EBook Gratis

APRENDIZAJE symfony2

Free unaffiliated eBook created from
Stack Overflow contributors.

#symfony2

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con symfony2

Observaciones

Esta sección proporciona una descripción general de qué es Symfony2 y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de Symfony2 y vincularlo a los temas relacionados. Dado que la Documentación para Symfony2 es nueva, es posible que necesites crear versiones iniciales de esos temas relacionados.

Versiones

La última versión estable durante el tiempo de escritura es **Symfony 3.1,** que se mantendrá hasta finales de julio de 2017.

Symfony tiene versiones de **soporte a largo plazo** que se mantienen por un total de 4 años (3 años para correcciones de errores, 1 año adicional para correcciones de errores de seguridad)

Una **versión menor estándar** se mantiene durante un período de ocho meses para las correcciones de errores, y durante un período de catorce meses para las correcciones de problemas de seguridad.

Versiones de soporte a largo plazo:

```
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)
```

A partir de la versión 3.X, las versiones menores estarán limitadas a 5 y la última versión menor será LTS.

Syfmony tiene doble modo de mantenimiento, lanzando versiones menores cada seis meses una vez en mayo y una vez en noviembre. Las versiones principales se lanzan cada dos años, lo que significa que habrá un período de tiempo de un año para pasar de la versión principal anterior a la más reciente, dando a los usuarios la posibilidad de elegir entre las características más recientes de la versión estándar o una versión LTS compatible con la corrección de errores.

Symfony mantiene una estricta compatibilidad con versiones anteriores, todo lo que se rompe BC se realiza en la próxima versión principal. Una implementación de características que es una mejora pero que se rompe BC se mantiene junto con la implementación antigua que quedará obsoleta

Lea más sobre las versiones y el proceso de desarrollo en detalle en la documentación oficial [aquí] [1]

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

Examples

Instalación o configuración

Symfony Framework, construido con componentes de Symfony, es uno de los principales framework de PHP utilizado para crear sitios web robustos y aplicaciones web.

Symfony se puede instalar rápidamente de dos maneras recomendadas.

1. La documentación oficial recomienda instalar el marco a través del ***instalador de Symfony,*** que es una pequeña aplicación php que se instala una vez en el sistema local y que ayuda a descargar el marco y a configurar la configuración del marco. El instalador de Symfony requiere PHP 5.4 o superior. Para instalar en la versión heredada de php use Composer.
2. A través del gestor de dependencias PHP ***Composer***

Instalación a través del instalador de Symfony

En Linux / Mac OS X ejecuta los siguientes comandos:

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

En Windows, vaya al directorio del proyecto y ejecute el siguiente comando:

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

El proyecto symfony se puede crear ejecutando el `symfony new my_project [2.8]` en Linux / Mac OS X

En Windows `php symfony new my_project [2.8]`

o alternativamente, el `symfony new my_project lts` usará la última versión de soporte a largo plazo de Symfony.

Instalación a través del compositor

- Descargar compositor

- Usa el comando `create-project` Composer para descargar Symfony

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

Excelente documentación oficial detallada aquí.

Ejecutando la aplicación Symfony

Para iniciar el servidor web interno de Symfony (disponible desde PHP 5.4), ve al directorio del proyecto y ejecuta este comando:

