Actividad

Symfony. Forms

1. Introducción formularios en Symfony

Crear y procesar formularios HTML es difícil y repetitivo. Debe ocuparse de la representación de los campos del formulario HTML, la validación de los datos enviados, la asignación de los datos del formulario a los objetos y mucho más. Symfony incluye una potente función de formulario que proporciona todas estas características y muchas más para escenarios realmente complejos.

2. Instalación

En aplicaciones que utilizan Symfony Flex, ejecuta este comando para instalar la función de formulario antes de usarla:

```
composer require symfony/form
```

3. Uso de formularios (flujo)

El flujo de trabajo recomendado cuando se trabaja con formularios de Symfony es el siguiente:

1. **Construir el formulario** en **un controlador Symfony** o utilizando una clase de formulario dedicada;
2. **Representar el formulario** en una plantilla para que el usuario pueda editarlo y enviarlo;
3. **Procesar el formulario** para validar los datos enviados, transformarlos en datos PHP y haga algo con ellos (por ejemplo, conservarlos en una base de datos).

Cada uno de estos pasos se explica en detalle en las siguientes secciones. Para que los ejemplos sean más fáciles de seguir, todos ellos asumen que está creando una pequeña aplicación de lista de tareas pendientes que muestra "tareas".

Los usuarios crean y editan tareas utilizando formularios de Symfony. Cada tarea es una instancia de la siguiente clase Tarea:

Esta clase es un " objeto PHP simple" porque, hasta ahora, no tiene nada que ver con Symfony ni con ninguna otra biblioteca. Es un objeto PHP normal que resuelve directamente un problema dentro *de su aplicación* (es decir, la necesidad de representar una tarea en su aplicación). Pero también puede editar entidades de Doctrine de la misma manera.

4. Tipos y Formulario

Antes de crear tu primer formulario de Symfony, es importante entender el concepto de "tipo de formulario". **En otros proyectos, es común diferenciar**

entre "formularios" y "campos de formulario". En Symfony, todos son "tipos ":

1. un solo campo de formulario <input type=""text"> es un "tipo" (por ejemplo, TextType);
2. un grupo de varios campos HTML utilizados para introducir una dirección postal es un "tipo" (por ejemplo, PostalAddressType);
3. un <formulario> completo con varios campos para editar un perfil de usuario es un "tipo " (por ejemplo, UserProfileType).

Esto puede ser confuso al principio, pero pronto se hará más fácil de entender . Además, simplificaráel código y hace que la "composición" e "incrustación" de los campos de formulario sea mucho más fácil de implementar.

Hay decenas de tipos de formularios proporcionados por Symfony y también puedes crear tus propios tipos de formularios.

Puedes ver todos los tipos existentes con este comando:

```
php bin/console debug:form
```

IMPORTANTE: deberás conocer los diferentes tipos especialmente cuando tengas que crearlos, luego recuerda este comando

5. Construcción de formularios

Symfony proporciona un objeto "constructor de formularios" que permite describir los campos del formulario utilizando una interfaz. Más tarde, este generador crea el objeto de formulario real que se usa para representar y procesar el contenido.

Existen varias formas de crear formularios. Básicamente veremos dos: creación en el controlador y creación del formulario como una clase

5.1. Creación de formularios en controladores

Si tienes un controlador se extiende desde AbstractController (en nuestros ejemplos es así), usa el asistente createFormBuilder():

Añade este nuevo método en el controlador de tareas.

```php
#[Route('/tarea_nueva', name: 'tarea_crear')]
public function nueva(Request $request): Response
{
    // crear un objeto para que el builder lo construya un formulario
    $task = new Tarea();
    $task->setNombre('tarea 1');
    $task->setPrioridad(1);
    $task->setDescripcion('descripción');

    // construir el formulario
    $form = $this->createFormBuilder($task)
        ->add('nombre', TextType::class)
        ->add('prioridad', IntegerType::class)
        ->add('descripcion', TextType::class)
        ->add('crear', SubmitType::class, ['label' => 'Crear tarea'])
        ->getForm();

    // renderizar formulario
        return $this->render('tarea/new.html.twig', [
            'form' => $form,
        ]);
}
```

Fíjate que en el método render se usa una plantilla. Debes crearla en la carpeta templates/tarea, con nombre templates/tarea/new.html.twig y su contenido es el siguiente

```
{# templates/tarea/new.html.twig #}
{{ form(form) }}
```

Esta plantilla únicamente llama a la función form que imprime el contenido del formulario en HTML.

Pruébalo

5.2. Creación de Clases que representan formularios

Symfony recomienda poner la menor lógica posible en los controladores. Es por eso por lo que **es mejor mover los formularios complejos y su crecaión a clases dedicadas** en **lugar de definirlos en acciones de controlador**. Además, los formularios definidos en clases se pueden reutilizar en múltiples acciones y servicios.

