eBook Gratuit

APPRENEZ symfony3

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

#symfony3

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: symfony3

It is an unofficial and free symfony3 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 symfony3.

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 symfony3

Remarques

Cette section fournit une vue d'ensemble de ce qu'est symfony3 et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans symfony3, et établir un lien avec les sujets connexes. La documentation de symfony3 étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Date de sortie
3.0.0	2015-11-30
3.1.0	2016-05-30
3.2.0	2016-11-30
3.2.5	2017-03-09
3.2.6	2017-03-10
3.2.7.	2017-04-05

Examples

3. Systèmes Windows

Vous devez ajouter php à votre variable d'environnement de chemin. Suivez ces étapes:

Windows 7 :

- Cliquez avec le bouton droit sur l'icône Poste de travail
- Cliquez sur Propriétés
- Cliquez sur Paramètres système avancés dans le menu de navigation de gauche.
- Cliquez sur l'onglet Avancé
- Cliquez sur le bouton Variables d'environnement
- Dans la section Variables système, sélectionnez Path (insensible à la casse) et cliquez sur le bouton Edit
- Ajoutez un point-virgule (;) à la fin de la chaîne, puis ajoutez le chemin complet du système de fichiers de votre installation PHP (par exemple, `C:\Program Files\PHP`)
- Continuez à cliquer sur OK jusqu'à ce que toutes les boîtes de dialogue aient disparu

- Fermez votre invite de commande et ouvrez-le à nouveau
- Trié

Windows 8 et 10

- Dans Rechercher, recherchez et sélectionnez ensuite: Système (Panneau de configuration)
- Cliquez sur le lien Paramètres système avancés.
- Cliquez sur Variables d'environnement.
- Dans la section Variables système, recherchez la variable d'environnement PATH et sélectionnez-la. Cliquez sur Modifier. Si la variable d'environnement PATH n'existe pas, cliquez sur Nouveau.
- Ajoutez le chemin complet du système de fichiers de votre installation PHP (par exemple, `C:\Program Files\PHP`)

Après cela, ouvrez votre console de commande et exécutez la commande suivante:

```
c:\> php –r "readfile('https://symfony.com/installer');" > symfony
```

Ensuite, déplacez le fichier symfony téléchargé dans le répertoire de votre projet et exécutez-le comme suit:

```
c:\> move symfony c:\projects
c:\projects\> php symfony
```

4. Création de l'application Symfony

Une fois le programme d'installation de Symfony disponible, créez votre première application Symfony avec la nouvelle commande:

```
# Linux, Mac OS X
$ symfony new my_project_name

# Windows
c:\> cd projects/
c:\projects\> php symfony new my_project_name
```

Cette commande peut être exécutée depuis n’importe où, pas nécessairement depuis le dossier `htdocs` .

Cette commande crée un nouveau répertoire appelé `my_project_name/` qui contient un nouveau projet basé sur la version stable la plus récente de Symfony disponible. De plus, le programme d'installation vérifie si votre système répond aux exigences techniques pour exécuter des applications Symfony. Sinon, vous verrez la liste des modifications nécessaires pour répondre à ces exigences.

1. Installation du programme d'installation de Symfony

> Le programme d'installation nécessite PHP 5.4 ou supérieur. Si vous utilisez toujours l'ancienne version de PHP 5.3, vous ne pouvez pas utiliser le programme d'installation

de Symfony. Lisez la section Création d'applications Symfony sans le programme d'installation pour savoir comment procéder. - source: http://symfony.com/doc/current/book/installation.html

5. Baser votre projet sur une version spécifique de Symfony

Si votre projet doit être basé sur une version spécifique de Symfony, utilisez le second argument facultatif de la nouvelle commande:

```
# use the most recent version in any Symfony branch
$ symfony new my_project_name 2.8
$ symfony new my_project_name 3.1

# use a specific Symfony version
$ symfony new my_project_name 2.8.1
$ symfony new my_project_name 3.0.2

# use a beta or RC version (useful for testing new Symfony versions)
$ symfony new my_project 3.0.0-BETA1
$ symfony new my_project 3.1.0-RC1
```

Le programme d'installation prend également en charge une version spéciale appelée lts qui installe la version la plus récente de Symfony LTS disponible:

```
$ symfony new my_project_name lts
```

Lisez le processus de publication de Symfony pour mieux comprendre pourquoi il existe plusieurs versions de Symfony et lesquelles utiliser pour vos projets.

Vous pouvez également créer des applications symfony sans le programme d'installation, mais ce n'était pas une bonne idée. Si vous voulez quand même, suivez le tutoriel original sur ce lien:

Oficial Symfony Docs, Configuration de Symfony sans le programme d'installation

2. Systèmes Linux et Mac OS X

Ouvrez votre console de commande et exécutez les commandes suivantes:

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

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
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>'
        );
    }
}
```

Créer une page

> Avant de continuer, assurez-vous d'avoir lu le chapitre Installation et pouvez accéder à votre nouvelle application Symfony dans le navigateur.

Supposons que vous vouliez créer une page - / lucky / number - qui génère un numéro de chance (bien, aléatoire) et l'imprime. Pour ce faire, créez une classe "Controller" et une méthode "controller" à l'intérieur de celle-ci qui seront exécutées lorsque quelqu'un accédera à / lucky / number

```
// src/AppBundle/Controller/LuckyController.php
namespace AppBundle\Controller;

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

class LuckyController
{
    /**
     * @Route("/lucky/number")
     */
    public function numberAction()
    {
        $number = rand(0, 100);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

Lire Démarrer avec symfony3 en ligne: https://riptutorial.com/fr/symfony3/topic/985/demarrer-avec-symfony3

Chapitre 2: Asset Management avec Assetic

Introduction

Lorsque vous utilisez le bundle Assetic, conformément à la documentation Symfony, veuillez prendre en compte les éléments suivants:

À partir de Symfony 2.8, Assetic n'est plus inclus par défaut dans Symfony Standard Edition. Avant d'utiliser l'une de ses fonctionnalités, installez AsseticBundle en exécutant cette commande de console dans votre projet:

$ composer nécessite symfony / assetic-bundle

Il y a d'autres étapes à suivre. Pour plus d'informations, rendez-vous sur: http://symfony.com/doc/current/assetic/asset_management.html

Paramètres

prénom	Exemple
Chemin	'static / images / logo / logo-default.png'

Remarques

Le dossier des actifs accessibles au public dans un projet Symfony3 standard est "/ web". Assetic utilise ce dossier en tant que dossier racine pour les actifs.