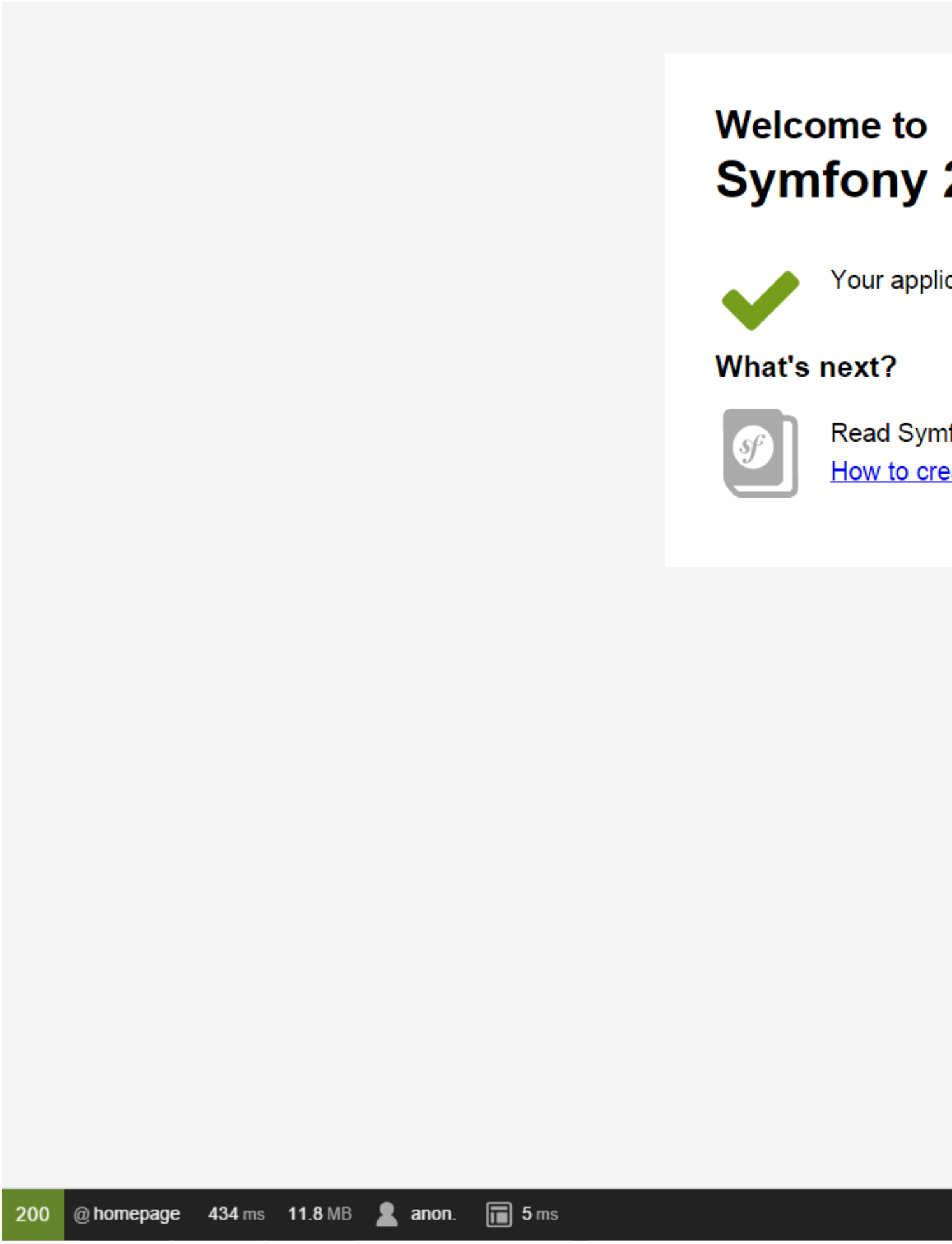
para Symfony <= 2.8

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

y para Symfony> = 3.0

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

Esto inicia el servidor web en `localhost:8000` en el fondo que sirve a su aplicación Symfony. Luego, abra su navegador y acceda a `http://localhost:8000/` URL para ver la página de bienvenida de Symfony:


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

El ejemplo más simple en Symfony.

1. Instala Symfony correctamente como se indica arriba.
2. Inicia el servidor Symfony si no estás instalado en el directorio www.
3. Asegúrate de que http: // localhost: 8000 esté funcionando si se usa el servidor Symfony.
4. Ahora está listo para jugar con el ejemplo más simple.
5. Agrega el siguiente código en un nuevo archivo **/src/AppBundle/Controller/MyController.php** en el directorio de instalación de Symfony.
6. Pruebe el ejemplo visitando http: // localhost: 8000 / hello
   (Si no estás utilizando el servidor http incorporado de Symfony, visita http: // localhost / (symfony-dir) /web/app_dev.php/hello)
7. Eso es todo. Siguiente: usa ramita para representar la respuesta.

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

NOTA: Todas las clases de controladores deben tener extremos con la palabra ' **Controlador** ' y los métodos relacionados con las rutas deben terminar con la palabra ' **Acción** '. Además, en qué controlador se colocan sus Acciones no es relevante hasta que defina un prefijo de enrutamiento para el controlador.

Instalando y Configurando Symfony

Requisitos de verificación

Ejecuta `bin/symfony_requirements` para verificar los requisitos de Symfony y la configuración de php cli. Instala todos los paquetes necesarios para ejecutar un proyecto de Symfony. Configuración de su php.ini, por ejemplo, configuración de zona horaria y short_open_tag. Configurando tanto php.ini para su servidor web php (por ejemplo: /etc/php/apache2/php.ini) y php cli (por ejemplo: /etc/php/cli/php.ini). Abra http: //localhost/config.php para verificar la configuración del servidor web php. Si todo ha pasado, estás listo para ejecutar tu proyecto de Symfony.

Proyecto en ejecución

Ejecute `composer install` para instalar todas las dependencias. A continuación, configure el permiso para `var/cache` , `var/logs` y `var/sessions` .

Documentación oficial detallada aquí.

Lea Empezando con symfony2 en línea: https://riptutorial.com/es/symfony2/topic/909/empezando-con-symfony2

Capítulo 2: Configuración de paquetes propios.

Introducción

Esta es una descripción de cómo puede crear una configuración para su propio paquete en /app/config/config.{yml,xml}

Examples

Crear configuración en la aplicación / config / config.yml

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

Este es un ejemplo básico para crear la configuración en formato yml, para seguir el formato yml puede tomar una configuración más profunda.

Establecer la configuración en el paquete creado

Por ejemplo, tienes un paquete generado por la consola de Symfony. En este caso, en DependencyInjection / Configuration.php, debe insertar su representación de configuración:

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

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

Básicamente, en DependencyInjection / AmazingserviceExtension.php verá las siguientes líneas:

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

No es suficiente para obtener la configuración en los servicios. Tienes que llevarlo al contenedor.

