FREE eBook

LEARNING symfony2

Free unaffiliated eBook created from
Stack Overflow contributors.

#symfony2

Table of Contents

About

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

Chapter 1: Getting started with symfony2

Remarks

This section provides an overview of what Symfony2 is and why a developer might want to use it.

It should also mention any large subjects within Symfony2 and link out to the related topics. Since the Documentation for Symfony2 is new, you may need to create initial versions of those related topics.

Versions

The latest stable version during the time of writing is **Symfony 3.1** which will be maintained until end of July 2017.

Symfony has **Long Term Support** versions which are maintained for a total of 4 years (3 years for bug fixes, 1 additional year for security bug fixes)

A **Standard Minor Version** is maintained for an eight month period for bug fixes, and for a fourteen month period for security issue fixes.

Long Term Support versions:

```
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)
```

Starting from 3.X version, minor versions will be limited to 5 and the last minor version will be LTS.

Syfmony has dual maintenance mode, releasing minor versions ever six months once in May and once in November. Major version are released every two years, meaning there will be one year time period to move form the previous major version to the latest one, giving user a choice between the latest features of standard version or an LTS version that is supported for bug fixes.

Symfony maintains strict backward compatibility, anything that breaks BC is done in the next major version. A feature implementation that is an improvement but breaks BC is kept alongside the old implementation which will be deprecated

Read more about versions and development process in detail from the official documentation [here][1]

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

Examples

Installation or Setup

Symfony Framework - built with symfony components, is one of the leading PHP framework used to create robust websites and web applications.

Symfony can be installed quickly through two recommended ways.

1. The official documentaion recommends to install the framework through the ***Symfony Installer*** which is a tiny php application that is installed once on the local system that helps in downloading the framework and setting up the configuration of the framework. Symfony Installer requires PHP 5.4 or higher. To install on legacy php version use Composer.
2. Through the PHP dependency manager ***Composer***

Installing through Symfony Installer

On Linux/Mac OS X run the following commands:

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

On Windows move to the project directory and run the following command:

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

symfony project can be created by running `symfony new my_project [2.8]` on Linux/Mac OS X

On Windows `php symfony new my_project [2.8]`

or alternatively `symfony new my_project lts` will use the latest long-term support version of Symfony.

Installation through Composer

- Download Composer
- Use Composer's `create-project` command to download Symfony

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

Excellent detailed official documentation here

Running the Symfony application

For starting symfony internal web server (available since PHP 5.4), go to the project directory and execute this command:

