EBook Gratis

APRENDIZAJE symfony3

Free unaffiliated eBook created from
Stack Overflow contributors.

#symfony3

Tabla de contenido

Acerca de

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

It is an unofficial and free symfony3 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official symfony3.

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

Capítulo 1: Empezando con symfony3

Observaciones

Esta sección proporciona una descripción general de qué es symfony3 y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de symfony3, y vincular a los temas relacionados. Dado que la Documentación para symfony3 es nueva, es posible que necesites crear versiones iniciales de esos temas relacionados.

Versiones

Versión	Fecha de lanzamiento
3.0.0	2015-11-30
3.1.0	2016-05-30
3.2.0	2016-11-30
3.2.5	2017-03-09
3.2.6	2017-03-10
3.2.7	2017-04-05

Examples

3. Sistemas de Windows

Debe agregar php a su variable de entorno de ruta. Sigue estos pasos:

Windows 7:

- Haga clic derecho en el ícono de Mi PC
- Haga clic en Propiedades
- Haga clic en Configuración avanzada del sistema en la barra izquierda
- Haga clic en la pestaña Avanzado
- Haga clic en el botón Variables de entorno
- En la sección Variables del sistema, seleccione Ruta (sin distinción de mayúsculas) y haga clic en el botón Editar
- Agregue un punto y coma (;) al final de la cadena, luego agregue la ruta completa del sistema de archivos de su instalación de PHP (por ejemplo, `C:\Program Files\PHP`)
- Siga haciendo clic en Aceptar, etc. hasta que todos los cuadros de diálogo hayan

desaparecido.
- Cierra el símbolo del sistema y ábrelo de nuevo.
- Ordenados

Windows 8 y 10

- En Buscar, busque y luego seleccione: Sistema (Panel de control)
- Haga clic en el enlace Configuración avanzada del sistema.
- Haga clic en Variables de entorno.
- En la sección Variables del sistema, busque la variable de entorno PATH y selecciónela. Haga clic en Editar. Si la variable de entorno PATH no existe, haga clic en Nuevo.
- Agregue la ruta completa del sistema de archivos de su instalación de PHP (por ejemplo, `C:\Program Files\PHP`)

Después de esto, abra su consola de comandos y ejecute el siguiente comando:

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

Luego, mueve el archivo symfony descargado al directorio de tu proyecto y ejecútalo de la siguiente manera:

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

4. Creando la aplicación Symfony

Una vez que el instalador de Symfony esté disponible, crea tu primera aplicación de Symfony con el nuevo comando:

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

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

Este comando se puede ejecutar desde cualquier lugar, no necesariamente desde la carpeta `htdocs` .

Este comando crea un nuevo directorio llamado `my_project_name/` que contiene un nuevo proyecto basado en la versión estable más reciente de Symfony. Además, el instalador verifica si su sistema cumple con los requisitos técnicos para ejecutar aplicaciones Symfony. De lo contrario, verá la lista de cambios necesarios para cumplir con esos requisitos.

1. Instalar el instalador de Symfony

> El instalador requiere PHP 5.4 o superior. Si aún usas la versión heredada de PHP 5.3, no puedes usar el instalador de Symfony. Lee la sección Cómo crear aplicaciones Symfony sin el instalador para saber cómo proceder. - fuente:

http://symfony.com/doc/current/book/installation.html

5. Basando tu proyecto en una versión específica de Symfony

En caso de que tu proyecto deba basarse en una versión específica de Symfony, usa el segundo argumento opcional del nuevo comando:

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

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

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

El instalador también admite una versión especial llamada lts que instala la versión más reciente de Symfony LTS disponible:

```
$ symfony new my_project_name lts
```

Lee el proceso de lanzamiento de Symfony para comprender mejor por qué hay varias versiones de Symfony y cuál usar para tus proyectos.

