EBook Gratuito

APPRENDIMENTO symfony3

Free unaffiliated eBook created from
Stack Overflow contributors.

#symfony3

Sommario

Di

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

Capitolo 1: Iniziare con symfony3

Osservazioni

Questa sezione fornisce una panoramica su cosa sia symfony3 e perché uno sviluppatore potrebbe volerlo usare.

Dovrebbe anche menzionare qualsiasi argomento di grandi dimensioni all'interno di symfony3 e collegarsi agli argomenti correlati. Poiché la documentazione di symfony3 è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Versioni

Versione	Data di rilascio
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. Sistemi Windows

È necessario aggiungere php alla variabile di ambiente del percorso. Segui questi passaggi:

Windows 7 :

- Fare clic con il tasto destro sull'icona Risorse del computer
- Clicca Proprietà
- Fai clic su Impostazioni di sistema avanzate dal menu di navigazione sinistro
- Fai clic sulla scheda Avanzate
- Fare clic sul pulsante Variabili d'ambiente
- Nella sezione Variabili di sistema, selezionare Percorso (senza distinzione tra maiuscole e minuscole) e fare clic sul pulsante Modifica
- Aggiungi un punto e virgola (;) alla fine della stringa, quindi aggiungi il percorso completo del file system dell'installazione PHP (ad esempio `C:\Program Files\PHP`)
- Continua a fare clic su OK ecc. Fino a quando tutte le finestre di dialogo non sono

scomparse
- Chiudi il prompt dei comandi e aprilo di nuovo
- smistato

Windows 8 e 10

- In Cerca, cerca e quindi seleziona: Sistema (Pannello di controllo)
- Fai clic sul link Impostazioni avanzate di sistema.
- Fai clic su Variabili d'ambiente.
- Nella sezione Variabili di sistema, trova la variabile d'ambiente PATH e selezionala. Fai clic su Modifica. Se la variabile di ambiente PATH non esiste, fare clic su Nuovo.
- Aggiungi il percorso completo del file system dell'installazione di PHP (ad es. `C:\Program Files\PHP`)

Fatto ciò, apri la console di comando ed esegui il seguente comando:

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

Quindi, sposta il file symfony scaricato nella directory del tuo progetto ed eseguilo come segue:

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

4. Creazione dell'applicazione Symfony

Una volta che Symfony Installer è disponibile, crea la tua prima applicazione Symfony con il nuovo comando:

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

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

Questo comando può essere eseguito da qualsiasi luogo, non necessariamente dalla cartella `htdocs` .

Questo comando crea una nuova directory chiamata `my_project_name/` che contiene un nuovo progetto basato sulla versione di Symfony stabile più recente disponibile. Inoltre, l'installer verifica se il proprio sistema soddisfa i requisiti tecnici per l'esecuzione delle applicazioni Symfony. In caso contrario, verrà visualizzato l'elenco delle modifiche necessarie per soddisfare tali requisiti.

1. Installazione di Symfony Installer

> L'installer richiede PHP 5.4 o versioni successive. Se si utilizza ancora la versione legacy di PHP 5.3, non è possibile utilizzare Symfony Installer. Leggi la creazione di Symfony Applications senza la sezione Installer per sapere come procedere. - fonte: http://symfony.com/doc/current/book/installation.html

5. Basare il tuo progetto su una versione specifica di Symfony

Nel caso in cui il progetto debba essere basato su una versione specifica di Symfony, utilizzare il secondo argomento opzionale del nuovo 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
```

L'installer supporta anche una versione speciale chiamata lts che installa la versione di Symfony LTS più recente disponibile:

```
$ symfony new my_project_name lts
```

Leggi il processo di rilascio di Symfony per capire meglio perché ci sono diverse versioni di Symfony e quale usare per i tuoi progetti.

Puoi anche creare applicazioni symfony senza il programma di installazione, ma non è stata una buona idea. Se vuoi comunque, segui il tutorial originale su questo link:

Oficial Symfony Docs, Configurazione di Symfony senza il programma di installazione

2. Sistemi Linux e Mac OS X.

Apri la console di comando ed esegui i seguenti comandi:

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

Esempio più semplice in Symfony

1. Installa symfony correttamente come spiegato sopra.
2. Avvia il server symfony se non sei installato nella directory www.
3. Accertarsi che http: // localhost: 8000 funzioni se viene utilizzato il server symfony.
4. Ora è pronto per giocare con l'esempio più semplice.
5. Aggiungi il seguente codice in un nuovo file /src/AppBundle/Controller/MyController.php nella directory di installazione di symfony.
6. Prova l'esempio visitando http: // localhost: 8000 / hello
7. È tutto. Successivo: usa il ramoscello per rendere la risposta.

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

Creazione della pagina

> Prima di continuare, assicurati di aver letto il capitolo Installazione e di poter accedere alla tua nuova app Symfony nel browser.

Supponiamo di voler creare una pagina - / lucky / number - che generi un numero fortunato (bene, casuale) e lo stampi. Per fare ciò, crea un "Controller class" e un metodo "controller" al suo interno che verrà eseguito quando qualcuno andrà in / 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>'
        );
    }
}
```

Leggi Iniziare con symfony3 online: https://riptutorial.com/it/symfony3/topic/985/iniziare-con-symfony3

Capitolo 2: analisi

Examples

Semplice test in Symfony3

Test unitario

I test unitari vengono utilizzati per garantire che il codice non abbia errori di sintassi e per testare la logica del codice affinché funzioni come previsto. Esempio veloce:

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%
    }
}
```

test / 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);
    }
}
```

Ho testato la mia classe BillCalculator per garantire che il mio BillCalculator restituisca il prezzo totale dei prodotti + 10% di tasse. Nel test unitario, creiamo il nostro caso di test. In questo test, fornisco 2 prodotti (i prezzi sono 100 e 200), quindi l'imposta sarà del 10% = 30. Mi aspetto che il TaxCalculator restituisca 30, in modo che il prezzo totale sia 300 + 30 = 330.

Test funzionale

I test funzionali vengono utilizzati per testare l'input e l'output. Con l'input fornito, mi aspettavo qualche output senza testare il processo per creare l'output. (questo è diverso con il test dell'unità perché nel test dell'unità testiamo il flusso del codice). Esempio veloce:

```
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'),
            // ...
        );
    }
}
```

Ho testato il mio controller in modo da garantire che il mio controller restituisca 200 risposte invece di 400 (non trovato) o 500 (errore interno del server) con l'url specificato.

Riferimenti:

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

Leggi analisi online: https://riptutorial.com/it/symfony3/topic/2881/analisi

Capitolo 3: Configurazione

introduzione

Esempi e buone pratiche per la configurazione dell'applicazione Symfony non presenti nella documentazione ufficiale.

Examples

Include tutti i file di configurazione da una directory

Dopo un po ', si finisce con molti elementi di configurazione in config.yml. Può semplificare la lettura della configurazione se dividi la tua configurazione su più file. Puoi facilmente includere tutti i file da una directory in questo modo:

config.yml:

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

Nella directory `includes` puoi inserire ad esempio doctrine.yml, swiftmailer.yml, ecc.

Utilizzare il nome di classe completo (FQCN) come id di servizio

In molti esempi, troverai un id di servizio come "acme.demo.service.id" (una stringa con punti). You `services.yml` sarà simile a questo:

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

Nel tuo controller, puoi utilizzare questo servizio:

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

Anche se non vi è alcun problema con questo, è possibile utilizzare un nome di classe pienamente qualificato (FQCN) come id di servizio:

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

Nel tuo controller puoi usarlo in questo modo:

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

Questo rende il tuo codice più comprensibile. In molti casi non ha senso avere un id di servizio che non sia solo il nome della classe.

A partire da Symfony 3.3, è possibile rimuovere anche l'attributo di `class` se l'id di servizio è un FQCN.

Non è necessaria l'interfaccia HTTP?