Examples

Créer un chemin relatif pour l'actif

```
<img src="{{ asset('static/images/logo-default.png') }}" alt="Logo"/>

<!--Generates path for the file "/web/static/images/logo-default.png" -->
```

Créer un chemin absolu pour l'actif

```
<img src="{{ app.request.getSchemeAndHttpHost() ~ asset('static/images/logo-default.png') }}"
alt="Logo"/>

<!--Generates path for the file "/web/static/images/logo-default.png" -->
```

Lire Asset Management avec Assetic en ligne: https://riptutorial.com/fr/symfony3/topic/6409/asset-management-avec-assetic

Chapitre 3: Configuration

Introduction

Exemples et bonnes pratiques pour configurer votre application Symfony qui ne figurent pas dans la documentation officielle.

Examples

Inclure tous les fichiers de configuration d'un répertoire

Après un certain temps, vous obtenez de nombreux éléments de configuration dans votre config.yml. Il peut vous rendre la configuration plus facile à lire si vous divisez votre configuration sur plusieurs fichiers. Vous pouvez facilement inclure tous les fichiers d'un répertoire de cette manière:

config.yml:

```
imports:
    - { resource: parameters.yml }
    - { resource: "includes/" }
```

Dans le `includes` répertoire que vous pouvez mettre par exemple doctrine.yml, swiftmailer.yml, etc.

Utiliser le nom de classe complet (FQCN) comme identifiant de service

Dans de nombreux exemples, vous trouverez un identifiant de service tel que "acme.demo.service.id" (une chaîne avec des points). Votre `services.yml` ressemblera à ceci:

```
services:
    acme.demo.service.id:
        class: Acme\DemoBundle\Services\DemoService
        arguments: ["@doctrine.orm.default_entity_manager", "@cache"]
```

Dans votre contrôleur, vous pouvez utiliser ce service:

```
$service = $this->get('acme.demo.service.id');
```

Bien qu'il n'y ait pas de problème avec cela, vous pouvez utiliser un nom de classe pleinement qualifié (FQCN) comme identifiant de service:

```
services:
    Acme\DemoBundle\Services\DemoService:
        class: Acme\DemoBundle\Services\DemoService
        arguments: ["@doctrine.orm.default_entity_manager", "@cache"]
```

Dans votre contrôleur, vous pouvez l'utiliser comme ceci:

```
use Acme\DemoBundle\Services\DemoService;
// ..
$this->get(DemoService::class);
```

Cela rend votre code plus compréhensible. Dans de nombreux cas, cela n'a aucun sens d'avoir un identifiant de service qui n'est pas seulement le nom de la classe.

A partir de Symfony 3.3, vous pouvez même supprimer l'attribut `class` si votre identifiant de service est un FQCN.

Aucune interface HTTP nécessaire?

Si votre application n'a besoin d'aucune interface HTTP (par exemple pour une application uniquement console), vous souhaiterez désactiver au moins `Twig` et `SensioFrameworkExtra`

Juste commenter ces lignes:

app / AppKernel.php

```
$bundles = [
//...
//    new Symfony\Bundle\TwigBundle\TwigBundle(),
//    new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
//...
if (in_array($this->getEnvironment(), ['dev', 'test'], true)) {
//...
//    $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
```

app / config / config.yml

```
framework:
#    ...
#    router:
#        resource: '%kernel.root_dir%/config/routing.yml'
#        strict_requirements: ~
#    ...
#    templating:
#        engines: ['twig']
#...
#twig:
#    debug: '%kernel.debug%'
#    strict_variables: '%kernel.debug%'
```

app / config / config_dev.yml

```
#framework:
#    router:
#        resource: '%kernel.root_dir%/config/routing_dev.yml'
#        strict_requirements: true
#    profiler: { only_exceptions: false }
```

```
#web_profiler:
#    toolbar: true
#    intercept_redirects: false
```

Vous pouvez également supprimer les exigences de fournisseur connexes de **composer.json** :

```
"sensio/framework-extra-bundle": "x.x.x",
"twig/twig": "x.x"
```

L'utilisation de Symfony dans un tel cas est discutable, mais au moins cela peut être temporaire.

Lire Configuration en ligne: https://riptutorial.com/fr/symfony3/topic/9175/configuration

Chapitre 4: Entités déclarantes

Examples

Déclaration d'une entité Symfony comme YAML

- AppBundle / Entity / Person.php

```
<?php

namespace AppBundle\Entity;

/**
 * Person
 */
class Person
{
    /**
     * @var int
     */
    private $id;

    /**
     * @var string
     */
    private $name;

    /**
     * @var int
     */
    private $age;


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

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

        return $this;
    }
```

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

    /**
     * Set age
     *
     * @param integer $age
     *
     * @return Person
     */
    public function setAge($age)
    {
        $this->age = $age;

        return $this;
    }

    /**
     * Get age
     *
     * @return int
     */
    public function getAge()
    {
        return $this->age;
    }
}
```

- AppBundle / Resources / config / doctrine / Person.orm.yml

```
AppBundle\Entity\Person:
    type: entity
    repositoryClass: AppBundle\Repository\PersonRepository
    table: persons
    id:
        id:
            type: integer
            id: true
            generator:
                strategy: AUTO
    fields:
        name:
            type: string
            length: 20
            nullable: false
            column: Name
            unique: true
        age:
            type: integer
            nullable: false
            column: Age
            unique: false
```

```
    lifecycleCallbacks: { }
```

- Créer la table

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

Ou utilisez la méthode recommandée (en supposant que doctrine-migrations-bundle soit déjà installé dans votre projet):