También puedes crear aplicaciones de Symfony sin el instalador, pero no fue una buena idea. Si quieres de todos modos, sigue el tutorial original en este enlace:

Documentos oficiales de Symfony, configurando Symfony sin el instalador

2. Sistemas Linux y Mac OS X.

Abre tu consola de comandos y ejecuta los siguientes comandos:

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

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
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>'
        );
    }
}
```

Creando página

> Antes de continuar, asegúrese de haber leído el capítulo de Instalación y de que pueda acceder a su nueva aplicación Symfony en el navegador.

Supongamos que desea crear una página - / lucky / number - que genera un número afortunado (bueno, aleatorio) y lo imprime. Para hacer eso, cree un método de "clase de controlador" y un "controlador" dentro de él que se ejecutará cuando alguien vaya a / lucky / number

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

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

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

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

Lea Empezando con symfony3 en línea: https://riptutorial.com/es/symfony3/topic/985/empezando-con-symfony3

Capítulo 2: Configuración

Introducción

Ejemplos y buenas prácticas para configurar su aplicación Symfony que no están en la documentación oficial.

Examples

Incluir todos los archivos de configuración de un directorio

Después de un tiempo, terminas con muchos elementos de configuración en tu config.yml. Puede hacer que su configuración sea más fácil de leer si divide su configuración en varios archivos. Puede incluir fácilmente todos los archivos de un directorio de esta manera:

config.yml:

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

En la `includes` directorio se puede poner por ejemplo doctrine.yml, swiftmailer.yml, etc.

Utilice el nombre de clase completo (FQCN) como ID de servicio

En muchos ejemplos, encontrará un ID de servicio como 'acme.demo.service.id' (una cadena con puntos). Usted `services.yml` se verá así:

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

En su controlador, puede utilizar este servicio:

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

Si bien no hay ningún problema con esto, puede usar un Nombre de clase totalmente calificado (FQCN) como ID de servicio:

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

En tu controlador puedes usarlo así:

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

Esto hace que su código sea mejor para entender. En muchos casos, no tiene sentido tener un ID de servicio que no sea solo el nombre de la clase.

A partir de Symfony 3.3, incluso puedes eliminar el atributo de `class` si tu ID de servicio es un FQCN.

No se necesita interfaz HTTP?

Si su aplicación no necesita ninguna interfaz HTTP (por ejemplo, solo para una aplicación de consola), deseará deshabilitar al menos `Twig` y `SensioFrameworkExtra`

Solo comenta esas líneas:

app / AppKernel.php

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

