

Manual Symfony

Programación Web
Grado en Ingeniería Informática



Luis Fernando Torres Moya

Índice

1.- Instalación y Configuración	— Página 3
2.- Lucky Number	— Página 5
3.- Renderizar una Plantilla	— Página 7
4.- Crear nuestros Controladores	— Página 10
5.- La Entidad Doctrine	— Página 14
6.- Entidades y migración a la BBDD	— Página 15
7.- Página Web Artículos	— Página 21

Explicación del Manual

En este documento he tomado una directriz más parecida a la hecha en manuales anteriores. Lo que he hecho ha sido seguir el tutorial para aprender las capacidades básicas de Symfony (Apartados 1 a 6), explicando paso a paso lo que se ha hecho para ello. De la misma manera el apartado 7 contendrá explicaciones mínimas y capturas de pantalla de la web. Explicaciones mínimas ya que siguiendo el tutorial ya he explicado con detalle lo que he hecho y arreglado los errores que me han ido surgiendo, por lo que volver a repetir esa explicación sería redundante.

1.- Instalación y Configuración

En este manual vamos a utilizar el instalador de aplicaciones Composer, que ya hemos utilizado en el manual anterior. Siguiendo el tutorial vamos a crear el proyecto:

```
luisfernandotorresmoya@wlanipr34215 Symfony % composer create-project symfony/website-skeleton tutorial3
Creating a "symfony/website-skeleton" project at "./tutorial3"
Info from https://repo.packagist.org: #StandWithUkraine
Installing symfony/website-skeleton (v6.0.99)
  - Downloading symfony/website-skeleton (v6.0.99)
  - Installing symfony/website-skeleton (v6.0.99): Extracting archive
Created project in /Users/luisfernandotorresmoya/Desktop/ESI/PW/Symfony/tutorial3
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking symfony/flex (v2.1.7)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading symfony/flex (v2.1.7)
  - Installing symfony/flex (v2.1.7): Extracting archive
Generating optimized autoload files
1 package you are using is looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.
```

```
tutorial3 -- zsh - 80x24

1. Install the messenger component by running composer require messenger;
2. Add 'Symfony\Component\Mailer\Messenger\SendEmailMessage': amqp to the
   config/packages/messenger.yaml file under framework.messenger.routing
   and replace amqp with your transport name of choice.

* Read the documentation at https://symfony.com/doc/master/mailers.html

doctrine/doctrine-bundle instructions:

* Modify your DATABASE_URL config in .env

* Configure the driver (postgresql) and
  server_version (13) in config/packages/doctrine.yaml

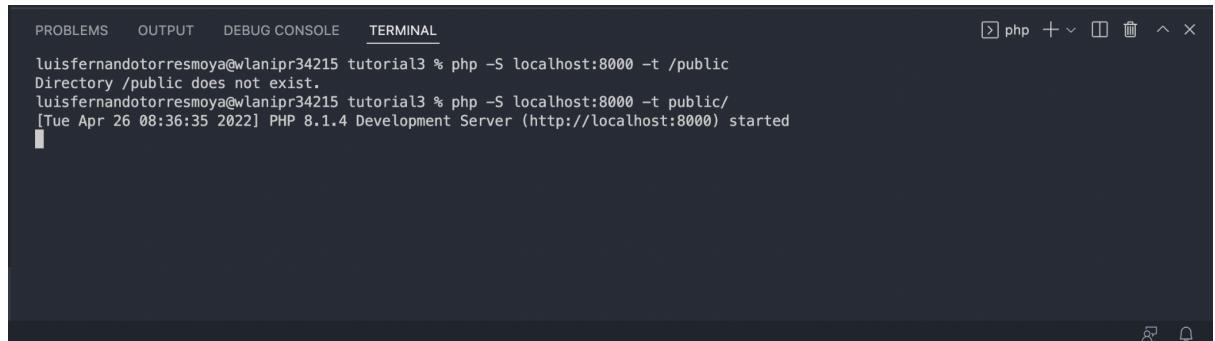
symfony/phpunit-bridge instructions:

* Write test cases in the tests/ folder
* Use MakerBundle's make:test command as a shortcut!
* Run the tests with php bin/phpunit

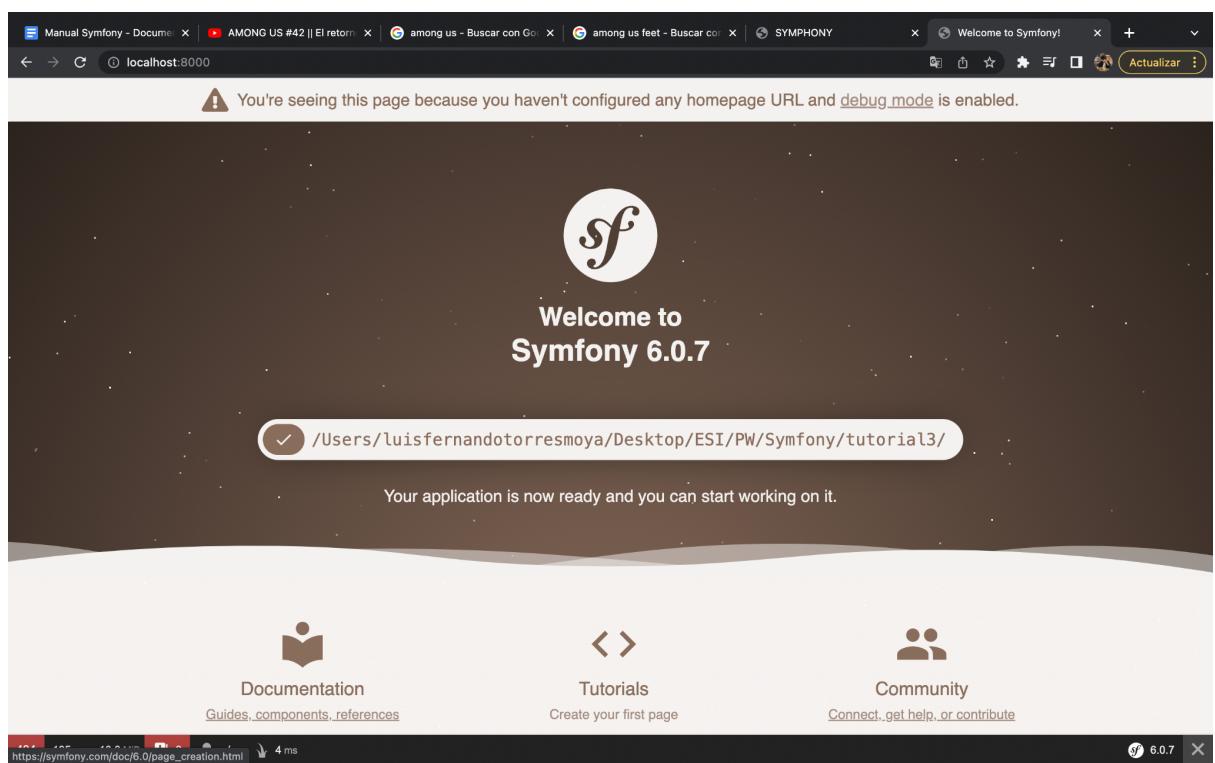
luisfernandotorresmoya@wlanipr34215 Symfony % cd tutorial3
luisfernandotorresmoya@wlanipr34215 tutorial3 %
```

Una vez hecho eso, podemos comprobar que todo haya funcionado haciendo igual que en los manuales anteriores, lanzando el proyecto con el comando:

```
php -S localhost:8000 -t /public
```

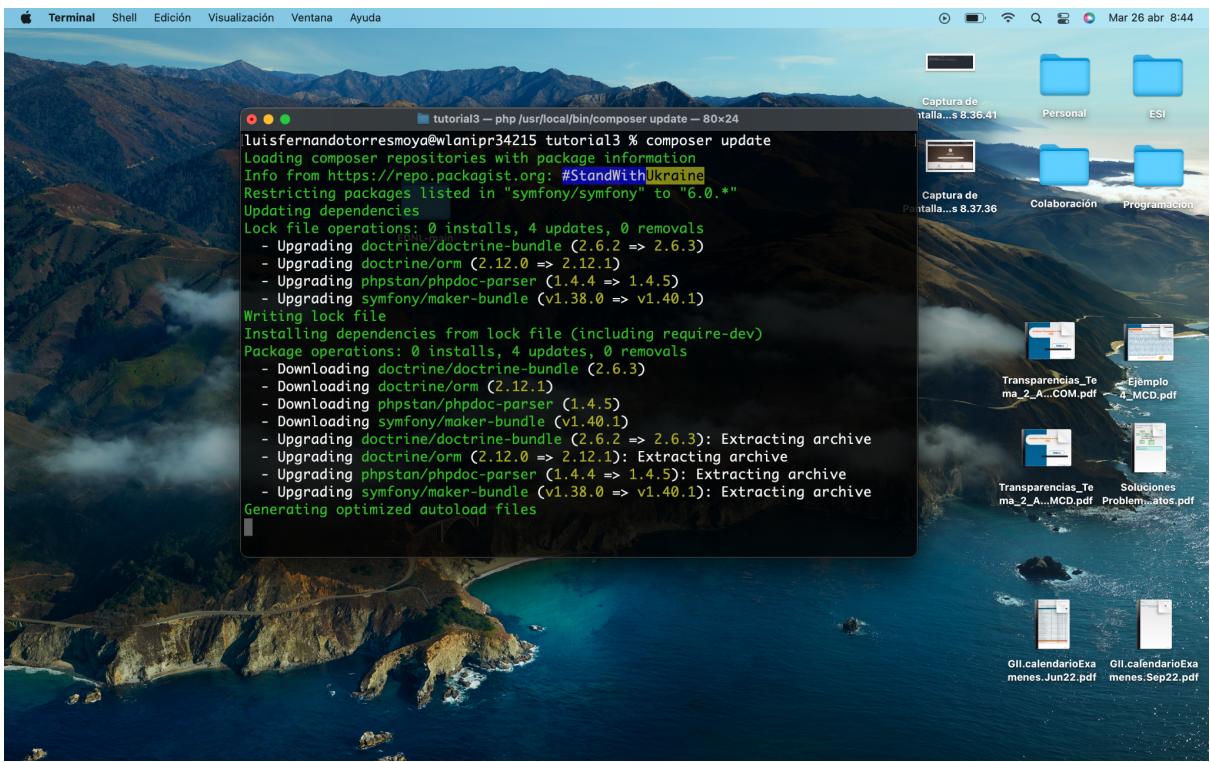


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
luisfernandotorresmoya@wlanipr34215 tutorial3 % php -S localhost:8000 -t /public
Directory /public does not exist.
luisfernandotorresmoya@wlanipr34215 tutorial3 % php -S localhost:8000 -t public/
[Tue Apr 26 08:36:35 2022] PHP 8.1.4 Development Server (http://localhost:8000) started
```



Ya lo tenemos todo listo para empezar a trabajar en nuestro proyecto usando Modelos, Vistas y Controladores. Pero antes vamos a hacer una pequeña cosita para quedarnos con la conciencia tranquila. Vamos a actualizar el composer por si algún casual tenemos alguna actualización que hacer.

