

SYMPHONY7

MANIPULACIÓN DE DATOS

ACCESO Y MANIPULACIÓN DATOS

Para trabajar con los datos necesitaremos nuestro **repositorio**, que es el encargado de poner a nuestra disposición los distintos cauces para poder obtener registros de nuestra entidad.

Este repositorio es el objeto que nos permite solicitar y actualizar datos y para obtenerlo simplemente usamos el nombre lógico de nuestra entidad de esta manera:

\$entityManager ->getRepository(nombre-entidad::class);

ACCESO Y MANIPULACIÓN DATOS

Para obtener datos de las tablas tenemos varios métodos mágicos:

findAll(): Obtiene todos los registros de la tabla. Retorna un array.

find(): Obtiene un registro **a partir de la clave primaria** de la tabla.

Con los siguientes, puedes extraer objetos bajo múltiples condiciones

findBy(): Obtiene los registros encontrados pudiendo pasar como argumentos los valores que irían dentro del WHERE. Retorna un array.

findOneBy(): obtiene un registro pudiendo pasar como argumentos los valores que irían dentro del WHERE

ACCESO Y MANIPULACIÓN DATOS

- Primero debemos acceder a nuestra entidad mediante el administrador, inyectando en el metodo:
 - **public function index2(EntityManagerInterface \$entityManager): Response**
- Luego aplicamos los métodos mágicos, por ejemplo:
 - para obtener **todos** los registros de la tabla a través del repositorio:
\$vble-recogida = \$entityManager ->getRepository (nombre-entidade::class')->findAll();
 - Para obtener un solo registro se accede mediante su pk (por ejem con id igual a 12):
 - **\$vble-recogida = \$entityManager ->getRepository (nombre-entidade::class)->find(12);**

ACCESO Y MANIPULACIÓN DATOS

- Para obtener los registros cuyo campo columna1 sea igual a " valor-1 "

```
$vble-recogida = $entityManager ->getRepository (nombre-entidade::class)  
->findOneBy(array('columna1' => 'valor1-1'));
```

- Para obtener todos los registros cuyo campo columna1 sea igual a " valor-1 " y columna2 sea igual a " valor-2 "

```
$vble-recogida = $entityManager ->getRepository (nombre-entidade::class')  
->findBy ( array( 'columna1' => 'valor1', 'columna2' => 'valor2' ) );
```

EJEMPLO GUIADO

Realizar un C.R.U.D. no automatizado, que permita insertar, borrar, modificar y listar.

De manera que todas las acciones realizadas se puedan ver mediante el listado automáticamente.

Las acciones que vamos a hacer son:

1. Insertar
2. Actualizar
3. Borrar
4. Listar

EJEMPLO GUIADO

Como ya sabemos las acciones las realizan los controlador, y son los encargados de que la respuesta se muestren por pantalla mediante las plantillas.

Por tanto haremos un controlador que en función de lo que queramos realizar, se dirija a esta acción, y se encargue de pedir que muestre el resultado por pantalla.

Las plantillas que utilizamos para mostrar los resultados serán:

- la plantilla base de los ejercicios anteriores.
- Y una específica donde se hará una tabla para mostrar los resultados en forma de tabla.

Como las inserciones ya las tenemos hechas del ejemplo anterior, vamos a aprovechar este controlador para diseñar todo el C.R.U.D. añadiendo los métodos que faltan.