```
php bin/console doctrine:migrations:diff
php bin/console doctrine:migrations:migrate
```

- Créer l'entité automatiquement à partir de YAML

  doctrine php bin / console: generate: entités AppBundle

Déclaration d'une entité Symfony avec des annotations

- AppBundle / Entity / Person.php

```
<?php

namespace AppBundle\Entity;

use DoctrineORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

/**
 * @ORM\Entity
 * @ORM\Table(name="persons")
 * @ORM\Entity(repositoryClass="AppBundle\Entity\PersonRepository")
 * @UniqueEntity("name")
 */
class Person
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="string", name="name", length=20, nullable=false)
     */
    protected $name;

    /**
     * @ORM\Column(type="integer", name="age", nullable=false)
     */
    protected $age;

    public function getId()
    {
        return $this->id;
```

```
    }

    public function getName()
    {
        return $this->name;
    }

    public function setName($name)
    {
        $this->name = $name;
        return $this;
    }

    public function getAge()
    {
        return $this->age;
    }

    public function setAge($age)
    {
        $this->age = $age;
        return $this;
    }
}
```

- Générer l'entité

```
php bin/console doctrine:generate:entities AppBundle/Person
```

- Pour vider les instructions SQL à l'écran

```
php bin/console doctrine:schema:update --dump-sql
```

- Créer la table

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

Ou utilisez la méthode recommandée (en supposant que doctrine-migrations-bundle soit déjà installé dans votre projet):

```
php bin/console doctrine:migrations:diff
php bin/console doctrine:migrations:migrate
```

Lire Entités déclarantes en ligne: https://riptutorial.com/fr/symfony3/topic/4443/entites-declarantes

Chapitre 5: Essai

Examples

Tests simples dans Symfony3

Test de l'unité

Les tests unitaires permettent de s'assurer que votre code n'a pas d'erreur de syntaxe et de tester la logique de votre code pour fonctionner comme prévu. Exemple rapide:

src / AppBundle / Calculatrice / BillCalculator.php

```
<?php

namespace AppBundle\Calculator;

use AppBundle\Calculator\TaxCalculator;

class BillCalculator
{
    private $taxCalculator;

    public function __construct(TaxCalculator $taxCalculator)
    {
        $this->taxCalculator = $taxCalculator;
    }

    public function calculate($products)
    {
        $totalPrice = 0;
        foreach ($products as $product) {
            $totalPrice += $product['price'];
        }
        $tax = $this->taxCalculator->calculate($totalPrice);

        return $totalPrice + $tax;
    }
}
```

src / AppBundle / Calculatrice / TaxCalculator.php

```
<?php

namespace AppBundle\Calculator;

class TaxCalculator
{
    public function calculate($price)
    {
        return $price * 0.1; // for example the tax is 10%
    }
}
```

tests / AppBundle / Calculatrice / BillCalculatorTest.php

```
<?php

namespace Tests\AppBundle\Calculator;

class BillCalculatorTest extends \PHPUnit_Framework_TestCase
{
    public function testCalculate()
    {
        $products = [
            [
                'name' => 'A',
                'price' => 100,
            ],
            [
                'name' => 'B',
                'price' => 200,
            ],
        ];
        $taxCalculator = $this->getMock(\AppBundle\Calculator\TaxCalculator::class);

        // I expect my BillCalculator to call $taxCalculator->calculate once
        // with 300 as the parameter
        $taxCalculator->expects($this->once())->method('calculate')->with(300)-
>willReturn(30);

        $billCalculator = new BillCalculator($taxCalculator);
        $price = $billCalculator->calculate($products);

        $this->assertEquals(330, $price);
    }
}
```

J'ai testé ma classe BillCalculator afin de m'assurer que mon BillCalculator retournera le prix total des produits + 10% de taxe. Dans le test unitaire, nous créons notre propre cas de test. Dans ce test, je fournis 2 produits (les prix sont 100 et 200), la taxe sera donc de 10% = 30. Je m'attends à ce que TaxCalculator retourne 30, de sorte que le prix total sera de 300 + 30 = 330.

Test fonctionnel

Les tests fonctionnels sont utilisés pour tester les entrées et les sorties. Avec l'entrée donnée, je m'attendais à une sortie sans tester le processus pour créer la sortie. (ceci est différent avec le test unitaire car dans le test unitaire, nous testons le flux de code). Exemple rapide:

```
namespace Tests\AppBundle;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ApplicationAvailabilityFunctionalTest extends WebTestCase
{
    /**
     * @dataProvider urlProvider
     */
    public function testPageIsSuccessful($url)
    {
        $client = self::createClient();
```

```
        $client->request('GET', $url);

        $this->assertTrue($client->getResponse()->isSuccessful());
    }

    public function urlProvider()
    {
        return array(
            array('/'),
            array('/posts'),
            array('/post/fixture-post-1'),
            array('/blog/category/fixture-category'),
            array('/archives'),
            // ...
        );
    }
}
```

J'ai testé mon contrôleur pour que mon contrôleur renvoie une réponse 200 au lieu de 400 (Introuvable) ou 500 (Erreur interne du serveur) avec l'URL indiquée.

Les références:

- http://symfony.com/doc/current/best_practices/tests.html
- http://symfony.com/doc/current/book/testing.html

Lire Essai en ligne: https://riptutorial.com/fr/symfony3/topic/2881/essai

Chapitre 6: Formulaires dynamiques

Examples

Comment étendre ChoiceType, EntityType et DocumentType pour charger des choix avec AJAX.

Dans Symfony, le ChoiceType intégré (et l'EntityType ou DocumentType l'extendant), fonctionnent de manière fondamentale avec une liste de choix constante.

Si vous voulez le faire fonctionner avec les appels ajax, vous devez les modifier pour accepter les choix supplémentaires cumulés.

- **Comment commencer avec une liste de choix vide?**

  Lorsque vous créez votre formulaire, définissez simplement l'option `choices` sur un `array()` vide `array()` :