app / config / config.yml

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

app / config / config_dev.yml

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

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

También puede eliminar los requisitos de proveedores relacionados de **composer.json** :

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

El uso de Symfony en todos los casos es discutible, pero al menos puede ser temporal.

Lea Configuración en línea: https://riptutorial.com/es/symfony3/topic/9175/configuracion

Capítulo 3: Despachador de eventos

Sintaxis

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

Observaciones

- A menudo es mejor usar una sola instancia de EventDispatcher en su aplicación que inyecte en los objetos que necesitan disparar eventos.
- Es una práctica recomendada tener una única ubicación en la que gestione la configuración y agregue escuchas de eventos a su EventDispatcher. El framework Symfony usa el Depósito de Inyección de Dependencias.
- Estos patrones le permitirán cambiar fácilmente sus escuchas de eventos sin necesidad de cambiar el código de cualquier módulo que esté enviando eventos.
- El desacoplamiento del envío de eventos de la configuración del detector de eventos es lo que hace que Symfony EventDispatcher sea tan poderoso
- EventDispatcher lo ayuda a satisfacer el principio abierto / cerrado.

Examples

Inicio rápido del despachador de eventos

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

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

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

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

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

Suscriptores de eventos

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

$dispatcher = new EventDispatcher();

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

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

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

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

Lea Despachador de eventos en línea:
https://riptutorial.com/es/symfony3/topic/5609/despachador-de-eventos

Capítulo 4: Enrutamiento

Introducción

Una ruta es como asignar una URL a una acción (función) en una clase de Controlador. El siguiente tema se centrará en la creación de rutas, pasando los parámetros a la clase Controller a través de una ruta, ya sea utilizando YAML o una anotación.

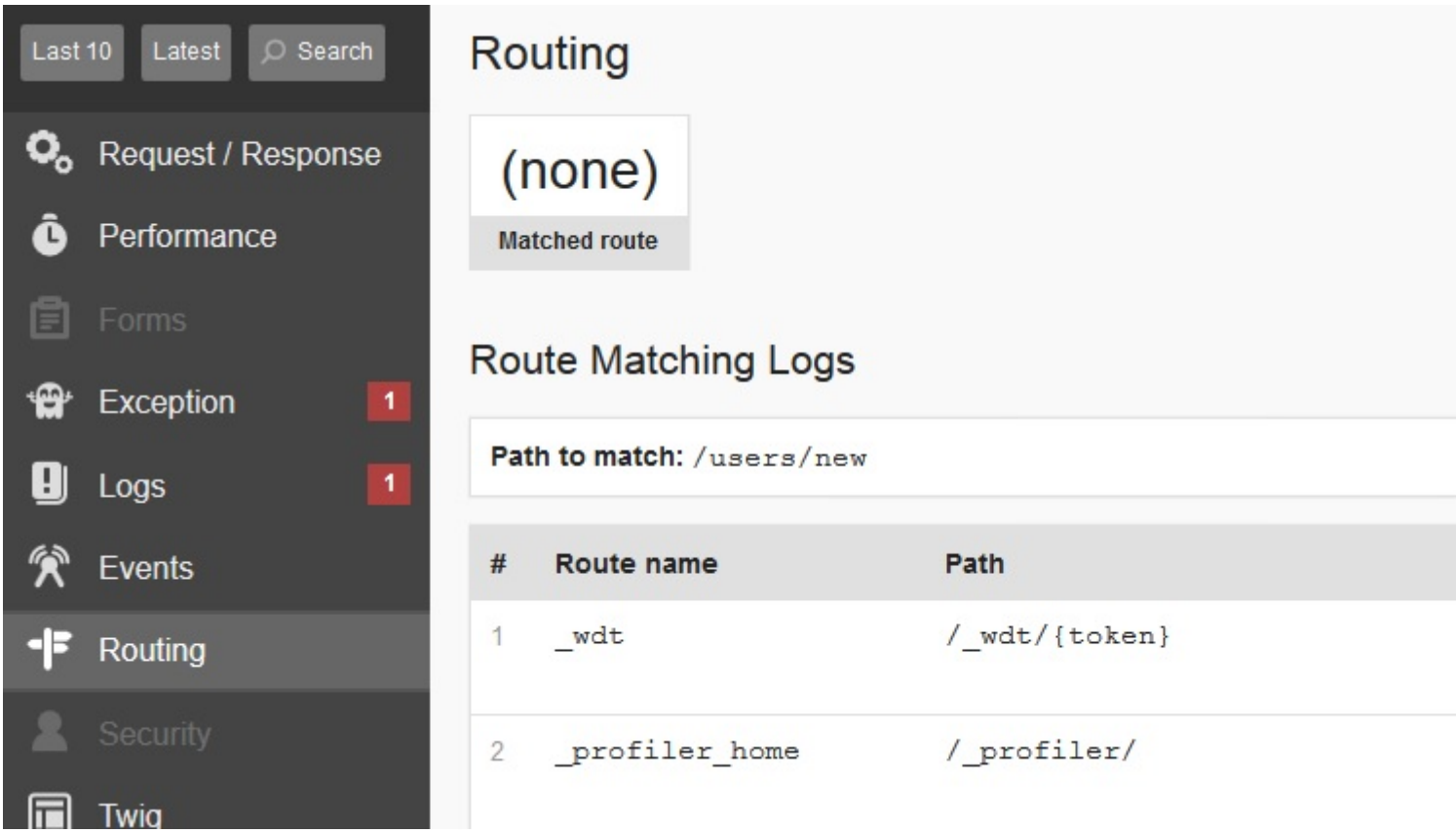
Observaciones

Es útil ver lo que genera Symfony Framework, este proporciona herramientas para ver todas las rutas de una aplicación específica.

Desde el Symfony Doc , use (en un shell):

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

Además, puede ver toda la información relevante de las rutas en el perfilador de Framework, en el menú de enrutamiento:



Examples

Enrutamiento utilizando YAML

La configuración de enrutamiento está incluida en su archivo `app/config/config.yml` , por defecto el archivo `app/config/routing.yml` .

Desde allí puede enlazar a su propia configuración de enrutamiento en un paquete