Las clases de formulario son tipos que implementan la interfaz FormTypeInterface. Sin embargo, es mejor extender desde AbstractType, que ya implementa la interfaz y proporciona algunas utilidades ya disponibles:

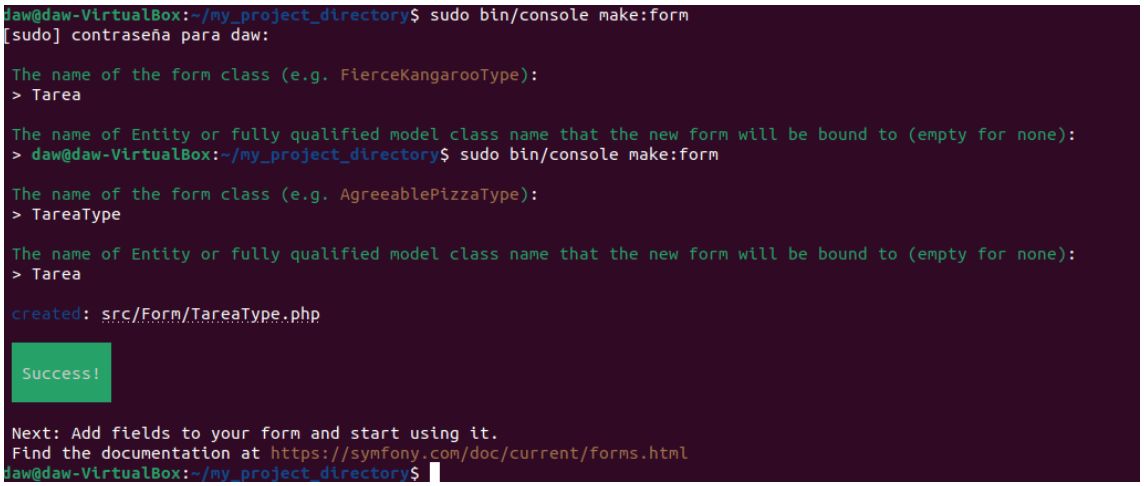
Vamos a reescribir el controlador de tarea para que el método anterior para crear un formulario para una nueva tarea se haga con una clase de tipo formulario tarea (TareaType)

Debes tener instalado (MakerBundle) para hacer lo siguiente (ya lo tenemos instalado cuando instalamos Doctrine).

Ejecutamos el comando siguiente

```
bin/console make:form
```

Este comando nos iniciará **un asistente para comenzar la creación una clase que represente al formulario, la tenemos que llamar** TareaType. Sigue las siguientes instrucciones para ello:



Si vamos a la clase generada podemos ver la clase creada TareaType

```php
<?php

namespace App\Form;

use App\Entity\Proceso;
use App\Entity\Tarea;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class TareaType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nombre')
            ->add('prioridad')
            ->add('descripcion')
            ->add('proceso', EntityType::class, [
                'class' => Proceso::class,
'choice_label' => 'id',
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Tarea::class,
        ]);
    }
}
```

IMPORTANTE: Fíjate que **en configureOptions es donde vinculamos el formulario con la clase que le da sentido**.

Ahora vas a cambiar el controlador para que use esta nueva forma de crear un formulario, pon el siguiente código en el controlador:

```php
#[Route('/tarea_nueva', name: 'tarea_crear')]
public function nueva(Request $request): Response
{

        $form = $this->createForm(TareaType::class);

        return $this->render('tarea/new.html.twig', [
            'form' => $form,
        ]);
}
```

Como ves, este código es más compacto y sencillo.

6. Formularios representación (render)

Internamente, **el método render()** llama a $form->createView() para transformar el formulario en una instancia de vista de formulario .

Existen muchas funciones funciones auxiliares para representar el contenido del formulario:

Lo más simple que puedes hacer es:

```
{# templates/tarea/new.html.twig #}
{{ form(form) }}
```

¡Eso es todo! La función form() representa todos los campos *y las* etiquetas de inicio y fin <form>. De forma predeterminada, el método de formulario es POST y la dirección URL de destino es la misma que mostraba el formulario, pero se puede cambiar ambas.

Vamos a hacer que el objeto que salgan datos por defecto en el formulario

```php
#[Route('/tarea_nueva', name: 'tarea_crear')]
public function nueva(Request $request): Response
{
    // crear un objeto de ejemplo
    $tarea = new Tarea();
    $tarea->setNombre('tarea 1');
    $tarea->setPrioridad(1);
    $tarea->setDescripcion('descripción');

    $form = $this->createForm(TareaType::class,$tarea);

    return $this->render('tarea/new.html.twig', [
        'form' => $form,
    ]);
}
```

Observa cómo el campo de entrada de la tarea tiene el valor de la propiedad de tarea del objeto $tarea (Este es el primer trabajo de un formulario: tomar datos de un objeto y traducirlos a un formato adecuado para ser representado en un formulario HTML.

El sistema de formularios es lo suficientemente inteligente como para acceder al valor de la propiedad través de los métodos get y set() en la clase Tarea. Toda entidad debe tener un método "getter" y "setter" para que Symfony pueda obtener y colocar datos en la propiedad.

A pesar de lo corto que es este renderizado, **no es muy flexible**. **Por lo general, necesitarás más control sobre el aspecto de todo el formulario** o algunos de sus campos. Por ejemplo, gracias a la integración de Bootstrap 5 con los formularios de Symfony, puedes configurar esta opción para generar formularios compatibles con el framework CSS de Bootstrap 5: