

PROGRAMACIÓN WEB

---

**Manual SYMFONY**

---



**Autor:**

Alejandro Guerrero Medina

**Grado en Ingeniería Informática**

Curso 2022 - 2023

## Índice

<b>1. Prolegómeno</b>	<b>2</b>
<b>2. Instalación de Symfony</b>	<b>2</b>
2.1. Requisitos del sistema . . . . .	2
2.2. Instalación de Symfony CLI . . . . .	2
2.3. Funcionamiento de la aplicación . . . . .	4
<b>3. Creación de la base de datos</b>	<b>6</b>
3.1. Creación de la Entidad Articulos . . . . .	6
3.2. Migración de la Entidad Artículos . . . . .	8
<b>4. Creación Controlador ArtículoController</b>	<b>9</b>
4.1. Vista <code>articulos</code> . . . . .	9
4.2. Funcionalidad para mostrar articulos en <code>articulos</code> . . . . .	11
4.3. Formulario para añadir artículos . . . . .	14
<b>5. Login y Registro de Usuario</b>	<b>18</b>
5.1. Creación de la Entidad Usuario . . . . .	18
5.2. Creación de la vista de registro . . . . .	19
5.3. Creacion de la vista de inicio de sesión . . . . .	20
5.4. Botón de cerrar sesión . . . . .	21
<b>6. Conclusiones</b>	<b>24</b>

## 1. Prolegómeno

Sea este documento una guía sobre el *framework* basado en PHP, SYMFONY, con objeto de especificar cada paso realizado para desplegar una aplicación web en correcto funcionamiento basándose en las funcionalidades realizadas con el *framework* anterior, LARAVEL.

## 2. Instalación de Symfony

### 2.1. Requisitos del sistema

Para el correcto funcionamiento de SYMFONY, se necesita:

- Versión de PHP igual o superior a la 8.1.0
- Extensión de `Ctype`
- Extensión de `iconv`
- Extensión de `PCRE`
- Extensión de `Session`
- Extensión de `SimpleXML`
- Extensión de `Tokenizer`

Además, necesitaremos de una herramienta como XAMPP o APPSERV, así como el manejador de dependencias para PHP, COMPOSER.

### 2.2. Instalación de Symfony CLI

A diferencia de los *frameworks* anteriores, SYMFONY puede instalarse mediante una herramienta propia que tiene el propósito de crear y configurar aplicaciones web de SYMFONY desde la terminal. Para ello, deberemos instalar el binario de `symfony` usando `scoop`. Para instalar `scoop`, introducimos los dos siguientes comandos en la terminal:

```
$ Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
$ irm get.scoop.sh | iex
```

Una vez instalado, procedemos con la instalación de la herramienta introduciendo el siguiente comando en la terminal:

```
$ scoop install symfony-cli
```

```
Windows PowerShell
PS C:\Users\aleja> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
PS C:\Users\aleja> irm get.scoop.sh | iex
Initializing...
Downloading...
Creating shim...
Adding ~\scoop\shims to your path.
Scoop was installed successfully!
Type 'scoop help' for instructions.
PS C:\Users\aleja> scoop install symfony-cli
Installing 'symfony-cli' (5.5.3) [64bit] from main bucket
symfony-cli_windows_amd64.zip (5,2 MB) [=====] 100%
Checking hash of symfony-cli_windows_amd64.zip ... ok.
Extracting symfony-cli_windows_amd64.zip ... done.
Linking ~\scoop\apps\symfony-cli\current => ~\scoop\apps\symfony-cli\5.5.3
Creating shim for 'symfony'.
'symfony-cli' (5.5.3) was installed successfully!
PS C:\Users\aleja>
```

Figura 1: Instalación de Scoop y SymfonyCLI

Podemos comprobar si nuestro sistema cumple los requisitos mínimos y óptimos para poder iniciar un proyecto en SYMFONY con el siguiente comando:

```
$ symfony check:requirements
```

```
PS C:\Users\aleja> symfony check:requirements

Symfony Requirements Checker

> PHP is using the following php.ini file:
C:\xampp\php\php.ini

> Checking Symfony requirements:

*****

[OK]
Your system is ready to run Symfony projects

Note The command console can use a different php.ini file
      than the one used by your web server.
      Please check that both the console and the web server
      are using the same PHP version and configuration.

PS C:\Users\aleja>
```