Se la tua applicazione non ha bisogno di alcuna interfaccia HTTP (ad esempio per un'app solo console), dovrai disabilitare almeno `Twig` e `SensioFrameworkExtra`

Basta commentare queste righe:

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
```

È inoltre possibile rimuovere i requisiti del fornitore correlato da **composer.json** :

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

L'uso di Symfony in questo caso è discutibile, ma almeno può essere temporaneo.

Leggi Configurazione online: https://riptutorial.com/it/symfony3/topic/9175/configurazione

Capitolo 4: Dichiarare le entità

Examples

Dichiarare un'entità Symfony come YAML

- AppBundle / Entità / 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: { }
```

- Crea la tabella

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

Oppure usa il metodo consigliato (ammesso che tu abbia già installato doctrine-migrations-bundle nel tuo progetto):

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

- Crea l'entità automaticamente da YAML

  php bin / console doctrine: generate: entità AppBundle

Dichiarare un'entità Symfony con annotazioni

- AppBundle / Entità / 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;
    }
}
```

- Genera l'entità

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

- Per scaricare le istruzioni SQL sullo schermo

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

- Crea la tabella

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

Oppure usa il metodo consigliato (ammesso che tu abbia già installato doctrine-migrations-bundle nel tuo progetto):

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

Leggi Dichiarare le entità online: https://riptutorial.com/it/symfony3/topic/4443/dichiarare-le-entita

Capitolo 5: Dispatcher di eventi

Sintassi

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

Osservazioni

- È spesso preferibile utilizzare una singola istanza di EventDispatcher nell'applicazione che si inietta negli oggetti che devono generare eventi.
- È consigliabile disporre di un'unica posizione in cui gestire la configurazione e aggiungere listener di eventi al proprio EventDispatcher. Il framework Symfony utilizza il contenitore di iniezione delle dipendenze.
- Questi pattern ti permetteranno di cambiare facilmente i tuoi listener di eventi senza dover cambiare il codice di qualsiasi modulo che sta inviando eventi.
- Il disaccoppiamento della distribuzione degli eventi dalla configurazione del listener di eventi è ciò che rende Symfony EventDispatcher così potente
- EventDispatcher ti aiuta a soddisfare il Principio Aperto / Chiuso.

Examples

Avvio rapido di Event Dispatcher

```
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);
```

Abbonati agli eventi

```
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
    }
});
```

Leggi Dispatcher di eventi online: https://riptutorial.com/it/symfony3/topic/5609/dispatcher-di-eventi

Capitolo 6: Gestione delle risorse con Assetic

introduzione

Quando si utilizza il pacchetto Assetic, in base alla documentazione di Symfony, tenere presente quanto segue:

A partire da Symfony 2.8, Assetic non è più incluso di default in Symfony Standard Edition. Prima di utilizzare una delle sue funzioni, installa AsseticBundle eseguendo questo comando della console nel tuo progetto:

$ compositore richiede symfony / assetic-bundle

Ci sono altri passi da fare. Per maggiori informazioni vai su:
http://symfony.com/doc/current/assetic/asset_management.html

Parametri

Nome	Esempio
Sentiero	'/ Immagini / marchio / logo-Default.png statica'

Osservazioni

La cartella per le risorse accessibili pubblicamente in un progetto standard di Symfony3 è "/ web". Assetic utilizza questa cartella come cartella principale per le risorse.

Examples

Crea un percorso relativo per la risorsa

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

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

Crea un percorso assoluto per la risorsa

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

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

Leggi Gestione delle risorse con Assetic online:
https://riptutorial.com/it/symfony3/topic/6409/gestione-delle-risorse-con-assetic

Capitolo 7: Lavorare con i servizi Web

Examples

API Rest

Ho precedentemente scritto la documentazione su questo sito per descrivere come creare servizi web su Symfony

Scriverò di nuovo un tutorial per la versione symfony> = 3.

Pensiamo di avere un web server installato su una versione configurata di Symfony Framework . È necessario avere anche il compositore (gestore dei pacchetti php).

Per renderlo semplice, se hai installato il compositore, digita questo in un terminale / prompt dei comandi:

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

Questo creerà una nuova directory chiamata "esempio" nella directory corrente, con un'installazione standard del framework symfony.

È necessario installare questo pacchetto 2: JMSSerializer Bundle (estende il serializzatore della componente framework) e FOSRest Bundle (estende il routing ei controller dei componenti framework ...)

Puoi farlo in questo modo (nella directory di esempio):

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

Non dimenticare di attivarli in AppKernel!

Qui non puoi usare:

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

Perché è basato sulla versione di Symfony 2.8.

Per prima cosa crea il tuo bundle ("Esempio") (nella directory di Symfony):

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

Immagina di voler creare CRUD (Crea / Leggi / Aggiorna / Elimina) di questa entità 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
```