```
# app/config/routing.yml

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

También puede contener varias rutas globales de aplicaciones.

En su propio paquete puede configurar rutas que sirven para dos propósitos:

- Coincidencia con una solicitud, de manera que se llame a la acción correcta para la solicitud.
- Generando una URL a partir de los parámetros de nombre y ruta.

A continuación se muestra un ejemplo de configuración de ruta YAML:

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

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

La ruta se llama `my_page` y llama a `myPageAction` del `DefaultController` en `AppBundle` cuando se solicita. Tiene un parámetro, denominado `parameter` con un valor predeterminado. El valor solo es válido cuando coincide con la expresión regular `\d+` . Para esta ruta, solo se aceptan los métodos HTTP `GET` y `PUT` . La `condition` es una expresión en el ejemplo que la ruta no coincidirá a menos que el encabezado User-Agent coincida con firefox. Puedes hacer cualquier lógica compleja que necesites en la expresión aprovechando dos variables que se pasan a la expresión: `context` (RequestContext) y `request` (Symfony Request).

Una ruta generada con el valor del parámetro 10 podría verse como `/application/content/page/10` .

Enrutamiento mediante anotaciones.

La configuración de enrutamiento está incluida en su archivo `app/config/config.yml` , por defecto el archivo `app/config/routing.yml` .

Desde allí puede enlazar a los controladores que tienen una configuración de enrutamiento anotada:

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

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

En su propio paquete puede configurar rutas que sirven para dos propósitos:

- Coincidencia con una solicitud, de manera que se llame a la acción correcta para la solicitud.
- Generando una URL a partir de los parámetros de nombre y ruta.

A continuación se muestra un ejemplo de configuración de ruta anotada:

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

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

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

El controlador está anotado con una ruta de *prefijo* , de manera que cualquier ruta configurada en este controlador se añadirá con el prefijo.

La ruta configurada se llama `my_page` y llama a la función `myPageAction` cuando se solicita. Tiene un parámetro, denominado `parameter` con un valor predeterminado. El valor solo es válido cuando coincide con la expresión regular `\d+` . Para esta ruta, solo se aceptan los métodos HTTP `GET` y `PUT` .

Observe que el parámetro se inyecta en la acción como un parámetro de función.

Una ruta generada con el valor del parámetro 10 podría verse como `/application/content/page/10` .

Rutas de descanso de gran alcance

Tradicionalmente, puede usar el enrutamiento para asignar una solicitud con el Componente de enrutamiento que manejó los parámetros de solicitud y respuesta por el Componente de HttpFoundation .

Además, se puede crear un parámetro de ruta personalizado utilizando FOSRestBundle para extender las funcionalidades predeterminadas del Componente de enrutamiento.

Esto es útil para crear rutas REST, y es realmente útil para especificar cómo transferir datos estructurados como XML o json en una solicitud (y una respuesta).

Consulte FOSRestBundle Doc para obtener más información, y especialmente esta anotación:

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

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

Lea Enrutamiento en línea: https://riptutorial.com/es/symfony3/topic/2287/enrutamiento

Capítulo 5: Entidades declarantes

Examples

Declarar una entidad de Symfony como YAML

- AppBundle / Entity / Person.php

```
<?php

namespace AppBundle\Entity;

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

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

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


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

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

        return $this;
    }
```

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

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

        return $this;
    }

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

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

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

```
    lifecycleCallbacks: { }
```

- Crear la tabla

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

O use la forma recomendada (asumiendo que ya tiene el paquete doctrina-migrations instalado en su proyecto):

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

- Crea la entidad automáticamente desde el YAML.

  doctrina php bin / console: generar: entidades AppBundle

Declarando una Entidad Symfony con anotaciones

- AppBundle / Entity / Person.php

```
<?php

namespace AppBundle\Entity;

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

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

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

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

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

```
    }

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

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

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

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

- Generar la entidad

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

- Para volcar las sentencias de SQL a la pantalla.

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

- Crear la tabla

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

O use la forma recomendada (asumiendo que ya tiene el paquete doctrina-migrations instalado en su proyecto):

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

Lea Entidades declarantes en línea: https://riptutorial.com/es/symfony3/topic/4443/entidades-declarantes

Capítulo 6: Formas dinamicas

Examples

Cómo extender ChoiceType, EntityType y DocumentType para cargar opciones con AJAX.

En Symfony, el ChoiceType integrado (y EntityType o DocumentType que lo extiende), funciona básicamente con una lista de opciones constante.

Si desea que funcione con llamadas ajax, debe cambiarlas para que acepte cualquier opción adicional resumida.

