

SYMPHONY 7

CREACIÓN DE FORMULARIOS

INTRODUCCIÓN:

Una de las mejores cosas que tiene Symfony, es el manejo y creación de los formularios.

Con symfony tenemos un componente que automatiza gran parte de la creación, y recepción de los datos de formularios, pudiendo añadir fácilmente tokens de seguridad, generar formularios responsivos con Bootstrap de forma automática, generarlos automáticamente a partir de la definición de la base de datos, y un largo muy largo etcétera.

Hay multitud de opciones disponibles para trabajar con los campos de los formularios

INTRODUCCIÓN:

- Mediante el componente ***symfony/form*** gran parte de la generación de formularios se hace en el servidor.
- Los formularios se confeccionan en PHP, despues se pasan a la plantilla Twig como un objeto, desde donde se dibujan para mostrarlos.
- De esta forma nos ahorramos generar todo el HTML, CSS y Javascript que antes teníamos que hacer. Incluso si usamos la plantilla Bootstrap para los formularios haremos que esta generación sea responsiva.
- Además, tenemos un montón de validaciones automáticas de los campos.

INTRODUCCIÓN:

- Un formulario siempre debe ser representado por un objeto que se le conoce como **Type**.
- Este objeto recibirá un Entity, que será donde se almacenan los datos cargados en el formulario.
- Hay multitud de opciones disponibles para trabajar con los campos de los formularios.

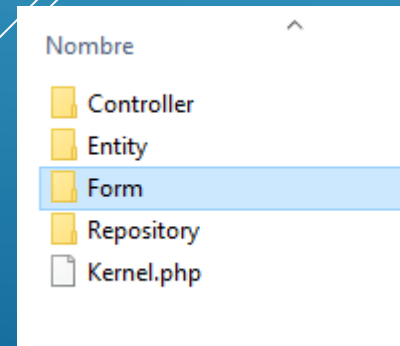
CREACIÓN DE FORMULARIOS

Los formularios se crean dentro del directorio scr, y dentro de este en el directorio Form,

- Los formularios por convenio se llaman con el nombre de la entidad seguido del sufijo Type: **EntidadType.php**
- Para nuestro ejemplo tomaremos el Entity Profesion, y crearemos un objeto Type para representar a este formulario.
- Pero lo **primero que hay que crear es el controlador ProfesionController** que es el recurso donde vamos a trabajar tanto con el formulario como con la entidad, si lo creamos después del formulario **dará error la creación del controlador**.
- El comando para generar el Controlador es:
 - **php bin/console make:controller ProfesionController**
- El comando para generar el formulario es:
 - **php bin/console make:form**

CREACIÓN DE FORMULARIOS

- Nos preguntará:
 - El nombre del formulario: ProfessionType
 - Con que entidad lo relaciono, esto lo hace para tomar sus columnas como entradas del formulario, de esta forma recoge los datos a través de estas entradas del formulario, y luego guardarlo en la bbdd(tabla profesion)
 - Con esto tengo hecho el formulario, puedo ver la estructura en dentro de scr en la nueva carpeta que se ha creado:Form



```
C:\wamp64\www\miAppSymfony2>php bin/console make:controller
```

```
Choose a name for your controller class (e.g. DeliciousChefController):
```

```
> ProfesionController
```

```
Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:
```

```
> no
```

```
created: src/Controller/ProfesionController.php
```

```
created: templates/profesion/index.html.twig
```

Success!

```
C:\wamp64\www\miAppSymfony2>php bin/console make:form
```

```
The name of the form class (e.g. GrumpyGnomeType):
```

```
> ProfesionType
```

```
The name of Entity or fully qualified model class name that the new form
```

```
> Profesion
```

```
created: src/Form/ProfesionType.php
```

Success!

```
Next: Add fields to your form and start using it.
```

```
Find the documentation at https://symfony.com/doc/current/forms.html
```