Configura il tuo pacchetto:

```
#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 }
```

Genera questa entità:

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

Crea un controller:

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

namespace ExampleBundle\Controller;

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

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

use ExampleBundle\Entity\StackOverFlower;

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

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

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

        return $user;
    }

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

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

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

        if (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);
    }
}
```

Non dirmi che è un controller grasso, è per l'esempio !!!

Crea il tuo modello:

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

Hai appena creato la tua prima API RESTFul !!!

Puoi testarlo su: http: //your-server-name/your-symfony-path/app_dev.php/stackoverflower/new/test .

Come puoi vedere nel database, è stato creato un nuovo utente con il nome: "test".

Puoi vedere un esempio funzionante completo di questo codice sul mio account GitHub , un ramo con più percorsi reali ...

Questo è un esempio molto semplice, non lasciarlo nell'ambiente di produzione, devi proteggere la tua API con apikey !!!

Un esempio futuro, potrebbe essere?

Leggi Lavorare con i servizi Web online: https://riptutorial.com/it/symfony3/topic/6972/lavorare-con-i-servizi-web

Capitolo 8: Moduli dinamici

Examples

Come estendere ChoiceType, EntityType e DocumentType per caricare le scelte con AJAX.

In Symfony, il ChoiceType incorporato (e EntityType o DocumentType che lo estendono), funzionano in modo basico con una lista di scelta costante.

Se vuoi farlo funzionare con le chiamate ajax, devi cambiarle per accettare qualsiasi altra scelta addizionale.

- **Come iniziare con una lista di scelta vuota?**

  Quando crei il tuo modulo, imposta l'opzione `choices` su un `array()` vuoto `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()));
      }
  }
  ```

  Quindi otterrai un input di selezione vuoto, senza scelte. Questa soluzione funziona per ChoiceType e tutti i suoi figli (EntityType, DocumentType, ...).

- **Come accettare le nuove scelte presentate** :

  Per accettare le nuove scelte, devi renderle disponibili nel menu di scelta del modulo. È possibile modificare il campo modulo in base ai dati inviati con l'evento FormEvent :: PRE_SUBMIT.

  Questo esempio mostra come farlo con un ChoiceType di base:

  ```
  namespace AppBundle\Form;

  use Symfony\Component\Form\AbstractType;

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

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

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

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

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

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

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

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

Le tue scelte inviate sono ora scelte consentite e la convalida incorporata di Symfony ChoiceType non le rifiuterà più.

Se vuoi fare lo stesso con un bambino ChoiceType (EntityType, DocumentType, ...), devi iniettare l'entityManager o il documentManager e fare il datatransformation quando compili le nuove scelte.

Compilare un campo di selezione in base al valore di un altro.

Questo è un esempio per mostrare come modificare le scelte consentite su un campo di selezione della sottocategoria in base al valore del campo di selezione della categoria. Per fare ciò devi rendere dinamiche le tue scelte di sottocategoria sia per lato client che server.

1. Rendi dinamico il modulo sul lato client per le interazioni display / utente

Esempio di modulo dinamico lato client (utilizzando 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);
        });
    });
```

Naturalmente potresti ottenere le scelte da una chiamata AJAX. Questo non è lo scopo di questo esempio.

