peut être très utile si vous avez des fonctions qui font la même chose mais qui ont des paramètres différents.

```
class TestController extends Controller
{
    /**
     * @Route("/test1/{id}", name="test")
     * @Route("/test2/{id}", name="test2")
     * Here you can define multiple routes with multiple names
     */
    public function testAction(Test $test)
    {
        if (!$test) {
            throw $this->createNotFoundException('Test record not found.');
        }
```

```
        return $this->render('::Test/test.html.twig', array('test' => $test));
    }
}
```

Redirection de demande POST

Lorsque vous êtes dans un **controllerAction** Et que vous recevez une **requête POST** , mais que vous souhaitez la **rediriger vers un autre itinéraire** , tout en **conservant la méthode POST et l'objet de requête** , vous pouvez utiliser les éléments suivants:

```
return $this->redirectToRoute('route', array(
    'request' => $request,
), 307);
```

Le code 307 conserve ici la méthode de requête.

Routage basé sur un sous-domaine

Le routage basé sur un sous-domaine peut être géré dans Symfony en utilisant le paramètre `host` . Par exemple, le paramètre `_locale` peut être utilisé comme valeur de sous-domaine.

En supposant

```
locale: en
domain: somedomain.com
```

les paramètres sont définis dans le fichier de configuration `parameters.yml` , route serait:

```
/**
 * @Route(
 *      "/",
 *      name="homepage",
 *      host="{_locale}.{domain}",
 *      defaults={"_locale" = "%locale%", "domain" = "%domain%"},
 *      requirements={"_locale" = "%locale%|de|fr", "domain" = "%domain%"}
 * )
 * @Route(
 *      "/",
 *      name="homepage_default",
 *      defaults={"_locale" = "%locale%"}
 * )
 */
```

De ce point, le routeur peut gérer les adresses URI telles que `http://de.somedomain.com` . La deuxième annotation `@Route` peut être utilisée comme `@Route` pour les paramètres régionaux par défaut et le sous-domaine void, `http://somedomain.com` .

Routes Symfony utilisant Routing.yml

```
    profile_user_profile:
        path:    /profile/{id}
```

```
        defaults: { _controller: ProfileBundle:Profile:profile }
        requirements:
            id: \d+
        methods: [get, delete]
```

Si vous décidez d'utiliser Routing.yml au lieu des Annotations, vous pouvez mieux voir **toutes les routes** et rechercher et en trouver plus facilement.

> C'est à vous de choisir entre **Routing.yml** et **Annotations** . Vous pouvez utiliser les deux pour différents itinéraires, mais ce n'est pas la meilleure solution.

Annotation `@Route()` est l'équivalent:

```
class ProfileController extends Controller
{
    /**
     * @Route("/profile/{id}", name="profile_user_profile", requirements={"id": "\d+"})
     * @Method("GET", "DELETE")
     */
    public function profileAction($id)
    {
        if (!$id) {
            throw $this->createNotFoundException('User not found.');
        }

        return $this->render('::Profile/profile.html.twig', array('id' => $id));
    }
}
```

Lire Le routage en ligne: https://riptutorial.com/fr/symfony2/topic/1691/le-routage

Chapitre 13: Modèles de conception Symfony

Examples

Modèle d'injection de dépendance

Imaginez que vous ayez un gestionnaire de classe pour gérer l'envoi des mails (s'appelle MailManager).

Dans ce cas, vous devez enregistrer les messages envoyés. Une bonne solution consiste à transformer la classe MailManager en un `service` , puis à injecter une classe pour créer des journaux (`Monolog` par exemple) dans MailManager en créant un service.