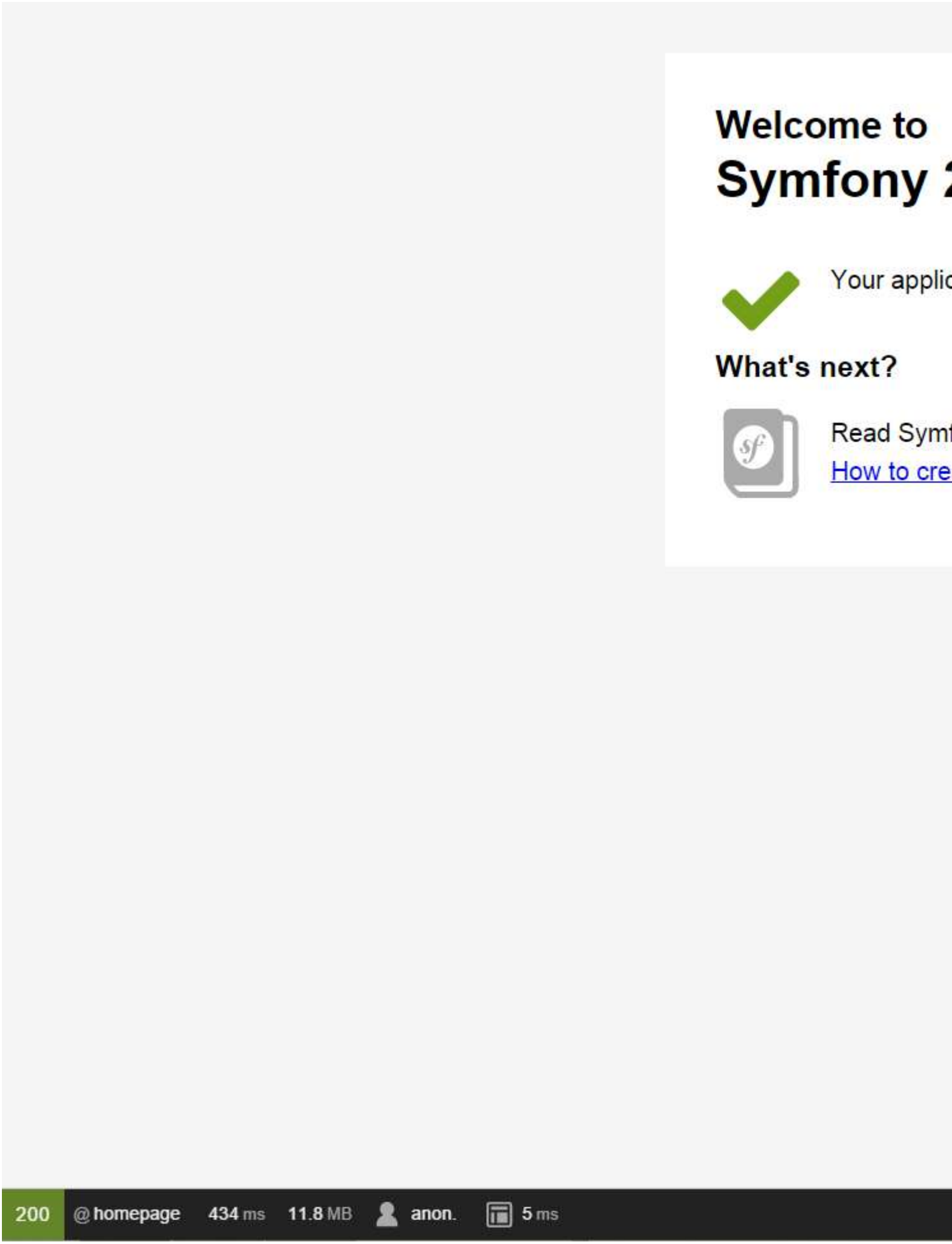
for symfony<=2.8

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

and for symfony >=3.0

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

This starts the web server at `localhost:8000` in the background that serves your Symfony application. Then, open your browser and access the `http://localhost:8000/` URL to see the Symfony welcome page:


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

Simplest example in Symfony

1. Install symfony correctly as guided above.
2. Start symfony server if you are not installed in www directory.
3. Ensure http://localhost:8000 is working if symfony server is used.
4. Now it is ready to play with simplest example.
5. Add following code in a new file **/src/AppBundle/Controller/MyController.php** in symfony installation dir.
6. Test the example by visiting http://localhost:8000/hello
   (If you are not using Symfony's built-in http server, visit http://localhost/(symfony-dir)/web/app_dev.php/hello)
7. That's all. Next: use twig to render the response.

```
<?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>'
        );
    }
}
```

NOTE: All controller classes should have ends with word '**Controller**' and methods related to routs should ends with word '**Action**'. Further, In which controller your Actions are placed is not relevant until you define a rout prefix for the controller.

Installing and Configuring Symfony

Checking Requirements

Run `bin/symfony_requirements` for checking symfony requirements and php cli setting. Install all packages that needed to run a symfony project. Setting your php.ini for example setting timezone and short_open_tag. Setting both php.ini for your php webserver (eg: /etc/php/apache2/php.ini) and php cli (eg: /etc/php/cli/php.ini). Open http://localhost/config.php for checking php webserver setting. If everything has passed, you are ready to run your symfony project.

Running Project

Run `composer install` to install all depedencies. Then setting up permission for `var/cache`, `var/logs` and `var/sessions`.

Detailed official documentation here

Read Getting started with symfony2 online: https://riptutorial.com/symfony2/topic/909/getting-started-with-symfony2

Chapter 2: Basic routing

Examples

Annotation-based routing

By default, all the controllers you generate with Symfony's built-in `generate:controller` command will make use of Symfony annotations for routing:

```
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()
    {
        // ...
    }
}
```

In order for the framework to handle these routes, you need to import them in your `routing.yml` as follows (notice the `annotation` type):

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

YAML routes

Instead of annotations, you can also specify your routes as YAML:

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

The same options apply to both annotations and YAML configurations. To import a YAML routing configuration in your root routing configuration, you don't need to specify a type:

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

Read Basic routing online: https://riptutorial.com/symfony2/topic/5881/basic-routing

Chapter 3: Configuration for own bundles

Introduction

This is a description how you can create configuration for you own bundle in the /app/config/config.{yml,xml}

Examples

Create configuration in the app/config/config.yml

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

This is a basic example for create configuration in yml format, for following the yml format you can take deeper configuration.

Set the config in the created bundle

For instance you have a bundle, which generated by the symfony console. In this case in the DependencyInjection/Configuration.php you have to insert your configuration representation:

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

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

Basically in the DependencyInjection/AmazingserviceExtension.php you will see the following lines:

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

It is not enough for getting the configuration in the Srevices. You have to take it into the container.

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

In this case the config in the container, so if your Service getting the container as a constructor

parameter:

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

Then you can get the configuration in the service with the following code, where the configuration will be an associative array:

```
private $config;

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

Read Configuration for own bundles online:
https://riptutorial.com/symfony2/topic/8153/configuration-for-own-bundles

Chapter 4: Create web services with Symfony using Rest

Examples

Using Symfony REST Edition

Symfony REST Edition is a fully-functional **Symfony2** application that you can use as the skeleton for your new applications.

Available on Github

It comes pre-configured with the following bundles:

Bundle	Description
FrameworkBundle	The core Symfony framework bundle.
SensioFrameworkExtraBundle	Adds several enhancements such as template and routing annotation capability.
DoctrineBundle	Adds support for the Doctrine ORM.
TwigBundle	Adds support for the Twig templating engine.
SecurityBundle	Adds security by integrating Symfony's security component.
SwiftmailerBundle	Adds support for Swiftmailer, a library for sending emails.
MonologBundle	Adds support for Monolog, a logging library.
AsseticBundle	Adds support for Assetic, an asset processing library.
WebProfilerBundle (in dev/test environment)	Adds profiling functionality and the web debug toolbar.
SensioDistributionBundle (in dev/test environment)	Adds functionality for configuring and working with **Symfony** distributions.
SensioGeneratorBundle (in dev/test environment)	Adds code generation capabilities.
AcmeDemoBundle (in dev/test environment)	A demo bundle with some example code.

Bundle	Description
FOSRestBundle	The FOSRestBundle adds REST functionality.
FOSHttpCacheBundle	This bundle offers tools to improve HTTP caching with **Symfony2**.
NelmioApiDocBundle	Adds API documentation features.
BazingaHateoasBundle	Adds HATEOAS support.
HautelookTemplatedUriBundle	Adds Templated URIs (RFC 6570) support.
BazingaRestExtraBundle	BazingaRestExtraBundle Provides extra features for REST APIs built using Symfony2.

Read Create web services with Symfony using Rest online:
https://riptutorial.com/symfony2/topic/6020/create-web-services-with-symfony-using-rest

Chapter 5: Creating Web-Services with Symfony 2.8

Examples

Work with RESTFul API

REpresentational State Transfer (REST) is an architectural style used for web development, introduced and defined in 2000 by Roy Fielding.

See it on wiki : REST wiki

It's based on HTTP protocol (HTTP on Wiki), HTTP requests (GET, POST, PATCH, DELETE...) / responses codes (404, 400, 200, 201, 500...) and bodies structure.

This is a great way to expose your datas to an another system on Internet.

Imagine you want to make a RESTFul api to manage your StackOverFlower (User) on your local database.

Let's make the example !

Symfony 2.8 framework

1. Web server :

You must install and configure a web server on your local machine, see Wamp or Lamp or Mamp : You must have a recent version of PHP (**!!! Symfony requirements !!!**)

2. Php cli and Composer :

You must configure PHP cli (varying on our system), type this "PHP cli [OS-NAME] how-to" in our friend Google! You must install composer, see Composer install

3. Symfony :

You must install Symfony 2.8 (with composer, it's the better way), open a terminal (or cmd on windows) and go to your web server path.

Symfony 2 works with the one of the better structure types: Bundles. All are Bundles on Symfony! We can test it above.

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

Go to the tree structure an see : Symfony 2.8 is installed on "example" directory.

4. FOSRest (for FriendsOfSymfony) on JMSSerializer Bundle :

You must install these two Bundles :

JMSSerializer (Install) :

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

FosRestBundle (Install) :

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

Don't forget to activate them in AppKernel.php !

5. Basic configuration :

Make your own "Example" bundle and create the database.

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

Go to the bottom of your Symfony 2.8 application configuration file, and paste it :