  ```
  namespace AppBundle\Form;

  use Symfony\Component\Form\AbstractType;

  use Symfony\Component\Form\FormBuilderInterface;

  use Symfony\Component\Form\Extension\Core\Type\ChoiceType;

  class FooType extends AbstractType
  {
      public function buildForm(FormBuilderInterface $builder, array $options)
      {
          $builder
              ->add('tag', ChoiceType::class, array('choices'=>array()));
      }
  }
  ```

  Vous obtiendrez donc une entrée de sélection vide, sans choix. Cette solution fonctionne pour ChoiceType et tous ses enfants (EntityType, DocumentType, ...).

- **Comment accepter les nouveaux choix soumis** :

  Pour accepter les nouveaux choix, vous devez les rendre disponibles dans la liste de sélection des champs de formulaire. Vous pouvez modifier votre champ de formulaire en fonction des données soumises avec l'événement FormEvent :: PRE_SUBMIT.

  Cet exemple montre comment le faire avec un ChoiceType de base:

  ```
  namespace AppBundle\Form;

  use Symfony\Component\Form\AbstractType;
  ```

```
use Symfony\Component\Form\FormBuilderInterface;

use Symfony\Component\Form\Extension\Core\Type\ChoiceType;

class FooType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('tag', ChoiceType::class, array('choices'=>array()))
        ;

        $builder->addEventListener(
            FormEvents::PRE_SUBMIT,
            function(FormEvent $event){
                // Get the parent form
                $form = $event->getForm();

                // Get the data for the choice field
                $data = $event->getData()['tag'];

                // Collect the new choices
                $choices = array();

                if(is_array($data)){
                    foreach($data as $choice){
                        $choices[$choice] = $choice;
                    }
                }
                else{
                    $choices[$data] = $data;
                }

                // Add the field again, with the new choices :
                $form->add('tag', ChoiceType::class, array('choices'=>$choices));
            }
        );
    }
}
```

Vos choix soumis sont maintenant des choix autorisés et la validation intégrée de Symfony ChoiceType ne les rejettera plus.

Si vous souhaitez faire la même chose avec un enfant ChoiceType (EntityType, DocumentType, ...), vous devez injecter entityManager ou le documentManager et effectuer la transformation de données lors du remplissage des nouveaux choix.

Remplir un champ de sélection en fonction de la valeur autre.

Cet exemple montre comment modifier les choix autorisés dans un champ de sélection de sous-catégorie en fonction de la valeur du champ de sélection de catégorie. Pour ce faire, vous devez rendre vos choix de sous-catégories dynamiques à la fois pour le client et pour le serveur.

1. Rendre le formulaire dynamique du côté client pour les interactions d'affichage / utilisateur

Exemple de formulaire dynamique côté client (en utilisant Javascript / JQuery):

```
$('#category').change(function(){
    switch($(this).val()){
        case '1': // If category == '1'
            var choice = {
                'choice1_1':'1_1',
                'choice1_2':'1_2',
                'choice1_3':'1_3',
            };
        break;
        case '2': // If category == '2'
            var choice = {
                'choice2_1':'2_1',
                'choice2_2':'2_2',
                'choice2_3':'2_3',
            };
        break;
        case '3': // If category == '3'
            var choice = {
                'choice3_1':'3_1',
                'choice3_2':'3_2',
                'choice3_3':'3_3',
            };
        break;
    }

    var $subCategorySelect = $('#subCategory');

    $subCategorySelect.empty();
    $.each(choice, function(key, value) {
        $subCategorySelect.append($('<option></option>')).attr('value',value).text(key);
    });
});
```

Bien sûr, vous pouvez choisir parmi un appel AJAX. Ce n'est pas le but de cet exemple.

2. Rendre le formulaire dynamique du côté serveur pour l'initialisation / validation

Exemple de formulaire dynamique côté serveur:

```
namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;

use Symfony\Component\Form\Extension\Core\Type\ChoiceType;

use Symfony\Component\Form\FormEvent;
use Symfony\Component\Form\FormEvents;

class MyBaseFormType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
```

```
            $builder
                ->add('category',ChoiceType::class,array('choices'=>array(
                        'choice1'=>'1',
                        'choice2'=>'2',
                        'choice3'=>'3',
                    )))
            ;

            $addSubCategoryListener = function(FormEvent $event){
                $form = $event->getForm();
                $data = $event->getData();

                switch($data['category']){
                    case '1': // If category == '1'
                        $choices = array(
                            'choice1_1'=>'1_1',
                            'choice1_2'=>'1_2',
                            'choice1_3'=>'1_3',
                        );
                    break;
                    case '2': // If category == '2'
                        $choices = array(
                            'choice2_1'=>'2_1',
                            'choice2_2'=>'2_2',
                            'choice2_3'=>'2_3',
                        );
                    break;
                    case '3': // If category == '3'
                        $choices = array(
                            'choice3_1'=>'3_1',
                            'choice3_2'=>'3_2',
                            'choice3_3'=>'3_3',
                        );
                    break;
                }

                $form->add('subCategory',ChoiceType::class,array('choices'=>$choices));
            };

            // This listener will adapt the form with the data passed to the form during
construction :
            $builder->addEventListener(FormEvents::PRE_SET_DATA, $addSubCategoryListener);