LISTAR.HTML.TWIG

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

{% block title %}LISTADO DE PROFESIONES{% endblock %}
{% block style %}  {{parent()}}{% endblock %}
{% block body %}
    <h2> PROFESIONES</h2>
    <table border='10px'>
        <tr>
            <td> ID</TD><td> NOMBRE</TD>
        </TR>
        {% for valor in indice%}
            <tr>
                <td>{{valor.id}}</td>
                <td>{{valor.nombre}}</td>
            </tr>
        {%endfor%}
    </table>
{% endblock %}
```



```

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);

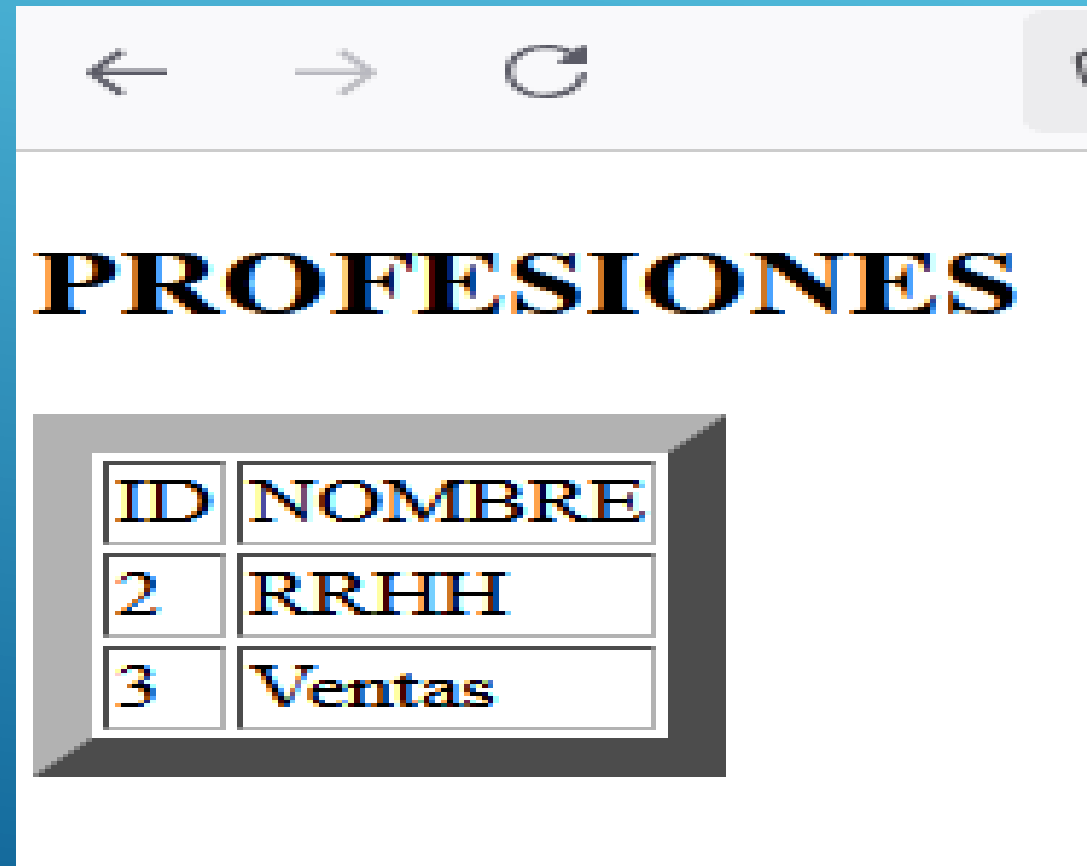
        if ($request->isMethod('POST')) {
            $form->handleRequest($request);
            if ($form->isSubmitted() && $form->isValid()) {
                $entityManager->persist($profesion);
                $entityManager->flush();
                return $this->redirect($this->generateUrl('app_listar'));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

    #[Route('/listar', name: 'app_listar')]
    public function listar(Request $request, EntityManagerInterface $entityManager): Response
    {
        $profesion= $entityManager->getRepository(Profesion::class)->findAll();
        return $this->render('profesion/listar.html.twig', [ 'controller_name' => 'ProfesionController','indice'=>$profesion ]);
    }
}

```

EJEMPLO GUIADO (DE LISTADO)

Para obtener todos los items de una entidad utilizamos el método `findAll()` por lo que tendremos el siguiente código dentro del método `listar()` para luego enviar el array del usuario a la vista que se encargará de mostrarlos en una tabla.



GENERAR URL Y REDIRECCIONAR

El método `generateUrl()` está definido en la clase base de `AbstractController`, y es solo un acceso directo a una parte del código.

Para redirigir al usuario un punto del código se usa el método `redirect()`, y se le pasa como argumento `generateUrl()` el cual contiene el nombre lógico de la ruta. .

Si en los métodos sustituimos la respuesta de operación realizada con éxito por una redirección al name del método de listado, me mostrará la bbdd con la actualización hecha:

```
➤return $this->redirect($this->generateUrl('app_listar'));
```

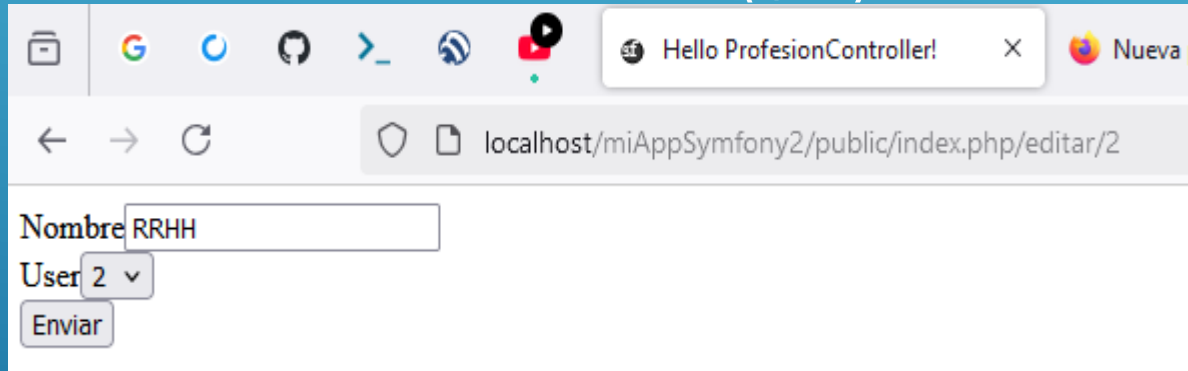
```
#[Route('/listar', name: 'app_listar')]
public function listar(Request $request, EntityManagerInterface $entityManager): Response
{
    $profesion= $entityManager->getRepository(Profesion::class)->findAll();
    return $this->render('profesion/listar.html.twig', [ 'controller_name' => 'ProfesionController', 'indice'=>$profesion ]);
}

#[Route('/editar/{id}', name: 'app_editar')]
public function editar(Request $request, EntityManagerInterface $entityManager, $id): Response
{
    $profesion= $entityManager->getRepository(Profesion::class)->find($id);
    $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()
    ]);
}
```

EJEMPLO GUIADO (DE ACTUALIZACIÓN)

La actualización de datos es exactamente igual a la inserción, con la diferencia que estamos modificando un objeto de usuario ya existente, para este ejemplo crearemos el método editar, al que pasaremos la primary key del usuario a modificar (\$id).

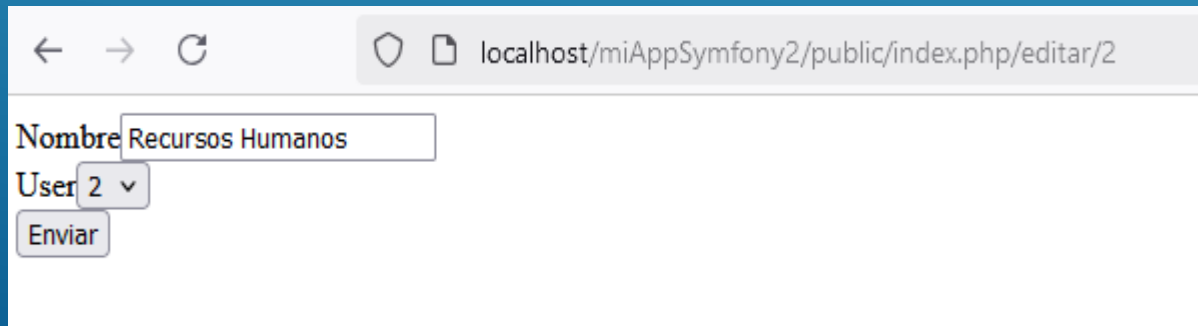


Nombre RRHH

User 2 ▾

Enviar

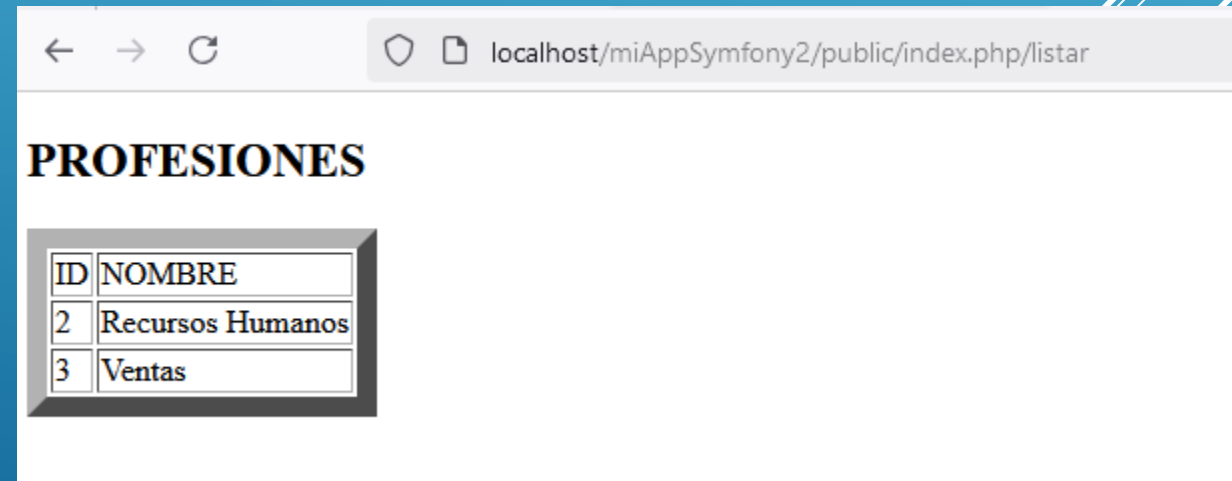
->



Nombre Recursos Humanos

User 2 ▾

Enviar

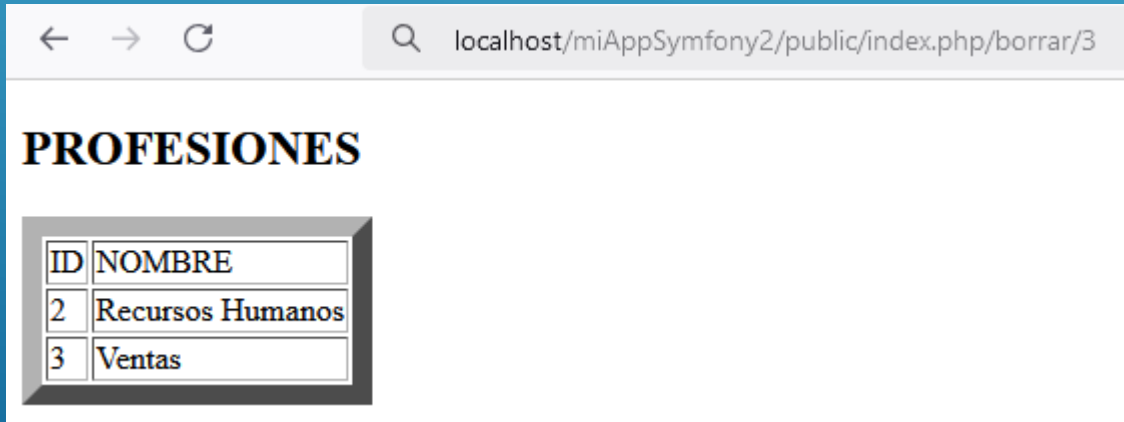


ID	NOMBRE
2	Recursos Humanos
3	Ventas

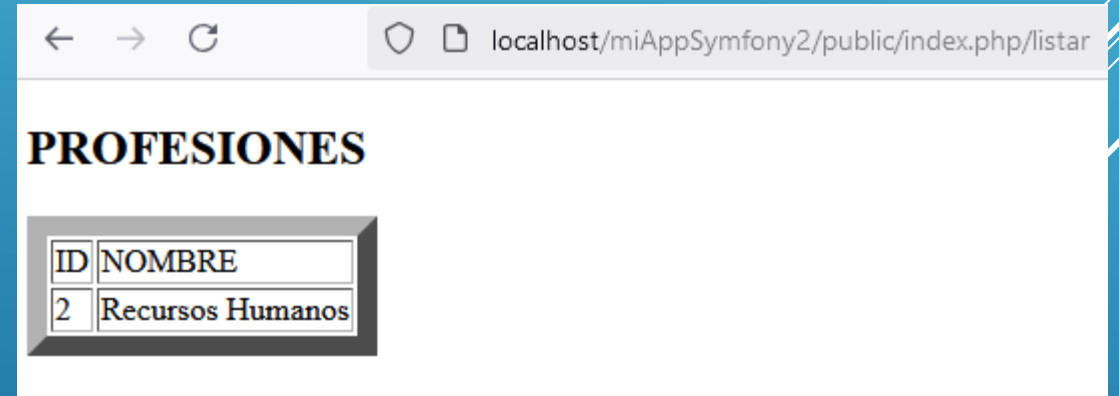
EJEMPLO GUIADO (DE BORRADO)

Para eliminar un usuario simplemente lo obtenemos e invocamos el método `remove()`, que marcará el usuario que va a ser borrado hasta que ejecutemos el método `flush()`.

Este código lo pondremos en el método `borrar()`.



->