DEFINICIÓN DEL FORMULARIO

- Una vez creado, en `ProfesionType`, voy a tener las siguientes clases directamente importadas
 - `namespace AppBundle\Form`: Define el espacio de nombres de formulario
 - `use Symfony\Component\Form\AbstractType`: Nos incluye las funcionalidades base de los formularios
 - `use Symfony\Component\Form\FormBuilderInterface`: Es el que nos permite construir el formulario y agregar sus campos con el método `add`
 - `use Symfony\Component\OptionsResolver\OptionsResolver`: Indica a la entidad de donde vamos a coger y /o guardarlos datos.

DEFINICIÓN DEL FORMULARIO

A las clases que vienen predefinidos hay que añadir las clases de los tipo de entradas que queremos seleccionar, ya que si las entradas las dejamos tal cual, symfony lo construye, y asume que todos son de tipo text, si queremos dar otro formato hay que importar sus clases:

- ✓ use `Symfony\Component\Form\Extension\Core\Type\EmailType`
- ✓ use `Symfony\Component\Form\Extension\Core\Type>PasswordType`
- ✓ use `Symfony\Component\Form\Extension\Core\Type\SubmitType`
- ✓ use `Symfony\Component\Form\Extension\Core\Type\TextType`
- ✓ use `Symfony\Component\Form\Extension\Core\Type\HiddenType`
- ✓ use `Symfony\Component\Form\Extension\Core\Type\DateType`
- ✓ use `Symfony\Component\Form\Extension\Core\Type\CheckboxType`;

Todos los tipos de campos disponibles están en la documentación de symfony:

<https://symfony.com/doc/current/reference/forms/types.html>

CAMPOS NO MAPEADOS.

Al editar un objeto a través de un formulario, todos los campos del formulario se consideran propiedades del objeto. Cualquier campo del formulario que no exista en el objeto provocará que se genere una excepción.

Si necesita campos adicionales en el formulario que no se almacenarán en el objeto (por ejemplo, para agregar una casilla de verificación "Estoy de acuerdo con estos términos"), configure la opción `mapped` a `false` en estos campos que no están incluidos en la `bbdd`

```
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('task')
            ->add('dueDate')
            ->add('agreeTerms', CheckboxType::class, ['mapped' => false])
            ->add('save', SubmitType::class)
        ;
    }
}
```

DEFINICIÓN DEL FORMULARIO

En este punto se revisan los campos que tengo en el formulario y se personalizan.

- Ahora hay que decir si quiero visualizar todo lo incluido en el formulario, o si quiero añadir algún campo de los no mapeados, solo tendré que añadir los add necesarios con sus clases correspondientes.
- El formato general de los campos a añadir es:
->add(NombreCampo, tipo de dato , Array de opciones).

DEFINICIÓN DEL FORMULARIO

Por lo que si queremos, por ejemplo, introducir campos no vinculados a ninguna entidad, como un checkbox de aceptar condiciones , se incluye su clase con el use de datos correspondiente, y en el formulario se pone el mapeo a false para que no de errores en la comprobación de los campos en la tabla :

.....

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

.....

```
->add('acepto', CheckboxType :: class, array('required'=>false, 'mapped'=>false))
```

```
use App\Entity\Profesion;
use App\Entity\User;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ProfesionType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nombre')
            ->add('user', EntityType::class, [
                'class' => User::class,
                'choice_label' => 'id',
            ])
        ;
    }

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

Vamos a añadir un submit para poder y añadir profesiones:

```
use App\Entity\Profesion;
use App\Entity\User;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class ProfesionType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nombre')
            ->add('user', EntityType::class, [
                'class' => User::class,
                'choice_label' => 'id',
            ])
            ->add('enviar', SubmitType::class );
    }