            // This listener will adapt the form with the submitted data :
            $builder->addEventListener(FormEvents::PRE_SUBMIT, $addSubCategoryListener);
        }
    }
```

Lire Formulaires dynamiques en ligne: https://riptutorial.com/fr/symfony3/topic/5855/formulaires-dynamiques

Chapitre 7: Le routage

Introduction

Une route est comme mapper une URL à une action (fonction) dans une classe de contrôleur. La rubrique suivante se concentrera sur la création d'itinéraires, en passant des paramètres à la classe Controller via une route, à l'aide de YAML ou d'une annotation.

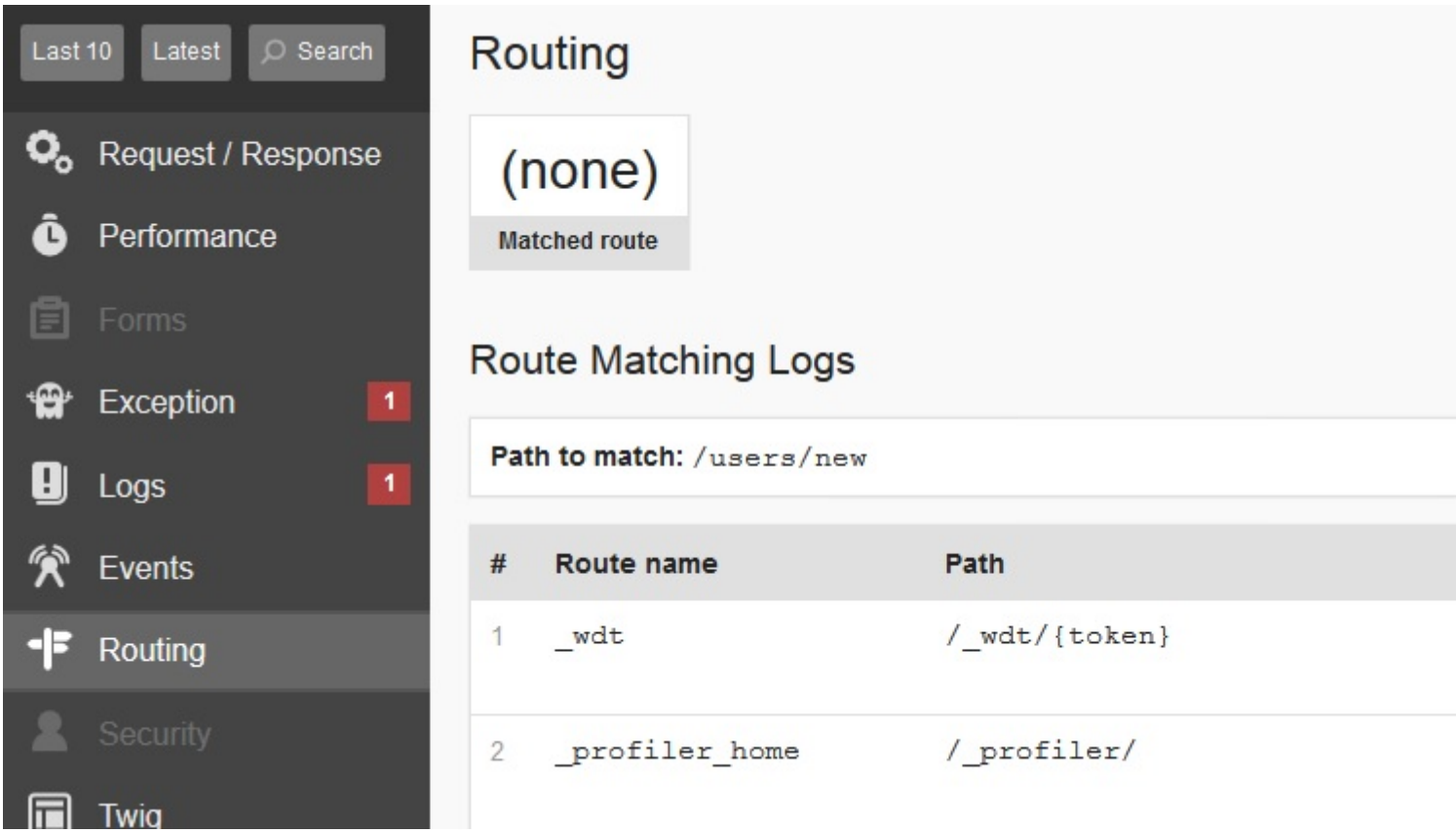
Remarques

Il est utile de voir ce qui est généré par le framework Symfony, celui-ci fournit des outils pour regarder toutes les routes d'une application spécifique.

A partir du document Symfony , utilisez (dans un shell):

```
php bin/console debug:router
```

De plus, vous pouvez voir toutes les informations de routes pertinentes dans le profileur Framework, dans le menu de routage:



Examples

Routage à l'aide de YAML

La configuration du routage est incluse dans votre fichier `app/config/config.yml` , par défaut dans le fichier `app/config/routing.yml` .

De là, vous pouvez créer un lien vers votre propre configuration de routage dans un bundle

```
# app/config/routing.yml

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

Il peut également contenir plusieurs routes globales d'application.

Dans votre propre ensemble, vous pouvez configurer des itinéraires qui servent deux objectifs:

- Correspondance à une demande, de sorte que l'action correcte soit appelée pour la demande.
- Générer une URL à partir des paramètres name et route.

Voici un exemple de configuration d'itinéraire YAML:

```
# src/AppBundle/Resources/config/routing.yml

my_page:
    path: /application/content/page/{parameter}
    defaults:
        _controller: AppBundle:Default:myPage
        parameter: 42
    requirements:
        parameter: '\d+'
    methods: [ GET, PUT ]
    condition: "request.headers.get('User-Agent') matches '/firefox/i'"
```

La route s'appelle `my_page` et appelle `myPageAction` du `DefaultController` dans `AppBundle` à la demande. Il a un paramètre, nommé `parameter` avec une valeur par défaut. La valeur n'est valide que lorsqu'elle correspond à l'expression rationnelle `\d+` . Pour cette route, seules les méthodes HTTP `GET` et `PUT` sont acceptées. La `condition` est une expression dans l'exemple, la route ne correspondra pas, sauf si l'en-tête User-Agent correspond à Firefox. Vous pouvez effectuer toute logique complexe dont vous avez besoin dans l'expression en tirant parti de deux variables transmises à l'expression: `context` (RequestContext) et `request` (Symfony Request).

Un itinéraire généré avec la valeur du paramètre 10 peut ressembler à `/application/content/page/10` .

Routage à l'aide d'annotations

La configuration du routage est incluse dans votre fichier `app/config/config.yml` , par défaut dans le fichier `app/config/routing.yml` .