```
composer update
```



Pues ya está todo listo. Vamos a pasar a hacer nuestro primer controlador.

2.- Lucky Number

Vamos a crear nuestro primer controlador, y ya de paso (como se necesita para crearlo), también crearemos nuestra primera ruta al controlador.

Primero que todo nos vamos a la carpeta *controllers* y creamos un fichero php que se llama como el controlador que queremos y dentro hacemos la función y lo que queremos que haga el controlador.

Como todavía no le hemos echado un vistazo a las vistas (valga la redundancia), vamos a chapucear un *echo* que enseñe lo que nosotros queremos. Ya luego cuando veamos la parte de las vistas podemos volver a esta parte y hacer una adecuada para este controlador, pero ya vamos viendo.

El fichero *Controllers/LuckyController.php* queda de la siguiente manera:

```

src > Controller > LuckyController.php
1 <?php
2
3 namespace App\Controller;
4
5 use Symfony\Component\HttpFoundation\Response;
6
7 class LuckyController{
8
9     public function number(): Response{
10
11         $number = random_int(0, 100);
12
13         return new Response(
14             '<html><body>Lucky number: ' . $number . '</body></html>'
15         );
16
17     }
18
19 }

```

Ln 2, Col 1 Spaces: 4 UTF-8 LF PHP

Como se puede ver en la imagen, es bastante parecido a lo que hemos visto ya con Laravel y Codeigniter, solo que algo más simple. Vamos a usar la variable que hemos sacado con el `use`, `response` para enseñar por pantalla lo que vamos a hacer. Dentro de la clase creamos la función que realiza el random del número y retornamos con la variable anteriormente mencionada.

Pero todavía no tenemos una ruta definida para poder ver este número. ¿Veis el patrón? Todos los Framework que hemos visto funcionan igual, necesitamos siempre crear primero el controlador, y luego darle un espacio en la web mediante una ruta.

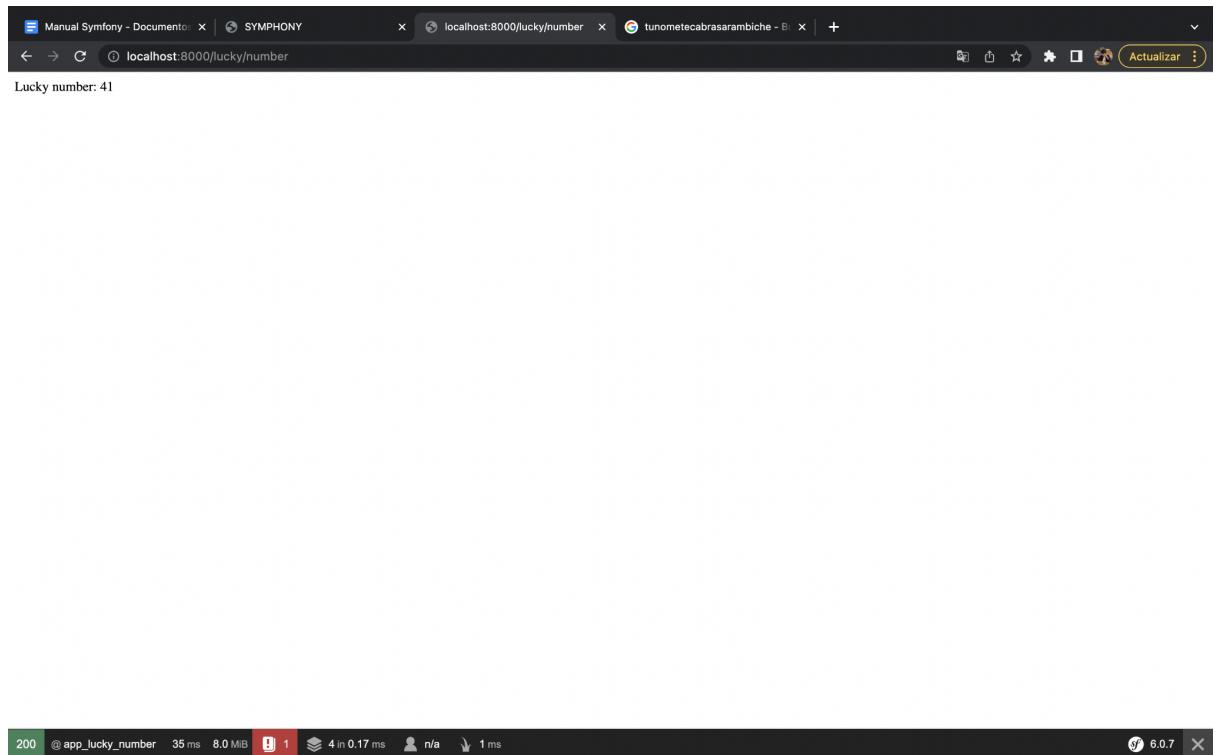
En este caso las rutas las definimos en `config/routes.yaml` con el siguiente código:

```

config > ! routes.yaml
1 controllers:
2     resource: ../src/Controller/
3     type: annotation
4
5 kernel:
6     resource: ../src/Kernel.php
7     type: annotation
8
9 app_lucky_number:
10    path: /lucky/number
11    controller: App\Controller\LuckyController::number
12

```

Ahora si todo funciona adecuadamente, si utilizamos el comando que vimos anteriormente para inicializar la página y accedemos a la ruta `localhost:8000/lucky/number` obtendremos una solución:



Un detalle a tener en cuenta es la barra de abajo del todo, la cual nos da algo de información sobre el estado actual de nuestra aplicación, el manual no se explora mucho en eso pero a mi me parece que es una característica muy útil de Symfony y debería de tenerla más en cuenta.

3.- Renderizar una Plantilla

Lo que acabamos de hacer, aunque funciona, es una chapuza más grande que los ordenadores de la ESI. Ya que lo que hacemos es una especie de `echo` que hace que nos enseñe el resultado. Eso ya no lo vamos a usar más, vamos a aprender a crear plantillas que rendericen nuestra información y la enseñen por pantalla, algo más bonito. Pero antes que nada requerimos de un par de cositas más del `composer`. Necesitamos `twig` y `annotations`.

Como siempre, vamos a la carpeta del proyecto y usamos require.

```
luisfernandotorresmoya@wlanipr34215 tutorial3 % composer require twig
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^1.0 for symfony/twig-pack
./composer.json has been updated
Running composer update symfony/twig-pack
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking symfony/twig-pack (v1.0.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing symfony/twig-pack (v1.0.1): Extracting archive
Generating optimized autoload files
107 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

Unpacking Symfony packs
- Unpacked symfony/twig-pack
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
```

```
luisfernandotorresmoya@wlanipr34215 tutorial3 % composer require annotations
Using version ^6.2 for sensio/framework-extra-bundle
./composer.json has been updated
Running composer update sensio/framework-extra-bundle
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
106 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

Executing script cache:clear [OK]
Executing script assets:install public [OK]

luisfernandotorresmoya@wlanipr34215 tutorial3 %
```

Con eso ya podemos hacer nuestra plantilla en el fichero en el cual creamos nuestro controlador *LuckyNumber*:

The screenshot shows the Visual Studio Code (VS Code) interface. The left sidebar contains the Explorer view with a tree structure of files and folders, including 'Tutorial3' (bin, config, migrations, public, src), 'Controller' (LuckyController.php, .gitignore), 'Entity', 'Repository', 'Kernel.php', 'templates' (lucky, number.html.twig, base.html.twig), 'tests', 'translations', 'var', 'vendor', '.env', '.env.test', '.gitignore', composer.json, composer.lock, phpunit.xml.dist, and symfony.lock. The center is the code editor with the file 'LuckyController.php' open. The code defines a controller class that extends AbstractController, has a route annotation, and returns a rendered twig template. The right side features the Terminal tab showing logs from a local server, and the bottom right shows status information like 'Ln 19, Col 77' and file encoding.

```

LuckyController.php
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

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

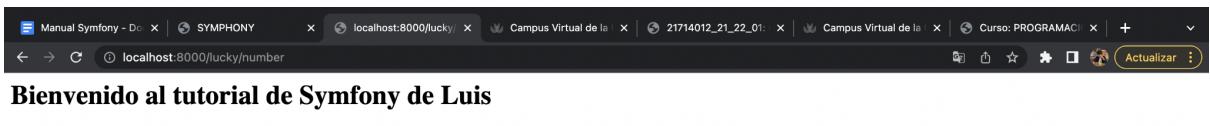
class LuckyController extends AbstractController{
    /**
     * @Route("/lucky/number")
     */
    public function number(): Response{
        $number = random_int(0, 100);

        return new $this->render('number.html.twig', ['number' => $number]);
    }
}

```

Como se puede ver incluimos más contenido para poder usarlo, y extendemos la clase por defecto del controlador abstracto. Luego se le pasa la ruta donde está el template de *twig* que va a devolver el controlador.

En este fichero *number.html.twig* tenemos un código pseudo-html que llamamos para que se enseñe el controlador de forma más o menos visible.

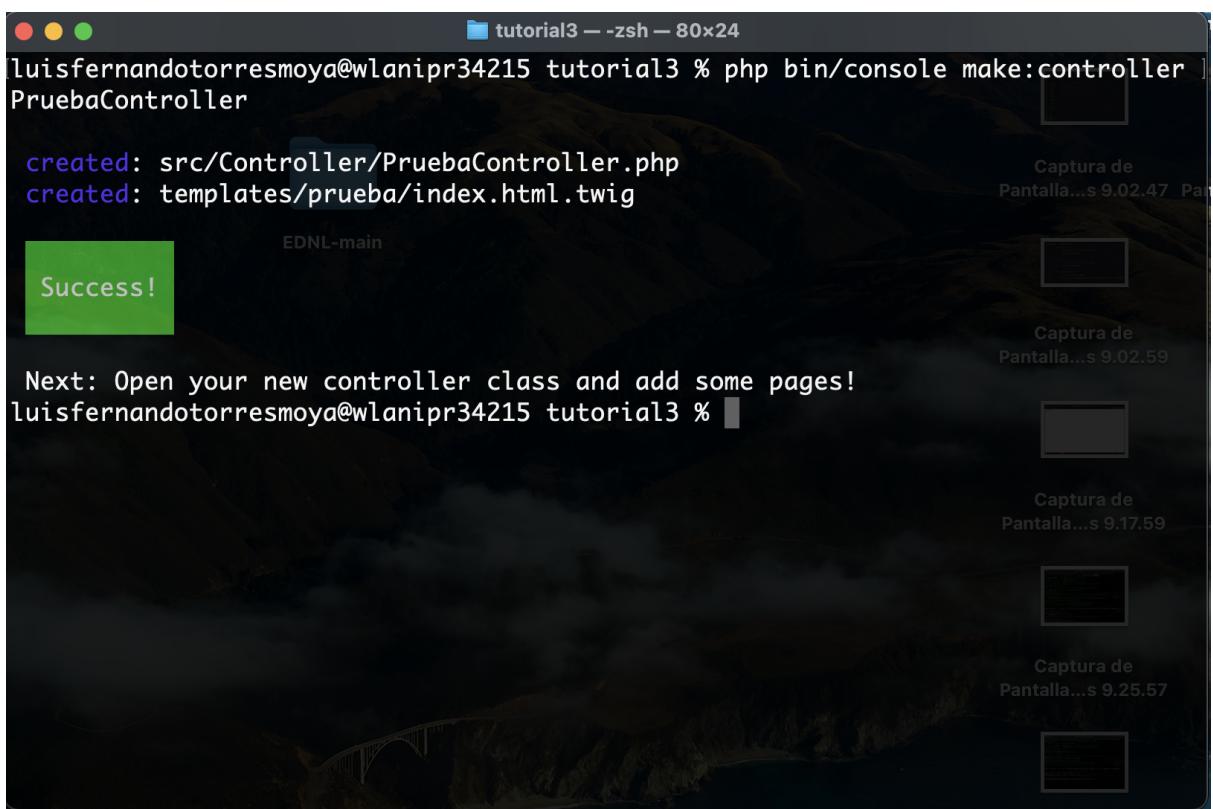


4.- Crear nuestros Controladores

Hemos aprendido a crear controladores muy básicos que enseñan por pantalla un numerito. Bastante lejos de nuestro sueño de crear páginas web funcionales. Así que vamos a subir una marcha y vamos a crear controladores más complejos y que hagan lo que nosotros queramos.

Empecemos por hacer una pequeña introducción, una prueba *of sorts*, de como crear un controlador funcional. Para esto, vamos a usar la terminal, que nos va a crear un controlador de forma automática si introducimos el siguiente comando:

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



The screenshot shows a terminal window titled 'tutorial3 -- zsh -- 80x24'. The command 'php bin/console make:controller PruebaController' is being run. The output indicates that two files were created: 'src/Controller/PruebaController.php' and 'templates/prueba/index.html.twig'. A green 'Success!' message is displayed at the bottom left. The background of the terminal window features a dark landscape with mountains and a bridge.

Como se puede ver se nos han creado dos archivos, no solo el controlador, sino que también el template. Bastante mejor que hacerlo a mano ¿no?

Sin siquiera empezar a añadir código, nos encontramos con que los archivos están ya llenos:

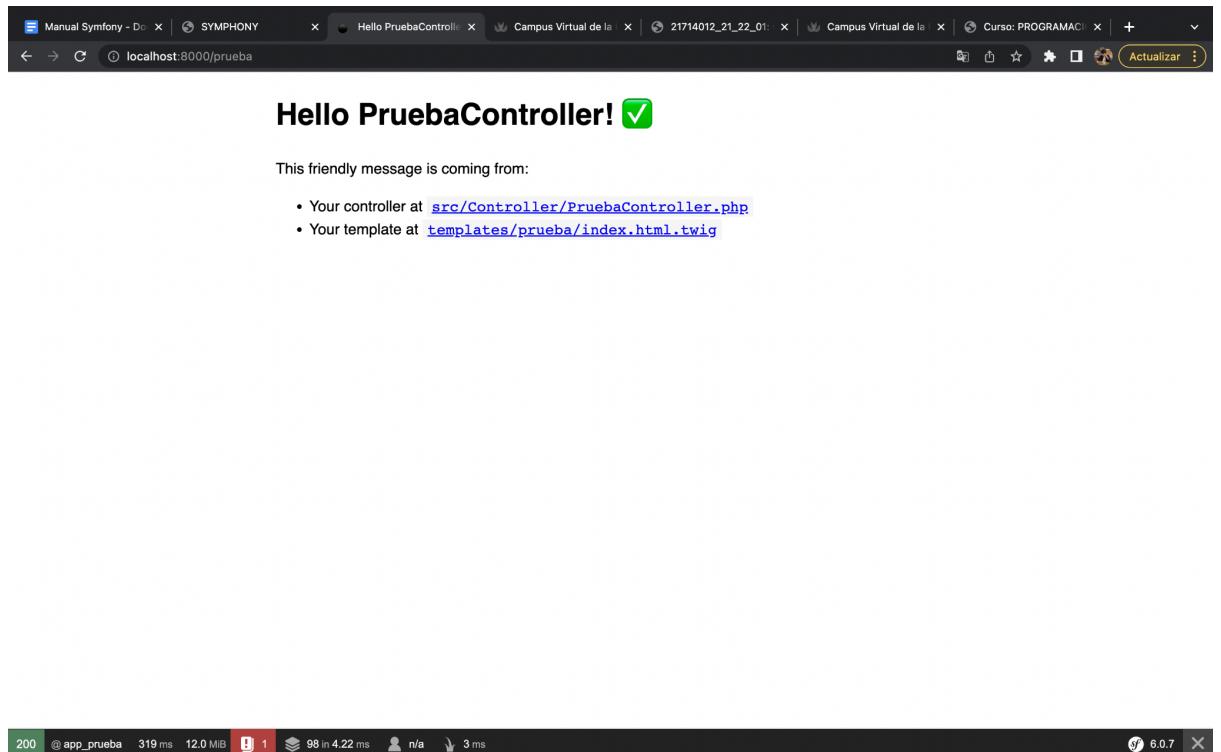
The screenshot shows the VS Code interface with the following details:

- EXPLORER** panel on the left showing the project structure:
 - TUTORIAL...
 - bin
 - config
 - migrations
 - public
 - src
 - Controller
 - .gitignore
 - LuckyController.php
 - PruebaController.php
 - Entity
 - Repository
 - Kernel.php
 - templates
 - lucky
 - number.html.twig
 - prueba
 - base.html.twig
 - tests
 - translations
 - var
 - vendor
 - .env
 - .env.test
 - .gitignore
 - composer.json
 - composer.lock
 - phpunit.xml.dist
 - symfony.lock
- PruebaController.php** file open in the main editor pane, showing PHP code for a controller.
- TERMINAL** pane at the bottom showing logs from a Symfony application.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** panel on the left showing the project structure, identical to the previous screenshot.
- index.html.twig** file open in the main editor pane, showing Twig template code.
- TERMINAL** pane at the bottom showing logs from a Symfony application.

Y si ahora (primero nos aseguramos de que el servidor está activado) nos vamos a la ruta del controlador, vemos que hasta está ya enrutado. Impresionante.



Antes de irnos al siguiente apartado, el cual es bastante importante, vamos a crear los archivos necesarios para el CRUD con el siguiente comando:

```
php bin/console make:crud Product
```

A screenshot of a terminal window titled 'tutorial3 — zsh — 80x24'. The command 'php bin/console make:crud Product' is being run. The output shows 'Success!' in a green box, followed by instructions to open the new controller class and add pages. It then prompts for a controller class name ('ProductController') and shows an error message: '[ERROR] There are no registered entities; please create an entity before using this command.' The terminal prompt 'luisfernandotorresmoya@wlanipr34215 tutorial3 %' is visible at the bottom.

Como se puede ver, estamos cometiendo un error. Lo que está pasando es que nos dice que no tenemos entidades creadas. Si volvemos al manual de la asignatura vemos una anotación a pie de página que nos indica que antes de hacer el paso, tenemos que crear la entidad *Doctrine*. Pues bien, tenemos que añadir el doctrine en nuestro proyecto tal que así:

```
luisfernandotorresmoya@wlanipr34215 tutorial3 % composer require symfony/orm-pack
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^2.2 for symfony/orm-pack
./composer.json has been updated
Running composer update symfony/orm-pack
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking symfony/orm-pack (v2.2.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing symfony/orm-pack (v2.2.0): Extracting archive
Generating optimized autoload files
107 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

Unpacking Symfony packs
- Unpacked symfony/orm-pack
Loading composer repositories with package information
Updating dependencies
```

```
luisfernandotorresmoya@wlanipr34215 tutorial3 % composer require --dev symfony/maker-bundle
Using version ^1.40 for symfony/maker-bundle
./composer.json has been updated
Running composer update symfony/maker-bundle
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
106 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

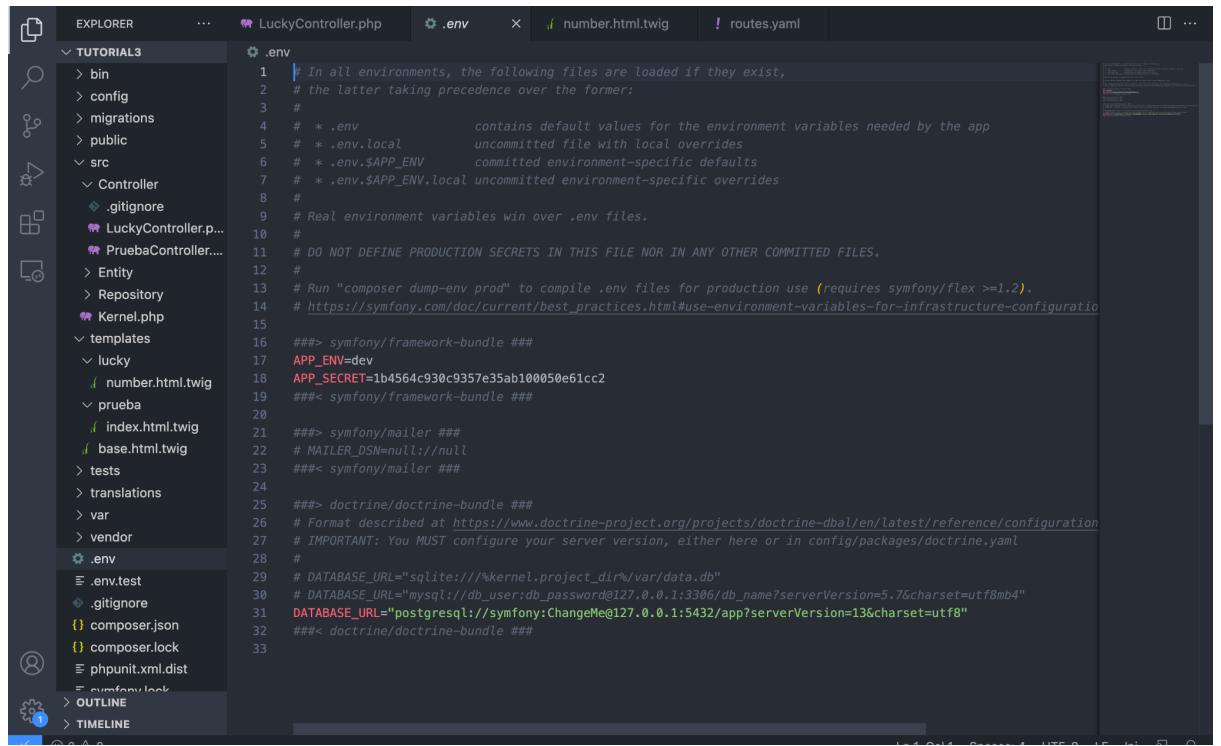
Executing script assets:install public [OK]
Executing script cache:clear [OK]
Executing script assets:install public [OK]
```

Ya tenemos instalado el doctrine, le he echado un vistazo y es super útil, así que he decidido crear una sección en la cual me dedicaré a explicar lo que hace (ya que para la web que construiremos más tarde es muy MUY útil).

5.- La Entidad Doctrine

Una de las facilidades que Doctrine pone a nuestra disposición es manipular la base de datos de una forma más simple.

Antes que nada, vamos a ver que tenemos en el fichero `.env` y a partir de ahí configuraremos la conexión a nuestra base de datos.



```
In all environments, the following files are loaded if they exist,
the latter taking precedence over the former:
#
# * .env contains default values for the environment variables needed by the app
# * .env.local uncommitted file with local overrides
# * .env.$APP_ENV committed environment-specific defaults
# * .env.$APP_ENV.local uncommitted environment-specific overrides
#
# Real environment variables win over .env files.
#
# DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
#
# Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex >=1.2).
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration
##> symfony/framework-bundle ##
APP_ENV=dev
APP_SECRET=1b4564c930c9357e35ab100050e61cc2
##< symfony/framework-bundle ##