```
#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 }
```

Make your doctrine directory ("example/src/ExampleBundle/Entity") and resource file ("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
```

Generate Entity and Update Schema :

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

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

Make a default controller :

```
#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);
    }
}
```

Make your default Twig view :

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

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

You have just made your first RESTFul API!

You can test it on : http://your-server-name/your-symfony-path/app_dev.php/stackoverflower/new/test.

As you can see in the databse, a new user has been created with the name "test".

You can get the list of stackoverflower on : http://your-server-name/your-symfony-path/app_dev.php/stackoverflowers

You have a full example on my github account of this example : Git Hub example, at the "master" branch this example, and on the "real-routes" branche an example with more appropriate URL (like POST and DELETE).

See you later for an example with SOAP!

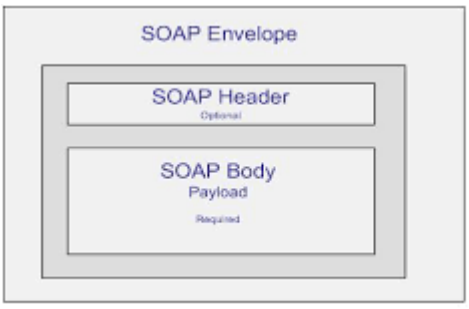
Best Regards,

Mathieu

Work with SOAP API

SOAP (Simple Access Object Protocol) is XML based, like XML-RPC, *is ancestor*, with file called **WSDL**, what describe the method to be exposed.

This protocol is often based with **SOAP-Enveloppe**, a **SOAP-Body**, and alternatively **SOAP-Header**, the data is envelopped in a structure and be interpreted as the same way from different langages.



For more information, see : SOAP on wiki

As described above, the most important to describe your web service is the **WSDL** file, see : WSDL explanation on wiki

The basic of the work will be to define what is exposed on your SOAP API, your class and your business process will be automatically handled by the basic PHP SOAPServer class. You still

need the code!

Let's see how the file is constructed :

1. Service : Set the API URI and what will be associated.
2. Binding : It define the operations associated with the service
3. Operations : Some methods you want to expose to the Web
4. PortTypes : Define queries and responses
5. Requests and Responses : what you expect input and output
6. Messages : what formt you expect (parameters) on each IO, they can be simple (string, integer, float...) or complex type (structured format)

With this basic information, you can achieve all API you want.

Imagine you want to make a SOAP api to manage your StackOverFlower (User) on your local database.

Let's make the example !

Install Web server, Php cli, Composer, Symfony 2.8, create a new Bundle "ExampleBundle" and build the schema like described above.