De là, vous pouvez accéder aux contrôleurs dotés d'une configuration de routage annotée:

```
# app/config/routing.yml
```

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

Dans votre propre ensemble, vous pouvez configurer des itinéraires qui servent deux objectifs:

- Correspondance à une demande, de sorte que l'action correcte soit appelée pour la demande.
- Générer une URL à partir des paramètres name et route.

Voici un exemple de configuration de route annotée:

```
// src/AppBundle/Controller/DefaultController.php

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

/**
 * @Route("/application")
 */
class DefaultController extends Controller {
    /**
     * @Route("/content/page/{parameter}",
     *        name="my_page",
     *        requirements={"parameter" = "\d+"},
     *        defaults={"parameter" = 42})
     * @Method({"GET", "PUT"})
     */
    public function myPageAction($parameter)
    {
        // ...
    }
}
```

Le contrôleur est annoté avec un itinéraire de *préfixe* , de sorte que tout itinéraire configuré dans ce contrôleur sera ajouté en préfixe.

L'itinéraire configuré s'appelle `my_page` et appelle la fonction `myPageAction` à la demande. Il a un paramètre, nommé `parameter` avec une valeur par défaut. La valeur n'est valide que lorsqu'elle correspond à l'expression rationnelle `\d+` . Pour cette route, seules les méthodes HTTP `GET` et `PUT` sont acceptées.

Notez que le paramètre est injecté dans l'action en tant que paramètre de fonction.

Un itinéraire généré avec la valeur du paramètre 10 peut ressembler à `/application/content/page/10` .

Routes de repos puissantes

Traditionnellement, vous pouvez utiliser le routage pour mapper la demande avec le composant de routage qui traitait les paramètres de demande et de réponse par le composant HttpFoundation .

De plus, un paramètre de route personnalisé peut être créé en utilisant FOSRestBundle pour étendre les fonctionnalités par défaut du composant de routage.

Ceci est utile pour créer des routes REST, ce qui est très utile pour spécifier comment transférer des données structurées comme un fichier XML ou json dans une requête (et une réponse).

Voir FOSRestBundle Doc pour plus d'informations, et en particulier cette annotation:

```
use FOS\RestBundle\Controller\Annotations\FileParam;

/**
 * @FileParam(
 *   name="",
 *   key=null,
 *   requirements={},
 *   default=null,
 *   description="",
 *   strict=true,
 *   nullable=false,
 *   image=false
 * )
 */
```

Lire Le routage en ligne: https://riptutorial.com/fr/symfony3/topic/2287/le-routage

Chapitre 8: Répartiteur d'événements

Syntaxe

- $ dispatcher-> dispatch (string $ eventName, Event $ event);
- $ dispatcher-> addListener (string $ eventName, callable $ listener, int $ priority = 0);
- $ dispatcher-> addSubscriber (EventSubscriberInterface $ abonné);

Remarques

- Il est souvent préférable d'utiliser une seule instance de EventDispatcher dans votre application que vous injectez dans les objets devant déclencher des événements.
- Il est recommandé d'avoir un seul emplacement où vous gérez la configuration et d'ajouter des écouteurs d'événement à votre EventDispatcher. Le framework Symfony utilise le conteneur d'injection de dépendances.
- Ces modèles vous permettront de changer facilement vos écouteurs d'événement sans avoir à modifier le code d'un module qui distribue des événements.
- Le découplage de la répartition des événements de la configuration de l'écouteur d'événements est ce qui rend le Symfony EventDispatcher si puissant
- EventDispatcher vous aide à satisfaire le principe ouvert / fermé.

Examples

Démarrage rapide du répartiteur d'événements

```
use Symfony\Component\EventDispatcher\EventDispatcher;
use Symfony\Component\EventDispatcher\Event;
use Symfony\Component\EventDispatcher\GenericEvent;

// you may store this in a dependency injection container for use as a service
$dispatcher = new EventDispatcher();

// you can attach listeners to specific events directly with any callable
$dispatcher->addListener('an.event.occurred', function(Event $event) {
    // process $event
});

// somewhere in your system, an event happens
$data = // some important object
$event = new GenericEvent($data, ['more' => 'event information']);

// dispatch the event
// our listener on "an.event.occurred" above will be called with $event
// we could attach many more listeners to this event, and they too would be called
$dispatcher->dispatch('an.event.occurred', $event);
```

Abonnés aux événements

```
use Symfony\Component\EventDispatcher\EventDispatcher;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\EventDispatcher\Event;

$dispatcher = new EventDispatcher();

// you can attach event subscribers, which allow a single object to subscribe
// to many events at once
$dispatcher->addSubscriber(new class implements EventSubscriberInterface {
    public static function getSubscribedEvents()
    {
        // here we subscribe our class methods to listen to various events
        return [
            // when anything fires a "an.event.occurred" call "onEventOccurred"
            'an.event.occurred' => 'onEventOccurred',
            // an array of listeners subscribes multiple methods to one event
            'another.event.happened' => ['whenAnotherHappened', 'sendEmail'],
        ];
    }

    function onEventOccurred(Event $event) {
        // process $event
    }

    function whenAnotherHappened(Event $event) {
        // process $event
    }

    function sendEmail(Event $event) {
        // process $event
    }
});
```

Lire Répartiteur d'événements en ligne: https://riptutorial.com/fr/symfony3/topic/5609/repartiteur-d-evenements

Chapitre 9: Travailler avec des services Web

Examples

Rest Rest API

J'ai déjà écrit de la documentation sur ce site afin de décrire comment créer des services Web sur Symfony

Je vais écrire à nouveau un tutoriel pour la version symfony> = 3.

Nous pensons que nous avons un serveur Web installé sur une version configurée de Symfony Framework . Vous devez avoir un compositeur (gestionnaire de paquets php) également installé.

Pour simplifier, si vous avez un composeur installé, tapez ceci dans une invite de terminal / commande:

```
composer create-project symfony/framework-standard-edition example "3.1.*"
```

Cela créera un nouveau répertoire appelé "exemple" dans le répertoire actuel, avec une installation standard du framework symfony.

Vous devez installer ces 2 bundles: JMSSerializer Bundle (étend le sérialiseur de composant de structure) et FOSRest Bundle (étend le routage et les contrôleurs de composants de structure ...)

Vous pouvez le faire comme ceci (dans le répertoire exemple):

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

N'oubliez pas de les activer dans AppKernel!

Ici vous ne pouvez pas utiliser:

```
composer create-project gimler/symfony-rest-edition --stability=dev example
```

Parce qu'il est basé sur la version Symfony 2.8.

Tout d'abord, créez votre propre ensemble ("Example") (dans le répertoire Symfony):

```
php bin/console generate:bundle
php bin/console doctrine:create:database
```

Imaginez que nous voulions créer CRUD (Create / Read / Update / Delete) de cette entité StackOverFlower:

```
# 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
```

Configurez votre ensemble:

```
#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 }
```

Générez cette entité:

```
php bin/console doctrine:generate:entity StackOverFlower
php bin/console doctrine:schema:update --force
```

Faire un contrôleur:

```
#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 (0 === count($errors_list)) {

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

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

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

Ne me dites pas que c'est un gros contrôleur, c'est pour l'exemple !!!

Créez votre modèle:

```
#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 voir un exemple complet de ce code sur mon compte GitHub , une branche avec plus de routes réelles ...

Voici un exemple très simple, ne le laissez pas dans un environnement de production, vous devez protéger votre api avec apikey !!!

Un exemple futur, peut-être?

Lire Travailler avec des services Web en ligne: https://riptutorial.com/fr/symfony3/topic/6972/travailler-avec-des-services-web

Chapitre 10: Validation

Remarques

En fait, la validation de formulaire est basée sur un composant nommé " **Validator Component** ".

Vous pouvez souvent utiliser le service dédié si vous n'avez pas à afficher un formulaire dans un modèle. Comme les API. Vous pouvez valider les données de la même manière, comme ceci:

Par exemple, *basé sur le doc de symfony* :

```
$validator = $this->get('validator');
$errors = $validator->validate($author);

if (count($errors) > 0) {
    /*
     * Uses a __toString method on the $errors variable which is a
     * ConstraintViolationList object. This gives us a nice string
     * for debugging.
     */
    $errorsString = (string) $errors;
}
```

Examples

Validation Symfony à l'aide d'annotations

- Activer la validation à l'aide d'annotations dans le fichier `app/config/config.yml`

```
framework:
    validation: { enable_annotations: true }
```

- Créez un `AppBundle/Entity` entité dans `AppBundle/Entity` . Les validations sont effectuées avec les annotations `@Assert` .

```
<?php
# AppBundle/Entity/Car.php


namespace AppBundle\Entity;


use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * Car
 *
 * @ORM\Table(name="cars")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\CarRepository")
 */
```

```
class Car
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=50)
     * @Assert\NotBlank(message="Please provide a name")
     * @Assert\Length(
     *     min=3,
     *     max=50,
     *     minMessage="The name must be at least 3 characters long",
     *     maxMessage="The name cannot be longer than 50 characters"
     * )
     * @Assert\Regex(
     *     pattern="/^[A-Za-z]+$/",
     *     message="Only letters allowed"
     * )
     */
    private $name;

    /**
     * @var string
     *
     * @ORM\Column(name="number", type="integer")
     * @Assert\NotBlank(message="Please provide a number")
     * @Assert\Length(
     *     min=1,
     *     max=3,
     *     minMessage="The number field must contain at least one number",
     *     maxMessage="The number field must contain maximum 3 numbers"
     * )
     * @Assert\Regex(
     *     pattern="/^[0-9]+$/",
     *     message="Only numbers allowed"
     * )
     */
    private $number;


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

    /**
     * Set name
     *
```

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

        return $this;
    }

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

    /**
     * Set number
     *
     * @param integer $number
     *
     * @return Car
     */
    public function setNumber($number)
    {
        $this->number = $number;

        return $this;
    }

    /**
     * Get number
     *
     * @return integer
     */
    public function getNumber()
    {
        return $this->number;
    }
}
```

- Créez un nouveau formulaire dans le `AppBundle/Form` .

```
<?php
# AppBundle/Form/CarType.php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\IntegerType;
```

```
class CarType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', TextType::class, ['label'=>'Name'])
            ->add('number', IntegerType::class, ['label'=>'Number'])
        ;
    }

    /**
     * @param OptionsResolver $resolver
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'AppBundle\Entity\Car'
        ));
    }

    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        // TODO: Implement setDefaultOptions() method.
    }

    public function getName()
    {
        return 'car_form';
    }
}
```

- Créez une nouvelle route et une nouvelle méthode d'action dans `AppBundle/Controller/DefaultController.php` . La route sera également déclarée avec des annotations, alors assurez-vous d'avoir importé cette route dans le fichier de route principal (`app/config/routing.yml`).

```
<?php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use AppBundle\Entity\Car;
use AppBundle\Form\CarType;

class DefaultController extends Controller
{
    /**
     * @Route("/car", name="app_car")
     */
    public function carAction(Request $request)
    {
        $car = new Car();
```

```
        $form = $this->createForm(
            CarType::class,
            $car,
            [
                'action' => $this->generateUrl('app_car'),
                'method'=>'POST',
                'attr'=>[
                    'id'=>'form_car',
                    'class'=>'car_form'
                ]
            ]
        );

        $form->handleRequest($request);

        return $this->render(
            'AppBundle:Default:car.html.twig',[
                'form'=>$form->createView()
            ]
        );
    }
}
```

- Créez la vue dans `AppBundle/Resources/views/Default/car.html.twig` .