##> symfony/mailer ##
MAILER_DSN=null://null
##< symfony/mailer ##

##> doctrine/doctrine-bundle ##
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///kernel.project_dir%var/data.db"
# DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7&charset=utf8mb4"
DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
##< doctrine/doctrine-bundle ##
```

Abajo del todo donde pone URL es donde linkeamos la base de datos. Seguiremos usando la entidad doctrine más adelante.

De momento vamos a seguir con el tutorial y vamos a aprender a crear una entidad, migrarla a la base de datos y posteriormente trabajar con ella.

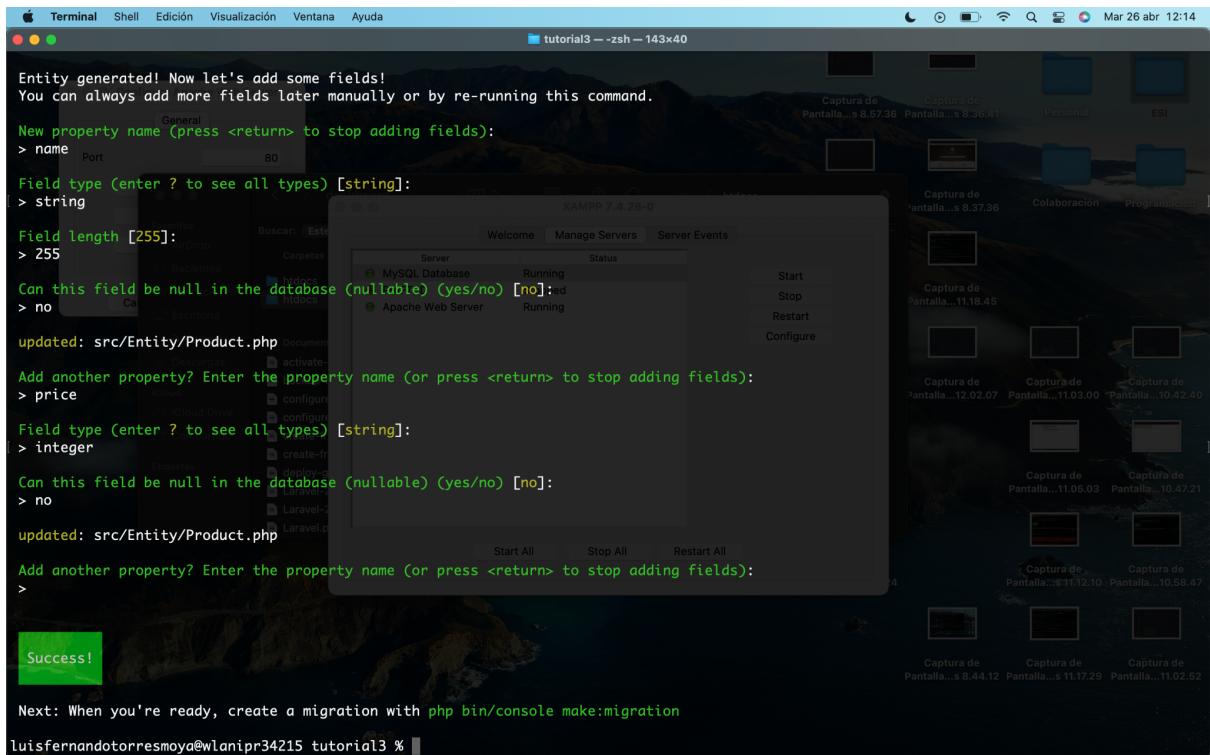
6.- Entidades y Migración a la BBDD

Este apartado será el anterior a la creación de la página web que hicimos con Laravel pero ahora con Symfony. Vamos a crear una entidad Product para probar las entidades.

Una vez hecho esto, migraremos lo que hemos creado a la base de datos, la cual anteriormente conectaremos y configuraremos.

Para crear la entidad Product, introducimos el comando siguiente en una terminal dentro del directorio de trabajo de Symfony:

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



```
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):
> name
  Port: 80
  General

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/Product.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> price
  Field type (enter ? to see all types) [string]:
> integer
  Can this field be null in the database (nullable) (yes/no) [no]: no
  updated: src/Entity/Product.php

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

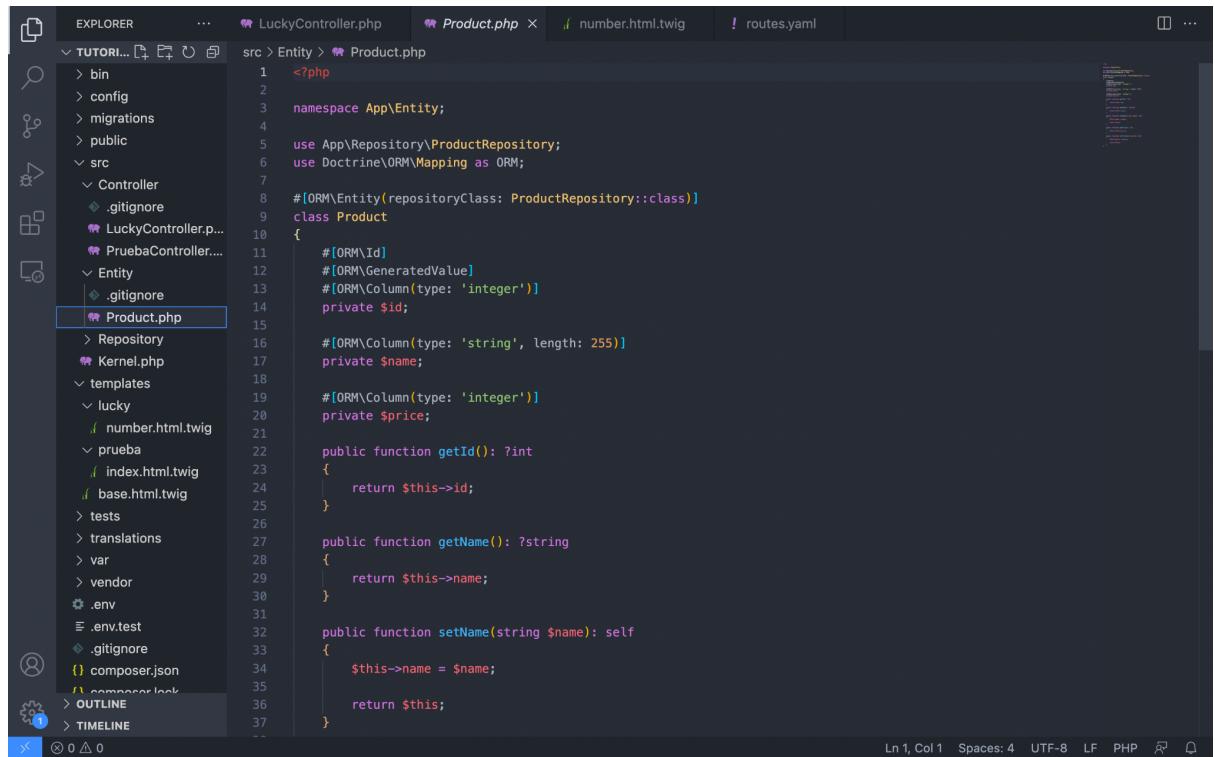
Success!
```

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

luisfernandotorresmoya@Wlanipr34215 tutorial3 %

Como se puede observar, es una herramienta de lo más útil. De esta forma no tenemos que estar yendo todo el rato a PHPMYADMIN para añadir nuevas entidades, sino que se puede hacer de forma mucho más simple de esta forma. Incluso nos avisa que tenemos que migrar para que se guarde.

Ahora se nos ha creado un archivo en la carpeta de entidades con el nombre que hemos elegido. Como ha pasado antes, se nos ha rellenado de forma automática:

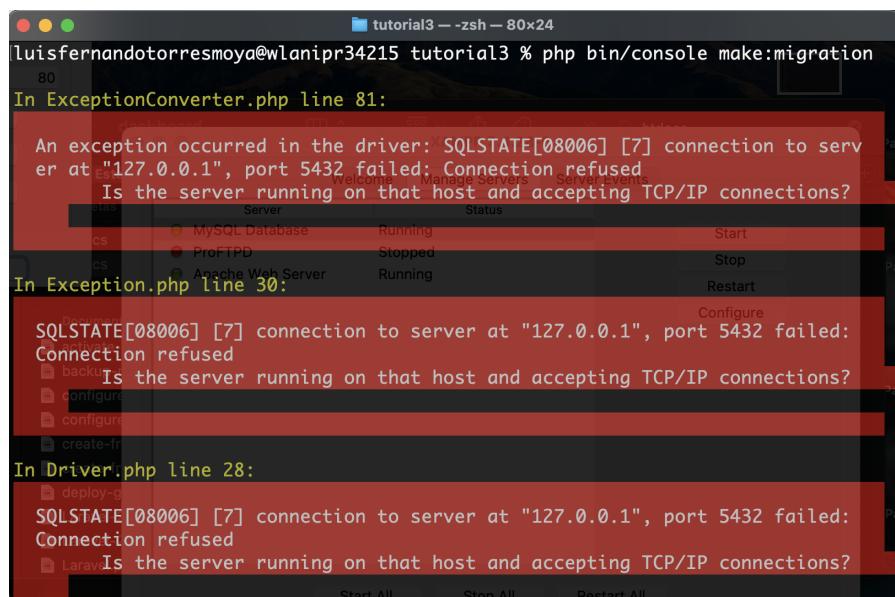


The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure. The `src` directory contains `Entity`, `Controller`, `Kernel.php`, `templates`, `lucky`, `prueba`, `tests`, `translations`, `var`, `vendor`, and configuration files like `.env`, `.env.test`, `.gitignore`, `composer.json`, and `composer.lock`.
- Editor View:** The `Product.php` file is open. It is a PHP class named `Product` with annotations for Doctrine ORM. The code includes methods for `getId`, `getName`, and `setName`.
- Status Bar:** Shows "Ln 1, Col 1" and "Spaces: 4" and "UTF-8 LF PHP".

Ahora nos falta hacer la migración a la Base de Datos, esto se puede hacer también con doctrine con el siguiente comando:

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



The terminal window shows the following output:

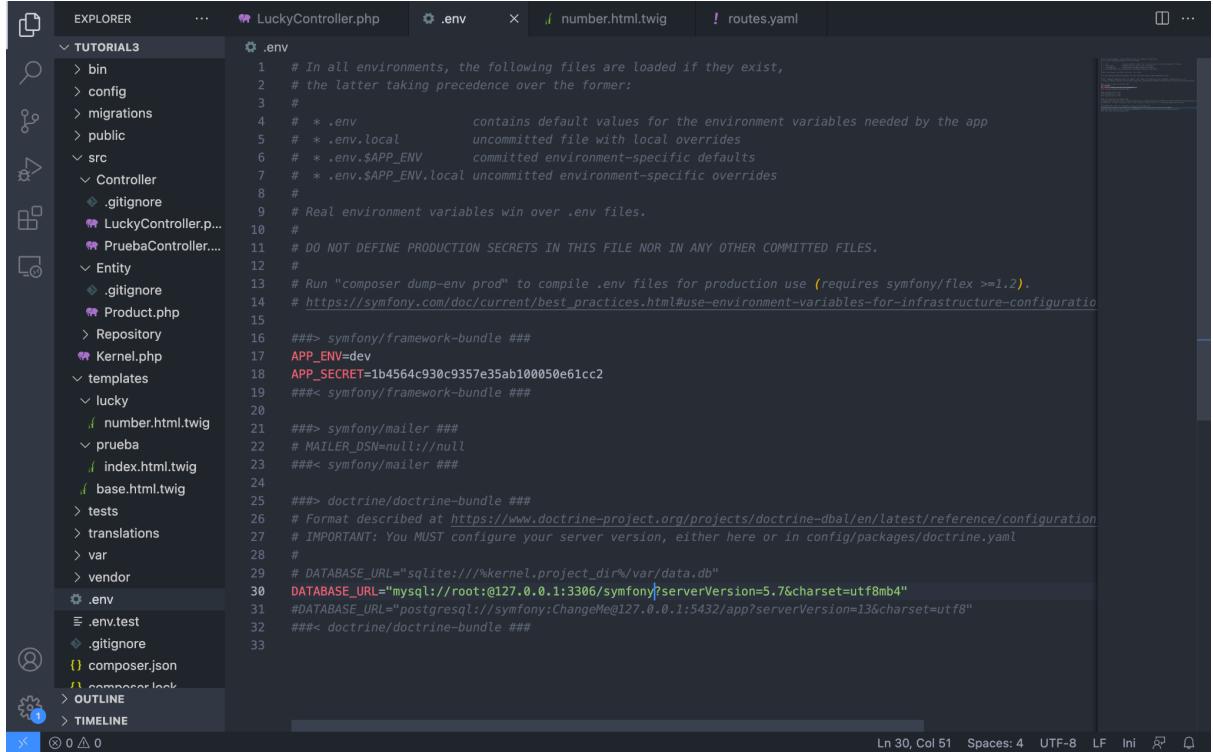
```
[luisfernandotorresmoya@wlanipr34215 tutorial3 % php bin/console make:migration ]
```

An exception occurred in the driver: SQLSTATE[08006] [7] connection to server at "127.0.0.1", port 5432 failed: Connection refused
Is the server running on that host and accepting TCP/IP connections?

SQLSTATE[08006] [7] connection to server at "127.0.0.1", port 5432 failed:
Connection refused
Is the server running on that host and accepting TCP/IP connections?

SQLSTATE[08006] [7] connection to server at "127.0.0.1", port 5432 failed:
Connection refused
Is the server running on that host and accepting TCP/IP connections?

Nos sale este error. Vamos a analizarlo. Si lo vemos de forma más detenida podemos ver que directamente no está encontrando la base de datos. Así que nos vamos otra vez al fichero `.env` y hacemos la siguiente modificación en el código especificando la base de datos a la que queremos que se conecte.

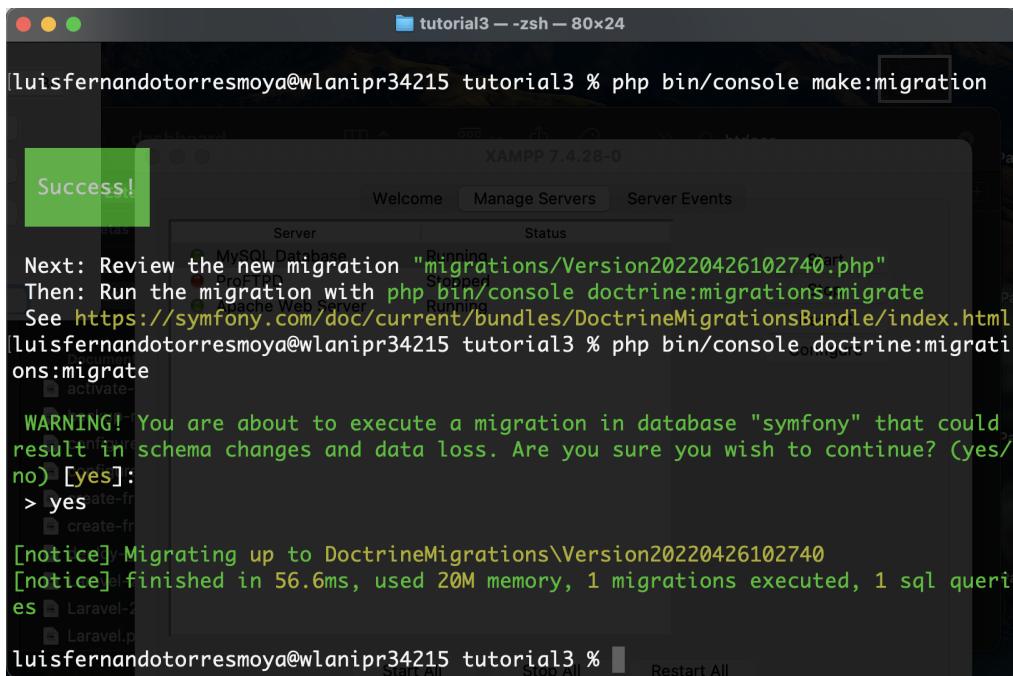