```

#[Route('/listar', name: 'app_listar')]
public function listar(Request $request, EntityManagerInterface $entityManager): Response
{
    $profesion= $entityManager->getRepository(Profesion::class)->findAll();
    return $this->render('profesion/listar.html.twig', [ 'controller_name' => 'ProfesionController' ]);
}

#[Route('/editar/{id}', name: 'app_editar')]
public function editar(Request $request, EntityManagerInterface $entityManager, $id): Response
{
    $profesion= $entityManager->getRepository(Profesion::class)->find($id);
    $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()
    ]);
}

#[Route('/borrar/{id}', name: 'app_borrar')]
public function borrar(Request $request, EntityManagerInterface $entityManager, $id): Response
{
    $profesion= $entityManager->getRepository(Profesion::class)->find($id);
    $entityManager->remove($profesion);
    $entityManager->flush();
    return $this->redirect($this->generateUrl('app_listar'));
}

```

```

}

```

SI EL BORRADO IMPLICA UN BORRADO EN CASCADA
PUEDE OCURRIR EL ERROR:

An exception occurred while executing a query: SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails (`ejemplo1`.`pedido`, CONSTRAINT `FK_C4EC16CE19EB6921` FOREIGN KEY (`client_id`) REFERENCES `usuario` (`id`))

➤ No deja borrar x tener nodos hijos apuntando a la pk y no se ha elegido el on delete cascade

OTROS ERRORES FRECUENTES:

No route found for

"GET http://localhost/miAppSymfony/public/index.php/modificar/3"

- Por no encontrar el registro pedido(**el 3 ya estaba borrado o no existe**)
- Por tener el nombre lógico de la ruta repetido(**name de la ruta**)
- O x saturación de caché(**se debe volver a ejecutar solo con el directorio public para lanzarlo limpio**)