2. Rendere dinamico il modulo sul lato server per l'inizializzazione / convalida

Esempio di modulo dinamico lato server:

```
    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);
    }
}
```

Leggi Moduli dinamici online: https://riptutorial.com/it/symfony3/topic/5855/moduli-dinamici

Capitolo 9: Routing

introduzione

Una rotta è come associare un URL a un'azione (funzione) in una classe Controller. Il seguente argomento si concentrerà sulla creazione di percorsi, passando i parametri alla classe Controller tramite una rotta usando YAML o annotazione.

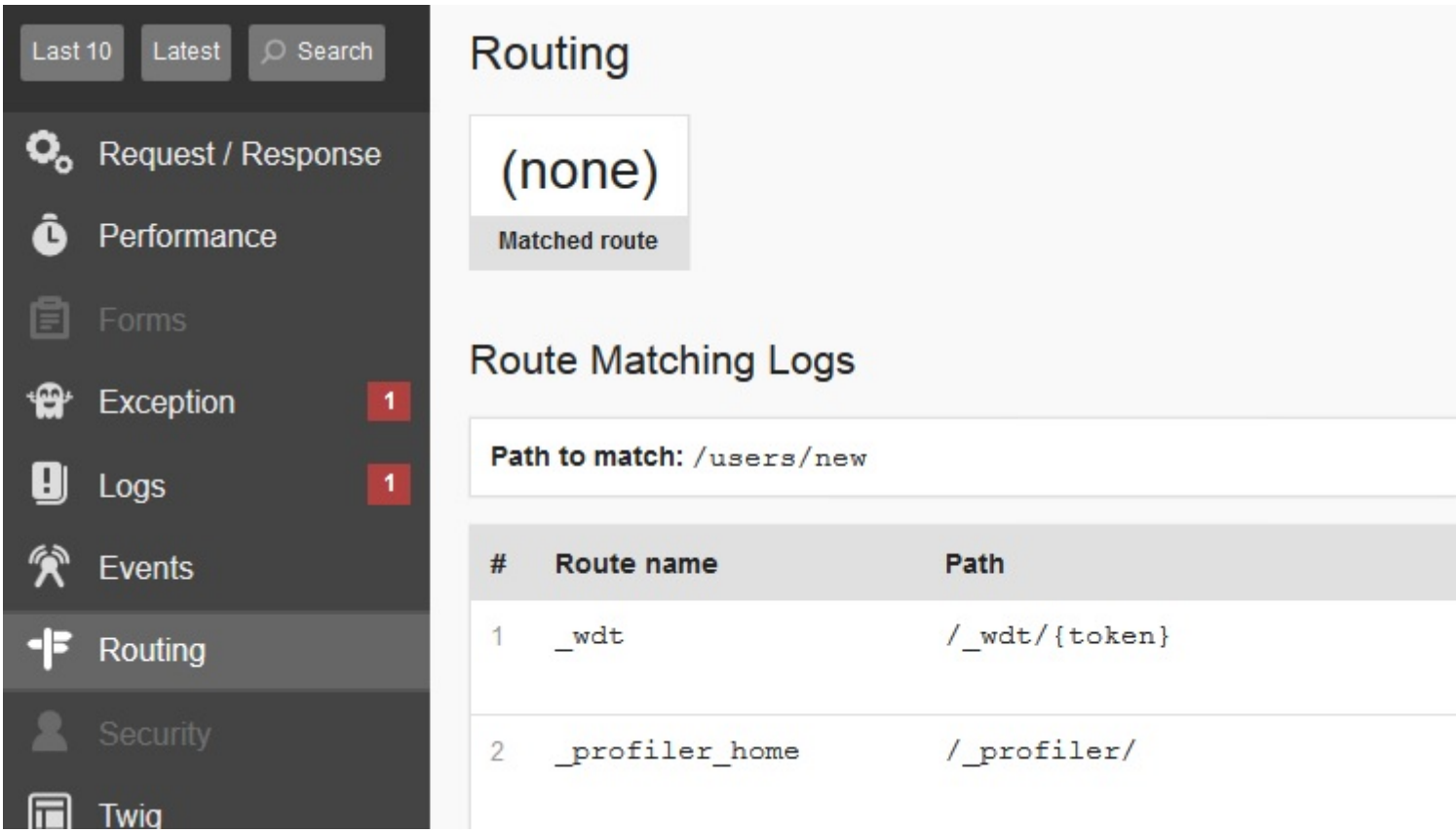
Osservazioni

È utile vedere cosa è generato dal framework Symfony, questo fornisce strumenti per guardare tutti i percorsi di un'applicazione specifica.

Da Symfony Doc , usa (in una shell):

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

Inoltre, puoi vedere tutte le informazioni relative ai percorsi rilevanti nel profiler Framework, nel menu di routing:



Examples

Routing utilizzando YAML

La configurazione del routing è inclusa nel tuo file `app/config/config.yml` , di default il file `app/config/routing.yml` .

Da lì puoi collegare alla tua configurazione di routing in un pacchetto

```
# app/config/routing.yml

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

Potrebbe anche contenere diverse rotte globali dell'applicazione.

Nel tuo bundle puoi configurare, percorsi che hanno due scopi:

- Corrispondenza con una richiesta, in modo tale che venga richiesta l'azione corretta per la richiesta.
- Generazione di un URL dal nome e parametri del percorso.

Di seguito è riportata una configurazione di route YAML di esempio:

```
# 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'"
```

Il percorso è denominato `my_page` e, quando richiesto, richiama `myPageAction` del `DefaultController` in `AppBundle` . Ha un parametro, chiamato `parameter` con un valore predefinito. Il valore è valido solo quando corrisponde alla regex `\d+` . Per questa rotta, sono accettati solo i metodi HTTP `GET` e `PUT` . La `condition` è un'espressione nell'esempio in cui la route non corrisponde a meno che l'intestazione User-Agent non corrisponda a firefox. È possibile eseguire qualsiasi logica complessa necessaria nell'espressione sfruttando due variabili che vengono passate nell'espressione: `context` (RequestContext) e `request` (`request` symfony).

Una rotta generata con il valore del parametro 10 potrebbe essere simile a `/application/content/page/10` .

Routing tramite annotazioni