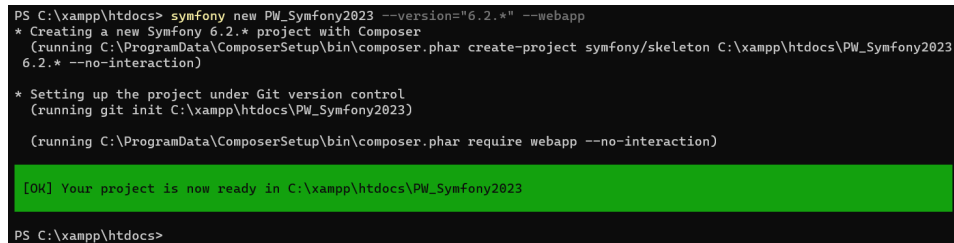
Figura 2: Comprobación de requisitos

- Se ha habilitado la extensión intl.
- Se ha habilitado la extensión zend = opache.
- Se ha asignado más capacidad a la variable post\_max\_size a **60M**, siendo upload\_max\_size la que viene por defecto.
- Se ha asignado a realpath\_cache\_size unos **5M**.

A continuación, crearemos nuestro proyecto de SYMFONY en el directorio deseado –la elección será:

C:\xampp\htdocs–, dirigiéndonos a dicho directorio en la terminal e introduciendo el siguiente comando:

```
$ symfony new PW_Symfony2023 --version="6.2.*" --webapp
```



```
PS C:\xampp\htdocs> symfony new PW_Symfony2023 --version="6.2.*" --webapp
* Creating a new Symfony 6.2.* project with Composer
(running C:\ProgramData\ComposerSetup\bin\composer.phar create-project symfony/skeleton C:\xampp\htdocs\PW_Symfony2023
6.2.* --no-interaction)

* Setting up the project under Git version control
(running git init C:\xampp\htdocs\PW_Symfony2023)

(running C:\ProgramData\ComposerSetup\bin\composer.phar require webapp --no-interaction)

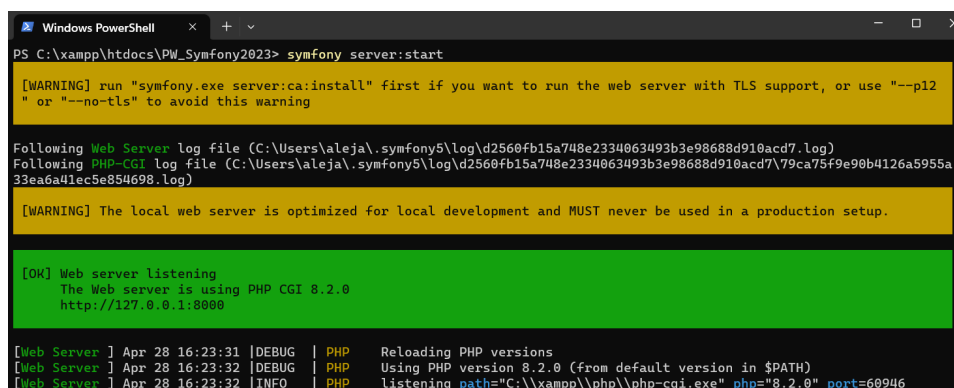
[OK] Your project is now ready in C:\xampp\htdocs\PW_Symfony2023
PS C:\xampp\htdocs>
```