```
{% extends '::base.html.twig' %}

{% block body %}
    {{ form_start(form, {'attr': {'novalidate':'novalidate'}}) }}
    {{ form_row(form.name) }}
    {{ form_row(form.number) }}
    <button type="submit">Go</button>
    {{ form_end(form) }}
{% endblock %}
```

- Lancez le serveur intégré de Symfony (`php bin/console server:run`) et accédez à la route `127.0.0.1:8000/car` dans votre navigateur. Il devrait y avoir un formulaire composé de deux zones de saisie et d'un bouton d'envoi. Si vous appuyez sur le bouton d'envoi sans saisir de données dans les zones de saisie, les messages d'erreur s'afficheront.

Validation Symfony avec YAML

- Créez un `AppBundle/Entity` entité dans `AppBundle/Entity` . Vous pouvez le faire manuellement ou en utilisant la commande `php bin/console doctrine:generate:entity` Symfony `php bin/console doctrine:generate:entity` et en renseignant les informations requises à chaque étape. Vous devez spécifier l'option `yml` au niveau du `Configuration format (yml, xml, php or annotation)` .

```
<?php
# AppBundle/Entity/Person.php

namespace AppBundle\Entity;

/**
 * Person
```

```
 */
class Person
{
    /**
     * @var int
     */
    private $id;

    /**
     * @var string
     */
    private $name;

    /**
     * @var int
     */
    private $age;


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

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

        return $this;
    }

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

    /**
     * Set age
     *
     * @param integer $age
     *
     * @return Person
     */
    public function setAge($age)
```

```
    {
        $this->age = $age;

        return $this;
    }

    /**
     * Get age
     *
     * @return int
     */
    public function getAge()
    {
        return $this->age;
    }
}
```

- Créez les informations de mappage d'entité pour la classe d'entité. Si vous utilisez la commande `php bin/console doctrine:generate:entity` Symfony `php bin/console doctrine:generate:entity` , le code suivant sera généré automatiquement. Sinon, si vous n'utilisez pas la commande, vous pouvez créer le code suivant à la main.

```
# AppBundle/Resources/config/doctrine/Person.orm.yml

AppBundle\Entity\Person:
type: entity
table: persons
repositoryClass: AppBundle\Repository\PersonRepository
id:
    id:
        type: integer
        id: true
        generator:
            strategy: AUTO
fields:
    name:
        type: string
        length: '50'
    age:
        type: integer
lifecycleCallbacks: {  }
```

- Créez la validation pour la classe Entity.

```
# AppBundle/Resources/config/validation/person.yml

AppBundle\Entity\Person:
    properties:
        name:
            - NotBlank:
                message: "Name is required"
            - Length:
                min: 3
                max: 50
                minMessage: "Please use at least 3 chars"
                maxMessage: "Please use max 50 chars"
            - Regex:
                pattern: "/^[A-Za-z]+$/"
```

```
                message: "Please use only letters"
        age:
            - NotBlank:
                message: "Age is required"
            - Length:
                min: 1
                max: 3
                minMessage: "The age must have at least 1 number in length"
                maxMessage: "The age must have max 3 numbers in length"
            - Regex:
                pattern: "/^[0-9]+$/"
                message: "Please use only numbers"
```

- Créez un nouveau formulaire dans le `AppBundle/Form` .

```
<?php
# AppBundle/Form/PersonType.php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\IntegerType;

class PersonType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name', TextType::class, ['label'=>'Name'])
            ->add('age', IntegerType::class, ['label'=>'Age'])
        ;
    }

    /**
     * @param OptionsResolver $resolver
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'AppBundle\Entity\Person'
        ));
    }

    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        // TODO: Implement setDefaultOptions() method.
    }

    public function getName()
    {
        return 'person_form';
    }
```

```
}
```

- Créez une nouvelle route dans `AppBundle/Resources/config/routing.yml`

```
app_person:
    path: /person
    defaults: { _controller: AppBundle:Default:person }
```

- Créez maintenant une nouvelle méthode d'action pour cette route.

```
<?php
# AppBundle/Controller/DefaultController.php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use AppBundle\Entity\Person;
use AppBundle\Form\PersonType;

class DefaultController extends Controller
{
    public function personAction(Request $request)
    {
        $person = new Person();

        $form = $this->createForm(
            PersonType::class,
            $person,
            [
                'action' => $this->generateUrl('app_person'),
                'method'=>'POST',
                'attr'=>[
                    'id'=>'form_person',
                    'class'=>'person_form'
                ]
            ]
        );

        $form->handleRequest($request);

        return $this->render(
            'AppBundle:Default:person.html.twig', [
                'form'=>$form->createView()
            ]
        );
    }
}
```

- Créez la vue dans `AppBundle/Resources/views/Default/person.html.twig`

```
{% extends '::base.html.twig' %}

{% block body %}
    {{ form_start(form, {'attr': {'novalidate':'novalidate'}}) }}
        {{ form_row(form.name) }}
        {{ form_row(form.age) }}
```

```
        <button type="submit">Go</button>
    {{ form_end(form) }}
{% endblock %}
```

- Lancez le serveur intégré de Symfony (`php bin/console server:run`) et accédez à la route `127.0.0.1:8000/person` dans votre navigateur. Il devrait y avoir un formulaire composé de deux zones de saisie et d'un bouton d'envoi. Si vous appuyez sur le bouton d'envoi sans saisir de données dans les zones de saisie, les messages d'erreur s'afficheront.

Lire Validation en ligne: https://riptutorial.com/fr/symfony3/topic/6457/validation

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec symfony3	Community, insertusernamehere, Paweł Kolanowski, Raphael Schubert, Tartare2240, TRiNE, uddhab, YoannFleuryDev
2	Asset Management avec Assetic	Aaron Belchamber, Orlando
3	Configuration	Pierre de LESPINAY, Stephan Vierkant
4	Entités déclarantes	Dan Costinel, janek1, Nicodemuz, rubenj
5	Essai	Hendra Huang, Pierre de LESPINAY
6	Formulaires dynamiques	Alsatian
7	Le routage	Alvin Bunk, Anjana Silva, Hidde, Mathieu Dormeval, Matteo, Renato Mefi
8	Répartiteur d'événements	Chris Tickner
9	Travailler avec des services Web	Mathieu Dormeval, Pascal, Pierre de LESPINAY
10	Validation	Dan Costinel, Mathieu Dormeval