```

1 # In all environments, the following files are loaded if they exist,
2 # the latter taking precedence over the former:
3 #
4 # * .env
5 #   contains default values for the environment variables needed by the app
6 # * .env.local
7 #   uncommitted file with local overrides
8 # * .env.$APP_ENV
9 #   committed environment-specific defaults
10 #
11 # Real environment variables win over .env files.
12 #
13 # DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
14 #
15 ##> symfony/framework-bundle ##
16 APP_ENV=dev
17 APP_SECRET=1b4564c930c9357e35ab100050e61cc2
18 ##< symfony/framework-bundle ##
19
20 ##> symfony/mailer ##
21 # MAILER_DSN=null://null
22 ##< symfony/mailer ##
23
24 ##> doctrine/doctrine-bundle ##
25 # Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration
26 # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
27 #
28 #
29 # DATABASE_URL="sqlite:///kernel.project_dir/var/data.db"
30 DATABASE_URL="mysql://root:@127.0.0.1:3306/symfony?serverVersion=5.7&charset=utf8mb4"
31 #DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
32 ##< doctrine/doctrine-bundle ##
33

```

Y ahora sí podemos hacer las migraciones que son necesarias para que la entidad se vea reflejada en la BBDD:



```

luisfernandotorresmoya@wlanipr34215 tutorial3 % php bin/console make:migration
Success!
Next: Review the new migration "migrations/Version20220426102740.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
luisfernandotorresmoya@wlanipr34215 tutorial3 % php bin/console doctrine:migrations:migrate
WARNING! You are about to execute a migration in database "symfony" that could
result in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
> yes
[notice] Migrating up to DoctrineMigrations\Version20220426102740
[notice] finished in 56.6ms, used 20M memory, 1 migrations executed, 1 sql queries
luisfernandotorresmoya@wlanipr34215 tutorial3 %

```

Y como se puede ver si entramos al servidor MySQL de PHPMyAdmin, ya tenemos la entidad en la base de datos para trabajar con el:

The screenshot shows the phpMyAdmin interface for the 'symfony' database. The left sidebar lists databases: 'Nueva', 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', 'symfony', 'test', and 'doctrine_migration_versions'. The 'symfony' database is selected. The main area shows the 'product' table with 1 row. The table structure includes columns: 'id' (Type: INT(11), Value: 1), 'name' (Type: VARCHAR(255), Value: 'Keyboard'), and 'price' (Type: DECIMAL(10,2), Value: 1999.00). Below the table, there is a 'Crear tabla' (Create Table) form with 'Nombre:' set to 'Nuevo' and 'Número de columnas:' set to 4.

Listo, pero nos queda todavía una cosa, vamos a ver cómo guardamos cosas en la BBDD desde el código. Es decir, vamos a hablar de la Persistencia en la BBDD.

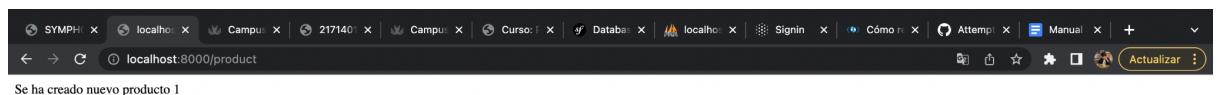
Para empezar, necesitamos un controlador para la entidad que acabamos de crear, ósea ProductController:

```

LuckyController.php | .env | ProductController.php | Product.php | number.html.twig | routes.yaml | ...
src > Controller > ProductController.php
1  <?php
2      namespace App\Controller;
3      use App\Entity\Product;
4      use Doctrine\Persistence\ManagerRegistry as PersistenceManagerRegistry;
5      use Doctrine\ORM\EntityManagerInterface;
6      use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
7      use Symfony\Component\HttpFoundation\Response;
8      use Symfony\Component\Routing\Annotation\Route;
9
10     /**
11      * @Route("/product", name="create_product")
12     */
13     public function createProduct(PersistenceManagerRegistry $doctrine): Response{
14         $entityManager = $doctrine->getManager();
15
16         $product = new Product();
17         $product->setName('Keyboard');
18         $product->setPrice(1999);
19
20         $entityManager->persist($product);
21
22         $entityManager->flush();
23
24         return new Response('Se ha creado nuevo producto ' . $product->getId());
25     }
26
27 }
28
29 ?>
30

```

Ahora si accedemos a localhost:8000/product podemos ver como se ha añadido el producto keyboard con un precio de 1999.

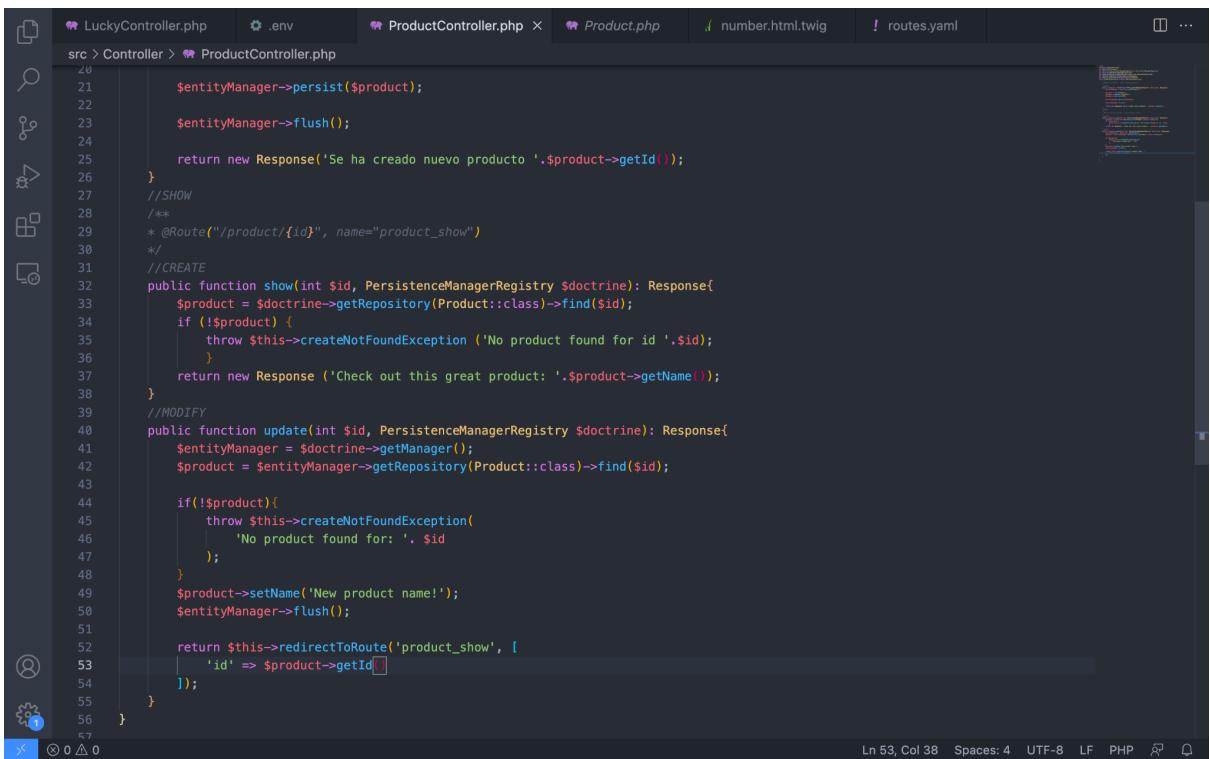


A screenshot of the phpMyAdmin interface. The left sidebar shows databases like 'Nueva', 'information_schema', 'mysql', etc., and the 'product' database is selected. The main panel shows the 'product' table with one row: id 1, name 'Keyboard', price 1999. There are buttons for Editar, Copiar, Borrar, and Exportar.

Ya sabemos como guardar cosas en la BBDD. Solo nos faltan un par de pasos, el obtener, modificar y borrar datos de la BBDD. Pues veamos cómo se hace.

Lo que tenemos que hacer es lo de siempre, vamos a crear una función de enseñar, modificar y borrar en el controlador de Product.

Veamos en una sola captura de pantalla todo el código que hace falta para tener listas estas funciones.

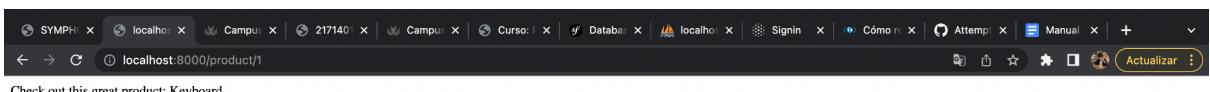


```
LuckyController.php .env ProductController.php Product.php number.html.twig routes.yaml

src > Controller > ProductController.php
26     $entityManager->persist($product);
27
28     $entityManager->flush();
29
30     return new Response('Se ha creado nuevo producto '.$product->getId());
31 }
32 //SHOW
33 /**
34 * @Route("/product/{id}", name="product_show")
35 */
36 //CREATE
37 public function show(int $id, PersistenceManagerRegistry $doctrine): Response{
38     $product = $doctrine->getRepository(Product::class)->find($id);
39     if (!$product) {
40         throw $this->createNotFoundException ('No product found for id '.$id);
41     }
42     return new Response ('Check out this great product: '.$product->getName());
43 }
44 //MODIFY
45 public function update(int $id, PersistenceManagerRegistry $doctrine): Response{
46     $entityManager = $doctrine->getManager();
47     $product = $entityManager->getRepository(Product::class)->find($id);
48
49     if(!$product){
50         throw $this->createNotFoundException(
51             'No product found for: '. $id
52         );
53     }
54     $product->setName('New product name!');
55     $entityManager->flush();
56
57     return $this->redirectToRoute('product_show', [
58         'id' => $product->getId()
59     ]);
56 }
57 }