- **¿Cómo empezar con una lista de opciones vacía?**

  Cuando cree su formulario, simplemente configure la opción de `choices` en una `array()` vacía `array()` :

```
namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\FormBuilderInterface;

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

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

  Así obtendrá una entrada de selección vacía, sin opciones. Esta solución funciona para ChoiceType y todos sus hijos (EntityType, DocumentType, ...).

- **Cómo aceptar nuevas opciones enviadas** :

  Para aceptar las nuevas opciones, debe hacer que estén disponibles en la lista de selección de campos de formulario. Puede cambiar el campo de formulario según los datos enviados con el evento FormEvent :: PRE_SUBMIT.

  Este ejemplo muestra cómo hacerlo con un ChoiceType básico:

```
namespace AppBundle\Form;

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

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

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

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

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

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

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

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

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

Tus opciones enviadas ahora son opciones permitidas y la validación integrada de Symfony ChoiceType no las rechazará más.

Si desea hacer lo mismo con un hijo ChoiceType (EntityType, DocumentType, ...), debe inyectar el entityManager o el documentManager y realizar la transformación de datos al completar las nuevas opciones.

Rellene un campo de selección en función del valor de otro.

Este es un ejemplo para mostrar cómo cambiar las opciones permitidas en un campo de selección de subcategoría dependiendo del valor del campo de selección de categoría. Para hacer esto, tiene que hacer que sus elecciones de subcategoría sean dinámicas tanto para el cliente como para el servidor.

1. Haga que el formulario sea dinámico en el lado del cliente para las interacciones entre la pantalla y el usuario.

Ejemplo de forma dinámica del lado del cliente (usando Javascript / JQuery):

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

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

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

Por supuesto, usted podría obtener las opciones de una llamada AJAX. Ese no es el propósito de este ejemplo.

2. Haga el formulario dinámico en el lado del servidor para la inicialización / validación

Ejemplo de forma dinámica del lado del servidor:

```
namespace AppBundle\Form;

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

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

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

class MyBaseFormType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
```

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

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

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

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

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

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

Lea Formas dinamicas en línea: https://riptutorial.com/es/symfony3/topic/5855/formas-dinamicas

Capítulo 7: Gestión de Activos con Assetic

Introducción

Cuando utilice el paquete Assetic, de acuerdo con la documentación de Symfony, tenga en cuenta lo siguiente:

A partir de Symfony 2.8, Assetic ya no se incluye de forma predeterminada en la edición estándar de Symfony. Antes de usar cualquiera de sus funciones, instale AsseticBundle ejecutando este comando de consola en su proyecto:

$ compositor requiere symfony / assetic-bundle

Hay otros pasos que debes seguir. Para obtener más información, visite: http://symfony.com/doc/current/assetic/asset_management.html

Parámetros

Nombre	Ejemplo
Camino	'static / images / logo / logo-default.png'

Observaciones

La carpeta para los activos de acceso público en un proyecto estándar de Symfony3 es "/ web". Assetic utiliza esta carpeta como carpeta raíz para los activos.

Examples

Crear una ruta relativa para el activo

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

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

Crear ruta absoluta para el activo

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

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

Lea Gestión de Activos con Assetic en línea: https://riptutorial.com/es/symfony3/topic/6409/gestion-de-activos-con-assetic

Capítulo 8: Pruebas

Examples

Pruebas simples en Symfony3

Prueba de unidad

Las pruebas unitarias se utilizan para asegurarse de que su código no tenga un error de sintaxis y para probar la lógica de su código para que funcione como lo esperaba. Ejemplo rápido:

src / AppBundle / Calculator / BillCalculator.php

```
<?php

namespace AppBundle\Calculator;

use AppBundle\Calculator\TaxCalculator;

class BillCalculator
{
    private $taxCalculator;

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

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

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

src / AppBundle / Calculator / TaxCalculator.php

```
<?php

namespace AppBundle\Calculator;