La configurazione del routing è inclusa nel tuo file `app/config/config.yml` , di default il file `app/config/routing.yml` .

Da lì puoi collegarti ai controller che hanno una configurazione di routing annotata:

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

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

Nel tuo bundle puoi configurare, percorsi che hanno due scopi:

- Corrispondenza con una richiesta, in modo tale che venga richiesta l'azione corretta per la richiesta.
- Generazione di un URL dal nome e parametri del percorso.

Di seguito è riportato un esempio di configurazione del percorso annotato:

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

Il controllore è annotato con un percorso *prefisso* , in modo tale che qualsiasi percorso configurato in questo controller verrà anteposto utilizzando il prefisso.

Il percorso configurato è denominato `my_page` e richiama la funzione `myPageAction` quando richiesto. Ha un parametro, chiamato `parameter` con un valore predefinito. Il valore è valido solo quando corrisponde alla regex `\d+` . Per questa rotta, sono accettati solo i metodi HTTP `GET` e `PUT` .

Si noti che il parametro viene iniettato nell'azione come parametro di funzione.

Una rotta generata con il valore del parametro 10 potrebbe essere simile a `/application/content/page/10` .

Potenti percorsi di riposo

Tradizionalmente, è possibile utilizzare il routing per mappare una richiesta con il componente di routing che ha gestito i parametri di richiesta e risposta tramite HttpFoundation Component .

Inoltre, è possibile creare un parametro del percorso personalizzato utilizzando FOSRestBundle

per estendere le funzionalità predefinite del componente di instradamento.

Questo è utile per creare percorsi REST, un modo davvero utile per specificare come trasferire dati strutturati come XML o file json in una richiesta (e una risposta).

Vedi FOSRestBundle Doc per maggiori informazioni, e specialmente questa annotazione:

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

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

Leggi Routing online: https://riptutorial.com/it/symfony3/topic/2287/routing

Capitolo 10: Validazione

Osservazioni

In effetti, la convalida del modulo si basa su un componente, denominato " **Componente Validatore** ".

È spesso possibile utilizzare il servizio dedicato se non è necessario mostrare un modulo in un modello. Mi piacciono le API. Puoi convalidare i dati nello stesso modo, come questo:

Ad esempio, *basato su symfony doc* :

```
$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

Convalida di symfony usando le annotazioni

- Abilita la convalida usando le annotazioni nel file `app/config/config.yml`

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

- Creare un'entità nella `AppBundle/Entity` . Le convalide vengono eseguite con annotazioni `@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;
    }
}
```

- Crea un nuovo modulo nella `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';
    }
}
```

- Crea un nuovo percorso e un nuovo metodo di azione in `AppBundle/Controller/DefaultController.php` . Anche il percorso verrà dichiarato con annotazioni, quindi assicurati di aver importato questa rotta nel file di percorso principale (`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()
            ]
        );
    }
}
```

- Creare la vista in `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 %}
```

- Avvia il server integrato di Symfony (`php bin/console server:run`) e accedi al percorso `127.0.0.1:8000/car` nel tuo browser. Ci dovrebbe essere una forma composta da due caselle di input e un pulsante di invio. Se si preme il pulsante di invio senza immettere alcun dato nelle caselle di immissione, verranno visualizzati i messaggi di errore.

Convalida di symfony usando YAML

- Creare un'entità nella `AppBundle/Entity` . Puoi farlo manualmente o usando il comando di Symfony `php bin/console doctrine:generate:entity` e riempiendo le informazioni richieste in ogni passaggio. È necessario specificare l'opzione `yml` al passaggio del `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;
    }
}
```

- Creare le informazioni di mapping Entità per la classe Entity. Se stai usando il comando di Symfony `php bin/console doctrine:generate:entity` , il seguente codice verrà generato automaticamente. Altrimenti, se non si utilizza il comando, è possibile creare manualmente il seguente codice.

```
# 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: {  }
```

- Creare la convalida per la classe Entity.

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

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

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

- Crea un nuovo modulo nella `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';
```

```
    }
}
```

- Crea un nuovo percorso in `AppBundle/Resources/config/routing.yml`

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

- Ora crea un nuovo metodo di azione per quella rotta.

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

- Creare la vista in `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 %}
```

- Avvia il server integrato di Symfony (`php bin/console server:run`) e accedi al percorso `127.0.0.1:8000/person` nel tuo browser. Ci dovrebbe essere una forma composta da due caselle di input e un pulsante di invio. Se si preme il pulsante di invio senza immettere alcun dato nelle caselle di immissione, verranno visualizzati i messaggi di errore.

Leggi Validazione online: https://riptutorial.com/it/symfony3/topic/6457/validazione

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con symfony3	Community, insertusernamehere, Paweł Kolanowski, Raphael Schubert, Tartare2240, TRiNE, uddhab, YoannFleuryDev
2	analisi	Hendra Huang, Pierre de LESPINAY
3	Configurazione	Pierre de LESPINAY, Stephan Vierkant
4	Dichiarare le entità	Dan Costinel, janek1, Nicodemuz, rubenj
5	Dispatcher di eventi	Chris Tickner
6	Gestione delle risorse con Assetic	Aaron Belchamber, Orlando
7	Lavorare con i servizi Web	Mathieu Dormeval, Pascal, Pierre de LESPINAY
8	Moduli dinamici	Alsatian
9	Routing	Alvin Bunk, Anjana Silva, Hidde, Mathieu Dormeval, Matteo, Renato Mefi
10	Validazione	Dan Costinel, Mathieu Dormeval