Ln 53, Col 38 Spaces: 4 UTF-8 LF PHP ⌂ ⌂
```

Si lo probamos, vemos que todo funciona perfectamente.

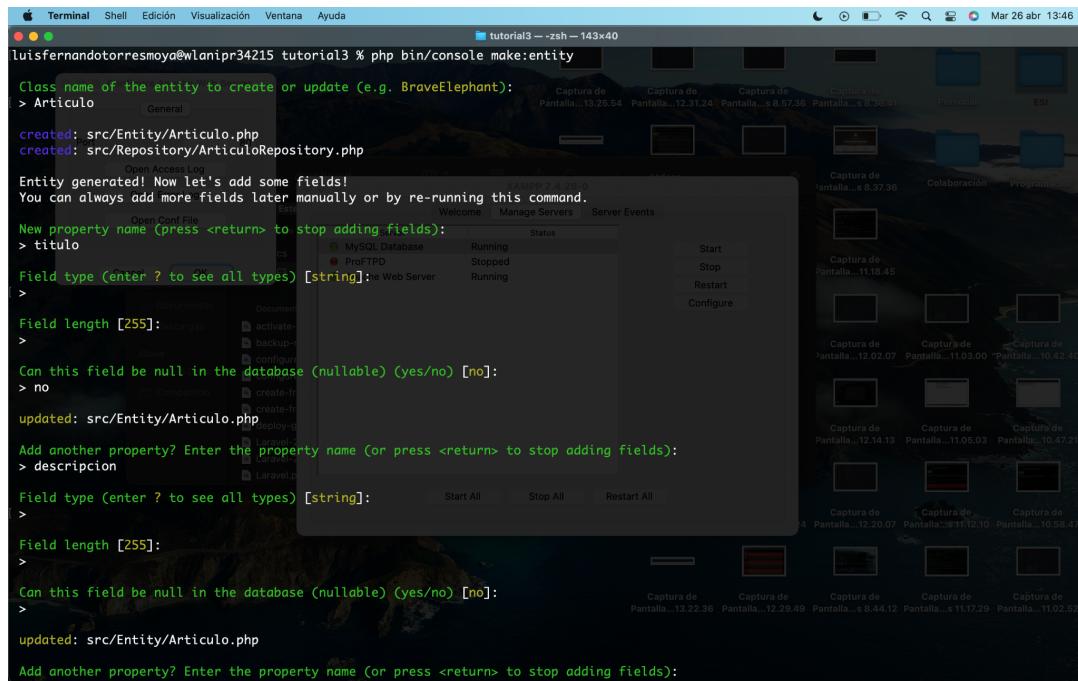


Así que ya lo tenemos todo listo para crear nuestra página web de artículos.

7.- Página Web Artículos

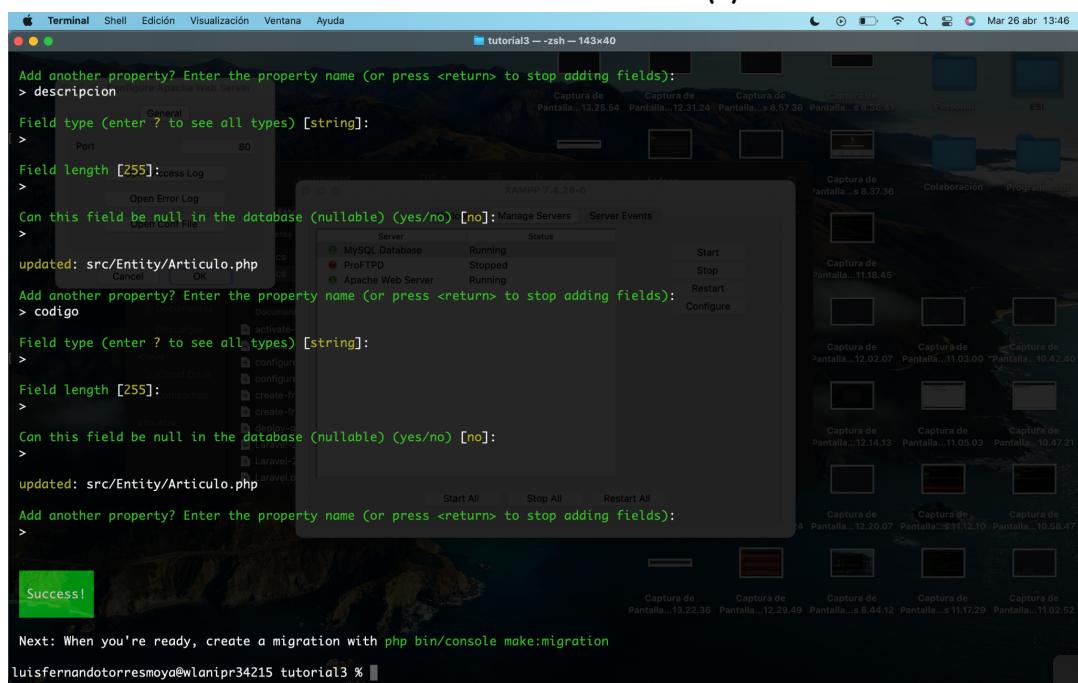
7 . 1.- Creación entidad Artículos

-Salida de la creación de la entidad Articulos (1)



```
luisfernandotorresmoya@wlanipr34215 tutorial3 % php bin/console make:entity
Class name of the entity to create or update (e.g. BraveElephant):
> Artículo
General
created: src/Entity/Articulo.php
created: src/Repository/ArticuloRepository.php
Open Access Log
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):
> título
Field type (enter ? to see all types) [string]: The Web Server
>
Field length [255]:
>
Can this field be null in the database (nullable) (yes/no) [no]:
> no
updated: src/Entity/Articulo.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> descripción
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/Articulo.php
Add another property? Enter the property name (or press <return> to stop adding fields):
>
```

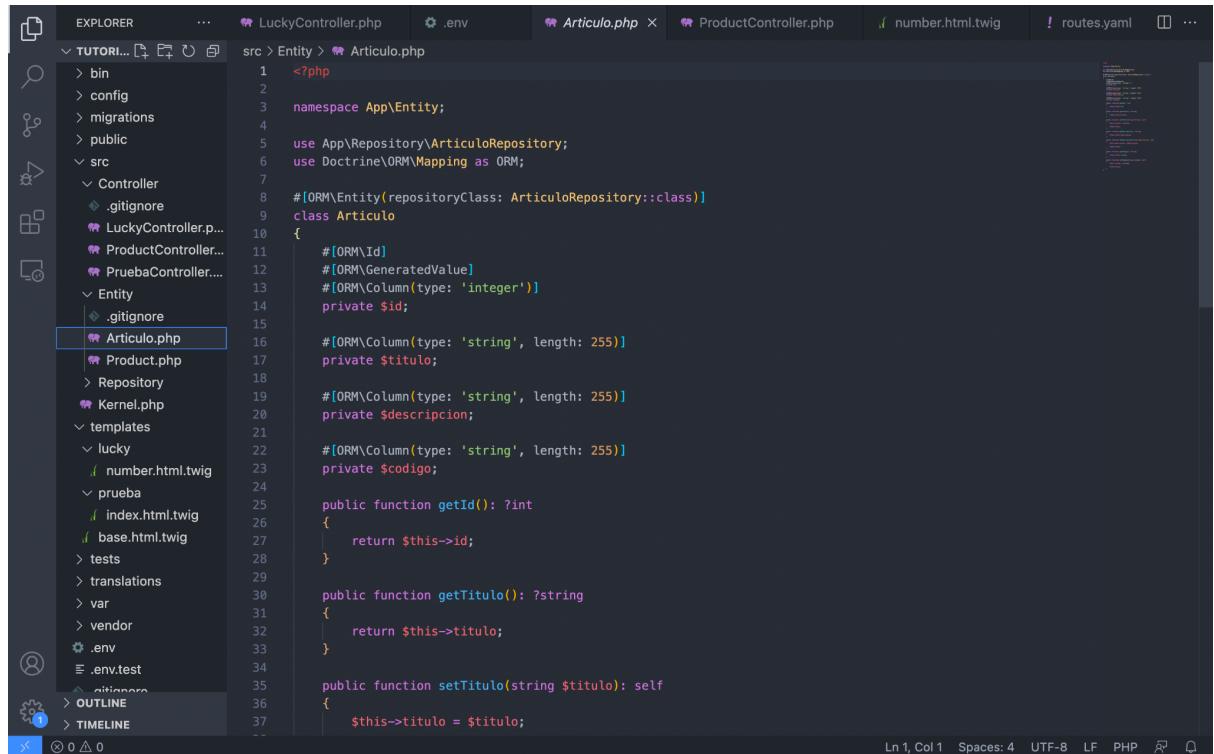
-Salida de la creación de la entidad Articulos (2)



```
Add another property? Enter the property name (or press <return> to stop adding fields):
> descripción
General
Field type (enter ? to see all types) [string]:
> Port
80
Field length [255]: access Log
>
Open Error Log
Can this field be null in the database (nullable) (yes/no) [no]:
>
updated: src/Entity/Articulo.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> codigo
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/Articulo.php
Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!
Next: When you're ready, create a migration with php bin/console make:migration
luisfernandotorresmoya@wlanipr34215 tutorial3 %
```

-Archivo Articulo.php



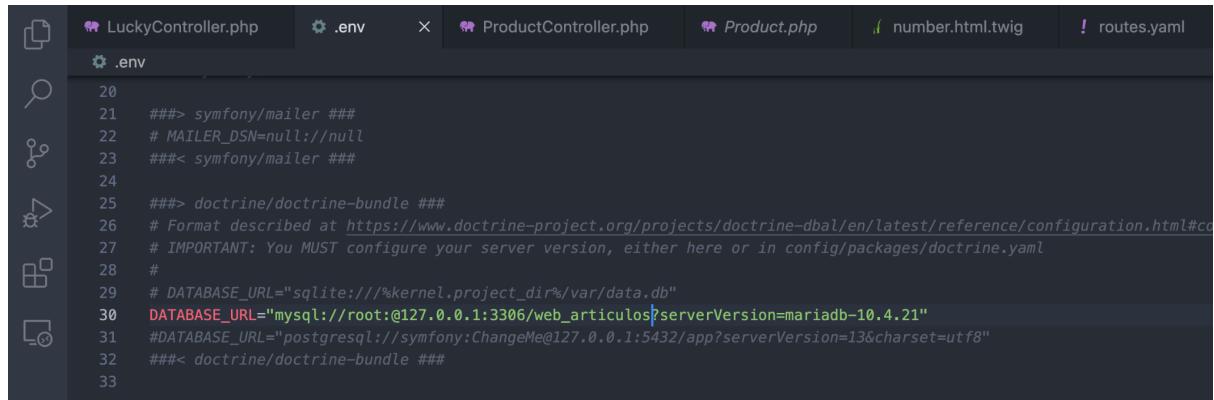
The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure with the following files and folders:
 - TUTORIAL...
 - src > Entity > Articulo.php
 - src > Controller > .gitignore
 - src > Controller > LuckyController.php
 - src > Controller > ProductController.php
 - src > Controller > PruebaController.php
 - src > Entity > .gitignore
 - src > Entity > Articulo.php
 - src > Entity > Product.php
 - src > Repository
 - src > Kernel.php
 - src > templates > lucky > number.html.twig
 - src > templates > prueba > index.html.twig
 - src > templates > base.html.twig
 - src > tests
 - src > translations
 - src > var
 - src > vendor
 - .env
 - .env.test
- Articulo.php Content:**

```
<?php  
namespace App\Entity;  
  
use App\Repository\ArticuloRepository;  
use Doctrine\ORM\Mapping as ORM;  
  