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


←

→

↺

🛡️

📄

Nombre

User

1

▼

Enviar

RELACIONAR FORMULARIO Y CONTROLLER

- Con esto ya tengo el formulario hecho y relacionado con la entidad.
- Ahora para pasarlo a mi template, como ya sabemos se debe hacer a través del controlador ProfesionController, vamos a utilizar el ya creado, pero debo importar la clase del formulario que le une con esta entidad para que sean visibles cuando el controlador haga referencia a estos recursos.
- Ahora debo importar en el controlador la clase del formulario, la de la entidad, la respuesta request y el enlace con la BBDD
- Estos se hace con:
 - `use App\Entity\Profesion;`
 - `use App\Form\ProfesionType;`

Ahora ya son visibles para el controlador.

```
namespace App\Controller;
```

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

```
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\Routing\Attribute\Route;
```

```
use App\Entity\Profesion;
```

```
use App\Form\ProfesionType;
```

```
class ProfesionController extends AbstractController
```

```
{
```

```
    #[Route('/profesion', name: 'app_profesion')]
```

INVOCAR Y VISUALIZAR EL FORMULARIO

El código que debe contener nuestro método es muy sencillo.

- Primeramente creamos un objeto `profesion` y luego, por medio del método **`$this->createForm()`** invocamos a nuestro objeto `ProfesionType` pasándole como parámetro la instancia `$profesion`, recién creada, esto nos devolverá un objeto de tipo formulario.
- A continuación procesamos el formulario, la forma recomendada de procesar formularios de Symfony es, primero detectar que se ha enviado por método `post`, y después utilizar el método `handleRequest()` para detectar que se ha enviado el formulario, es decir se ha pulsado el submit. Esto se realiza con el método **`handleRequest($request)`**.
- Finalmente invocamos a la vista como siempre hacemos, y pasamos como parámetro el resultado de ejecutar **`$form->createView()`**.

CÓDIGO DE LA VISTA(INDEX.HTML.TWIG)

\$form->createView(): Nos permite utilizar un conjunto de funciones helper que definimos en la plantilla.

- En templates/registro se nos ha creado la plantilla index.html.twig donde podemos utilizar los helpers para poder representar el formulario:

```
{{ form(form, {'method': 'GET'}) }}
```

Representa el fomulario completo

```
{{ form_start(form) }}
```

Representa la etiqueta de inicio de un formulario. Este asistente se encarga de imprimir el método configurado y la acción de destino del formulario. También incluirá la propiedad enctype correcta si el formulario contiene campos de carga.

```
{{ form_label(form.name) }}
```

Representa la etiqueta del campo indicado.

```
{{ form_row(form.submit, { 'label': 'Submit me' }) }}
```

función muestra el contenido completo del campo, incluida la etiqueta, el mensaje de ayuda, los elementos HTML y los mensajes de error.

```
{{ form_end(form) }}
```