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

tests / AppBundle / Calculator / BillCalculatorTest.php

```
<?php

namespace Tests\AppBundle\Calculator;

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

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

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

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

Probé mi clase BillCalculator para asegurarme de que mi BillCalculator devolverá el precio total de los productos + 10% de impuestos. En prueba de unidad, creamos nuestro propio caso de prueba. En esta prueba, proporciono 2 productos (los precios son 100 y 200), por lo que el impuesto será del 10% = 30. Espero que la calculadora de impuestos devuelva 30, por lo que el precio total será 300 + 30 = 330.

Prueba funcional

Las pruebas funcionales se utilizan para probar la entrada y la salida. Con la entrada dada, esperaba alguna salida sin probar el proceso para crear la salida. (Esto es diferente con la prueba de unidad porque en la prueba de unidad, probamos el flujo de código). Ejemplo rápido:

```
namespace Tests\AppBundle;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ApplicationAvailabilityFunctionalTest extends WebTestCase
{
    /**
     * @dataProvider urlProvider
     */
    public function testPageIsSuccessful($url)
    {
```

```
        $client = self::createClient();
        $client->request('GET', $url);

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

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

Probé mi controlador para asegurarme de que mi controlador devolverá 200 respuestas en lugar de 400 (No encontrado) o 500 (Error interno del servidor) con la url dada.

Referencias:

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

Lea Pruebas en línea: https://riptutorial.com/es/symfony3/topic/2881/pruebas

Capítulo 9: Trabajar con servicios web

Examples

API de descanso

He escrito previamente documentación en este sitio para describir cómo hacer servicios web en Symfony

Volveré a escribir un tutorial para la versión de symfony> = 3.

Creemos que tenemos un servidor web instalado en una versión configurada de Symfony Framework . Usted debe tener el compositor (administrador de paquetes php) instalado también.

Para hacerlo simple, si tiene el compositor instalado, escriba esto en un terminal / indicador de comando:

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

Esto creará un nuevo directorio llamado "ejemplo" en el directorio actual, con una instalación estándar de framework Symfony.

Debe instalar estos 2 paquetes: JMSSerializer Bundle (extiende el componente serializer del framework) y FOSRest Bundle (extiende el enrutamiento y los controladores de los componentes del framework ...)

Puedes hacer esto así (en el directorio de ejemplo):

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

¡No olvides activarlos en AppKernel!

Aquí no puedes usar:

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

Porque está basado en la versión Symfony 2.8.

Primero, crea tu propio ("Ejemplo") Bundle (en el directorio de Symfony):

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

Imagina que queremos hacer CRUD (Crear / Leer / Actualizar / Eliminar) de esta Entidad StackOverFlower:

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

Configure su paquete:

```
#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 }
```

Generar esta entidad:

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

Hacer un controlador:

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

namespace ExampleBundle\Controller;

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

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

use ExampleBundle\Entity\StackOverFlower;

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

        $id = $request->get('id');
        $user = $this->getDoctrine()->getManager()-
```

```
>getRepository("ExampleBundle:StackOverFlower")->findOneBy(array('id' => $id));

        return $user;
    }

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

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

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

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

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

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

            $em->flush();

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

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

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

        }

        return $view;
    }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    /**
     * getStackOverFlowerAction
     *
     * @Get("/stackoverflowers")
     *
     * @param Request $request
```

```
     * @return type
     */
    public function getStackOverFlowerAction(Request $request) {

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

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

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

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

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

¡No me digas que es un controlador gordo, es para el ejemplo!

Crea tu plantilla:

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

Puede ver un ejemplo completo de este código en mi cuenta de GitHub , una sucursal con más rutas reales ...

Este es un ejemplo muy básico, no deje que en el entorno de producción, debe proteger su api con apikey.

Un ejemplo de futuro, puede ser?

Lea Trabajar con servicios web en línea: https://riptutorial.com/es/symfony3/topic/6972/trabajar-con-servicios-web

Capítulo 10: Validación

Observaciones

De hecho, la validación de formularios se basa en un componente, denominado " **Componente Validador** ".

A menudo puede usar el servicio dedicado si no tuvo que mostrar un formulario en una plantilla. Al igual que las API. Puedes validar los datos de la misma manera, de esta manera:

Por ejemplo, *basado en el documento symfony* :

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

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

Examples

Validación de Symfony usando anotaciones.

- Habilite la validación usando anotaciones en el archivo `app/config/config.yml`

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

- Crear una entidad en el directorio `AppBundle/Entity` . Las validaciones se realizan con anotaciones `@Assert` .

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


namespace AppBundle\Entity;


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

/**
 * Car
 *
 * @ORM\Table(name="cars")
```

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

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

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


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

    /**
```

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

        return $this;
    }

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

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

        return $this;
    }

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

- Crear un nuevo formulario en el directorio `AppBundle/Form` .

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

namespace AppBundle\Form;

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

```
use Symfony\Component\Form\Extension\Core\Type\IntegerType;

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

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

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

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

- Cree una nueva ruta y un nuevo método de acción en `AppBundle/Controller/DefaultController.php` . La ruta también se declarará con anotaciones, así que asegúrese de haber importado esta ruta en el archivo de ruta principal (`app/config/routing.yml`).

```
<?php

namespace AppBundle\Controller;

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

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

```
        $car = new Car();

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

        $form->handleRequest($request);

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

- Cree la vista en `AppBundle/Resources/views/Default/car.html.twig` .

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

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

- Inicia el servidor integrado de Symfony (`php bin/console server:run`) y accede a la ruta `127.0.0.1:8000/car` en tu navegador. Debe haber un formulario que consta de dos cuadros de entrada y un botón de envío. Si presiona el botón Enviar sin ingresar ningún dato en los cuadros de entrada, se mostrarán los mensajes de error.

Validación de Symfony usando YAML

- Crear una entidad en el directorio `AppBundle/Entity` . Puedes hacerlo manualmente, o usando el comando `php bin/console doctrine:generate:entity` Symfony `php bin/console doctrine:generate:entity` y completa la información requerida en cada paso. Debe especificar la opción `yml` en el paso `Configuration format (yml, xml, php or annotation)` .

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

namespace AppBundle\Entity;

/**
```

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

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

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


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

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

        return $this;
    }

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

    /**
     * Set age
     *
     * @param integer $age
     *
     * @return Person
     */
```

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

        return $this;
    }

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

- Cree la información de asignación de entidad para la clase de entidad. Si estás usando el comando `php bin/console doctrine:generate:entity` Symfony, el siguiente código se generará automáticamente. De lo contrario, si no usa el comando, puede crear el siguiente código a mano.

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

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

- Crear la validación para la clase Entidad.

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

AppBundle\Entity\Person:
    properties:
        name:
            - NotBlank:
                message: "Name is required"
            - Length:
                min: 3
                max: 50
                minMessage: "Please use at least 3 chars"
                maxMessage: "Please use max 50 chars"
            - Regex:
```

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

- Crear un nuevo formulario en el directorio `AppBundle/Form` .

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

namespace AppBundle\Form;

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

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

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

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

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

```
    }
}
```

- Cree una nueva ruta en `AppBundle/Resources/config/routing.yml`

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

- Ahora crea un nuevo método de acción para esa ruta.

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

namespace AppBundle\Controller;

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

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

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

        $form->handleRequest($request);

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

- Cree la vista en `AppBundle/Resources/views/Default/person.html.twig`

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

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

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

- Inicia el servidor integrado de Symfony (`php bin/console server:run`) y accede a la ruta `127.0.0.1:8000/person` en tu navegador. Debe haber un formulario que consta de dos cuadros de entrada y un botón de envío. Si presiona el botón Enviar sin ingresar ningún dato en los cuadros de entrada, se mostrarán los mensajes de error.

Lea Validación en línea: https://riptutorial.com/es/symfony3/topic/6457/validacion

Creditos

S. No	Capítulos	Contributors
1	Empezando con symfony3	Community, insertusernamehere, Paweł Kolanowski, Raphael Schubert, Tartare2240, TRiNE, uddhab, YoannFleuryDev
2	Configuración	Pierre de LESPINAY, Stephan Vierkant
3	Despachador de eventos	Chris Tickner
4	Enrutamiento	Alvin Bunk, Anjana Silva, Hidde, Mathieu Dormeval, Matteo, Renato Mefi
5	Entidades declarantes	Dan Costinel, janek1, Nicodemuz, rubenj
6	Formas dinamicas	Alsatian
7	Gestión de Activos con Assetic	Aaron Belchamber, Orlando
8	Pruebas	Hendra Huang, Pierre de LESPINAY
9	Trabajar con servicios web	Mathieu Dormeval, Pascal, Pierre de LESPINAY
10	Validación	Dan Costinel, Mathieu Dormeval