Before we start to build our business logic, we had to know what to expose of our controller. This job is done by using the WSDL. This is an example of a good syntax of an 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>
```

We must take this on your web symfony directory (in soap subdirectory, and name this "stackoverflower.wsdl").

Really inspired from WSDl Example. You can validate that with an Online WSDl Validator

After this, we can make our basic service and controller, inspirated from SOAP Symfony 2.8 Doc.

Service, that is handled by 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";
  }
}
```

Configure this service :

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

As you can see, we inject the Doctrine Entity Manger as a dependency because we have to use this to CRUD StackOverFlower Object.

Controller, that expose the service object :

```
#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;
  }
}
```

To learn more about services, see :Service container on Symfony doc

The route :

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

The basic Twig Template :

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

We have made your first SOAP API with Symfony 2.8 !

Before you expose it, we have to test !!

In your StackOverFlowerController, add this :

```
  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);
```

The 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 }
```

You can type this in your browser :

1. getList
2. new
3. edit
4. delete

This is a very basic example of a non secured API with SOAP, I can do an example of a secured example behind a api key authentication later.

That all folks...

Mathieu

Read Creating Web-Services with Symfony 2.8 online:
https://riptutorial.com/symfony2/topic/6355/creating-web-services-with-symfony-2-8

Chapter 6: Deployment of Symfony2

Examples

Steps to move Symfony 2 project to hosting manually

It depends on kind of hosting that you have:

1. If you have SSH console, then you can do it on hosting after step 2, if you haven't then do it locally: run command

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

2. Suppose you have on you hosting folders `youdomain/public_html`, so in `public_html` must be located all web files. So you must upload all from Symfony project (folders: `app`, `src`, `vendors`, `bin`; files: `deps`, `deps.lock`), except for folder `web` in folder `youdomain`. Everything from folder `web` upload to folder `public_html`.

3. Check CHMOD for folders `app/cache` and `app/logs`, there should be write access.

4. If there is no file .htaccess in public_html, then create it and add such code in it: https://raw.github.com/symfony/symfony-standard/master/web/.htaccess

5. Now you should use `youdomain.com/index` instead of `youdomain.com/app_dev.php/index`, that you use locally. If a site still did not works, you can open file `web/config.php` and find a code where a check for IP performs, you find there only IP `127.0.0.1`. Add your current IP to this list and upload new config on the server. Then you can open path `yourdomain/config.php` and check what's wrong. If `config.php` shows that everything ok, but it still didn't work, you can enable `app_dev.php` to debug: open `app/app_dev.php` and your IP the same way as in `config.php`. Now you can run scripts as locally using `app_dev.php`.

Read Deployment of Symfony2 online: https://riptutorial.com/symfony2/topic/6039/deployment-of-symfony2

Chapter 7: Doctrine Entity Relationships

Examples

One-To-Many, Bidirectional

This bidirectional mapping requires the `mappedBy` attribute on the `OneToMany` association and the `inversedBy` attribute on the `ManyToOne` association.

A bidirectional relationship has both an owning and inverse side. `OneToMany` relationships can use join tables, so you have to specify an owning side. The `OneToMany` association is always the inverse side of a bidirectional association.

```
<?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();
    }
}
```

Read Doctrine Entity Relationships online: https://riptutorial.com/symfony2/topic/3043/doctrine-entity-relationships

Chapter 8: Doctrine Entity Repository

Examples

Creating a new Repository

You can create a new Repository where ever you want, but it's recommended to create them in a seperate `Repository` folder.

While you could name the Repository file and class as you wish, it's recommended to name the Repository `EntityNameRepository`, to that you could quickly find those in your folder.

Let's assume we have an `Project` Entity, stored in `AppBundle\Entity`, it would look like this:

```
<?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
}

?>
```

The important part here is the line `@ORM\Entity(repositoryClass="AppBundle\Repository\ProjectRepository")`, because it connects this Entity with the given Repository class.

Also you need to use the `\Doctrine\ORM\Mapping` class to use the mapping options.

The repository itself is pretty 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();
    }
}

?>
```

It's important to notice that the Repository class must extend the `\Doctrine\ORM\EntityRepository`, so that it can work properly. Now you can add as many functions for different querys as you want.

ExpressionBuilder IN() function

If you want to use the MySQL command `IN()` in the QueryBuilder, you can do it with the `in()` function of the ExpressionBuilder class.

```
// 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();
```

Make a Query with a Sub-Query

As example, only for demonstrate HOW-TO use a subquery select statement inside a select statement, suppose we what to find all user that not yet have compile the address (no records exists in the address table):

```
 // 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();
```

Read Doctrine Entity Repository online: https://riptutorial.com/symfony2/topic/6032/doctrine-entity-repository

Chapter 9: Form Validation

Examples

Simple Form Validation using constraints

Example Controller action

```
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(),
    ));
}
```

Example constraints for object

@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
```

Validation Constraints Reference: https://symfony.com/doc/current/reference/constraints.html

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

Read Form Validation online: https://riptutorial.com/symfony2/topic/7813/form-validation

Chapter 10: Install Symfony2 on localhost

Examples

Using the command prompt

The best way to install and configure a Symfony2 project is described in the official documentation as follows:

Mac OS X / Linux

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

Windows

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

Then you could use the Symfony binary to build the right scaffolding:

```
$ symfony new my_project
```

Using composer over console

Given you've already installed composer up and running and it's accessible globally, you can simply create new Symfony projects as stated in the official documentation.

Now you can create a new Symfony project with composer:

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

This will create the project in the current directory you're in.

If you want to create the project with a specific version of Symfony, you can add a parameter with the version number:

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

Hint: If you're thinking that composer won't do anything, add the `-vvv` flag to the command. That way, you'll receive detailed information about what composer is doing right now.

Read Install Symfony2 on localhost online: https://riptutorial.com/symfony2/topic/6340/install-symfony2-on-localhost

Chapter 11: Managing the Symfony firewalls and security

Examples

Managing Security

Security was a part of the dark side of the symfony documentation, it has a dedicated component named **Security Component**.

This component is configured in the **security.yml** file of the main application project.

The default configuration is like this one :

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

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

        default:
            anonymous: ~
```

You can define specific **Firewalls** to restrict access to some URL to specific **Roles** based on a hierarchy for your **Users** that are defined by a **Provider** and **Encoders** that manage the password security.

For example, if you want to create a custom **Provider**, from your database engine, you can define you **security.yml** like this :

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

This is detailled in the symfony Documentation : How to define a custom UserProvider and from the database or against LDAP for example.

After that, you can defined **firewall** to restrict some URL based on your custom user provider (security.yml) explicitely like this :

```
firewalls:
    secured_area:
```

```
        pattern: ^/admin
```

Or with **access control** :

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

See more detailled documentation here.

The best way to manage user is to use FosUserBundle that extends some framework functionnalities.

Read Managing the Symfony firewalls and security online: https://riptutorial.com/symfony2/topic/7486/managing-the-symfony-firewalls-and-security

Chapter 12: Monolog : improve your logs

Examples

Add user's details and posted parameters sent to logs

Logs are very important. Recreate an error context can be sometimes very painful due to the lack of information about how and when the error occurred.

This example shows:

- How to add user's data in the error logs
- How to add post parameters sent when an error occurred
- How to use WebProcessor in order to add all data regarding the request like :
    - url
    - ip
    - http method
    - server
    - referrer

Service Configuration

```
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 }
```

Service code

```
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;
    }
```

```
}
```

Read Monolog : improve your logs online: https://riptutorial.com/symfony2/topic/6671/monolog---improve-your-logs

Chapter 13: Request

Remarks

API documentation links (master):

- Request
- RequestStack

Request object contains several significant data like current Locale and matched Controller. You can use and manage them by HttpKernel events. For reliable understanding of Request-Responce live cycle read this HttpKernel Component doc page (very helpful!).

Examples

Access to Request in a Controller

```
<?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();
   }
}
```

Note that injected Request object applies to current request (it may or may not equal to master request).

Access to Request in a Twig or PHP template.

In Twig template, Request object is available at

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

When you want display request method in Twig, try this:

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

In PHP template

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

Read Request online: https://riptutorial.com/symfony2/topic/4870/request

Chapter 14: Response

Parameters

Parameter	Details
`string` content	The response content.
`integer` status	The HTTP status code.
`array` headers	Array of response headers.

Examples

Simple usage

```
public function someAction(){

    // Action's code

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

Set status code

```
public function someAction(){

    // Action's code

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

Another example:

```
public function someAction(){

    // Action's code

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

Set header

See list of http headers.

```
 public function someAction(){

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

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

    return $response;
}
```

JsonResponse

Return JSON formated response:

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

public function someAction(){

    // Action's code

    $data = array(
        // Array data
    );

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

Read Response online: https://riptutorial.com/symfony2/topic/9218/response

Chapter 15: Routing

Examples

Return a 404 response

404 responses are returned when a resource is not found on the server, in Symfony this status can be created by throwing a `NotFoundHttpException` exception. To avoid an extra `use` statement inside a controller use the `createNotFoundException()` provided by the `Controller` class

```
<?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));
    }

}
```

Multiple Routes

In Symfony it's possible to define multiple routes for one action. This can be very helpful if you have functions that do the same but have different parameters.

```
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));
```

```
    }
}
```

POST request redirect

When you are in a **controllerAction** And have a **POST request coming in**, but want to **redirect it, to a different route**, while still **maintaining the POST method and the request object**, you can use the following:

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

Code 307 here preserves the request method.

Subdomain-based routing

Subdomain-based routing can be handled in Symfony using `host` parameter. For example, `_locale` parameter can be used as subdomain value.

Assuming

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

parameters are defined in `parameters.yml` config file, route would be:

```
/**
 * @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%"}
 * )
 */
```

From this point router can handle URI's such as `http://de.somedomain.com`. Second `@Route` annotation can be used as a fallback for default locale and void subdomain, `http://somedomain.com`.

Symfony routes using Routing.yml

```
    profile_user_profile:
        path:    /profile/{id}
        defaults: { _controller: ProfileBundle:Profile:profile }
        requirements:
```

```
            id: \d+
        methods: [get, delete]
```

If you decide to use Routing.yml instead of Annotations You can get better view of **all routes** and easier to search and find one.

> It is up to you to chose between **Routing.yml** and **Annotations**. You can use both for different routes but this is not best solution.

Annotation `@Route()` equivalent is:

```
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));
    }
}
```

Read Routing online: https://riptutorial.com/symfony2/topic/1691/routing

Chapter 16: Sending options to a form class

Syntax

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

Parameters

Parameter	Definition
HouseholdType::class	custom form class for the Household entity
$household	an instance of the Household entity (usually created by `$household = new Household();`)
$formOptions	an array of user-defined options to be passed to the form class, e.g., `$formOptions = array('foo' => 'bar');`

Remarks

When you create a form class the form fields are added in the `public function buildForm(FormBuilderInterface $builder, array $options) {...}` function. The `$options` parameter includes a set of default options such as `attr` and `label`. To enable your custom options to be available in the form class the options need to be initialized in `configureOptions(OptionsResolver $resolver)`

So for our real-world example:

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

Examples

A real-world example from a Household controller

Background: The Household entity includes a set of options, each of which is an entity that is managed in an admin backend. Each option has a boolean `enabled` flag. If a previously enabled option is set to disabled it will need to be persisted in later Household edits, but cannot be edited away. To accomplish this the field definition in the form class will display field as a disabled choice field if the option has `enabled = false` (but is persisted because the submit button triggers a

javascript that removes the `disabled` attribute.) The field definition also prevents disabled options from being displayed.

The form class then needs to know, for a given Household entity, which of its options have been disabled. A service has been defined that returns an array of the names of option entities that have been disabled. That array is `$disabledOptions`.

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

How the custom options are used in the form class

```
        ->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');
                }
            },
        ))
```

Housing entity