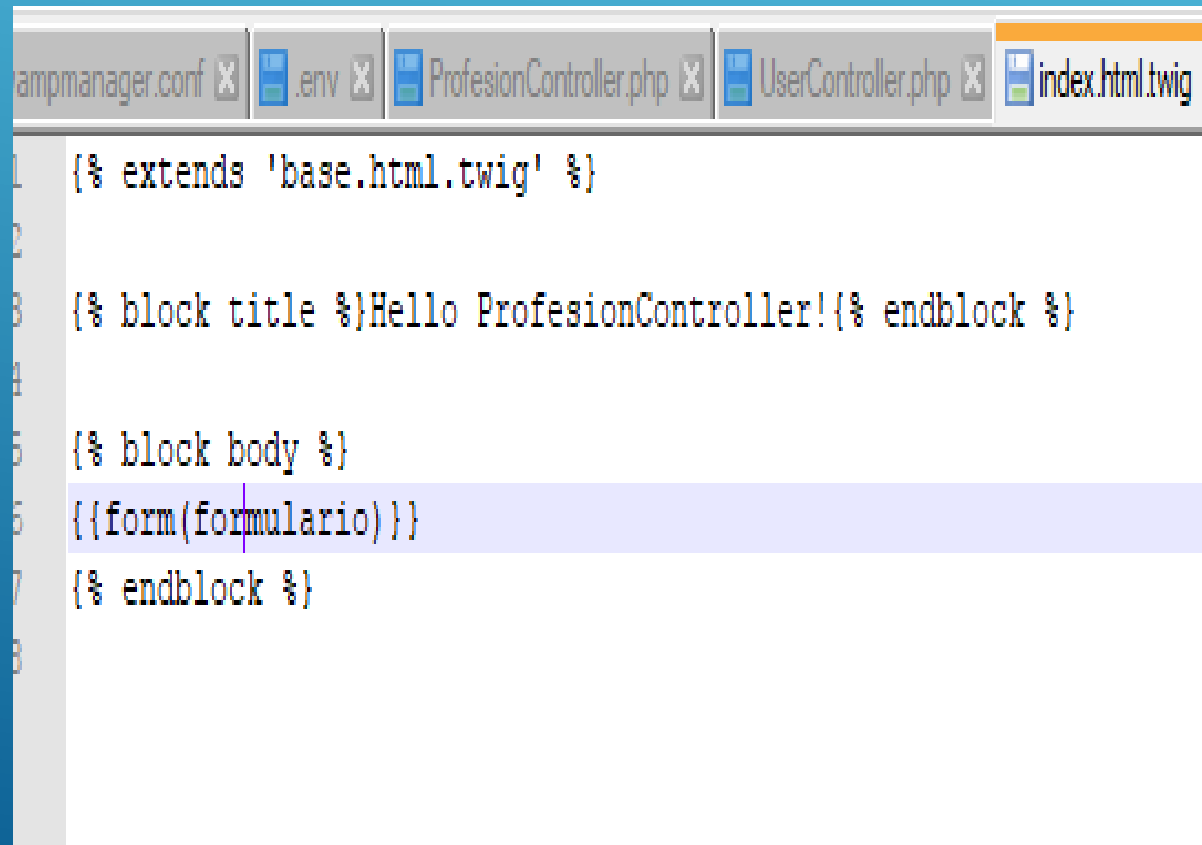
Representa la etiqueta final de un formulario.

```
#[Route('/profesion', name: 'app_profesion')]
public function index(Request $request, EntityManagerInterface $entityManager): Response
{
    $profesion= new Profesion();
    $form= $this->createForm(ProfesionType::class,$profesion);

    if ($request->isMethod('POST')) {
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            //INSTRUCCIONES PARA INSERTAR DATOS A PARTIR DEL FORMULARIO
            return $this->redirect($this->generateUrl('app_listar'));
        }
    }
    return $this->render('profesion/index.html.twig', [
        'controller_name' => 'ProfesionController',
        'formulario'=>$form->createView()
    ]);
}
```

CÓDIGO DE LA VISTA(INDEX.HTML.TWIG)