#[ORM\Entity(repositoryClass: ArticuloRepository::class)]  
class Articulo  
{  
    #[ORM\Id]  
    #[ORM\GeneratedValue]  
    #[ORM\Column(type: 'integer')]  
    private $id;  
  
    #[ORM\Column(type: 'string', length: 255)]  
    private $titulo;  
  
    #[ORM\Column(type: 'string', length: 255)]  
    private $descripcion;  
  
    #[ORM\Column(type: 'string', length: 255)]  
    private $codigo;  
  
    public function getId(): ?int  
    {  
        return $this->id;  
    }  
  
    public function getTitulo(): ?string  
    {  
        return $this->titulo;  
    }  
  
    public function setTitulo(string $titulo): self  
    {  
        $this->titulo = $titulo;  
        return $this;  
    }  
}
```
- Status Bar:** Shows "Ln 1, Col 1" and "Spaces: 4" and "UTF-8 LF PHP".

7 . 2.- Migración a la BBDD

-Archivo .env con la nueva configuración de la base de datos



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure with the following files and folders:
 - LuckyController.php
 - .env
 - ProductController.php
 - Product.php
 - number.html.twig
 - routes.yaml
- .env Content:**

```
20  
21    ##> symfony/mailer ##  
22    # MAILER_DSN=null://null  
23    ##< symfony/mailer ##  
24  
25    ##> doctrine/doctrine-bundle ##  
26    # Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connection  
27    # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml  
28    #  
29    # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"  
30    DATABASE_URL="mysql://root:@127.0.0.1:3306/web_articulos|serverVersion=mariadb-10.4.21"  
31    #DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"  
32    ##< doctrine/doctrine-bundle ##  
33
```

-Migración realizada en la consola

```
luisfernandotorresmoya@wlanipr34215 tutorial3 % php bin/console make:migration
[WARNING] You have 1 available migrations to execute.

Are you sure you wish to continue? [Yes/no] [yes]:
>

Success!
Start All Stop All Restart All

Next: Review the new migration "migrations/Version20220426115405.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

-PHPMYAdmin con la tabla artículos ya creada

The screenshot shows the phpMyAdmin interface. On the left, there's a tree view of databases: SYMPHO, localhost, Campus, 2171401, Campus, Curso: F, Databases, and a local host entry. The 'web_articulos' database is selected. Inside it, the 'articulo' table is shown. The table has four columns: 'id', 'titulo', 'descripcion', and 'codigo'. A query 'SELECT * FROM `articulo`' is run, and the results show an empty set of values. There are also sections for creating a view and saving the query as a favorite.

7 . 3.- Creación del Controlador

Aquí voy a explicar un poco lo que voy a hacer, ya que puede ser un poco lioso. Voy a crear el controlador base ArticuloController y dentro del mismo voy a añadir todas las funcionalidades básicas de una web, es decir el CRUD.

Dentro del controlador voy a añadir funciones de Crear, Visualizar, Modificar y Borrar. Voy a enseñar el código completo de este archivo para que se pueda ver lo que he hecho. En el último apartado haré las capturas del CRUD totalmente funcional.

También incluimos en este fichero que vamos a enseñar a continuación una función index que enseñara todos los artículos que tengamos al momento. Creemos unos cuantos en PHPMyAdmin.

-PHPMyAdmin con Articulos de prueba añadidos

The screenshot shows the PHPMyAdmin interface with the following details:

- Database:** web_articulos
- Table:** articulo
- SQL Query:** SELECT * FROM `articulo`
- Results:** 3 rows displayed
- Table Headers:** id, titulo, descripcion, codigo
- Table Data:**

	id	titulo	descripcion	codigo
<input type="checkbox"/>	1	El roldan lo deja con la parienta	Diosotnio lo han dejao	0001
<input type="checkbox"/>	2	El roldan VUELVE con la parienta	Ya creo en el amor	0002
<input type="checkbox"/>	3	Codeigniter deja de existir	Que alivio picha	0003
- Operations:** Imprimir, Copiar al portapapeles, Exportar, Mostrar gráfico, Crear vista
- Buttons:** Guardar esta consulta en favoritos, Consola

Veamos entonces el código que vamos a tener dentro del controlador que vamos a utilizar para artículos:

-Fichero ArticuloController.php (1)

```
<?php
namespace App\Controller;
use App\Entity\Articulo;
use Doctrine\Persistence\ManagerRegistry as PersistenceManagerRegistry;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
class ArticuloController extends AbstractController{



    /**
     * @Route("/articulo/index_articulos")
     */
    public function index_articulos(PersistenceManagerRegistry $doctrine): Response{
        $articulos = $doctrine->getRepository(Articulo::class)->findAll();
        return $this->render('articulo/index.html.twig', [
            'articulos' => $articulos
        ]);
    }

    /**
     * @Route("/articulo/{nombre}", name="crea_articulo")
     */
    //CREATE

    public function crearArticulo(string $nombre, PersistenceManagerRegistry $doctrine): Response{
        $entityManager = $doctrine->getManager();

        $articulo = new articulo();
        $articulo->setTitulo($titulo);
        $articulo->setPrice(1999);
    }
}
```

Ln 78, Col 5 Spaces: 4 UTF-8 LF PHP

-Fichero ArticuloController.php(2)

```
//CREATE

public function enseñaArticulo(int $id, PersistenceManagerRegistry $doctrine): Response{
    $articulo = $doctrine->getRepository(Articulo::class)->find($id);
    if (!$articulo) {
        throw $this->createNotFoundException ('No articulo found for id '.$id);
    }
    return new Response ('Check out this great articulo: '.$articulo->getName());
}

//MODIFY
//SHOW
/**
 * @Route("/articulo/{id}", name="cambia_articulo")
 */
public function cambia_articulo(int $id, PersistenceManagerRegistry $doctrine): Response{
    $entityManager = $doctrine->getManager();
    $articulo = $entityManager->getRepository(Articulo::class)->find($id);

    if(!$articulo){
        throw $this->createNotFoundException(
            'No articulo found for: '. $id
        );
    }
    $articulo->setName('New articulo name!');
    $entityManager->flush();

    return $this->redirectToRoute('articulo_show', [
        'id' => $articulo->getId()
    ]);
}

```

Ln 78, Col 5 Spaces: 4 UTF-8

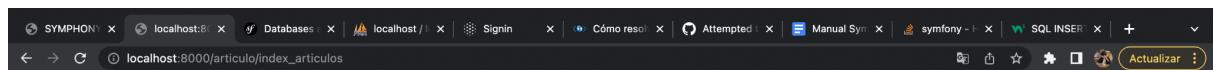
Y ahora vamos a enseñar en la web la página localhost:8000/articulo/index_articulos la cual listará todos los artículos que encuentre en la base de datos y los enseñara por pantalla, pero antes vamos a enseñar el .twig que enseña esto.

-Fichero index.html.twig



```
templates > articulo > index.html.twig
1  {% for articulo in articulos %}
2    <h1>{{ articulo.titulo }}</h1>
3    <h2>{{ articulo.descripcion }}</h2>
4    <h3>{{ articulo.codigo }}</h3>
5    <br><br>
6  {% endfor %}
```

-Resultado de ir a localhost:8000/articulo/index_articulo



El roldan lo deja con la parienta

Diositomio lo han dejao

00001

El roldan VUELVE con la parienta

Ya creo en el amor

00002

CodeIgniter deja de existir

Que alivio picha

00003

Esto claramente puede ser mucho más bonito, pero bueno eso ya sería comerse la cabeza con HTML y CSS así que por funcionalidad lo vamos a dejar así el index.

7 . 4.- Forms en Symfony

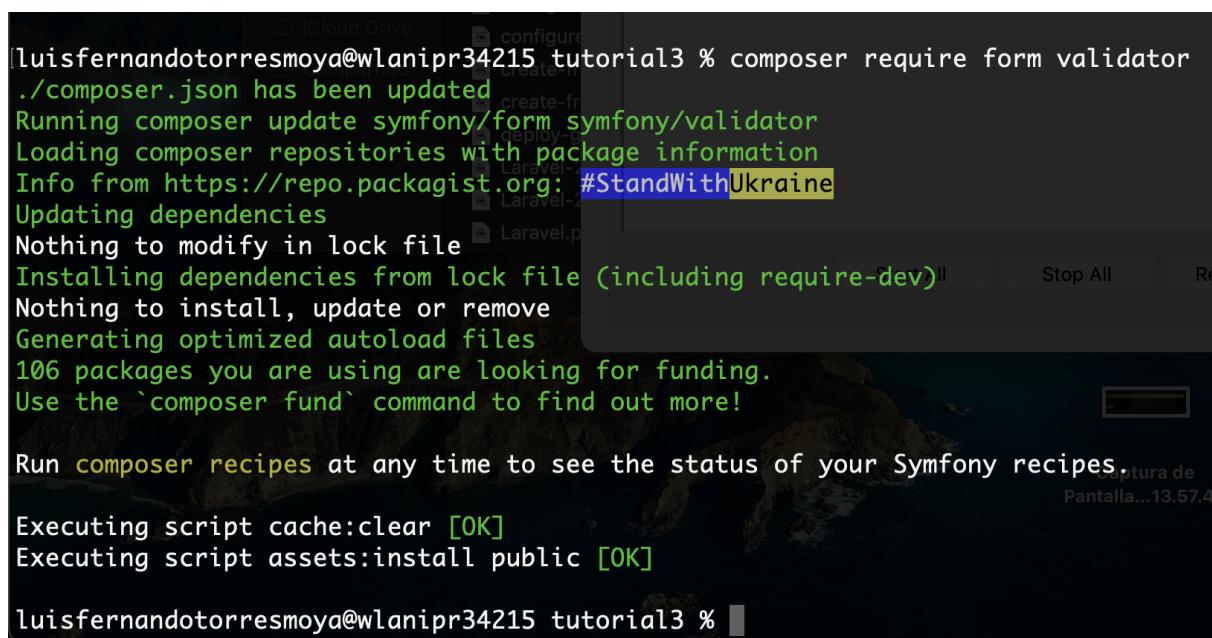
Este apartado técnicamente no es necesario, pues ya está todo hecho, pero creo que si aprendemos a hacer formularios y enviarlos para poder así añadir de un tirón un artículo en nuestra web, podemos así, aumentar la funcionalidad de nuestra página web.

Como esta parte no se ha explicado en el manual que hemos ido recorriendo, la voy a documentar desde cero, es decir, voy a escribir todo lo que vaya averiguando y las capturas de pantallas de lo mismo. A su vez, iremos documentando los errores que vayan surgiendo.