Figura 3: Instalación de SYMFONY

## 2.3. Funcionamiento de la aplicación

El binario `symfony` provee, entre otras cosas, un servidor web local para poder probar la aplicación web a desarrollar. Para ello, en el directorio del proyecto, introducimos el siguiente comando:

```
$ symfony server:start
```



```
Windows PowerShell
PS C:\xampp\htdocs\PW_Symfony2023> symfony server:start

[WARNING] run "symfony.exe server:ca:install" first if you want to run the web server with TLS support, or use "--p12" or "--no-tls" to avoid this warning

Following Web Server log file (C:\Users\aleja\.symfony5\log\d2560fb15a748e2334063493b3e98688d910acd7.log)
Following PHP-CGI log file (C:\Users\aleja\.symfony5\log\d2560fb15a748e2334063493b3e98688d910acd7\79ca75f9e90b4126a5955a33ea6a41ec5e854698.log)

[WARNING] The local web server is optimized for local development and MUST never be used in a production setup.

[OK] Web server listening
The Web server is using PHP CGI 8.2.0
http://127.0.0.1:8000

[Web Server ] Apr 28 16:23:31 | DEBUG | PHP | Reloading PHP versions
[Web Server ] Apr 28 16:23:32 | DEBUG | PHP | Using PHP version 8.2.0 (from default version in $PATH)
[Web Server ] Apr 28 16:23:32 | INFO | PHP | listening path="C:\xampp\php\php-cgi.exe" php="8.2.0" port=60946
```

Figura 4: Arranque de Servidor Local

Con esto, `symfony` iniciará un servidor web en la dirección `http://127.0.0.1:8000`, y si abrimos dicha dirección en nuestro navegador nos aparecerá la siguiente vista:

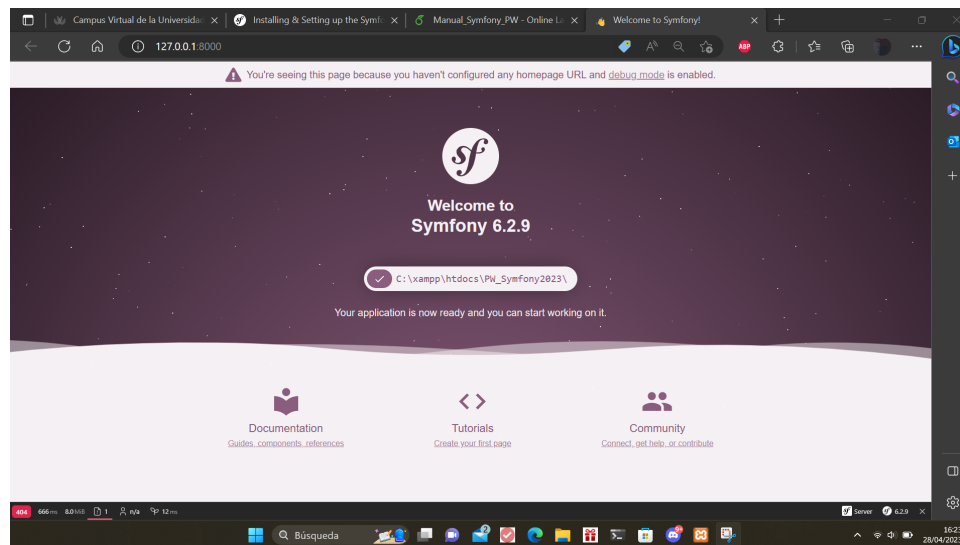


Figura 5: Página principal de SYMFONY

Una vez accedamos a esta página, tendremos SYMFONY perfectamente configurado en nuestro sistema para poder empezar a trabajar con él.

### 3. Creación de la base de datos

Para crear la base de datos, podemos directamente crearla desde la terminal de VISUAL STUDIO o en phpMyAdmin, si nos es más cómodo. Usaremos la primera opción propuesta:

```
PS C:\xampp\htdocs\PW_Symfony2023> mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 62
Server version: 10.4.27-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database pw_symfony2023
-> ;
```

Figura 6: Creación de la base de datos pw\_symfony2023

Ahora, en el fichero `.env` del proyecto, modificamos el valor de la variable `DATABASE_URL` donde debemos indicar el SGBD usado `mysql`, el usuario `root`, la URL y el nombre de la base de datos:

```
###> 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/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://app:ChangeMe!@127.0.0.1:3306/app?serverVersion=8&charset=utf8mb4"
DATABASE_URL="mysql://root@127.0.0.1:3306/pw_symfony2023"
###< doctrine/doctrine-bundle ###
```