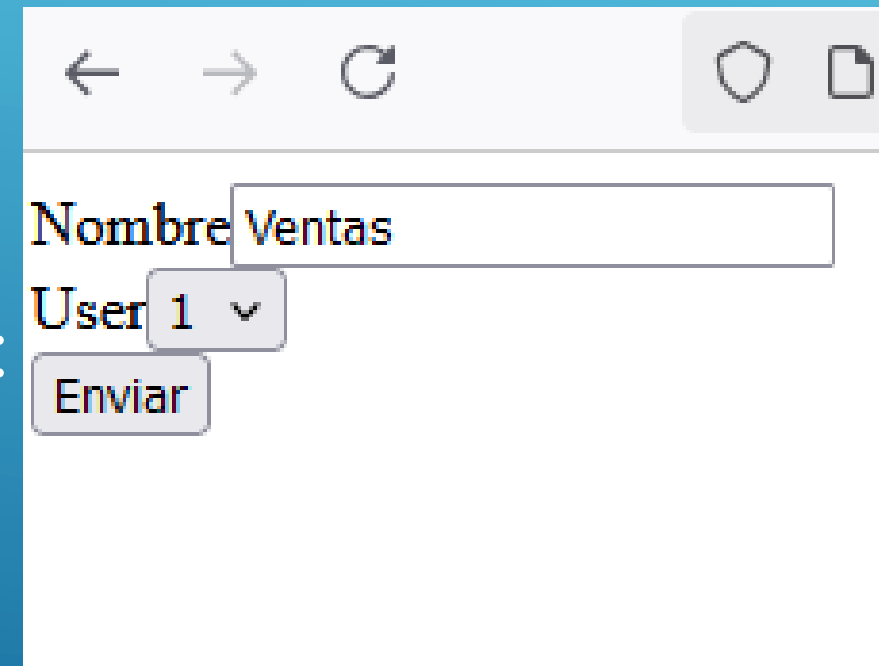
Ahora para visualizar el formulario solo tengo que ir a index.html.twig, y reescribir el body visualizando mi variable formulario:



```
1 {% extends 'base.html.twig' %}
2
3 {% block title %}Hello ProfesionController!{% endblock %}
4
5 {% block body %}
6 {{form(formulario)}}
7 {% endblock %}
8
```

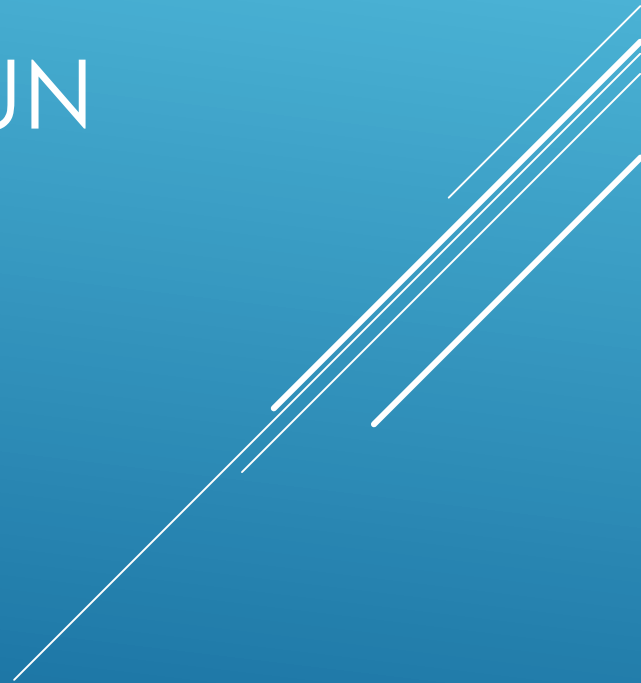

VISUALIZACIÓN DEL FORMULARIO

No tiene ningún estilo, ni bootstrap, está tal y como lo genera el framework:



A screenshot of a web browser window displaying a form. The browser's address bar shows navigation icons (back, forward, refresh) and security icons (shield, document). The form contains two input fields: the first is labeled 'Nombre Ventas' and contains the text 'Nombre Ventas'; the second is a dropdown menu labeled 'User 1' with a downward arrow. Below these fields is a button labeled 'Enviar'.

GENERAR REGISTROS A TRAVÉS DE UN FORMULARIO



GENERAR REGISTROS A TRAVÉS DE UN FORMULARIO".

Para poder hacer algo con el formulario, como por ejemplo insertar en la bbdd, en el controller debemos generar el código que nos permita almacenar información con doctrine.

Almacenar información con doctrine es muy sencillo , para ello necesitamos hacer una referencia al administrador, que es el que une el formulario con las entidades y la bbdd, y llamar a un método que me insertará de forma automática el registro.

El acceso a nuestras entidades se hará por medio de un objeto de Doctrine llamado **EntityManager** que es el administrador de las entidades y el encargado de interactuar con ellas.