Pour faire ça :

1- Déclarez la future classe MailManager en tant que service (dans services.yml)

```
services:
    mail.manager.class:
        class:     Vendor/YourBundle/Manager/MailManager
```

2- Injecter le service existant du logger en utilisant la méthode des `argument`

```
services:
    mail.manager.class:
        class:    Project/Bundle/Manager/MailManager
        arguments: ["@logger"]    # inject logger service into constructor
```

3- Créer une classe MailManager

```
<?php

namespace Project\Bundle\Manager;

use Symfony\Component\HttpKernel\Log\LoggerInterface;

class MailManager
{
  protected $logger;

  //initialized logger object
  public function __construct(LoggerInterface $logger)
  {
     $this->logger = $logger;
  }

   public function sendMail($parameters)
   {
      //some codes to send mail

      //example using logger
      $this->logger->info('Mail sending');
```

```
    }
}
```

4- Appelez MailManager dans un contrôleur par exemple

```
<?php

class TestController extends Controller
{
    public function indexAction()
    {
        //some codes...

        //call mail manager service
        $mailManager = $this->get('mail.manager.class');
        //call 'sendMail' function from this service
        $mailManager->sendMail($parameters);

    }
}
```

Lire Modèles de conception Symfony en ligne:
https://riptutorial.com/fr/symfony2/topic/6289/modeles-de-conception-symfony

Chapitre 14: Monolog: améliore tes logs

Examples

Ajouter les détails de l'utilisateur et les paramètres publiés envoyés aux journaux

Les journaux sont très importants. Recréer un contexte d'erreur peut parfois être très pénible en raison du manque d'informations sur le moment et le moment où l'erreur s'est produite.

Cet exemple montre:

- Comment ajouter des données utilisateur dans les journaux d'erreurs
- Comment ajouter des paramètres postaux envoyés en cas d'erreur
- Comment utiliser WebProcessor pour ajouter toutes les données concernant la requête comme:
    - URL
    - ip
    - méthode http
    - serveur
    - référent

Configuration du service

```
services:
    # Permits to convert logs in HTML format for email notification
    monolog.formatter.html:
        class: Monolog\Formatter\HtmlFormatter

    # Add request data (url, ip, http method, server, referrer)
    monolog.processor.web_processor:
        class: Monolog\Processor\WebProcessor
        tags:
            - { name: monolog.processor, method: __invoke }

    # Custom class to include user's data and posted parameters in the logs
    monolog.processor.user:
        class: Company\ToolBoxBundle\Services\Monolog\ExtraProcessor
        arguments:  ["@security.token_storage"]
        tags:
            - { name: monolog.processor }
            - { name: kernel.event_listener, event: kernel.request, method: onKernelRequest }
```

Code de service

```
namespace Company\ToolBoxBundle\Services\Monolog;
```

```
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface;

class ExtraProcessor
{
    /**
    * @var string
    */
    private $postParams = null;

    /**
    * @var TokenStorageInterface
    */
    private $tokenStorage = null;

    /**
    * @var \Company\UserBundle\Entity\User
    */
    private $user = null;

    public function __construct(TokenStorageInterface $tokenStorage)
    {
        $this->tokenStorage = $tokenStorage;
    }

    // Called when an error occurred and a log (record) is creating
    public function __invoke(array $record)
    {
        if (null !== $this->user) {
            // $this->user is your user's entity. Extract all pertinent data you would need.
In this case, getUserDetails method create a summary including alias, name, role, ...
            $record['extra']['user'] = $this->user->getUserDetails();
        }

        if (null !== $this->postParams) {
            // Includes all posted parameter when the error occurred
            $record['extra']['postParams'] = $this->postParams;
        }

        return $record;
    }

    public function onKernelRequest(GetResponseEvent $event)
    {
        // Retain post parameters sent (serialized) in order to log them if needed
        $postParams = $event->getRequest()->request->all();
        if(false === empty($postParams)){
            $this->postParams = serialize($postParams);
        }

        // Do not continue if user is not logged
        if (null === $token = $this->tokenStorage->getToken()) {
            return;
        }

        if (!is_object($user = $token->getUser())) {
            // e.g. anonymous authentication
            return;
        }
```

```
        // Retain the user entity in order to use it
        $this->user = $user;
    }
}
```

Lire Monolog: améliore tes logs en ligne: https://riptutorial.com/fr/symfony2/topic/6671/monolog--ameliore-tes-logs

Chapitre 15: Relations d'entité de doctrine

Examples

Un-à-plusieurs, bidirectionnel

Cette mise en correspondance bidirectionnelle nécessite l' `mappedBy` attribut sur l' `OneToMany` association et le `inversedBy` attribut sur l' `ManyToOne` association.

Une relation bidirectionnelle possède à la fois un côté propriétaire et un côté inverse . `OneToMany` relations `OneToMany` peuvent utiliser des tables de jointure, vous devez donc spécifier un côté propriétaire. L'association `OneToMany` est toujours l'inverse d'une association bidirectionnelle.

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="users")
 */
class User
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

     /**
     * @var string
     *
     * @ORM\Column(name="username", type="string", length=255)
     */
    protected $username;

    /**
     * @var Group|null
     *
     * @ORM\ManyToOne(targetEntity="AppBundle\Entity\Group", inversedBy="users")
     * @ORM\JoinColumn(name="group_id", referencedColumnName="id", nullable=true)
     */
    protected $group;

    /**
     * @param string $username
     * @param Group|null $group
     */
    public function __construct($username, Group $group = null)
    {
```

```
        $this->username = $username;
        $this->group = $group;
    }

    /**
     * Set username
     *
     * @param string $username
     */
    public function setUsername($username)
    {
        $this->username = $username;
    }

    /**
     * Get username
     *
     * @return string
     */
    public function getUsername()
    {
        return $this->username;
    }

    /**
     * @param Group|null $group
     */
    public function setGroup(Group $group = null)
    {
        if($this->group !== null) {
            $this->group->removeUser($this);
        }

        if ($group !== null) {
            $group->addUser($this);
        }

        $this->group = $group;
    }

    /**
     * Get group
     *
     * @return Group|null
     */
    public function getGroup()
    {
        return $this->group;
    }
}
```

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;
```

```
/**
 * @ORM\Entity
 * @ORM\Table(name="groups")
 */
class Group
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(name="name", type="string", length=255)
     */
    protected $name;

    /**
     * @ORM\OneToMany(targetEntity="AppBundle\Entity\User", mappedBy="group")
     */
    protected $users;

    /**
     * @param string $name
     */
    public function __construct($name)
    {
        $this->name = $name;
        $this->users = new ArrayCollection();
    }

    /**
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * @param string $name
     */
    public function setName($name)
    {
        $this->name = $name;
    }

    public function addUser(User $user)
    {
        if (!$this->getUsers()->contains($user)) {
            $this->getUsers()->add($user);
        }
    }

    public function removeUser(User $user)
    {
        if ($this->getUsers()->contains($user)) {
            $this->getUsers()->removeElement($user);
        }
    }
```

```
    public function getUsers()
    {
        return $this->users;
    }

    public function __toString()
    {
        return (string) $this->getName();
    }
}
```

Lire Relations d'entité de doctrine en ligne: https://riptutorial.com/fr/symfony2/topic/3043/relations-d-entite-de-doctrine

Chapitre 16: Répertoire d'entités de doctrine

Examples

Créer un nouveau référentiel

Vous pouvez créer un nouveau référentiel où vous voulez, mais il est recommandé de les créer dans un dossier distinct du `Repository` .

Bien que vous puissiez nommer le fichier de référentiel et la classe comme vous le souhaitez, il est recommandé de nommer le référentiel `EntityNameRepository` , pour que vous puissiez rapidement les trouver dans votre dossier.

Supposons que nous ayons une entité de `Project` , stockée dans `AppBundle\Entity` , cela ressemblerait à ceci:

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Project Entity - some information
 *
 * @ORM\Table(name="project")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\ProjectRepository")
 */
class Project
{
   // definition of the entity with attributes, getters, setter whatsoever
}

?>
```

La partie importante ici est la ligne `@ORM\Entity(repositoryClass="AppBundle\Repository\ProjectRepository")` , car elle connecte cette entité à la classe de référentiel donnée.

Vous devez également utiliser la classe `\Doctrine\ORM\Mapping` pour utiliser les options de mappage.

Le dépôt lui-même est assez simple

```
<?php

namespace AppBundle\Repository;

class ProjectRepository extends \Doctrine\ORM\EntityRepository
{
    public function getLastTenProjects()
```

```
    {
        // creates a QueryBuilder instance
        $qb = $this->_em->createQueryBuilder()
            ->select('p')
            ->from($this->_entityName, 'p')
            ->orderBy('p.id', 'DESC')
            ->setMaxResults(10)
        ;
        // uses the build query and gets the data from the Database
        return $qb->getQuery()->getResult();
    }
}

?>
```

Il est important de noter que la classe Repository doit étendre le `\Doctrine\ORM\EntityRepository` pour que cela fonctionne correctement. Vous pouvez maintenant ajouter autant de fonctions que vous le souhaitez pour différentes requêtes.

Fonction ExpressionBuilder IN ()

Si vous souhaitez utiliser la commande MySQL `IN()` dans QueryBuilder, vous pouvez le faire avec la fonction `in()` de la classe ExpressionBuilder .

```
// get an ExpressionBuilder instance, so that you
$expressionBulder = $this->_em->getExpressionBuilder();
$qb = $this->_em->createQueryBuilder()
->select('p')
->from($this->_entityName, 'p');
->where($expressionBuilder->in('p.id', array(1,2,3,4,5)));

return $qb->getQuery()->getResult();
```

Faire une requête avec une sous-requête

Par exemple, uniquement pour démontrer comment utiliser une instruction select de sous-requête dans une instruction select, supposons que nous trouvions tout utilisateur n'ayant pas encore compilé l'adresse (aucun enregistrement n'existe dans la table d'adresses):

```
 // get an ExpressionBuilder instance, so that you
$expr = $this->_em->getExpressionBuilder();

// create a subquery in order to take all address records for a specified user id
$sub = $this->_em->createQueryBuilder()
    ->select('a')
    ->from($this->_addressEntityName, 'a')
    ->where('a.user = u.id');


$qb = $this->_em->createQueryBuilder()
    ->select('u')
    ->from($this->_userEntityName, 'u')
    ->where($expr->not($expr->exists($sub->getDQL())));

return $qb->getQuery()->getResult();
```

Lire Répertoire d'entités de doctrine en ligne:
https://riptutorial.com/fr/symfony2/topic/6032/repertoire-d-entites-de-doctrine

Chapitre 17: Réponse

Paramètres

Paramètre	Détails
contenu de `string`	Le contenu de la réponse.
statut `integer`	Le code de statut HTTP.
`array` têtes de `array`	Tableau des en-têtes de réponse.

Examples

Usage simple

```
public function someAction(){

    // Action's code

    return new Response('<span>Hello world</span>');
}
```

Définir le code d'état

```
public function someAction(){

    // Action's code

    return new Response($error, 500);
}
```

Un autre exemple:

```
public function someAction(){

    // Action's code

    $response = new Response();
    $response->setStatusCode(500)
    $response->setContent($content);
    return $response;
}
```

Définir l'en-tête

Voir la liste des en- têtes http .

```
 public function someAction(){

    // Action's code
    $response = new Response();

    $response->headers->set('Content-Type', 'text/html');

    return $response;
}
```

JsonResponse

Retourne la réponse formulée JSON:

```
use Symfony\Component\HttpFoundation\JsonResponse;

public function someAction(){

    // Action's code

    $data = array(
        // Array data
    );

    return new JsonResponse($data);
}
```

Lire Réponse en ligne: https://riptutorial.com/fr/symfony2/topic/9218/reponse

Chapitre 18: Routage de base

Examples

Routage basé sur les annotations

Par défaut, tous les contrôleurs que vous générez avec la commande `generate:controller` de Symfony utiliseront les annotations Symfony pour le routage:

```
namespace AppBundle\Controller;

// You have to add a use statement for the annotation
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class AcmeController
{
    /**
     * @Route("/index")
     */
    public function indexAction()
    {
        // ...
    }
}
```

Pour que le framework puisse gérer ces routes, vous devez les importer dans votre `routing.yml` comme suit (notez le type d' `annotation`):

```
app:
    resource: "@AppBundle/Controller"
    type:     annotation
```

Itinéraires YAML

Au lieu des annotations, vous pouvez également spécifier vos itinéraires en tant que YAML:

```
app_index:
    path: /index
    defaults: { _controller: AppBundle:Acme:index }
```

Les mêmes options s'appliquent aux annotations et aux configurations YAML. Pour importer une configuration de routage YAML dans votre configuration de routage racine, vous n'avez pas besoin de spécifier un type:

```
app:
    prefix: /app
    resource: "@AppBundle/Resources/config/routing.yml"
```

Lire Routage de base en ligne: https://riptutorial.com/fr/symfony2/topic/5881/routage-de-base

Chapitre 19: Services Symfony

Examples

Comment déclarer, écrire et utiliser un service simple dans Symfony2

Déclaration de services:

```
# src/Acme/YourBundle/Resources/config/services.yml

services:
    my_service:
        class: Acme\YourBundle\Service\MyService
        arguments: ["@doctrine", "%some_parameter%", "@another_service"]
    another_service:
        class: Acme\YourBundle\Service\AnotherService
        arguments: []
```

Code de service:

```
<?php
namespace Acme\YourBundle\Service\Service;

class MyService
{
    /**
     * Constructor
     * You can had whatever you want to use in your service by dependency injection
     * @param $doctrine Doctrine
     * @param $some_parameter Some parameter defined in app/config/parameters.yml
     * @param $another_service Another service
     */
    public function __construct($doctrine, $some_parameter, $another_service)
    {
        $this->doctrine = $doctrine;
        $this->some_parameter = $some_parameter;
        $this->another_service = $another_service;
    }

    public function doMagic()
    {
        // Your code here
    }
}
```

Utilisez-le dans un contrôleur:

```
<?php

namespace Acme\YourBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Acme\YourBundle\Service\Service\MyService;
```

```
class MyController extends Controller
{
  /**
     * One action
     */
    public function oneAction(Request $request)
    {
        $myService = $this->get('my_service');
        $myService->doMagic();
        // ...
    }
}
```

Lire Services Symfony en ligne: https://riptutorial.com/fr/symfony2/topic/4587/services-symfony

Chapitre 20: Validation de formulaire

Examples

Validation de formulaire simple à l'aide de contraintes

Exemple d'action du contrôleur

```
use Symfony\Component\HttpFoundation\Request;

public function exampleAction(Request $request)
{
    /*
     * First you need object ready for validation.
     * You can create new object or load it from database.
     * You need to add some constraints for this object (next example)
     */
    $book = new Book();

    /*
     * Now create Form object.
     * You can do it manually using FormBuilder (below) or by creating
     * FormType class and passing it to builder.
     */
    $form = $this->createFormBuilder($book)
        ->add('title', TextType::class)
        ->add('pages', IntegerType::class)
        ->add('save', SubmitType::class, array('label' => 'Create Book'))
        ->getForm();

    /*
     * Handling Request by form.
     * All data submitted to form by POST(default) is mapped to
     * to object passed to FormBuilder ($book object)
     */
    $form->handleRequest($request);