```
/**
 * 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;
    }
}
```

Read Sending options to a form class online: https://riptutorial.com/symfony2/topic/7237/sending-options-to-a-form-class

Chapter 17: Symfony Design Patterns

Examples

Dependency Injection pattern

Imagine you have a class manager to manages sending mails (be called MailManager).

In this, you have to log mails that are sent. A good solution is to transform the MailManager class into a `service` and then inject class for creating logs (`Monolog` for example) into the MailManager creating a service.

To do this :

1- Declare future MailManager class as service (in services.yml)

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

2- Inject logger existant service using `argument` method

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

3- Create MailManager class

```
<?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- Call MailManager in a Controller for example

```
<?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);

    }
}
```

Read Symfony Design Patterns online: https://riptutorial.com/symfony2/topic/6289/symfony-design-patterns

Chapter 18: Symfony Services

Examples

How to declare, write and use a simple service in Symfony2

Services declaration :

```
# 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: []
```

Service code :

```
<?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
    }
}
```

Use it in a controller :

```
<?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();
        // ...
    }
}
```

Read Symfony Services online: https://riptutorial.com/symfony2/topic/4587/symfony-services

Chapter 19: Symfony Twig Extensions

Examples

A Simple Twig Extension -- Symfony 2.8

Before creating any extension, always check if it has already **been implemented**.

The first thing one would have to do is define the extension class which will house the twig filters and/or functions.

```
<?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);
    }
}
```

One then alerts the service container of the newly created twig extension.

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

With this you have all you need to be able to use your newly created twig filter or function in your twig templates

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

Make a number short e.g. 1 000 -> 1k, 1 000 000 -> 1M 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';
    }
}
```

Add your extension to services.yml

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

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

Use it in TWIG

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

Read Symfony Twig Extensions online: https://riptutorial.com/symfony2/topic/6000/symfony-twig-extensions

Chapter 20: Symfony Twig Extenstion Example

Parameters

Parameters	Description
$doctrine	doctrine object that we pass from the service.

Examples

Symfony Twig Extension Basic Example

In this example I define two custom function. 1 - countryFilter function get's the country short code as input and return the country full name. 2 - _countPrinterTasks is used to calculate the no of tasks assigned to a particular user.

```
<?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';
    }
}
```

To call it from the Twig, We just have to used as below;

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

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

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

And Declare this extenstion as a service in your `bundle/resource/config/service.yml` file.

```
services:

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

Read Symfony Twig Extenstion Example online:
https://riptutorial.com/symfony2/topic/5891/symfony-twig-extenstion-example

Credits

S. No	Chapters	Contributors
1	Getting started with symfony2	althaus, COil, Community, Dheeraj, Hendra Huang, Jakub Zalas , karel, kix, mgh, Redjan Ymeraj, TRiNE, uddhab
2	Basic routing	kix
3	Configuration for own bundles	PumpkinSeed
4	Create web services with Symfony using Rest	Barathon, Sunitrams'
5	Creating Web-Services with Symfony 2.8	Barathon, Mathieu Dormeval
6	Deployment of Symfony2	sphinks
7	Doctrine Entity Relationships	Federkun, palra
8	Doctrine Entity Repository	doydoy44, KhorneHoly, Matteo
9	Form Validation	Griva
10	Install Symfony2 on localhost	Barathon, KhorneHoly, sentenza
11	Managing the Symfony firewalls and security	Mathieu Dormeval
12	Monolog : improve your logs	sdespont
13	Request	A.L, Arkemlar, DOZ, Hermann Döppes, Mathieu Dormeval, mgh , Paweł Brzoski
14	Response	CStff
15	Routing	afkplus, Bob, Dheeraj, Farahmand, Kid Binary, kix, pinch boi

		triggered af, Sam Janssens, Scott Flack, Stephan Vierkant, Stevan Tosic, Stony, tchap
16	Sending options to a form class	geoB
17	Symfony Design Patterns	DOZ
18	Symfony Services	DonCallisto, enricog, moins52
19	Symfony Twig Extensions	Sand, Strabek
20	Symfony Twig Extenstion Example	Muhammad Taqi