GENERAR REGISTROS A TRAVÉS DE UN FORMULARIO

- Para tenerlo accesible hay que añadir al controlador:

use Doctrine\ORM\EntityManagerInterface;

- Para obtener este objeto , dentro de nuestro método lo inyectamos en nuestro método de la siguiente manera:

➤ **public function index(EntityManagerInterface \$entityManager): Response**

Con esto ya tenemos la referencia al administrador de la tarea dentro de una variable que hemos llamado “\$entityManager” .

GENERAR REGISTROS A TRAVÉS DE UN FORMULARIO

- Con el método `persist()` le decimos que el objeto pasado por argumento sea guardado para ser insertado, y la inserción en sí se realizará cuando ejecutemos el método `flush()` que es el que ejecuta las ordenes sql pasadas.
- Con esta orden Doctrine generará la sentencia `INSERT` necesaria.
- Ahora, ya sólo quedaría ejecutarlo desde el navegador y después ir a la base de datos a ver si realmente tenemos los datos de prueba generados en el controlador.

PASOS PARA EJECUCIÓN EN EL CONTROLLER

- Lo primero que tenemos que pensar es que si para procesar el formulario llamamos al mismo método en el que metemos los datos ¿Cómo sabemos cuándo mostrar el formulario y cuándo procesarlo?.
- La respuesta es sencilla, cuando el request fue de tipo GET lo deberíamos de mostrar, pero en caso de que se haya dado click en el botón submit se ejecuta un request de tipo POST y por lo tanto se debería procesar.

GENERAR REGISTROS A TRAVÉS DE UN FORMULARIO

- Como se ha comentado anteriormente ,la forma recomendada de procesar formularios de Symfony es utilizar el método **handleRequest ()** para detectar cuándo se ha enviado el formulario. Para dotar de esta funcionalidad hay que introducir en el controller

- **use Symfony\Component\HttpFoundation\Request;**

Tambien inyectamos el Request en el método de la siguiente forma:

- **public function index(Request \$request, EntityManagerInterface \$entityManager): Response**

Además, también se debe usar el método **isSubmit ()** para tener un mejor control sobre cuándo se envía exactamente su formulario y qué datos se le pasan.

- Junto a la función **isValid ()** que me dá los datos enviados validados a través del formulario.

GENERAR REGISTROS A TRAVÉS DE UN FORMULARIO

- El controlador que vamos a reescribir sigue un patrón común para manejar formularios y tiene tres pasos posibles:
- Al cargar inicialmente la página en un navegador, el formulario aún no se ha enviado por lo que `if ($request->isMethod('POST'))` y `$form->isSubmitted()` devuelven `false`, entonces se crea el formulario y se renderiza.
- Cuando el usuario envía el formulario lo reconoce con `handleRequest()` y `isSubmitted()=true`.
- Cuando el usuario envía el formulario con datos válidos el método `isValid()` = `true`, y por tanto los datos presentados se deben escribir en la base de datos.



localhost/miAppSymfony2/public/index.php/profesion

Nombre

User

1 ▾

Enviar

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use App\Entity\Profesion;
use App\Form\ProfesionType;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;

class ProfesionController extends AbstractController
{
    #[Route('/profesion', name: 'app_profesion')]
    public function index(Request $request, EntityManagerInterface $entityManager): Response
    {
        $profesion= new Profesion();
        $form= $this->createForm(ProfesionType::class,$profesion);

        if ($request->isMethod('POST')) {
            $form->handleRequest($request);
            if ($form->isSubmitted() && $form->isValid()) {
                $entityManager->persist($profesion);
                $entityManager->flush();
                return $this->redirect($this->generateUrl('app_listar'));
            }
        }
        return $this->render('profesion/index.html.twig', [
            'controller_name' => 'ProfesionController',
            'formulario'=>$form->createView()
        ]);
    }
}
```