Symfony 6

Tabla de contenidos

1. Introducción

Tabla de contenidos

- Recursos:
  - https://symfony.com/doc/current/index.html
  - https://www.ediciones-eni.com/libro/symfony-5-desarrolle-sitios-web-php-estructurados-y-eficientes-9782409041402
  - https://jnjsite.com/tutoriales-para-aprender-symfony/
- Instalar ejemplo Ediciones eni:

1. Descargar los archivos de ejemplo y ponerlos en el servidor

```
cd ~/Descargas
mkdir ENI && cd ENI
wget https://www.ediciones-eni.com/libro/symfony-5-desarrolle-sitios-web-php-estructurados-y-eficientes-9782409041402/descargar-los-ejemplos-del-libro-700-ko.zip
unzip descargar-los-ejemplos-del-libro-700-ko.zip
cd EI5SYM/Proyecto_Symfony
mkdir /var/www/html/ENISymfony
sudo mv midiario /var/www/html/ENISymfony/
```

2. Configurar Apache

```
sudo nano /etc/apache2/sites-available/midiario.local.conf

# Y dentro del archivo
<VirtualHost *>
  ServerName  midiario.local
  DocumentRoot  /var/www/html/ENISymfony/midiario/public

  <Directory /var/www/html/ENISymfony/midiario/public>
    AllowOverride all
    Require all granted
  </Directory>

</VirtualHost>
```

3. Habilitar el nuevo sitio:

```
sudo a2ensite midiario.local.conf
sudo service apache2 restart
sudo nano /etc/hosts
```

```
# Poner al final del todo...
127.0.0.1    midiario.local
```

4. Instalar la BBDD:

```
# NOTA: Hay un error en el script
# Debemos cambiar constrasenia -> contrasenia
cd ~/Descargas/ENI/EI5SYM/Script_Base_De_Datos
code midiario.sql
# En la línea 67 está el error. CAMBIAR!!
mysql -u root -p # root
SOURCE midiario.sql
```

5. Instalamos symfony mediante composer:

```
cd /var/www/html/ENISymfony/midiario
# NOTA: Hay un error en el proyecto!
sudo nano config/packages/doctrine.yaml
# Buscamos: auto_generate_proxy_clases
# Y cambiamos por: auto_generate_proxy_classes
# CTRL + O y CTRL + X
php bin/console cache:clear --no-warmup
  composer update
# NOTA: Saldrá un error en las traducciones. IGNORAMOS
symfony server:start
# Y lo vemos en: http://127.0.0.1:8000
```

```
#
```

Instalación

Tabla de contenidos

- De forma resumida, comandos para crear un proyecto
  NOTA: COMPOSER se instala de forma global

```
# Crear proyecto e iniciar Servidor
cd /var/www/html/
# Windows -> cd c:/xampp/htdocs
symfony new symfony6 --version="6.4.*" --webapp
cd /var/www/html/symfony6
symfony server:start  # O pasar a /var/www/html

# Parar el servidor (En otra pestaña de consola!!)
# symfony server:stop

# Instalar las dependencias
composer require --dev symfony/maker-bundle
composer require twig
composer require symfony/form
```

- Nos puede dar un error a la hora de instalar DOCTRINE (orm-pack) o el annotations. Para evitarlo, seguir estos pasos:

```
# Crear proyecto e iniciar Servidor
cd /var/www/html/symfony6

composer require symfony/orm-pack
composer require annotations

# Si nos da un error:
# code config/services.yaml
# Añadir lo siguiente en la sección services:
# annotation_reader:
#        class: Doctrine\Common\Annotations\AnnotationReader
php bin/console cache:clear
composer update sensio/framework-extra-bundle
composer require symfony/orm-pack
composer require annotations
```

- Comandos adicionales de trabajo

```
# Crear controlador
php bin/console make:controller
# Visualizar ENDPOINTs
php bin/console debug:router
```

```
# Para las BBDD
# MODIFICAR .env!
php bin/console doctrine:database:create

# Crear entidad
php bin/console make:entity

# Crear migración y ejecutarla
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

2. Instalar Symfony CLI

- Previamente debemos tener instalado y actualizado composer

```
# Bajamos el instalador
cd ~/Descargas
wget https://get.symfony.com/cli/installer -O - | bash
# Instalamos globalmente
sudo mv ~/.symfony5/bin/symfony /usr/local/bin/symfony

# Comprobamos
symfony
# Comprobar requerimientos
symfony check:requirements

# Si mas adelante nos sale DEPRECATED
sudo rm /usr/local/bin/symfony
# Y repetimos los pasos de la instalación
```

- Extensiones recomenddas para Visual Studio Code
  - Twig Language 2 (mblode)
  - yaml (Red Hat)
  - Symfony for VSCode (TheNouillet)
    IMPORTANTE Reiniciar VSCode

Primer proyecto

Tabla de contenidos

```
# IMPORTANTE: Symfony NO tiene que estar obligatoriamente
# en el directorio de publicación de Apache!
# Estructura del comando
# symfony new <carpeta> --version --webapp
symfony new /var/www/html/symfony6 --version="6.4.*" --webapp
# Como alternativa, podemos instalar la última versión LTS
symfony new /var/www/html/symfony6 --version=lts

# Nos metemos en el directorio del proyecto e iniciamos el servidor
cd /var/www/html/symfony6
symfony server:start

# Si mas adelante queremos parar el servidor debemos ABRIR OTRA PESTAÑA
# de la consola y poner lo siguiente
### symfony server:stop

# Opcional: podemos instalar la demo (OJO, usa SQLite)
# sudo apt install php php-common
# sudo apt-get install php-sqlite3
# sudo a2enmod proxy_fcgi setenvif
# sudo a2enconf php8.3-fpm
# sudo service apache2 restart
# sudo apt install sqlite3
# cd /var/www/html
# git clone https://github.com/symfony/demo
# sudo mv demo/ symfony6-demo/ && cd symfony6-demo
# composer install
# symfony server:start
```

- Y ya podemos ver en el navegador como quedaría:
  - http://127.0.0.1:8000

Carpetas Básicas

Tabla de contenidos

- bin -> Ejecutables principales del sistema
  - console -> php/bin console...
- config -> Archivos de configuración
  - routes -> Listado de rutas
  - services -> Listado de Servicios creados
- migrations -> creación de migraciones de BBDD
- public -> Páginas publicas
- src -> Recursos del sistema
  - Controller -> Controladores (MVC)
  - Entity -> Entidades (objetos)
  - Repository -> Gestión de consultas
- templates -> plantillas (twig)
- var -> caché de la aplicación y registros (logs)
- vendor -> Dependencias
  - bin -> Ejecutables de dependencias
    - doctrine -> BBDD
    - var-dump-server -> Backup BBDD
    - phpunit -> Test Unitarios
  - Symfony -> núcleo de la aplicación
  - sension -> Maker bundle

Crear repositorio

Tabla de contenidos

- IMPORTANTE: En Agosto de 2021 se dejará de tener acceso a través de usuario/contraseña a Github.
    - https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/
    - https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token
    - Para poder acceder, tenemos que hacerlo a través de usuario/token.
    - Para generar nuestro token debemos hacer lo siguiente:
    1. Pulsamos en nuestra foto > Settings > Developer settings
    2. Personal Access token > [Generate New token]
    3. En la sección New Personal access token poneos un nombre. Ej: mirepo
    4. Marcamos las opciones de acceso. Recomiendo marcar [x] Repo
    5. Pulsamos en Generate token.
    6. Copiamos el token generado en algún lado (mas tarde lo usaremos).
- Desde el primer momento vamos a usar GITHub para crear versiones de nuestros proyectos.

1. Lo primero será crearnos una cuenta en Github: https://github.com/join?ref_cta=Sign+up
2. Lo siguiente, será crearnos el repositorio para el proyecto:
   a) Nos logamos en Github con nuestros datos.
   b) Nos vamos a la izquierda de la interfaz donde poner Repositories y le damos a New
   c) En Repository name ponemos: symfony5-manual
   d) En Description ponemos, por ejemplo: Repositorio para el manual de Symfony5
   e) Marcamos [x] Private (¡solo este!)
   f) Le damos a Create Repository

Crear Proyecto

Tabla de contenidos

1. Nos creamos la aplicación de Symfony (completa):

```
cd $HOMEPROJECTS  #NO tiene que ir en el servidor!
# En cualquier momento podemos ver si lo tenemos todo para instalar una aplicac
ión Symfony:
symfony check:requirements
# Y procedemos a crear un proyecto para un microservicio, aplicación de copnsol
a o API
symfony new symfony5-manual
```

2. Sincronizamos la carpeta actual con la remota
   NOTA: Cambiar el user por nuestro usuario:

```
cd $HOMEPROJECTS/symfony5-manual
git remote add origin https://github.com/USER/symfony5-manual.git
git branch -M main
git push -u origin main
touch README.md
git add .
git commit -m "Add README.md"
git push
```

3. Lo siguiente que debemos hacer es meter nuestras credenciales en el equipo (para no tener que estar poniéndolas cada vez que la usemos).

- https://git-scm.com/docs/git-credential-store
- Siguiendo estos pasos, guardaremos nuestras credenciales en un archivo oculto dentro de nuestra carpeta de usuario llamado git-credentials.

```
cd $HOMEPROJECTS/symfony5-manual
touch README2.md
git add .
git commit -m "Add README2.md"
git config credential.helper 'store --file ~/.git-credentials'
git push https://github.com/USER/symfony5-manual.git
Username for 'https://github.com': <user>
Password for 'https://ivanrguez1@github.com': <token>
Everything up-to-date
```

4. Por último, vamos a ver un archivo que es MUY IMPORTANTE: .gitignote

- Con este archivo vamos a omitir la subida de determinados elementos al repositorio.

- Por ejemplo tendremos los siguientes archivos y carpetas:
  - .env.local -> Archivo con las configuraciones de entorno locales
  - /vendor -> Carpeta con el núcleo de symfony y paquetes adicionales

```
###> symfony/framework-bundle ###
/.env.local
/.env.local.php
/.env.*.local
/config/secrets/prod/prod.decrypt.private.php
/public/bundles/
/var/
/vendor/
/builds/
.idea/
###< symfony/framework-bundle ###
composer.lock
symfony.lock
###> symfony/phpunit-bridge ###
.phpunit
.phpunit.result.cache
/phpunit.xml
###< symfony/phpunit-bridge ###
/phpunit.phar

###> squizlabs/php_codesniffer ###
/.phpcs-cache
/phpcs.xml
###< squizlabs/php_codesniffer ###

###> symfony/web-server-bundle ###
/.web-server-pid
###< symfony/web-server-bundle ###
```

Clonar Proyecto

Tabla de contenidos

1. Vamos a descargarnos nuestro proyecto del repositorio.

- Evidentemente, **NO** hay que hacerlo cada vez vayamos a trabajar con nuestro proyecto de Symfony
- El ejemplo que vamos a ver es para, llegado el caso, seguir con el proyecto en otro equipo:

```
cd $HOMEPROJECTS/
# Borramos el contenido de la carpeta symfony5-manual
rm -rf symfony5-manual
git clone https://github.com/ivanrguez1/symfony5-manual.git
# Y ponemos nuestro usuario y contraseña...
```

2. El siguiente paso será descargarnos las dependencias y arrancar el servidor de symfony

- IMPORTANTE: Cada vez que bajemos una nueva versión de nuestro repositorio tendremos que descarganos las dependencias

```
cd $HOMEPROJECTS/symfony5-manual
# Descargamos las dependencias
composer install
# Iniciamos el servidor
symfony server:start
```

3. Vemos nuestra página en el navegador (por ejemplo Firefox):

- http://127.0.0.1:8000\

4. Además, en cualquier momento podemos ver las posibles vulnerabilidades de nuestro proyecto:

- Están sacadas de la Base de datos oficial del propio Symfony:

```
cd $HOMEPROJECTS/symfony5-manual
symfony check:security
```

5. Por último tenemos disponible la aplicación de ejemplo oficial junto al manual completo de desarrollo rápido:

- https://symfony.com/doc/5.0/the-fast-track/es/index.html

```
cd $HOMEPROJECTS
symfony new my_project_name --demo
```

3. Crear páginas

Tabla de contenidos

- Recursos:
  - https://symfony.com/doc/current/page_creation.html
  - https://symfony.com/bundles/SymfonyMakerBundle/current/index.html
- Nombre de la rama: 2-Crear-Paginas
- Llegados a este punto tenemos dos opciones:
  - 1. Clonar nuestro proyecto de GitHub y añadir una rama nueva por cada tema
  - 2. Seguir con nuestro proyecto en local y añadir una rama nueva por cada tema
- Vamos a presuponer que iremos trabajando en local y, al final, subiremos los cambios a GitHub
- Para crear las páginas tenemos dos opciones:
  - a) Definir las rutas en el archivo config/routes.yaml, asociandolas a su controlador
  - b) Añadiendo anotaciones a los controladores.
- Veremos ambos casos, automatizando el proceso lo máximo posible.
- Para ambas opciones añadiremos las dependencias y arrancaremos el servidor:

```
cd $HOMEPROJECTS/symfony6-manual
composer require annotations                  # Anotaciones para añadir rutas a
l controlador
composer require twig                         # Para usar TWIG en el Frontend
composer require --dev symfony/maker-
bundle   # Para crear controladores (y mas...) por consola
symfony server:start
```

- Con la instalación de maker-bundle tenemos disponible multitud de comandos para ejecutar
  - Los iremos viendo, poco a poco, mas adelante. Pero el listado está disponible por consola:

```
php bin/console
```

Usar routes.yaml

Tabla de contenidos

1. Debemos abrir otra pestaña de consola y cuando nos pregunte por el nombre, pondremos Aleatorio:
   - IMPORTANTE: OJO a las mayúsculas. Aleatorio tiene la primera en mayúscula.

```
php bin/console make:controller
Choose a name for your controller class (e.g. GrumpyElephantController):
 > Aleatorio

 created: src/Controller/AleatorioController.php
 created: templates/aleatorio/index.html.twig

  Success!
```

2. Editamos el archivo src/Controller/AleatorioController.php
   - En este archivo nos saldrá una anotación #[Route] que usaremos mas tarde. Por ahora lo comentamos:

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class AleatorioController extends AbstractController
{
  //#[Route('/aleatorio', name: 'app_aleatorio')]
    public function index(): Response
    {
        $numeroAleatorio = random_int(0, 100);

        return new Response(
            '<html><body>Lucky number: ' . $numeroAleatorio . '</body></html>'
        );
    }
}
```

3. Editamos el archivo config/routes.yaml
   NOTA: podemos añadir tantas URLs (ENDPOINTS) como queramos. Pero NUNCA podemos repetir el nombre de cada uno.

```yaml
controllers:
  resource:
    path: ../src/Controller/
    namespace: App\Controller
```

```
  type: attribute

num_aleatorio1:
  path: /aleatorio
  controller: App\Controller\AleatorioController::index

# Si queremos añadir otra ruta (ENDPOINT)
# Copiamos, pegamos, y cambiamos nombre y path.
num_aleatorio2:
  path: /mi-aleatorio
  controller: App\Controller\AleatorioController::index
```

4. Ya solo nos queda probarlo en el navegador:
    - http://localhost:8000/aleatorio
5. Para ver el listado con todas las rutas del sistema, iremos a la consola:

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

Usar Anotaciones

Tabla de contenidos

- Vamos a implementar el mismo ejemplo anterior pero usando anotaciones:

1. Usamos el controlador del paso anterior:
   AleatorioController.php
   NOTA: Podemos agregar varias páginas al mismo controlador. Ya lo veremos.

2. Editamos la página src/Controller/AleatorioController.php

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AleatorioController extends AbstractController
{
    // public function index(): Response
    // ...

    // En la anotación ponemos la ruta: http://localhost:8000/aleatorio
    #[Route('/aleatorio2', name: 'app_aleatorio2')]
    public function index2(): Response
    {
        $numeroAleatorio = random_int(0, 100);

        // Aquí usaremos Twig, y le pasamos el parámetro numeroAleatorio
        // ATENCIÓN: aleatorio se recogerá en el twig entre llaves: {{ aleatori
o }}
        return $this->render('aleatorio/index.html.twig', [
            'controller_name' => 'AleatorioController',
            'aleatorio' => $numeroAleatorio,
        ]);
    }
}
```

3. Como hemos visto en el código, vamos a usar Twig, un motor de plantillas.
   - Dicho motor está mantenido por Fabien Potencier, el creador de Symfony
   - Mas información aqui: https://twig.symfony.com/
   - Instalamos el plugin Twig Language (5estrellas)
   - Editamos la plantilla: templates/aleatorio/index.html.twig
   - IMPORTANTE: la variable {{ aleatorio }} se corresponde con la clave 'aleatorio' => $numeroAleatorio del controlador.

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

{% block title %}Hola AleatorioController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-
width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Hola {{ controller_name }}! ✅</h1>

    <!-- El resto del código lo dejamos o lo borramos...-->
    <p> Tu número aleatorio es: {{ aleatorio }} </p>
</div>
{% endblock %}
```

4. Como vimos antes, podemos probarlo en el navegador:

- http://localhost:8000/aleatorio2

5. Por último, vamos a añadir una tercera ruta, en el mismo controlador:

- En este caso, pasamos un parámetro para que nos devuelva varios aleatorios:

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AleatorioController extends AbstractController
{
    // public function index(): Response
    // public function index2(): Response
    // ...

    #[Route('/aleatorio3/{num}', name: 'app_aleatorio3')]
    public function index3(int $num): Response
    {
        $numerosAleatorios = "<br>";
        for ($i = 1; $i <= $num; $i++) {
            $numerosAleatorios .= random_int(0, 100) . "<br>";
        }
        return new Response(
            '<html><body>Números aleatorios: ' . $numerosAleatorios . '</body>
</html>'
        );
```

```
    }
}
```

6. Como vimos antes, podemos probarlo en el navegador:

- http://localhost:8000/aleatorio3/10
  - Nos sacaria 10 aleatorios

7. ¿Y si queremos mas de un parámetros?

- Pues igual, metemos /{param1}/{param2}...
- En este caso, pasamos dos parámetros para que nos devuelva varios aleatorios con un límite. Eso si, esta vez vamos a usar el renderizado de la plantilla TWIG:

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AleatorioController extends AbstractController
{
    // public function index(): Response
    // public function index2(): Response
    // public function index3(): Response
    // ...

    #[Route('/aleatorio4/{num1}/{limite}', name: 'app_aleatorio4')]
    public function index4(int $num): Response
    {
        $numeroAleatorio = "";
        for ($i = 1; $i <= $num1; $i++) {
            $numeroAleatorio .= random_int(0, $limite) . "-";
        }
        return $this->render('aleatorio/index.html.twig', [
            'controller_name' => 'AleatorioController',
            'aleatorio' => $numeroAleatorio,
        ]);
    }
}
```

8. Como vimos antes, podemos probarlo en el navegador:

- http://localhost:8000/aleatorio4/5/20
  - Nos sacaria 5 aleatorios entre el 0 y el 20.

9. Una vez finalizado todo el trabajo, es hora de subirlo todo al repositorio:

> NOTA: Hay que recordar que en usuario pondremos el nuestro, pero en password, pondremos nuestro TOKEN

```
git add composer.json
git add config/bundles.php
git add config/routes.yaml
git add config/packages/sensio_framework_extra.yaml
git add config/packages/test/twig.yaml
git add config/packages/twig.yaml
git add config/routes/annotations.yaml
git add src/Controller/Aleatorio2Controller.php
git add src/Controller/AleatorioController.php
git add templates/
git commit -m "Cap-02 Crear Paginas"
git push --set-upstream origin 2-Crear-Paginas
git push
```

4. Doctrine

- Symfony proporciona todas las herramientas necesarias para usar bases de datos en las aplicaciones gracias a Doctrine , el mejor conjunto de bibliotecas PHP para trabajar con bases de datos.
- Estas herramientas admiten bases de datos relacionales como MySQL y PostgreSQL (o SQLite, que será el que usemos) y también bases de datos NoSQL como MongoDB.
- Recursos:
  - https://symfony.com/doc/current/doctrine.html
  - https://symfony.com/doc/current/doctrine/dbal.html

Instalación de Doctrine

Tabla de contenidos

1. Creamos la rama y nos introducimos en ella

```
git branch 3-Doctrine
git checkout 3-Doctrine
git push --set-upstream origin 3-Doctrine
```

2. Por consola, dentro del proyecto:

```
cd $HOMEPROJECTS/symfony6
composer require symfony/orm-pack
composer require --dev symfony/maker-bundle
```

3. Ahora toca configurar la Base de datos que vamos a usar. Para ello tenemos el archivo .env.

- MUY IMPORTANTE: Si queremos cambiar la configuración de trabajo en local, debemos crearnos un archivo llamado .env.local que tendrá la configuración que deseemos.
- Para nuestro caso, los cambios los haremos en el .env, trabajando siempre con MySQL.
- Como ya veremos en el código, tan solo debemos cambiar una linea para trabajar, por ejemplo, con SQLite
- Una ultima cosa: en la línea sin comentar (DATABASE_URL="mysql...) debemos poner el nombre de la base de datos que queramos usar. Por defecto es db_name, pero nosotros pondremos symfony6

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=72984f7cb2d89365e92bb4dd28f0c02f
###< symfony/framework-bundle ###
```

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-
dbal/en/latest/reference/configuration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in config/p
ackages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://root:root@127.0.0.1:3306/test?serverVersion=10.2.32-
MariaDB-log"

#DATABASE_URL="mysql://root:root@127.0.0.1:3306/db_name?serverVersion=5.7"
DATABASE_URL="mysql://root:root@127.0.0.1:3306/symfony_anidi?
serverVersion=8&charset=utf8mb4"

# DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?
serverVersion=13&charset=utf8"
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
###< doctrine/doctrine-bundle ###
```

- Para los que tengan Raspberry

```
DATABASE_URL="mysql://root:root@127.0.0.1:3306/symfony6"
```

Estructura de la BD

Tabla de contenidos

1. Ahora nos crearnos la base de datos:
   > IMPORTANTE: En vez de db_name debe estar puesta la cadena con symfony6 o el nombre que queramos

```
php bin/console doctrine:database:create
> Created database `symfony6` for connection named default
# Para ver todos los comandos disponibles de Doctrine
php bin/console list doctrine
# Si queremos borrar la BBDD
# php bin/console doctrine:database:drop --force
```

2. Ahora vamos a crearnos una tabla. En Doctrine se le llama Entity (entidad) y no es mas que una clase que incluye atributos (campos) y sus correspondientes métodos setter y getter.
   > Por convencion, las entidades empiezan por mayúsculas y los campos van en minúsculas

```
php bin/console make:entity
> Articulos

New property name (press <return> to stop adding fields):
> titulo

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/Articulos.php

Add another property? Enter the property name (or press <return> to stop adding
 fields):
>
# INTRO!

Success!

Next: When you're ready, create a migration with php bin/console make:migration
```

3. Evidentemente, en cualquier momento podemos añadir campos adicionales a la tabla:

- Para ello sólo debemos ejecutar el comando de creación de la entidad con el MISMO NOMBRE...

  > Vamos a ver con ? TODAS las opciones de campo disponibles...

```
php bin/console make:entity
> Articulos

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
 > publicado

 Field type (enter ? to see all types) [string]:
 > ?

Main types
  * string
  * text
  * boolean
  * integer (or smallint, bigint)
  * float

Relationships / Associations
  * relation (a wizard 🧙 will help you build the relation)
  * ManyToOne
  * OneToMany
  * ManyToMany
  * OneToOne

Array/Object Types
  * array (or simple_array)
  * json
  * object
  * binary
  * blob

Date/Time Types
  * datetime (or datetime_immutable)
  * datetimetz (or datetimetz_immutable)
  * date (or date_immutable)
  * time (or time_immutable)
  * dateinterval

Other Types
  * ascii_string
  * decimal
  * guid
  * json_array

Field type (enter ? to see all types) [string]:
 > boolean

 Can this field be null in the database (nullable) (yes/no) [no]:
```

```
 >

 updated: src/Entity/Articulos.php

 Add another property? Enter the property name (or press <return> to stop addin
g fields):
 >
# INTRO!!

  Success!
```

4. Vale, ya tenemos la primera tabla, ahora vamos por la segunda:

- En este caso será la tabla Autor, con los campos nombre (string) y edad (integer)

```
php bin/console make:entity
Class name of the entity to create or update (e.g. OrangeKangaroo):
 > Autores

 created: src/Entity/Autores.php
 created: src/Repository/AutoresRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

 New property name (press <return> to stop adding fields):
 > nombre

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Autores.php

 Add another property? Enter the property name (or press <return> to stop addin
g fields):
 > edad

 Field type (enter ? to see all types) [string]:
 > integer

 Can this field be null in the database (nullable) (yes/no) [no]:
 >
  # INTRO!
 updated: src/Entity/Autores.php

 Add another property? Enter the property name (or press <return> to stop addin
g fields):
```

```
 >

  # INTRO!

  Success!
```

5. El siguiente paso será relacionar ambas tablas.

- Autores será la tabla principal y Articulos la tabla derivada
- Por tanto, para crear la relación NOS VAMOS A LA DERIVADA (Articulos)
- El tipo será ManyToOne (muchos Articulos son de 1 Autor)
- Mas información: https://symfony.com/doc/current/doctrine/associations.html

```
php bin/console make:entity
Class name of the entity to create or update (e.g. DeliciousChef):
 > Articulos

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > Autor

Field type (enter ? to see all types) [string]:
 > ManyToOne

 What class should this entity be related to?:
 > Autores

 Is the Articulos.autor property allowed to be null (nullable)? (yes/no) [yes]:
 > no

 Do you want to add a new property to Autores so that you can access/update Art
iculos objects from it - e.g. $autores->getArticulos()? (yes/no) [yes]:
 >

 A new property will also be added to the Autores class so that you can access
the related Articulos objects from it.

 New field name inside Autores [articulos]:
 >

 Do you want to activate orphanRemoval on your relationship?
 A Articulos is "orphaned" when it is removed from its related Autores.
 e.g. $autores->removeArticulos($articulos)

 NOTE: If a Articulos may *change* from one Autores to another, answer "no".

 Do you want to automatically delete orphaned App\Entity\Articulos objects (orp
hanRemoval)? (yes/no) [no]:
 >

 updated: src/Entity/Articulos.php
```

```
 updated: src/Entity/Autores.php

 Add another property? Enter the property name (or press <return> to stop addin
g fields):
 >

  Success!

 Next: When you're ready, create a migration with php bin/console make:migratio
n
```

6. Por último, todo la lógica dentro de Symfony debemos trasladarla al SGBD:

```
php bin/console make:migration
Success!

 Next: Review the new migration "migrations/Version20220615185503.php"
```

- Como nos dice la consola podemos ver el archivo migrations/Version20220615185503.php" para personalizar los comandos SQL que se van a ejecutar respecto al SGBD.
  - Por experiencia, suele ser buena idea cambiar los nombres de los FK (Foreign Key) e IDX (Index)
- Y lo trasladamos:

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

WARNING! You are about to execute a migration in database "symfony5" that could
 result in schema changes and data loss. Are you sure you wish to continue? (ye
s/no) [yes]:
 >
  # INTRO!

[notice] Migrating up to DoctrineMigrations\Version20220615185503
[notice] finished in 57.7ms, used 14M memory, 1 migrations executed, 3 sql quer
ies
```

- Dejo aquí la relación de comandos para doctrine:

```
    doctrine:database:create                 Creates the configured database
    doctrine:database:drop                   Drops the configured database
    doctrine:query:sql                       Executes arbitrary SQL directly f
rom the command line.
    doctrine:database:import                 Import SQL file(s) directly to Da
tabase.
    doctrine:cache:clear-
metadata            Clears all metadata cache for an entity manager
    doctrine:cache:clear-
query               Clears all query cache for an entity manager
```

```
    doctrine:cache:clear-
result               Clears result cache for an entity manager
    doctrine:cache:clear-collection-region    Clear a second-
level cache collection region
    doctrine:mapping:convert                  Convert mapping information betwe
en supported formats
    doctrine:schema:create                    Executes (or dumps) the SQL neede
d to generate the database schema
    doctrine:schema:drop                      Executes (or dumps) the SQL neede
d to drop the current database schema
    doctrine:ensure-production-
settings       Verify that Doctrine is properly configured for a production env
ironment
    doctrine:cache:clear-entity-region        Clear a second-
level cache entity region
    doctrine:mapping:info
    doctrine:cache:clear-query-region         Clear a second-
level cache query region
    doctrine:query:dql                        Executes arbitrary DQL directly f
rom the command line
    doctrine:schema:update                    Executes (or dumps) the SQL neede
d to update the database schema to match the current mapping metadata
    doctrine:schema:validate                  Validate the mapping files
    doctrine:mapping:import                   Imports mapping information from
an existing database
    doctrine:migrations:diff                  Generate a migration by comparing
 your current database to your mapping information.
    doctrine:migrations:sync-metadata-
storage Ensures that the metadata storage is at the latest version.
    doctrine:migrations:list                  Display a list of all available m
igrations and their status.
    doctrine:migrations:current               Outputs the current version
    doctrine:migrations:dump-
schema           Dump the schema for your database to a migration.
    doctrine:migrations:execute               Execute one or more migration ver
sions up or down manually.
    doctrine:migrations:generate              Generate a blank migration class.
    doctrine:migrations:latest                Outputs the latest version
    doctrine:migrations:migrate               Execute a migration to a specifie
d version or the latest available version.
    doctrine:migrations:rollup                Rollup migrations by deleting all
 tracked versions and insert the one version that exists.
    doctrine:migrations:status                View the status of a set of migra
tions.
    doctrine:migrations:up-to-date            Tells you if your schema is up-
to-date.
    doctrine:migrations:version               Manually add and delete migration
 versions from the version table.."
```

- Podemos comprobarlo todo en MySQL (abrimos sensión):

```
USE symfony6;
SHOW TABLES;
```

```
SHOW CREATE TABLE Articulos;
SHOW CREATE TABLE Autores;
```

Persistir Objetos

Tabla de contenidos

1. Para ello podemos crearnos un Controlador (también podemos hacerlo con un servicio):

```
# Ahora usamos
# php bin/console make:controller <nombre_controlador>
php bin/console make:controller AutoresController
```

2. Dentro de dicho controlador ponemos toda la lógica para insertar un registro:

- En src/Controller/AutoresController

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

// Debemos añadir las siguientes clases...
use App\Entity\Autores;
use Doctrine\Persistence\ManagerRegistry;

class AutoresController extends AbstractController
{
    #[Route('/crea-autor', name: 'crea-autor')]
    public function crearAutor(ManagerRegistry $doctrine): Response
    {
        // Creamos el objeto Gestor de Entidad
        $entityManager = $doctrine->getManager();

        // Defino un objeto autor
        $autor = new Autores();
        $autor->setNombre('Iván Rodríguez');
        $autor->setEdad(46);

        // Y lo guardo
        $entityManager->persist($autor);
        $entityManager->flush();

        return new Response('Guardado Autor con ID -> ' . $autor->getId());
    }
}
```

3. Tambien podemos meter un autor directamente por la url:

- En src/Controller/AutoresController

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

// Debemos añadir las siguientes clases...
use App\Entity\Autores;
use Doctrine\Persistence\ManagerRegistry;

class AutoresController extends AbstractController
{
    #[Route('/crea-autor', name: 'crea-autor')]
    //public function crearAutor(ManagerRegistry $doctrine): Response

    #[Route('/crea-autor/{nombre}/{edad}', name: 'crea-autor2')]
    public function crearAutor2(ManagerRegistry $doctrine, String $nombre, int
$edad): Response
    {
        // Creamos el objeto Gestor de Entidad
        $entityManager = $doctrine->getManager();

        // Defino un objeto autor
        $autor = new Autores();
        $autor->setNombre($nombre);
        $autor->setEdad($edad);

        // Y lo guardo
        $entityManager->persist($autor);
        $entityManager->flush();

        return new Response('Guardado Autor con ID -> ' . $autor->getId());
    }
}
```

- Y lo probamos en el navegador
  - http://localhost:8000/crea-autor/Luis/7

4. Ahora repetimos la operación con Articulos

```
php bin/console make:controller ArticulosController
```

5. En el controlador usaremos una construcción diferente:

- En src/Controller/ArticulosController

> PENDIENTE (NO ESTUDIAR!)
>
> - a) Agregamos el $entityManager como parámetro de la función

> SI ESTUDIAR!!
>
> - a) Usamos un array bidimensional para añadir varios registros de golpe
> - b) Empleamos foreach para hacer una carga de varios registros

```php
<?php

namespace App\Controller;

// Nuevas clases para añadir
use App\Entity\Articulos;
use App\Entity\Autores;
use Doctrine\Persistence\ManagerRegistry;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ArticulosController extends AbstractController
{
    /**
     * @Route("/crea-articulos", name="create_articles")
     */
    public function crearArticulos(ManagerRegistry $doctrine): Response
    {
        $entityManager = $doctrine->getManager();

        $registros = array(
            "articulo1" => array(
                "titulo" => 'Manual de Symfony5',
                "publicado" => 1,
                "autor" => 1
            ),

            "articulo2" => array(
                "titulo" => 'Notas sobre GIT',
                "publicado" => 0,
                "autor" => 1
            ),

            "articulo3" => array(
                "titulo" => 'Bienvenidos',
                "publicado" => 1,
                "autor" => 1
            )
        );

        foreach ($registros as $clave => $registro) {
            $articulo = new Articulos();
            $articulo->setTitulo($registro['titulo']);
            $autor = $entityManager->getRepository(Autores::class)-
>findOneBy(['id' => $registro['autor']]);
            $articulo->setAutor($autor);
```

```
            $articulo->setPublicado($registro['publicado']);

            $entityManager->persist($articulo);
            $entityManager->flush();
        }


        return new Response('Guardados Articulos!');
    }
}
```

6. Una vez creados los controladores, tan solo tenemos que ejecutarlos. Para ello ponemos en el navegador:

- http://localhost:8000/crea-autor
- http://localhost:8000/crea-articulos

7. Y si queremos, lo comprobamos directamente con estos comandos de Symfony:

```
php bin/console dbal:run-sql 'SELECT * FROM articulos'
php bin/console dbal:run-sql 'SELECT * FROM autores'
```

8. Inserción de datos por parámetros: en este caso, vamos pasar los 3 campos por el ENDPOINT.
   ATENCIÓN: en este caso debemos insertar el autor como un objeto, buscando primero su registro por la ID pasada por el parámetro y asignado luego su valor en el registro de articulos.

Nos creamos un nuevo método crearArticulo:

- En src/Controller/ArticulosController

```
#[Route('/crea-articulo/{titulo}/{publicado}/{autor}', name: 'crea-articulo')]
    public function crearArticulo(
        ManagerRegistry $doctrine,
        String $titulo,
        int $publicado,
        int $autor
    ): Response {
        $entityManager = $doctrine->getManager();

        $articulo = new Articulos();
        $articulo->setTitulo($titulo);
        $articulo->setPublicado($publicado);

        // Para el caso del autor, debemos buscar el autor
        // con la ID pasada por parámetro
        $autor = $entityManager->getRepository(Autores::class)->find($autor);
        $articulo->setAutor($autor);
```

```
        $entityManager->persist($articulo);
        $entityManager->flush();

        return new Response('Articulo agregado');
    }
```

Consultar Objetos

Tabla de contenidos

- Ya hemos visto como realizar el INSERT en el apartado anterior.
- A partir de ahora vamos a realizar el resto de componentes del CRUD, comenzando por el SELECT

1. Para empezar sacaremos el SELECT * FROM articulos
   usando como salida una tabla HTML usando el findAll():

- En src/Controller/ArticulosController

```
// Añadimos el repositorio a las clases que usamos:
use App\Repository\ArticulosRepository;
```

- Vamos a crearnos un nuevo método en el controlador de Articulos:

```
#[Route('/ver-articulos', name: 'ver-articulos')]
public function mostrarArticulos(ArticulosRepository $repo): Response
{
    $articulos = $repo->findAll();
    $respuesta = "<html>
    <body>
        <table border=1>
            <th>ID</th>
            <th>Titulo</th>
            <th>publicado</th>
            <th>autor</th>";
    // con getAutor obtenemos el autor como objeto
    // Con eso, sacamos lo que queramos, por ejemplo el nombre
    foreach ($articulos as $articulo) {
        $respuesta .= "<tr>
        <td> " . $articulo->getId() . "</td>
        <td> " . $articulo->getTitulo() . "</td>
        <td> " . $articulo->isPublicado() . "</td>
        <td> " . $articulo->getAutor()->getNombre() . "</td>
        </tr>";
    }
    $respuesta .= "</table>
    </body>
    </html>";
    return new Response($respuesta);
}
```

- Lo podemos ver en el navegador
  - http://localhost:8000/ver-articulos

2. Ahora vamos a sacar SELECT * FROM articulos
   WHERE id = 1 usando como salida JSON, usando el find($id):

- Creamos otro método en el mismo controlador:
  - En src/Controller/ArticulosController

```
// Añadimos el JsonResponse para sacarlo en formato JSON (Para API REST)
use Symfony\Component\HttpFoundation\JsonResponse;
```

- Y creamos el nuevo método:

```
#[Route('/articulo/{id}', name: 'ver-articulo')]
public function verArticulo(ManagerRegistry $doctrine, int $id): Response
{
    $articulo = $doctrine->getRepository(Articulos::class)->find($id);
    // De nuevo, sacamos el autor con el objeto completo...
    return new JsonResponse([
        'id' => $articulo->getId(),
        'titulo' => $articulo->getTitulo(),
        'publicado' => $articulo->isPublicado(),
        'autor' => $articulo->getAutor()->getNombre(),
    ]);
}
```

- Lo visualizamos en el navegador, usando el JsonViewer de Chrome:
  - http://localhost:8000/articulo/1
- https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh?hl=es

3. Vamos a sacar SELECT * FROM articulos usando como salida JSON, usando el findAll():

- Creamos otro método en el mismo controlador:
  - En src/Controller/ArticulosController
- Debemos tener estas clases al principio del controlador:

```
namespace App\Controller;

use App\Entity\Articulos;
use App\Entity\Autores;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\JsonResponse;
use App\Repository\ArticulosRepository;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
```

- Añadimos un nuevo método con su ruta:

```
#[Route('/consultar-articulos', name: 'consultar-articulos')]
public function consultarArticulos(
    ManagerRegistry $doctrine
): JsonResponse {
    $articulos = $doctrine->getRepository(Articulos::class)->findAll();
    $json = array();
    foreach ($articulos as $articulo) {
        $json[] = array(
            'id' => $articulo->getId(),
            'titulo' => $articulo->getTitulo(),
            'publicado' => $articulo->isPublicado(),
        );
    }

    return new JsonResponse($json);
}
```

- Lo visualizamos en el navegador, usando el JsonViewer de Chrome:
  - http://localhost:8000/consultar-articulos

4. Por último vamos a sacar SELECT * FROM articulos
   WHERE publicado=1 AND titulo = "Bienvenidos" usando como salida JSON y el método findBy():

- Creamos otro método en el mismo controlador:
  - En src/Controller/ArticulosController
- Añadimos el nuevo método:

```
#[Route('/articulos/{publicado}/{titulo}', name: 'ver-articulo2')]
public function verArticulo2(
    ManagerRegistry $doctrine,
    bool $publicado,
    String $titulo
): JsonResponse {

  // Dentro del findBy metemos 2 arrays
  // El 1er array es para filtrar por varios campos
  // El 2º array es para cambiar la ordenación
  // En este caso, la id irá al revés: 3,2,1...
    $articulos = $doctrine->getRepository(Articulos::class)->findBy(
        [
            'publicado' => $publicado,
            'titulo' => $titulo
        ],
        ['id' => 'DESC']
    );

    $json = array();
    foreach ($articulos as $articulo) {
        $json[] = array(
            'id' => $articulo->getId(),
```

```
            'titulo' => $articulo->getTitulo(),
            'publicado' => $articulo->isPublicado(),
        );
    }
    return new JsonResponse($json);
}
```

- Lo visualizamos en el navegador, usando el JsonViewer de Chrome:
  - http://localhost:8000/articulos/1/Bienvenidos

    > UN TRUCO!! Podemos poner lo de antes asi también:
    > http://localhost/symfony6/public/index.php/articulos/1/Bienvenidos
    > servidor - ruta_carpeta - endpoint

5. Las 3 consultas anteriores con salida en formato JSON lo podemos comprobar en mysql

```
mysql> SELECT * FROM articulos WHERE id = 1;
+----+----------+--------------------+-----------+
| id | autor_id | titulo             | publicado |
+----+----------+--------------------+-----------+
|  1 |        1 | Manual de Symfony5 |         1 |
+----+----------+--------------------+-----------+
1 row in set (0,00 sec)

mysql> SELECT * FROM articulos;
+----+----------+--------------------+-----------+
| id | autor_id | titulo             | publicado |
+----+----------+--------------------+-----------+
|  1 |        1 | Manual de Symfony5 |         1 |
|  2 |        1 | Notas sobre GIT    |         0 |
|  3 |        1 | Bienvenidos        |         1 |
|  4 |        1 | Manual de Symfony5 |         1 |
|  5 |        1 | Notas sobre GIT    |         0 |
|  6 |        1 | Bienvenidos        |         1 |
+----+----------+--------------------+-----------+
6 rows in set (0,00 sec)

mysql> SELECT * FROM articulos
    -> WHERE publicado=1 AND titulo = "Bienvenidos";
+----+----------+-------------+-----------+
| id | autor_id | titulo      | publicado |
+----+----------+-------------+-----------+
|  3 |        1 | Bienvenidos |         1 |
|  6 |        1 | Bienvenidos |         1 |
+----+----------+-------------+-----------+
2 rows in set (0,00 sec)
```

Consultar Objetos (Avanzado)

Tabla de contenidos

1. Vamos a crearnos un nuevo método en el controlador de Articulos:

```
#[Route('/ver-articulos-autores/{publicado}/{titulo}', name: 'ver-articulos-
autores')]
    public function consultarArticulos3(
        ManagerRegistry $doctrine,
        int $publicado,
        String $titulo
    ): JsonResponse {

      // En este caso no usamos el gestor de entidades
      // Usamos sólo la conexión
        $connection = $doctrine->getConnection();

        // IMPORTANTE: si queremos personalizar la salida
        // JSON, poner alias en los campos del SELECT
        $articulos = $connection
            ->prepare("SELECT articulos.id as articulos_id,
                       titulo, publicado,
                       nombre as Escritor
                       FROM articulos, autores
                       WHERE autor_id = autores.id
                       AND publicado = $publicado
                       AND titulo = '" . $titulo . "'")
            ->executeQuery()
            ->fetchAllAssociative();

        // Con el dump, sacamos el array completo
```

```
        return new JsonResponse(dump($articulos));
    }
```

2. Y lo probamos en el navegador

- http://localhost:8000/ver-articulos-autores/1/Bienvenidos

```
```

Actualizar Objetos

Tabla de contenidos

- Vamos a seguir con el CRUD. Ahora el UPDATE.
  En src/Controller/ArticulosController

1. Vamos a crearnos un nuevo método en el controlador de Articulos:
   Sacaremos el UPDATE articulos
   SET titulo="nuevo titulo"
   WHERE id=2;

```
#[Route('/cambia-articulo/{id}/{titulo}', name: 'actualizar-articulo')]
public function cambiarArticulo(ManagerRegistry $doctrine, int $id, String $tit
ulo): Response
{
    $entityManager = $doctrine->getManager();
    $articulo = $entityManager->getRepository(Articulos::class)->find($id);

    if (!$articulo) {
        throw $this->createNotFoundException(
            'Articulo NO existe con ID: ' . $id
        );
    }

    $articulo->setTitulo($titulo);
    $entityManager->flush();

    // Aprovechamos el método anterior para presentar el registro cambiado
    return $this->redirectToRoute('ver-articulo', [
            'id' => $articulo->getId()
    ]);
}
```

2. Sacamos en primer lugar el error, intentando cambiar un artículo que NO existe:
   http://localhost:8000/cambia-articulo/5/aloha

3. Y ahora probamos con uno que SI existe:
   http://localhost:8000/cambia-articulo/3/aloha

- Incluso podemos poner un título con varias palabras
  http://localhost:8000/cambia-articulo/3/Bievenidos a Symfony

Eliminar Objetos

Tabla de contenidos

- Vamos a seguir con el CRUD. Ahora el DELETE.
  En src/Controller/ArticulosController

1. . Vamos a crearnos un nuevo método en el controlador de Articulos:
   Sacaremos el DELETE FROM articulos WHERE id = 2

```
#[Route('/elimina-articulo/{id}', name: 'eliminar-articulo')]
public function eliminarArticulo(ManagerRegistry $doctrine, int $id): Response
{
    $entityManager = $doctrine->getManager();
    $articulo = $entityManager->getRepository(Articulos::class)->find($id);

    if (!$articulo) {
        throw $this->createNotFoundException(
            'Articulo NO existe con ID: ' . $id
        );
    }

    $entityManager->remove($articulo);
    $entityManager->flush();

    return new Response("Articulo con ID " . $id . " eliminado!");
}
```

2. Y probamos en el navegador:
   http://localhost:8000/elimina-articulo/2
   http://localhost:8000/consultar-articulos

5. Forms

Tabla de contenidos

- Recursos
  - https://diego.com.es/creacion-de-formularios-en-symfony
  - https://symfony.com/doc/current/forms.html
- Para crear formularios, lo primero es instalar el módulo correspondiente:

```
composer require symfony/form
```

El proceso completo será el siguiente:

1. Crear la entidad para gestionar el formulario.
   En nuestro caso, usaremos /src/Entity/Autores
2. Creamos el controlador para gestionar el formulario.
   En nuestro caso, usaremos /src/Controller/AutoresController que ya tenemos
3. Añadimos el siguiente método para visualizar la tabla autores:

```
use Symfony\Component\HttpFoundation\JsonResponse;

//...
// class AutoresController extends AbstractController
// ...

#[Route('/consultar-autores', name: 'consultar-autores')]
    public function consultarAutores(
        ManagerRegistry $doctrine
    ): JsonResponse {
        $autores = $doctrine->getRepository(Autores::class)->findAll();
        $json = array();
        foreach ($autores as $autor) {
            $json[] = array(
                'id' => $autor->getId(),
                'nombre' => $autor->getNombre(),
                'edad' => $autor->getEdad(),
            );
        }

        return new JsonResponse($json);
    }
```

4. Agregamos las siguientes clases adicionales al controlador:
   En src/Controller/AutoresController.php

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
```

4. Y creamos un método nuevo para definir el formulario:
   En src/Controller/AutoresController.php

```
#[Route('/nuevo-autor', name: 'nuevo-autor')]
public function nuevoAutor(
    Request $request,
    ManagerRegistry $doctrine
) {
    $autor = new Autores();
    $form = $this->createFormBuilder($autor)
        ->add('nombre', TextType::class)
        ->add('edad', NumberType::class)
        ->add(
            'Guardar',
            SubmitType::class,
            array('label' => 'Crear Autor')
        )
        ->getForm();

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $autor = $form->getData();

        $em = $doctrine->getManager();
        $em->persist($autor);
        $em->flush();

        // Cuando mandamos el formulario, vemos la tabla en JSON
        return $this->redirectToRoute('consultar-autores');
    }

    return $this->render('autores/index.html.twig', array(
        'form' => $form->createView(),
    ));
}
```

5. Tenemos que modificar el twig:
   En templates/autores/index.html.twig

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

{% block title %}Formulario Autores
{% endblock %}
```

```
{% block body %}
  <style>
    .example-wrapper {
      margin: 1em auto;
      max-width: 800px;
      width: 95%;
      font: 18px / 1.5 sans-serif;
    }
    .example-wrapper code {
      background: #F5F5F5;
      padding: 2px 6px;
    }
  </style>

  <div class="example-wrapper">
    {{ form_start(form) }}
    {{ form_widget(form) }}
    {{ form_end(form) }}
  </div>
{% endblock %}
```

6. (Opcional) Si queremos añadir un campo booleano:
   Usamos el CheckBox

- Primero añadimos el campo booleano sexo a Autores

```
php bin/console make:entity
Class name of the entity to create or update (e.g. BraveElephant):
 > Autores

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > sexo

 Field type (enter ? to see all types) [string]:
 > boolean

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Autores.ph
```

- Y ahora hacemos la migración

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

- Y agregamos el nuevo input al controlador:
  En src/Controller/AutoresController.php

```
// ...
//$form = $this->createFormBuilder($autor)
  ->add(
      'sexo',
      ChoiceType::class,
      [
          'choices' => [
              'mujer' => true,
              'hombre' => false,
          ],
          'expanded' => true
      ]
  )
  /*
  ->add(
      'Guardar',
      SubmitType::class,
      array('label' => 'Crear Autor')
  )
  ->getForm();
  */
```

7. (Opcional) Si queremos podemos crear un ENDPOINT para actualizar los datos de los autores
   En src/Controller/AutoresController.php

```
// ...
#[Route(
        '/cambia-autor/{sexo}/{id}',
        name: 'actualizar-autor'
    )]
public function cambiarAutor(
    ManagerRegistry $doctrine,
    int $sexo,
    int $id,
): Response {
    $entityManager = $doctrine->getManager();

    $autor =
        $entityManager->getRepository(Autores::class)->find($id);

    if (!$autor) {
        throw $this->createNotFoundException(
            'Autor NO existe'
        );
    }
    $autor->setSexo($sexo);
    $entityManager->flush();
    // Aprovechamos el método anterior para presentar el registro cambiado
```

```
        return $this->redirectToRoute('consultar-autores');
    }
```

Formulario -> Pokemons

Tabla de contenidos

- En templates/pokemons/index.html.twig

```
{% block title %}
  Hello PokemonsController!
{% endblock %}
{% form_theme formulario 'bootstrap_5_layout.html.twig' %}

{% block body %}
  <!-- ... -->

  <div class="example-wrapper">
    <h1>Formulario</h1>
    {{ form(formulario) }}
  </div>
{% endblock %}
```

- En src/Controller/PokemonsController

```
#[Route('/formulario-pokemons', name: 'pokemon7')]
    public function formularioPokemons(
        CategoriasRepository $repoCategorias,
        PokemonsRepository $repoPokemons,
        TiposRepository $repoTipos,
        ManagerRegistry $doctrine,
        Request $request
    ): Response {
        $pokemon = new Pokemons();
        $formulario = $this->createFormBuilder($pokemon)
            // El formato de campo será...
            // ->add (variable, tipoCampo, etiqueta)
            // variable debe ser la misma de las entidades!!
            ->add('id', NumberType::class, ['label' => 'ID'])
            ->add('nombre', TextType::class, ['label' => 'Nombre Pokemon'])
            ->add('sexo', ChoiceType::class, [
                'choices'  => [
                    'Ninguno' => null,
                    'Femenino' => true,
                    'Masculino' => false,
                ],
            ])
            ->add('sexo2', ChoiceType::class, [
                'choices'  => [
                    'Ninguno' => null,
                    'Femenino' => true,
                    'Masculino' => false,
                ],
```

```
            ])
            ->add('evolucion', ChoiceType::class, [
                'choices'  => [
                    'Si' => true,
                    'No' => false,
                ],
            ])
            ->add('idTipo', EntityType::class, [
                'class' => Tipos::class,
                'placeholder' => 'Elije Tipo2',
                'choice_label' => 'tipo',
                'label' => 'Tipo1 Pokemon',
            ])
            ->add('idTipo2', EntityType::class, [
                'class' => Tipos::class,
                'required'   => false,
                'empty_data' => null,
                'placeholder' => 'Elije Tipo2',
                'choice_label' => 'tipo',
                'label' => 'Tipo2 Pokemon',
            ])
            ->add('idCategoria', EntityType::class, [
                'class' => Categorias::class,
                'choice_label' => 'categoria',
                'label' => 'Categoria Pokemon',
            ])
            ->add(
                'Guardar',          // Ojo, es una variable! No meter caract es
peciales
                SubmitType::class,
                ['label' => 'Guardar Pokemon']
            )
            ->getForm();

        $formulario->handleRequest($request);
        if ($formulario->isSubmitted() && $formulario->isValid()) {
            $pokemon = $formulario->getData();
            $em = $doctrine->getManager();
            $em->persist($pokemon);
            $em->flush();
            // Cuando mandamos el formulario, vemos la tabla en JSON
            return $this->redirectToRoute('pokemon2');
        }

        return $this->render(
            "pokemons/index.html.twig",
            ["formulario" => $formulario->createView()]
        );
    }
```

6. GraphQL

Tabla de contenidos

Instalación

- Recursos
  - https://www.linkedin.com/pulse/graphql-server-symfony4-abhishek-mishra
  - https://betterprogramming.pub/graphql-api-symphony-mongodb-c866a79fdf48
  - https://latteandcode.medium.com/symfony-primeros-pasos-con-overblog-graphqlbundle-f4ef937c8fb

1. Creamos la rama y nos introducimos en ella

```
cd $HOMEPROJECTS/symfony5-manual
git branch 4-GraphQL
git checkout 4-GraphQL
git push --set-upstream origin 4-GraphQL
```

2. Entramos en el proyecto e instalamos el bundle de GraphQL

```
cd $HOMEPROJECTS/symfony5-manual
composer require overblog/graphql-bundle
```

> IMPORTANTE: Si nos sale en la instalación Do you want to execute this recipe?, pulsamos Y [INTRO]

3. Es muy recomendable instalar la interfaz gráfica para probar la API:

```
composer req --dev overblog/graphiql-bundle
symfony server:start
```

> NOTAS ADICIONALES
> Define your schema, read documentation
> https://github.com/overblog/GraphQLBundle/blob/master/docs/definitions/index.md
> If you want to see your dumped schema (really not necessary for bootstrap): run bin/console graphql:dump-schema

4. Podemos ver el editor de GrapQL llendo a esta dirección:
   http://127.0.0.1:8000/graphiql

5. Ahora debemos añadir un prefijo a nuestras rutas para graphQL:

- Nos vamos a config/routes/graphql.yaml y dejamos puesto esto:

```
overblog_graphql_endpoint:
  resource: "@OverblogGraphQLBundle/Resources/config/routing/graphql.yml"
  prefix: graphql
```

7. API REST

Tabla de contenidos

- https://www.itdo.com/blog/primeros-pasos-con-symfony-5-como-api-rest/
- https://soka.gitlab.io/angular/conceptos/http/consumir-servicios-api-rest/consumir-servicios-api-rest/
- https://codingpotions.com/angular-servicios-llamadas-http

1. Lo primero es instalar Postman

- https://bytexd.com/how-to-install-postman-on-ubuntu/

```
# Instalar Postman (ojo es 1 línea)
tar -C /tmp/ -xzf <(curl -
L https://dl.pstmn.io/download/latest/linux64) && sudo mv /tmp/Postman /opt/

# Ejecutamos Postman
/opt/Postman/Postman
```

8. SELECT

Tabla de contenidos

- Vamos a poner en este tema TODOS los elementos SELECT que hemos visto:
- Resumen de comandos SQL
    - USE -> Entrar en la BBDD
    - LIMIT -> Limita el nº de registros de salida
    - WHERE -> Filtrado de registros
    - WHERE...IN -> Filtro por varios registros
    - ORDER BY ASC/DESC -> Ordenación de datos
    - DISTINCT -> Distingue entre valores iguales
    - AS -> Alias, cambiar el nombre del campo en la salida
    - OPERADORES -> Son !=, <, >, <=, =>
    - Funciones AGREGACIÓN
        - AVG -> Media
        - COUNT -> Conteo de registros
        - MAX -> Valor máximo
        - MIN -> Valor mínimo
    - GROUP BY -> Agrupar registros
      -> OJO! Si se pone con el ORDER BY, ponerlo antes
- JOIN -> Unir tablas
    - Se emplea WHERE tabla1.campoA = tabla2. campoB

```
 USE ine;   # Entrar en BBDD

 # LIMIT -> Limita el nº de registros de salida
 SELECT * FROM municipios LIMIT 5;

 # SELECT campo1, campo2,..., campoN
 SELECT municipio, codprovincia FROM municipios LIMIT 5;

 # WHERE ->  Filtrado
 SELECT municipio FROM municipios WHERE codprovincia = 41;

 # WHERE...IN -> Filtro por varios registros
SELECT municipio FROM municipios WHERE codprovincia IN(41,11,14);

# ORDER BY ASC/DESC -> Ordenación de datos
SELECT municipio, codprovincia
FROM municipios
WHERE codprovincia IN(41,11,14)
```

```
ORDER BY codprovincia DESC;


# DISTINCT -> Distingue entre valores iguales
SELECT dc FROM municipios LIMIT 10;
SELECT DISTINCT dc FROM municipios LIMIT 10;

SELECT codprovincia FROM municipios LIMIT 200;
SELECT DISTINCT codprovincia FROM municipios LIMIT 200;

# Alias y operadores
SELECT municipio AS Poblaciones FROM municipios LIMIT 5;

SELECT DISTINCT codprovincia AS Provincias
FROM municipios
WHERE codprovincia > 41
LIMIT 5;

# El campo FILTRO no tiene porqué coincidir con la salida
SELECT municipio AS Poblaciones
FROM municipios
WHERE codprovincia = 41
LIMIT 10;

# Ejemplo: Las 5 provincias anteriores a Sevilla
DESCRIBE provincias;
SELECT provincia AS Provincias
FROM provincias
WHERE codprovincia < 41
ORDER BY codprovincia DESC
LIMIT 5;

# Funciones AGREGACIÓN -> AVG y COUNT
# Dame el número de poblaciones de Sevilla
SELECT COUNT(municipio)
FROM municipios
WHERE codprovincia = 41;

# Suma de poblaciones de Sevilla, Cádiz y Córdoba
SELECT COUNT(municipio)
FROM municipios
WHERE codprovincia IN (41,11,14);

# La misma consulta usand puertas lógica (peor rendimiento)
SELECT COUNT(municipio)
FROM municipios
WHERE codprovincia = 41
OR codprovincia = 11
OR codprovincia = 14;

# Dame la ultima ID de municipios de ALAVA (902-Lantarón)
SELECT MAX(codmunicipio)
FROM municipios
WHERE codprovincia = 1;
```

```
SELECT MAX(codmunicipio)
FROM municipios
WHERE codprovincia = 41;

# Sacar nº de poblaciones separadas de Sevilla, Cádiz y Córdoba
SELECT COUNT(codprovincia) AS "Nº Poblaciones",
codprovincia AS "Cod Provincia"
FROM municipios
WHERE codprovincia IN(41,11,14)
GROUP BY codprovincia;


# Nº de poblaciones de otras provincias DISTINTAS a Sevilla
SELECT COUNT(codprovincia)
FROM municipios
WHERE codprovincia <> 41;


# Sacar el Modelo (diagrama) de la BBDD
# Database > Reverse Engineer (Ingenieria Inversa)

USE ine;
SELECT *
FROM municipios
WHERE municipio = "Marbella";

SELECT provincia FROM provincias;

# Sacar nº de poblaciones separadas de
# Sevilla, Cádiz y Córdoba
# poniendo los nombres de las provincias
# y la región a la que pertenecen
# UNIMOS LAS 3 TABLAS de la BBDD!!
# Esto se llama JOIN

SELECT COUNT(codmunicipio) AS "Nº Poblaciones",
provincias.provincia AS "Provincia",
ccaa.comunidad AS "Región"
FROM ccaa, provincias, municipios
WHERE ccaa.codauto = provincias.codauto
AND provincias.codprovincia = municipios.codprovincia
AND municipios.codprovincia IN(41,11,14)
GROUP BY municipios.codprovincia;
```

9. CodeAnyWhere

Tabla de contenidos

- https://codeanywhere.com/signin
    - Pulsar en Sign Up y rellenar el formulario
    - Ir a nuestro correo y verificar pulsando el enlace
    - Entramos en la página con Sign In. Vemos el Dashboard (Tablero)
    - PUlsar New Container. Elegimos PHP, nombre: symfony y [Create]
- Una vez abierto el contenedor instalamos Symfony CLI y creamos el proyecto

```
curl -1sLf 'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' | sudo
 -E bash
sudo apt install symfony-cli
symfony
cd ~/workspace
symfony new symfony --version=5.4 --webapp
cd symfony
```

- Entramos en la carpeta del proyecto e instalamos las dependencias

```
cd symfony
composer require --dev symfony/maker-bundle
composer require twig
composer require annotations
composer require symfony/orm-pack
composer require symfony/form
```

- IMPORTANTE: EN el lado derecho de la pantalla sale la previsualziación de la página en el navegador.
  Hay que cambiar el puerto que viene por defecto 3000 a 8000
    - https://port-3000-symfony-diweb2022771506.preview.codeanywhere.com
    - https://port-8000-symfony-diweb2022771506.preview.codeanywhere.com
- MySQL dentro del contenedor

```
mysql -u root
```

10. ANEXO

Tabla de contenidos

- Os voy a mandar los métodos de CochesController con algunos comentarios de lo que hemos visto en clase:
- En primer lugar los use...

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

use App\Entity\Coches;
use Doctrine\Persistence\ManagerRegistry;
use App\Repository\CochesRepository;
use Symfony\Component\HttpFoundation\JsonResponse;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
```

- Y luego los método para ACTUALIZAR, BORRAR, y el formulario...

```
#[Route(
        '/cambia-coche/{potencia}/{electrico}',
        name: 'cambia-coche'
    )]
    public function cambiaCoche(
        ManagerRegistry $doctrine,
        int $potencia,
        bool $electrico
    ): Response {
        $entityManager = $doctrine->getManager();
        $coches =
            $entityManager->getRepository(Coches::class)->findAll();

        /* OJO! Si vamos a añadir potencia
        a la que tuviera el coche previamente
        obtenemos el dato de la BBDD y le sumamos
        la nueva potencias*/
        foreach ($coches as $coche) {
            $potenciaPrevia = $coche->getPotencia();
            $coche->setPotencia($potenciaPrevia + $potencia);
            $coche->setElectrico($electrico);
            $entityManager->persist($coche);
```

```
            $entityManager->flush();
        }

        // Aprovechamos el método anterior para presentar el registro cambiado
        return $this->redirectToRoute('ver-coches');
    }

    #[Route('/elimina-coche/{id}', name: 'elimina-coche')]
    public function eliminaCoche(
        ManagerRegistry $doctrine,
        int $id
    ): Response {
        $entityManager = $doctrine->getManager();
        $coche =
            $entityManager->getRepository(Coches::class)->find($id);
        if (!$coche) {
            throw $this->createNotFoundException(
                'Coche NO existe con ID: ' . $id
            );
        }
        $entityManager->remove($coche);
        $entityManager->flush();
        return $this->redirectToRoute('ver-coches');
    }

    #[Route(
        '/ver-coches-tipo/{tipo}',
        name: 'ver-coches-tipo'
    )]
    public function verCochesTipo(
        ManagerRegistry $doctrine,
        String $tipo
    ): JsonResponse {
        // En este caso no usamos el gestor de entidades
        // Usamos sólo la conexión
        $connection = $doctrine->getConnection();
        // IMPORTANTE: si queremos personalizar la salida
        // JSON, poner alias en los campos del SELECT
        // Mucho OJO! Esto puede entrar en el teórico
        $coches = $connection
            ->prepare("SELECT modelo, nombre
                    FROM coches, tipos
                    WHERE tipos.id = tipo_id
                    AND nombre = '" . $tipo . "'")
            ->executeQuery()
            ->fetchAllAssociative();
        // Con el dump, sacamos el array completo
        return new JsonResponse(dump($coches));
    }


    #[Route('/nuevo-coche', name: 'nuevo-coche')]
    public function nuevoCoche(
        Request $request,
        ManagerRegistry $doctrine
```

```
    ) {
        $coche = new Coches();
        $form = $this->createFormBuilder($coche)
            ->add('marca', TextType::class)
            ->add('modelo', TextType::class)
            ->add('potencia', NumberType::class)

            // Para añadir el radio...
            ->add(
                'electrico',
                ChoiceType::class,
                [
                    'choices' => [
                        'si' => true,
                        'no' => false,
                    ],
                    'expanded' => true
                ]
            )
            ->add(
                'Guardar',
                SubmitType::class,
                array('label' => 'Crear Coche')
            )
            ->getForm();
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $coche = $form->getData();
            $em = $doctrine->getManager();
            $em->persist($coche);
            $em->flush();
            // Cuando mandamos el formulario, vemos la tabla en JSON
            return $this->redirectToRoute('ver-coches');
        }
        return $this->render('coches/index.html.twig', array(
            'form' => $form->createView(),
        ));
    }
```

11. Angular en Symfony

Tabla de contenidos

- Vamos a ver una pequeña práctica para ver el contenido de un endpoint en Angular.
- Se da por sentado que Angular está globalmente instalado en el equipo (ver elementary.md)
- Para empezar preparamos un endpoint para listar la tabla Aulas, con los campos num_aula (int), capacidad (int), docente (string) y hadware (bool), que devolverá la tabla al completo en formato JSON

```php
<?php

namespace App\Controller;

use App\Entity\Aulas;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

// Importamos clases abstractas de gestión BBDD
use Doctrine\ORM\EntityManagerInterface;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\JsonResponse;

#[Route('/aulas', name: 'app_aulas_')]
class AulasController extends AbstractController
{
   //...

    // APLICACIÓN AULAS
    #[Route('/consultarAulas', name: 'consultarAulas')]
    public function consultarAulas(ManagerRegistry $gestorFilas): JsonResponse
    {
        // endpoint de ejemplo: http://127.0.0.1:8000/aulas/consultarAulas
        // Saco el gestor de entidades a partir del gestor de Filas (mas genéri
co)
        $gestorEntidades = $gestorFilas->getManager();
        // Desde el gestor de entidades, saco el repositorio de mi clase
        $repoAulas =  $gestorEntidades->getRepository(Aulas::class);
        $filasAulas = $repoAulas->findAll();


        $json = array();
        foreach ($filasAulas as $aula) {
            $json[] = array(
                'numAula' => $aula->getNumAula(),
                'capacidad' => $aula->getCapacidad(),
                'docente' => $aula->getDocente(),
                'hardware' => $aula->isHardware(),
            );
        }

        return new JsonResponse($json);
    }
}
```

- Una vez terminada la parte del Backend, preparamos todo para trabajar con Angular. Pasos:

1. Instalamos las dependencias necesarias y configuramos CORS:

```
cd /var/www/html/symfony6
composer require api
composer require nelmio/cors-bundle
sudo config/packages/nelmio_cors.yaml
```

2. En el archivo de CORS vamos a permitir TODOS los acceso para simplificar. Por supuesto, esto hay que restringirlo en producción.

> [!NOTE]
> A partir de este punto veremos en los comentarios APLICACIÓN AULAS, para tener un seguimiento en el proyecto.

```
# config/packages/nelmio_cors.yaml

nelmio_cors:
    defaults:
        origin_regex: true
        # allow_origin: ['%env(CORS_ALLOW_ORIGIN)%']
        # APLICACIÓN AULAS
        allow_origin: ['*']
        allow_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']
        allow_headers: ['Content-Type', 'Authorization']
        expose_headers: ['Link']
        max_age: 3600
    paths:
        '^/': null
```

3. Una vez acabado, es recomendable limpiar la caché

```
cd /var/www/html/symfony
php bin/console cache:clear
```

4. Instalamos la aplicación de Angular DENTRO de Symfony:

```
cd /var/www/html/symfony
ng new angular-aulas
cd angular-aulas
```

5. Añadimos el nuevo módulo CORS (HTTPClient) a los módulos que se cargan dentro de la aplicación

```
cd /var/www/html/symfony/angular-aulas
code app.module.ts
```

6. Editamos el archivo agregando en la sección de imports el nuevo elemento HTTPClientModule:

```
// APLICACIÓN AULAS
// src/app/app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AulaListComponent } from './components/aula-list/aula-list.component';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
    AulaListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,  // Asegúrate de que esto esté aquí
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

7. Creamos el modelo para aula en Angular:

```
# Generamos el modelo para aula
ng generate class models/aula --type=model
code src/app/models/aula.model.ts
```

8. Implementamos el modelo con TypeScript:

```
// APLICACIÓN AULAS
// src/app/models/aula.model.ts

export interface Aula {
    numAula: number;    // Primary Key
    capacidad: number;
    docente: string;
    hardware: boolean;
  }
```

9. Creamos el servicio para aula

```
cd /var/www/html/symfony/angular-aulas
ng generate service services/aula
code src/app/services/aula.service.ts
```

10. Desarrollamos el servicio, donde pondremos el endoint:

```
// APLICACIÓN AULAS
// src/app/services/aula.service.ts

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Aula } from '../models/aula.model';

@Injectable({
  providedIn: 'root'
})
export class AulaService {

  private apiUrl = 'http://127.0.0.1:8000/aulas/consultarAulas';

  constructor(private http: HttpClient) { }

  getAulas(): Observable<Aula[]> {
    return this.http.get<Aula[]>(this.apiUrl);
  }
}
```

11. Creamos un componente, por ejemplo aula-list:

```
cd /var/www/html/symfony/angular-aulas
ng generate component components/aula-list
cd src/app/components/aula-list/
code aula-list.component.ts
```

12. Configuramos el componente, que llamará a su HTML y CSS correspondiente:

```
// APLICACIÓN AULAS
// src/app/components/aula-list/aula-list.component.ts

import { Component, OnInit } from '@angular/core';
import { AulaService } from '../../services/aula.service';
import { Aula } from '../../models/aula.model';

@Component({
  selector: 'app-aula-list',
  templateUrl: './aula-list.component.html',
  styleUrls: ['./aula-list.component.css']
})
export class AulaListComponent implements OnInit {

  aulas: Aula[] = [];

  constructor(private aulaService: AulaService) { }
```

```
  ngOnInit(): void {
    this.aulaService.getAulas().subscribe(data => {
      this.aulas = data;
    });
  }
}
```

13. Modificamos el HTML aula-list.component.html

```
<!-- APLICACIÓN AULAS -->
<!-- src/app/components/aula-list/aula-list.component.html -->

<h2>Listado de Aulas</h2>
<table class="table">
  <tr>
    <th>Número de Aula</th>
    <th>Capacidad</th>
    <th>Docente</th>
    <th>Hardware</th>
  </tr>
  <tr *ngFor="let aula of aulas">
    <td>{{ aula.numAula }}</td>
    <td>{{ aula.capacidad }}</td>
    <td>{{ aula.docente }}</td>
    <td>{{ aula.hardware ? 'Sí' : 'No' }}</td>
  </tr>
</table>
```

14. Ya solo nos queda agregar el componente en el archivo que carga Angular al ejecutar su servidor. Nos vamos a /angular-aulas/src/app/app.component.html:

```
<!-- Toolbar -->
<div class="toolbar" role="banner">
  <!-- //... -->
  <span>¡Saludos desde Symfony 6 y Angular 16!</span>
    <!-- //... -->
</div>

<section class="w-50 p-3 m-5">
<app-aula-list></app-aula-list>
</section>
```

15. Para visualizarlo, abrirmos el servidor:

```
cd /var/www/html/symfony6/angular-aulas
ng serve
# Y lo veremos en el navegador: http://localhost:4200
```

16. Como alternativa, podemos crear una ruta para visualiar el endpoint a secas. Para ello editamos el archivo /angular-aulas/src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AulaListComponent } from './components/aula-list/aula-list.component';

const routes: Routes = [
  { path: '/verAulas', component: AulaListComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

17. Y podremos ver de nuevo la tabla en el navegador en otra URL distinta: http://localhost:4200/verAulas