

DAWS – 2º

Forms con Symfony

JESÚS MOLINA HERNÁNDEZ
jmolina@florida-uni.es

Índice

1. Introducción
2. Cómo generar un formulario
3. Entendiendo el código de un formulario
4. Modificar un formulario
5. Validaciones
6. Usar el formulario

1. Introducción a Forms en Symfony

Formularios en Symfony

- Integración completa con validaciones, plantillas **Twig** y base de datos.
- Soporte para formularios dinámicos, validaciones automáticas y temas personalizables.

Ventajas

- Generación rápida de formularios con lógica reusable.
- Validaciones integradas.
- Soporte para personalización en Twig.

2. Cómo generar un formulario

Tenemos 3 opciones

- (1) Manual
- (2) Mediante el comando `symfony console make:form`
- (3) Automáticamente al ejecutar `symfony console make:crud`

En todos casos, el resultado será
un fichero php en src/Form

3. Entendiendo el código de un formulario

```
$builder->add('nombre_campo', tipo_campo, opciones)
```

- **nombre_campo**: Es el nombre del campo en la entity
- **tipo_campo**: Tipo de campo a mostrar en el form
- **opciones**: Otras opciones de configuración

[Ver en código](#)

3. Entendiendo el código formulario – Tipos de campo

1. Tipos básicos

- `TextType::class`
 - *Entrada de texto estándar.*
- `TextareaType::class`
 - *Entrada de texto largo (multilínea).*
- `EmailType::class`
 - *Entrada de correo electrónico.*
- `PasswordType::class`
 - *Campo de contraseña (oculta el texto al escribir).*
- `NumberType::class`
 - *Campo para ingresar números.*
- `IntegerType::class`
 - *Campo para números enteros.*
- `MoneyType::class`
 - *Campo para manejar cantidades de dinero.*
- `UrlType::class`
 - *Campo para URLs.*

[Ver en código](#)

2. Tipos Booleanos y Selección

- `CheckboxType::class`
 - *Campo de selección única (checkbox).*
- `ChoiceType::class`
 - *Campo para seleccionar opciones.*
- `RadioType::class`
 - *Selección única con botones de radio (usado como parte de ChoiceType con expanded: true).*

3. Tipos Relacionados con Fechas

- `DateType::class`
 - *Campo para fechas.*
- `DateTimeType::class`
 - *Campo para fechas y horas.*
- `TimeType::class`
 - *Campo para horas.*

4. Tipos Relacionados con Archivos

- `FileType::class`
 - *Campo para subir archivos.*

5. Tipos Relacionados con Entidades

- `EntityType::class`
 - *Campo para seleccionar entidades de la base de datos.*

3. Entendiendo el código formulario – Opciones

- **label**: Cambia la etiqueta del campo.
- **required**: Indica si el campo es obligatorio.
- **attr**: Añade atributos HTML al campo.
- **help**: Añade un texto de ayuda debajo del campo.
- **data**: Define un valor por defecto.

[Ver en código](#)

4. Modificar un formulario

El formulario auto-generado por symfony se puede modificar en el código con todo lo visto en la sección anterior

5. Validaciones

Se pueden añadir validaciones en 3 sitios distintos:

- En la definición de la entidad usando #[Assert\...]

```
#[Assert\NotBlank]
private ?string $nombre = null;

#[Assert\GreaterThan(0)]
private ?float $precio = null;
```

- En la creación del formulario usando 'constraints' =>

```
$builder->add('nombre', TextType::class, [
    'constraints' => [
        new NotBlank(['message' => 'El nombre no puede estar vacío.']),
    ],
]);
```

- En Twig usando {{ form_errors() }}

```
 {{ form_errors(form.nombre) }}
```

6. Usar el formulario

Una vez creado el formulario (buildform), se puede usar en nuestro código:

- En el controlador se lanza y recibe la respuesta

```
$form = $this->createForm(ProductoType::class, $producto);
$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
    // Procesa los datos
```

- Renderizar en Twig

```
 {{ form_start(form) }}
 {{ form_widget(form) }}
 {{ form_end(form) }}
```

```
 {{ form_row(form.nombre, { 'label': 'Nombre del Producto' }) }}
 {{ form_row(form.precio, { 'label': 'Precio', 'attr': {'class': 'custom-class'}})}
```

Cambios en la vista

6. Usar el formulario

```
#[Route('/new', name: 'app_producto_new', methods: ['GET', 'POST'])]
public function new(Request $request, EntityManagerInterface $entityManager): Response
{
    $producto = new Producto();
    $form = $this->createForm( type: ProductoType::class, $producto);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager->persist($producto);
        $entityManager->flush();

        return $this->redirectToRoute( route: 'app_producto_index', [], status: Response::HTTP_SEE_OTHER);
    }

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