7 . 4 . 1.- Descargar los formularios con Composer

Igual que se hizo con anterioridad, vamos a usar el composer para quitarnos mucho del trabajo de programación. Al fin y al cabo, esto es parte de los Framework, la automatización del mucho trabajo común para muchos proyectos. Dicho esto, vamos a usar el siguiente comando para instalar los formularios:

```
composer require form validator
```



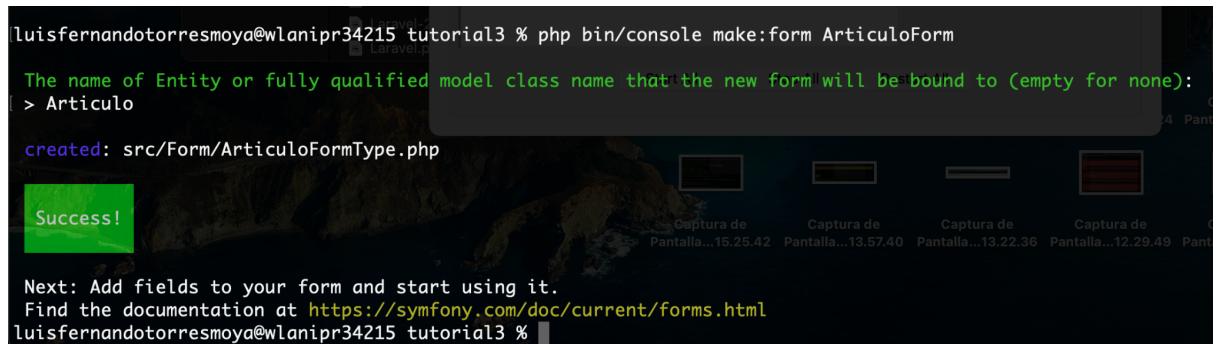
```
[luisfernandotorresmoya@wlanipr34215 tutorial3 % composer require form validator
./composer.json has been updated
Running composer update symfony/form symfony/validator
Loading composer repositories with package information
Info from https://repo.packagist.org: #StandWithUkraine
Updating dependencies
Nothing to modify in lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
106 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Run composer recipes at any time to see the status of your Symfony recipes.

Executing script cache:clear [OK]
Executing script assets:install public [OK]
luisfernandotorresmoya@wlanipr34215 tutorial3 %]
```

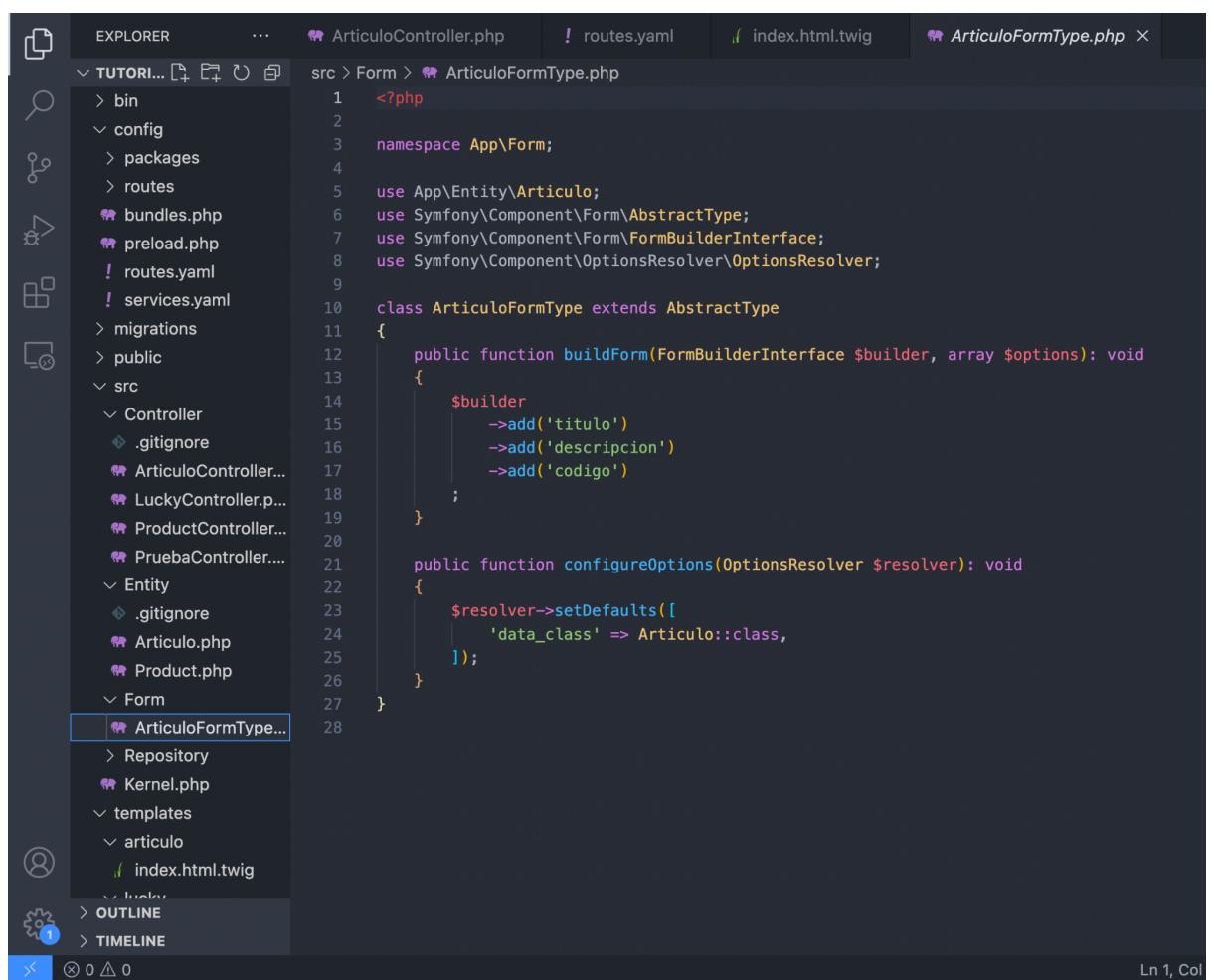
Ahora creamos el formulario deseado al igual que se creaban los controladores y las migraciones que se hicieron antes:

```
php bin/console make:form ArticuloForm
```



```
luisfernandotorresmoya@wlanipr34215 tutorial3 % php bin/console make:form ArticuloForm
The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> Articulo
created: src/Form/ArticuloFormType.php
Success!
Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html
luisfernandotorresmoya@wlanipr34215 tutorial3 %
```

Como podemos ver, se nos crea el formulario de forma automática, y además un documento en `src/Form/ArticuloFormType.php`. Veamos que tiene dentro:



```
<?php
namespace App\Form;
use App\Entity\Articulo;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
class ArticuloFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('titulo')
            ->add('descripcion')
            ->add('codigo')
    }

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

Hacemos una función en el controlador de Artículos para poder enseñar el formulario por pantalla. Así mismo se le da una ruta:

-Modificación en ArticuloController.php (para el form)

```
/**
 * @Route("/articulo/index_form")
 */

public function index_form(Environment $twig){

    $articulo = new Articulo();

    $form = $this->createForm(ArticuloFormType::class, $articulo);

    return new Response($twig->render('articulo/form.html.twig', [
        'articulo_form' => $form->createView()
    ]));
}
```

-Fichero form.html.twig

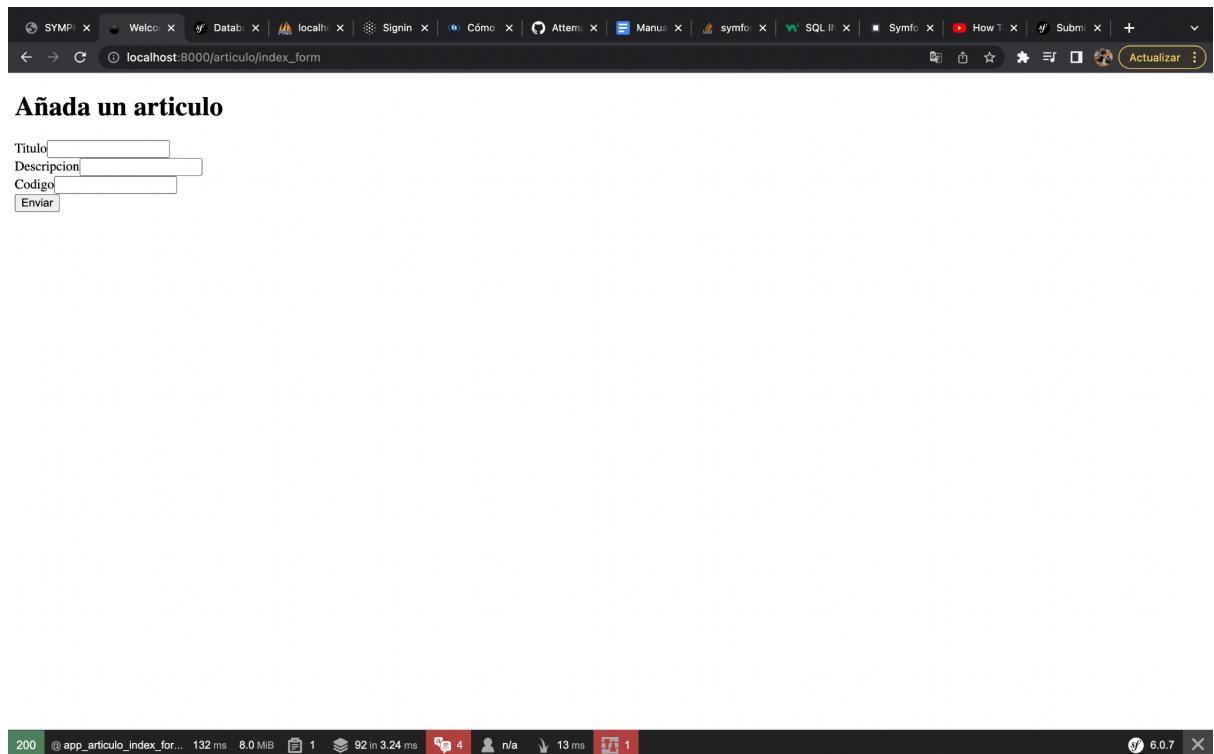


```
templates > articulo > form.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4
5      <h1>Añada un articulo</h1>
6
7      {{ form[articulo_form] }}
8
9
10     {% endblock %}
```

Ahora si entramos en la ruta que hemos definido deberíamos de tener nuestro formulario listo para recibir artículos nuevos.

Veámoslo en localhost:8000/articulo/index_form

-Visualización del formulario en la web



Añada un articulo

Titulo

Descripcion

Código

200 @app_articulo_index_for... 132 ms 8.0 MiB 1 92 in 3.24 ms 4 n/a 13 ms 1 6.0.7 X

Ya tenemos la vista del formulario creado, así que lo dicho, vamos ahora a guardar los artículos en la base de datos. Para esto modificamos la función que hemos creado anteriormente y le añadimos el siguiente código(Usando la funcionalidad request).

-Función final del formulario

```
public function index_form(Environment $twig, Request $request, EntityManagerInterface $entityManager){

    $articulo = new Articulo();

    $form = $this->createForm(ArticuloFormType::class, $articulo);

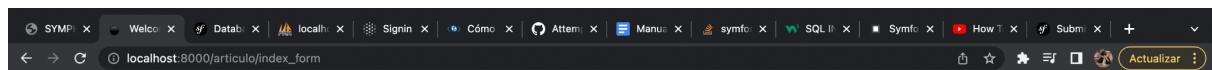
    $form->handleRequest($request);

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

        return new Response('Articulo numero: ' . $articulo->getId() . ' creado.');
    }

    return new Response($twig->render('articulo/form.html.twig', [
        'articulo_form' => $form->createView()
    ]));
}
```

Añadamos ahora un artículo y veamos si se guarda bien usando la visualización que hemos creado anteriormente.

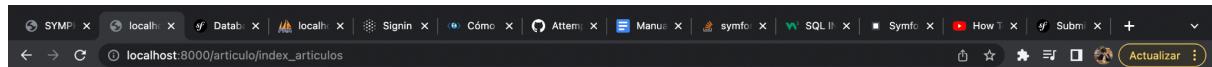


Añada un articulo

Título: DLH gana mejor profes
Descripción: que bien!!
Código: 00005



Y podemos ver que se ha creado el artículo de forma satisfactoria.



El roldan lo deja con la parienta

Diositomio lo han dejao
00001

El roldan VUELVE con la parienta

Ya creo en el amor
00002

CodeIgniter deja de existir

Que alivio picha
00003

DLH gana mejor profesor del año

que bien!!
00005

8.- Introspección del Manual

Al igual que en los manuales anteriores comenté los problemas que daban, este lo he disfrutado en cantidad. He de decir que este formato de que se den las cosas básicas y luego ya tu vas buscando lo demás es mucho más interactivo e incentiva el buscar las cosas en vez de estar todo el rato copiando código.

Vuelvo a explicar más o menos que del punto 1 al 6 he explicado en profundidad todo lo que he hecho, ya que he ido siguiendo el manual. Luego en el apartado 7, las primeras partes solo he enseñado código. Luego en el último apartado de este último, como era una cosa nueva la he explicado mejor. Creo que ha quedado bastante bien.