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

En este caso, la configuración en el contenedor, entonces si su Servicio obtiene el contenedor como un parámetro de constructor:

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

Luego, puede obtener la configuración en el servicio con el siguiente código, donde la configuración será una matriz asociativa:

```
private $config;

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

Lea Configuración de paquetes propios. en línea: https://riptutorial.com/es/symfony2/topic/8153/configuracion-de-paquetes-propios-

Capítulo 3: Crea servicios web con Symfony usando Rest.

Examples

Usando Symfony REST Edition

Symfony REST Edition es una aplicación **Symfony2** completamente funcional que puedes usar como esqueleto para tus nuevas aplicaciones.

Disponible en Github

Viene preconfigurado con los siguientes paquetes:

Haz	Descripción
FrameworkBundle	El núcleo de paquetes de Symfony.
SensioFrameworkExtraBundle	Agrega varias mejoras, como plantilla y capacidad de anotación de enrutamiento.
DoctrineBundle	Agrega soporte para el ORM de Doctrine.
TwigBundle	Agrega soporte para el motor de plantillas Twig .
Paquete de seguridad	Agrega seguridad al integrar el componente de seguridad de Symfony.
SwiftmailerBundle	Agrega soporte para Swiftmailer , una biblioteca para enviar correos electrónicos.
MonologBundle	Agrega soporte para Monolog , una biblioteca de registro.
AsseticBundle	Agrega soporte para Assetic , una biblioteca de procesamiento de activos.
WebProfilerBundle (en entorno dev / test)	Agrega funcionalidad de perfilado y la barra de herramientas de depuración web.
SensioDistributionBundle (en un entorno de desarrollo / prueba)	Agrega funcionalidad para configurar y trabajar con las distribuciones de **Symfony** .
SensioGeneratorBundle (en entorno dev / test)	Agrega capacidades de generación de código.

Haz	Descripción
AcmeDemoBundle (en el entorno de desarrollo / prueba)	Un paquete de demostración con algún código de ejemplo.
FOSRestBundle	El FOSRestBundle agrega funcionalidad REST.
FOSHttpCacheBundle	Este paquete ofrece herramientas para mejorar el almacenamiento en caché HTTP con **Symfony2** .
NelmioApiDocBundle	Agrega características de documentación de la API.
BazingaHateoasBundle	Añade soporte HATEOAS.
HautelookTemplatedUriBundle	Agrega soporte URI con plantilla (RFC 6570).
BazingaRestExtraBundle	BazingaRestExtraBundle Proporciona funciones adicionales para las API REST creadas con Symfony2.

Lea Crea servicios web con Symfony usando Rest. en línea:
https://riptutorial.com/es/symfony2/topic/6020/crea-servicios-web-con-symfony-usando-rest-

Capítulo 4: Creando servicios web con Symfony 2.8

Examples

Trabajar con RESTFul API

La Representación Estatal de Transferencia (REST) es un estilo arquitectónico utilizado para el desarrollo web, introducido y definido en 2000 por Roy Fielding.

Véalo en wiki: REST wiki

Se basa en el protocolo HTTP (HTTP en Wiki), solicitudes HTTP (GET, POST, PATCH, DELETE ...) / códigos de respuestas (404, 400, 200, 201, 500 ...) y estructura de cuerpos.

Esta es una excelente manera de exponer sus datos a otro sistema en Internet.

Imagine que desea crear una API RESTFul para administrar su StackOverFlower (usuario) en su base de datos local.

¡Hagamos el ejemplo!

Framework Symfony 2.8

1. Servidor web :

Debes instalar y configurar un servidor web en tu máquina local, consulta Wamp o Lamp o Mamp : debes tener una versión reciente de PHP (**!!! requisitos de Symfony !!!**)

2. Php cli y compositor:

Debe configurar PHP cli (que varía en nuestro sistema), escriba este "Cómo hacer PHP cli [OS-NAME]" en nuestro amigo Google. Debes instalar compositor, ver instalar compositor

3. Symfony:

Debes instalar Symfony 2.8 (con el compositor, es la mejor manera), abrir un terminal (o cmd en Windows) e ir a la ruta de tu servidor web.

Symfony 2 funciona con uno de los mejores tipos de estructura: paquetes. Todos son paquetes en Symfony! Podemos probarlo arriba.

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

Vaya a la estructura de árbol y vea: Symfony 2.8 está instalado en el directorio "ejemplo".

4. FOSRest (para FriendsOfSymfony) en el paquete JMSSerializer:

Debes instalar estos dos paquetes:

JMSSerializer (Instalar):

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

FosRestBundle (Instalar):

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

¡No olvides activarlos en AppKernel.php!

5. Configuracion basica :

Haga su propio paquete "Ejemplo" y cree la base de datos.

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

Ve a la parte inferior de tu archivo de configuración de la aplicación Symfony 2.8 y pégalo:

```
#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 }
```

Haga su directorio de doctrina ("example / src / ExampleBundle / Entity") y el archivo de recursos ("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
```

Generar entidad y actualizar esquema:

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

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

Hacer un controlador por defecto:

```
#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);
    }
}
```

Haga su vista predeterminada de Twig:

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

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

¡Acabas de hacer tu primera API RESTFul!