    /*
     * Form Validation
     * In this step we check if form is submitted = data passed in POST
     * and is your object valid. Object is valid only if it pass form validation
     * in function isValid(). Validation constraints are loaded from config files
     * depending on format (annotations, YAML, XML etc).
     * IMPORTANT - object passed (book) is validated NOT form object
     * Function isValid() using Symfony Validator component.
     */
    if ($form->isSubmitted() && $form->isValid()) {
        /*
         * Now object is valid and you can save or update it
         * Original object ($book) passed into form builder has been updated
         * but you can also get variable by function getData:
         * $book = $form->getData();
         */

        // You can now redirect user to success page
        return $this->redirectToRoute('book_success_route');
```

```
    }

    /*
     * If form is not submitted you show empty form to user.
     * If validation fail then the form object contains list of FormErrors.
     * Form errors are displayed in form_row template (read about form templates)
     */
    return $this->render('book/create.html.twig', array(
        'form' => $form->createView(),
    ));
}
```

Exemple de contraintes pour objet

@Annotations

```
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Book
{


    /**
     * @Assert\Length(
     *      min = 2,
     *      max = 100,
     *      minMessage = "Book title must be at least {{ limit }} characters long",
     *      maxMessage = "Book title cannot be longer than {{ limit }} characters"
     * )
     */
    private $title;

    /**
     * @Assert\Range(
     *      min = 3,
     *      max = 10000,
     *      minMessage = "Book must have at least {{ limit }} pages",
     *      maxMessage = "Book cannot have more than {{ limit }} pages"
     * )
     */
    private $pages;

    // [...] getters/setters
}
```

@YAML

```
# src/AppBundle/Resources/config/validation.yml
AppBundle\Entity\Book:
    properties:
        title:
            - Length:
                min: 2
                max: 50
                minMessage: 'Book title must be at least {{ limit }} characters long'
                maxMessage: 'Book title cannot be longer than {{ limit }} characters'
```

```
        pages:
            - Range:
                min: 3
                max: 10000
                minMessage: Book must have at least {{ limit }} pages
                maxMessage: Book cannot have more than {{ limit }} pages
```

Référence aux contraintes de validation:
https://symfony.com/doc/current/reference/constraints.html

Validation du formulaire: http://symfony.com/doc/current/forms.html#form-validation

Lire Validation de formulaire en ligne: https://riptutorial.com/fr/symfony2/topic/7813/validation-de-formulaire

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec symfony2	althaus, COil, Community, Dheeraj, Hendra Huang, Jakub Zalas, karel, kix, mgh, Redjan Ymeraj, TRiNE, uddhab
2	Configuration pour les bundles	PumpkinSeed
3	Création de services Web avec Symfony 2.8	Barathon, Mathieu Dormeval
4	Créer des services Web avec Symfony en utilisant Rest	Barathon, Sunitrams'
5	Demande	A.L, Arkemlar, DOZ, Hermann Döppes, Mathieu Dormeval, mgh, Paweł Brzoski
6	Déploiement de Symfony2	sphinks
7	Envoi d'options à une classe de formulaire	geoB
8	Exemple de Symfony Twig Extenstion	Muhammad Taqi
9	Extensions Symfony Twig	Sand, Strabek
10	Gestion des pare-feu et de la sécurité Symfony	Mathieu Dormeval
11	Installez Symfony2 sur localhost	Barathon, KhorneHoly, sentenza
12	Le routage	afkplus, Bob, Dheeraj, Farahmand, Kid Binary, kix, pinch boi triggered af, Sam Janssens, Scott Flack, Stephan Vierkant, Stevan Tosic, Stony, tchap
13	Modèles de	DOZ

	conception Symfony	
14	Monolog: améliore tes logs	sdespont
15	Relations d'entité de doctrine	Federkun, palra
16	Répertoire d'entités de doctrine	doydoy44, KhorneHoly, Matteo
17	Réponse	CStff
18	Routage de base	kix
19	Services Symfony	DonCallisto, enricog, moins52
20	Validation de formulaire	Griva