Puede probarlo en: http: //su-servidor-nombre/tu-simfonía-ruta/app_dev.php/stackoverflower/new/test .

Como puede ver en la base de datos, se ha creado un nuevo usuario con el nombre "prueba".

Puede obtener la lista de stackoverflower en: http: //su-servidor-nombre/tu-symfony-path/app_dev.php/stackoverflowers

Tiene un ejemplo completo en mi cuenta de github de este ejemplo: ejemplo de Git Hub , en la rama "maestra" de este ejemplo, y en la rama de "rutas reales", un ejemplo con una URL más apropiada (como POST y DELETE).

¡Nos vemos luego para un ejemplo con SOAP!

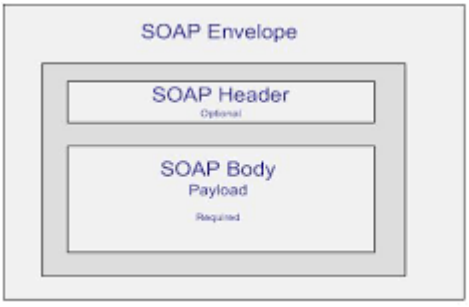
Atentamente,

Mathieu

Trabajar con API SOAP

SOAP (Simple Access Object Protocol) está basado en XML, como XML-RPC, *es ancestro* , con un archivo llamado **WSDL** , que describe el método a exponer.

Este protocolo a menudo se basa en **SOAP-Enveloppe** , un **SOAP-Body** , y alternativamente **SOAP-Header** , los datos se envuelven en una estructura y se interpretan de la misma manera desde diferentes idiomas.



Para más información, ver: SOAP en wiki.

Como se describió anteriormente, lo más importante para describir su servicio web es el archivo **WSDL** , consulte: explicación WSDL en wiki

El trabajo básico será definir qué se expone en su API SOAP, su clase y su proceso de negocio serán manejados automáticamente por la clase básica de SOAPServer de PHP. ¡Todavía

necesitas el código!

Veamos cómo se construye el archivo:

1. Servicio: Establezca la API URI y lo que se asociará.
2. Encuadernación: define las operaciones asociadas al servicio.
3. Operaciones: algunos métodos que desea exponer a la web.
4. PortTypes: definir consultas y respuestas
5. Solicitudes y respuestas: lo que esperas entrada y salida
6. Mensajes: qué formato espera (parámetros) en cada IO, pueden ser simples (cadena, entero, flotante ...) o tipo complejo (formato estructurado)

Con esta información básica, puede lograr todas las API que desee.

Imagina que quieres hacer una api de SOAP para administrar tu StackOverFlower (usuario) en tu base de datos local.

¡Hagamos el ejemplo!

Instala el servidor web, Php cli, Composer, Symfony 2.8, crea un nuevo paquete "ExampleBundle" y crea el esquema como se describe anteriormente.

Antes de comenzar a construir nuestra lógica de negocios, teníamos que saber qué exponer de nuestro controlador. Este trabajo se realiza mediante el uso de WSDL. Este es un ejemplo de una buena sintaxis de un WSDL:

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

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

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

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

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

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

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

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

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

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

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

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

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

      <output>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:example:get-list"
```

```
                use="encoded"/>
         </output>
      </operation>

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

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

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

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

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

Debemos tomar esto en tu directorio web de Symfony (en el subdirectorio de SOAP y nombrarlo como "stackoverflower.wsdl").

Realmente inspirado en el ejemplo WSDl . Puede validar eso con un validador WSDl en línea

Después de esto, podemos crear nuestro controlador y servicio básico, inspirados en SOAP Symfony 2.8 Doc .

Servicio, que es manejado por 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";
  }
}
```

Configura este servicio:

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

Como puede ver, inyectamos Doctrine Entity Manger como una dependencia porque tenemos que usar esto para el objeto StackOverFlower de CRUD.

Controlador, que expone el objeto de servicio:

```
#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;
  }
}
```

Para obtener más información sobre los servicios, consulte: Contenedor de servicios en Symfony doc.

La ruta :

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

La plantilla básica de la ramita:

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

¡Hemos creado tu primera API SOAP con Symfony 2.8!

Antes de exponerlo, ¡tenemos que probarlo!

En su StackOverFlowerController, agregue esto:

```
  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);
```

Las rutas:

```
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 }
```

Puede escribir esto en su navegador:

1. getList
2. nuevo
3. editar
4. borrar

Este es un ejemplo muy básico de una API no segura con SOAP. Puedo hacer un ejemplo de un ejemplo seguro detrás de una autenticación de clave api más adelante.

Eso es todo amigos...

Mathieu

Lea Creando servicios web con Symfony 2.8 en línea:
https://riptutorial.com/es/symfony2/topic/6355/creando-servicios-web-con-symfony-2-8

Capítulo 5: Despliegue de Symfony2

Examples

Pasos para mover el proyecto Symfony 2 a hosting manualmente

Depende del tipo de hosting que tengas:

1. Si tiene una consola SSH, puede hacerlo en el hosting después del paso 2, si no lo ha hecho localmente: ejecute el comando

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

2. Supongamos que tiene en sus carpetas de hosting `youdomain/public_html` , por lo que en `public_html` deben ubicarse todos los archivos web. Por lo tanto, debe cargar todos los proyectos de Symfony (carpetas: `app` , `src` , `vendors` , `bin` , archivos: `deps` , `deps.lock`), excepto la carpeta `web` en la carpeta `youdomain` . Todo, desde la carpeta de carga `web` a la carpeta `public_html` .

3. Compruebe CHMOD para las carpetas `app/cache` y `app/logs` , debe haber acceso de escritura.

4. Si no hay ningún archivo .htaccess en public_html, créelo y agregue dicho código en él: https://raw.github.com/symfony/symfony-standard/master/web/.htaccess

5. Ahora debe usar `youdomain.com/index` lugar de `youdomain.com/app_dev.php/index` , que usa localmente. Si un sitio aún no funciona, puede abrir el archivo `web/config.php` y encontrar un código en el que se realice una comprobación de IP, solo encontrará la IP `127.0.0.1` . Agregue su IP actual a esta lista y cargue la nueva configuración en el servidor. Luego puede abrir la ruta de acceso `yourdomain/config.php` y verificar qué está mal. Si `config.php` muestra que todo está bien, pero aún así no funcionó, puede habilitar `app_dev.php` para depurar: abra `app/app_dev.php` y su IP de la misma manera que en `config.php` . Ahora puede ejecutar scripts como localmente usando `app_dev.php` .

Lea Despliegue de Symfony2 en línea: https://riptutorial.com/es/symfony2/topic/6039/despliegue-de-symfony2

Capítulo 6: Doctrina Entidad Relaciones

Examples

Uno a muchos, bidireccional

Este mapeo bidireccional requiere el `mappedBy` atributo en el `OneToMany` asociación y el `inversedBy` atributo en el `ManyToOne` asociación.

Una relación bidireccional tiene un lado propietario e inverso . `OneToMany` relaciones `OneToMany` pueden usar tablas de unión, por lo que debe especificar un lado propietario. La asociación `OneToMany` es siempre el lado inverso de una asociación bidireccional.

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

Lea Doctrina Entidad Relaciones en línea: https://riptutorial.com/es/symfony2/topic/3043/doctrina-entidad-relaciones

Capítulo 7: Doctrine Entity Repository

Examples

Creando un nuevo repositorio

Puede crear un nuevo Repositorio donde lo desee, pero se recomienda crearlos en una carpeta de `Repository` separada.

Si bien puede nombrar el archivo y la clase del Repositorio como desee, se recomienda nombrar el Repository `EntityNameRepository` , para que pueda encontrarlos rápidamente en su carpeta.

Supongamos que tenemos una Entidad de `Project` , almacenada en `AppBundle\Entity` , se vería así:

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

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

?>
```

La parte importante aquí es la línea `@ORM\Entity(repositoryClass="AppBundle\Repository\ProjectRepository")` , porque conecta esta Entidad con la clase de Repository dada.

También debe usar la clase `\Doctrine\ORM\Mapping` para usar las opciones de mapeo.

El repositorio en sí es bastante 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();
    }
}

?>
```

Es importante tener en cuenta que la clase Repository debe extender `\Doctrine\ORM\EntityRepository` para que funcione correctamente. Ahora puede agregar tantas funciones para diferentes consultas como desee.

Función ExpressionBuilder IN ()

Si desea utilizar el comando MySQL `IN()` en QueryBuilder, puede hacerlo con la función `in()` de la clase ExpressionBuilder .

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

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

Hacer una consulta con una sub-consulta

Como ejemplo, solo para demostrar CÓMO usar una instrucción de selección de subconsulta dentro de una instrucción de selección, supongamos que encontramos a todos los usuarios que aún no han compilado la dirección (no existen registros en la tabla de direcciones):

```
 // 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();
```

Lea Doctrine Entity Repository en línea: https://riptutorial.com/es/symfony2/topic/6032/doctrine-

entity-repository

Capítulo 8: Enrutamiento

Examples

Devuelve una respuesta 404

Se devuelven 404 respuestas cuando no se encuentra un recurso en el servidor, en Symfony este estado se puede crear lanzando una excepción `NotFoundHttpException` . Para evitar una declaración de `use` adicional dentro de un controlador, use `createNotFoundException()` provista por la clase `Controller`

```
<?php

namespace Bundle\Controller;

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

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

}
```

Múltiples rutas

En Symfony es posible definir múltiples rutas para una acción. Esto puede ser muy útil si tiene funciones que hacen lo mismo pero tienen diferentes parámetros.

```
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));
    }
}
```

Solicitud de POST redireccionamiento

Cuando se encuentra en una **instancia de controllerAction** Y **recibe** una **solicitud POST** , pero desea **redirigirla a una ruta diferente** , mientras **mantiene el método POST y el objeto de solicitud** , puede usar lo siguiente:

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

El código 307 aquí conserva el método de solicitud.

Enrutamiento basado en subdominio

El enrutamiento basado en subdominios se puede manejar en Symfony usando el parámetro de `host` . Por ejemplo, el parámetro `_locale` se puede utilizar como valor de subdominio.

Asumiendo

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

los parámetros se definen en el archivo de configuración `parameters.yml` , la ruta sería:

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

Desde este punto, el enrutador puede manejar URI como `http://de.somedomain.com` . La segunda anotación de `@Route` se puede usar como una alternativa para la configuración regional predeterminada y el subdominio nulo, `http://somedomain.com` .

Rutas de Symfony usando Routing.yml

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

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

Si decide utilizar Routing.yml en lugar de Anotaciones, puede obtener una mejor vista de **todas las rutas** y es más fácil buscar y encontrar una.

> **Depende de** usted elegir entre **Routing.yml** y **Annotations** . Puedes usar ambos para diferentes rutas, pero esta no es la mejor solución.

Anotación `@Route()` equivalente es:

```
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));
    }
}
```

Lea Enrutamiento en línea: https://riptutorial.com/es/symfony2/topic/1691/enrutamiento

Capítulo 9: Enrutamiento básico

Examples

Enrutamiento basado en anotaciones

De forma predeterminada, todos los controladores que generas con el comando de `generate:controller` incorporado de Symfony utilizarán las anotaciones de Symfony para el enrutamiento:

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

Para que el marco pueda manejar estas rutas, debe importarlas en su `routing.yml` siguiente manera (observe el tipo de `annotation`):

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

Rutas de YAML

En lugar de anotaciones, también puede especificar sus rutas como YAML:

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

Las mismas opciones se aplican tanto a las anotaciones como a las configuraciones YAML. Para importar una configuración de enrutamiento YAML en su configuración de enrutamiento raíz, no necesita especificar un tipo:

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

Lea Enrutamiento básico en línea: https://riptutorial.com/es/symfony2/topic/5881/enrutamiento-basico

Capítulo 10: Extensiones Symfony Twig

Examples

Una extensión de ramita simple - Symfony 2.8

Antes de crear cualquier extensión, compruebe siempre si ya se **ha implementado** .

Lo primero que uno tendría que hacer es definir la clase de extensión que albergará los filtros y / o funciones de ramitas.

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

Uno luego alerta al contenedor de servicios de la extensión de ramita recién creada.

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

Con esto, tiene todo lo que necesita para poder utilizar su filtro o función de ramita recién creado en sus plantillas de ramita.

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

Haga un número corto, por ejemplo, 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';
    }
}
```

Añade tu extensión a services.yml

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

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

Úsalo en TWIG

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

Lea Extensiones Symfony Twig en línea:
https://riptutorial.com/es/symfony2/topic/6000/extensiones-symfony-twig

Capítulo 11: Gestionando los firewalls y la seguridad de Symfony.

Examples

Gestionando la seguridad

La seguridad era parte del lado oscuro de la documentación de Symfony, tiene un componente dedicado llamado **Componente de Seguridad** .

Este componente se configura en el archivo **security.yml** del proyecto principal de la aplicación.

La configuración por defecto es como esta:

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

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

        default:
            anonymous: ~
```

Puede definir **cortafuegos** específicos para restringir el acceso a algunas URL a **roles** específicos basados en una jerarquía para sus **usuarios** que están definidos por un **proveedor** y **codificadores** que administran la seguridad de la contraseña.

Por ejemplo, si desea crear un **proveedor** personalizado, desde su motor de base de datos, puede definir su **security.yml** así:

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

Esto se detalla en la documentación de Symfony: cómo definir un UserProvider personalizado y desde la base de datos o contra LDAP, por ejemplo.

Después de eso, puede definir un **firewall** para restringir algunas URL basadas en su proveedor de usuario personalizado (security.yml) explícitamente de esta manera:

```
firewalls:
    secured_area:
```

```
        pattern: ^/admin
```

O con **control de acceso** :

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

Ver más documentación detallada aquí .

La mejor manera de administrar usuarios es usar FosUserBundle que extiende algunas funcionalidades del marco.

Lea Gestionando los firewalls y la seguridad de Symfony. en línea: https://riptutorial.com/es/symfony2/topic/7486/gestionando-los-firewalls-y-la-seguridad-de-symfony-

Capítulo 12: Instala Symfony2 en localhost

Examples

Usando el símbolo del sistema

La mejor manera de instalar y configurar un proyecto Symfony2 se describe en la documentación oficial de la siguiente manera:

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'));"
```

Entonces podrías usar el binario de Symfony para construir el andamio correcto:

```
$ symfony new my_project
```

Usando compositor sobre consola

Dado que ya has instalado el compositor en funcionamiento y es accesible a nivel mundial, puedes crear nuevos proyectos de Symfony como se indica en la documentación oficial .

Ahora puedes crear un nuevo proyecto de Symfony con compositor:

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

Esto creará el proyecto en el directorio actual en el que estás.

Si quieres crear el proyecto con una versión específica de Symfony, puedes agregar un parámetro con el número de versión:

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

Sugerencia: si piensa que el compositor no hará nada, agregue el `-vvv` al comando. De esa manera, recibirás información detallada sobre lo que el compositor está haciendo en este momento.

Lea Instala Symfony2 en localhost en línea: https://riptutorial.com/es/symfony2/topic/6340/instala-symfony2-en-localhost

Capítulo 13: Monolog: mejora tus registros.

Examples

Agregue los detalles del usuario y los parámetros publicados enviados a los registros.

Los registros son muy importantes. Volver a crear un contexto de error puede ser a veces muy doloroso debido a la falta de información sobre cómo y cuándo ocurrió el error.

Este ejemplo muestra:

- Cómo agregar datos de usuario en los registros de error
- Cómo agregar parámetros de publicación enviados cuando se produjo un error
- Cómo usar WebProcessor para agregar todos los datos relacionados con la solicitud, como:
    - url
    - ip
    - método http
    - servidor
    - referente

Configuración del servicio

```
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 }
```

Código de servicio

```
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;
```

```
        }
}
```

Lea Monolog: mejora tus registros. en línea:
https://riptutorial.com/es/symfony2/topic/6671/monolog--mejora-tus-registros-

Capítulo 14: Opciones de envío a una clase de formulario

Sintaxis

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

Parámetros

Parámetro	Definición
Tipo de Hogar :: clase	clase de formulario personalizado para la entidad del hogar
$ hogar	una instancia de la entidad del hogar (generalmente creada por `$household = new Household();`)
$ formOptions	un conjunto de opciones definidas por el usuario para pasar a la clase de formulario, por ejemplo, `$formOptions = array('foo' => 'bar');`

Observaciones

Al crear una clase de formulario, los campos de formulario se agregan en la `public function buildForm(FormBuilderInterface $builder, array $options) {...}` . El parámetro `$options` incluye un conjunto de opciones predeterminadas, como `attr` y `label` . Para permitir que sus opciones personalizadas estén disponibles en la clase de formulario, las opciones deben inicializarse en `configureOptions(OptionsResolver $resolver)`

Así que para nuestro ejemplo del mundo real:

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

Examples

Un ejemplo del mundo real de un controlador de la casa

Antecedentes: la entidad del hogar incluye un conjunto de opciones, cada una de las cuales es una entidad que se administra en un backend de administración. Cada opción tiene una bandera

`enabled` booleana. Si una opción previamente habilitada está configurada como deshabilitada, deberá persistir en las ediciones posteriores de Household, pero no podrá ser eliminada. Para lograr esto, la definición de campo en la clase de formulario mostrará el campo como un campo de opción deshabilitado si la opción ha `enabled = false` (pero se mantiene porque el botón de envío activa un javascript que elimina el atributo `disabled`). La definición de campo también evita las opciones deshabilitadas de ser mostrado.

La clase de formulario debe saber, para una entidad del hogar determinada, cuál de sus opciones se ha deshabilitado. Se ha definido un servicio que devuelve una matriz de los nombres de las entidades de opción que se han deshabilitado. Esa matriz es `$disabledOptions` .

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

Cómo se usan las opciones personalizadas en la clase de formulario

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

Entidad de vivienda

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

Lea Opciones de envío a una clase de formulario en línea:
https://riptutorial.com/es/symfony2/topic/7237/opciones-de-envio-a-una-clase-de-formulario

Capítulo 15: Patrones de diseño de Symfony

Examples

Patrón de inyección de dependencia

Imagina que tienes un administrador de clase para administrar el envío de correos (se llamará MailManager).

En esto, tienes que registrar los correos que se envían. Una buena solución es transformar la clase MailManager en un `service` y luego inyectar la clase para crear registros (`Monolog` por ejemplo) en el MailManager creando un servicio.

Para hacer esto :

1- Declarar la futura clase de MailManager como servicio (en services.yml)

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

2- Inyectar el servicio existente logger utilizando el método de `argument`

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

3- Crear la clase MailManager.

```
<?php

namespace Project\Bundle\Manager;

use Symfony\Component\HttpKernel\Log\LoggerInterface;

class MailManager
{
  protected $logger;

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

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

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

```
    }
}
```

4- Llamar a MailManager en un controlador, por ejemplo.

```
<?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);

    }
}
```

Lea Patrones de diseño de Symfony en línea:
https://riptutorial.com/es/symfony2/topic/6289/patrones-de-diseno-de-symfony

Capítulo 16: Respuesta

Parámetros

Parámetro	Detalles
contenido de la `string`	El contenido de la respuesta.
estado `integer`	El código de estado HTTP.
encabezados de `array`	Array de cabeceras de respuesta.

Examples

Uso simple

```
public function someAction(){

    // Action's code

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

Establecer código de estado

```
public function someAction(){

    // Action's code

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

Otro ejemplo:

```
public function someAction(){

    // Action's code

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

Establecer encabezado

Ver lista de encabezados http .

```
 public function someAction(){

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

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

    return $response;
}
```

JsonResponse

Devuelva la respuesta con formato JSON:

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

public function someAction(){

    // Action's code

    $data = array(
        // Array data
    );

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

Lea Respuesta en línea: https://riptutorial.com/es/symfony2/topic/9218/respuesta

Capítulo 17: Servicios de Symfony

Examples

Cómo declarar, escribir y usar un servicio simple en Symfony2

Declaración de servicios:

```
# 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: []
```

Código de servicio :

```
<?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
    }
}
```

Úsalo en un controlador:

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

Lea Servicios de Symfony en línea: https://riptutorial.com/es/symfony2/topic/4587/servicios-de-symfony

Capítulo 18: Solicitud

Observaciones

Enlaces de documentación API (maestro):

- Solicitud
- RequestStack

El objeto de solicitud contiene varios datos significativos, como la configuración regional actual y el controlador coincidente. Puedes usarlos y gestionarlos por eventos de HttpKernel. Para una comprensión confiable del ciclo en vivo de la Solicitud-Respuesta, lea esta página de documentos del Componente HttpKernel (¡muy útil!).

Examples

Acceso a la solicitud en un controlador

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

Tenga en cuenta que el objeto Request inyectado se aplica a la solicitud actual (puede o no ser igual a la solicitud maestra).

Acceso a Solicitud en una plantilla Twig o PHP.

En la plantilla de Twig, el objeto de solicitud está disponible en

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

Cuando desee mostrar el método de solicitud en Twig, intente esto:

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

En plantilla de PHP

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

Lea Solicitud en línea: https://riptutorial.com/es/symfony2/topic/4870/solicitud

Capítulo 19: Symfony Twig Extenstion Ejemplo

Parámetros

Parámetros	Descripción
$ doctrina	Objeto de la doctrina que pasamos del servicio.

Examples

Ejemplo básico de la extensión de ramita de Symfony

En este ejemplo defino dos funciones personalizadas. 1 - La función countryFilter obtiene el código corto del país como entrada y devuelve el nombre completo del país. 2 - _countPrinterTasks se utiliza para calcular el no de tareas asignadas a un usuario en particular.

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

Para llamarlo desde la Ramita, solo tenemos que usarlo como se muestra abajo;

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

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

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

Y declare este extenstion como un servicio en su `bundle/resource/config/service.yml` .

```
services:

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

Lea Symfony Twig Extenstion Ejemplo en línea:
https://riptutorial.com/es/symfony2/topic/5891/symfony-twig-extenstion-ejemplo

Capítulo 20: Validación de formularios

Examples

Validación de formulario simple usando restricciones

Ejemplo de acción del controlador

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

Ejemplo de restricciones para objeto

@Anotaciones

```
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
```

Referencia de restricciones de validación:
https://symfony.com/doc/current/reference/constraints.html

Validación de formulario: http://symfony.com/doc/current/forms.html#form-validation

Lea Validación de formularios en línea: https://riptutorial.com/es/symfony2/topic/7813/validacion-de-formularios

Creditos

S. No	Capítulos	Contributors
1	Empezando con symfony2	althaus, COil, Community, Dheeraj, Hendra Huang, Jakub Zalas , karel, kix, mgh, Redjan Ymeraj, TRiNE, uddhab
2	Configuración de paquetes propios.	PumpkinSeed
3	Crea servicios web con Symfony usando Rest.	Barathon, Sunitrams'
4	Creando servicios web con Symfony 2.8	Barathon, Mathieu Dormeval
5	Despliegue de Symfony2	sphinks
6	Doctrina Entidad Relaciones	Federkun, palra
7	Doctrine Entity Repository	doydoy44, KhorneHoly, Matteo
8	Enrutamiento	afkplus, Bob, Dheeraj, Farahmand, Kid Binary, kix, pinch boi triggered af, Sam Janssens, Scott Flack, Stephan Vierkant, Stevan Tosic, Stony, tchap
9	Enrutamiento básico	kix
10	Extensiones Symfony Twig	Sand, Strabek
11	Gestionando los firewalls y la seguridad de Symfony.	Mathieu Dormeval
12	Instala Symfony2 en localhost	Barathon, KhorneHoly, sentenza
13	Monolog: mejora tus registros.	sdespont

14	Opciones de envío a una clase de formulario	geoB
15	Patrones de diseño de Symfony	DOZ
16	Respuesta	CStff
17	Servicios de Symfony	DonCallisto, enricog, moins52
18	Solicitud	A.L, Arkemlar, DOZ, Hermann Döppes, Mathieu Dormeval, mgh , Paweł Brzoski
19	Symfony Twig Extenstion Ejemplo	Muhammad Taqi
20	Validación de formularios	Griva