Figura 7: Configuración de la base de datos en SYMFONY

#### 3.1. Creación de la Entidad Artículos

A continuación, en la terminal, introducimos el siguiente comando:

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

Lo que creará una nueva **entidad** –clase que representa una tabla en la base de datos– dentro de la aplicación SYMFONY. Respondemos a las preguntas que nos haga la terminal hasta que hayamos terminado de agregar atributos. Hay que tener presente que queremos agregar lo siguiente:

- **Título.**
- **Descripción.**
- **Cuerpo.**

```
PS C:\xampp\htdocs\PW_Symfony2023> php bin/console make:entity

Class name of the entity to create or update (e.g. GentleChef):
> Articulos

created: src/Entity/Articulos.php
created: src/Repository/ArticulosRepository.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):
> titulo

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/Articulos.php

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

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

Success!
```

Figura 8: Configuración de la base de datos en SYMFONY

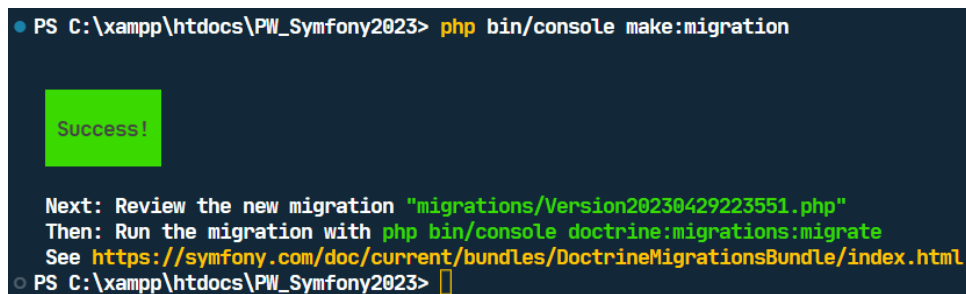


### 3.2. Migración de la Entidad Artículos

A continuación, introducimos el comando:

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

Con lo que podremos migrar las tablas a la base de datos pw\_symfony2023.



```
PS C:\xampp\htdocs\PW_Symfony2023> php bin/console make:migration

Success!

Next: Review the new migration "migrations/Version20230429223551.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
PS C:\xampp\htdocs\PW_Symfony2023>
```

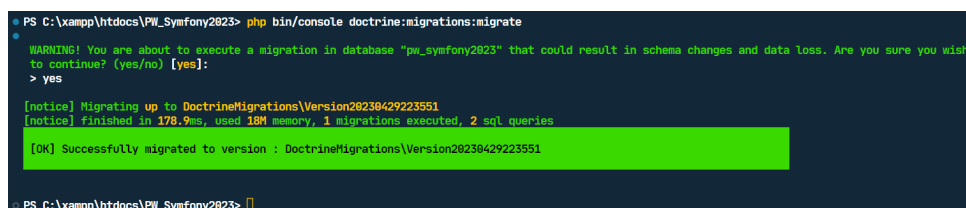
Figura 9: Creación del archivo de migración.

Antes de continuar con la siguiente sección, tenemos que introducir el comando:

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

Esto es debido a que el anterior comando, `php bin/console make:migration` crea el archivo donde se configura la migración a realizar. En dicho archivo se pueden modificar los valores atribuidos previamente durante la creación de una **Entidad**, tal y como hicimos unos pasos más atrás.

El uso del comando `php bin/console doctrine:migrations:migrate` termina de realizar la migración a la base de datos.



```
PS C:\xampp\htdocs\PW_Symfony2023> php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "pw_symfony2023" 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\Version20230429223551
[notice] finished in 178.9ms, used 18M memory, 1 migrations executed, 2 sql queries

[OK] Successfully migrated to version : DoctrineMigrations\Version20230429223551
PS C:\xampp\htdocs\PW_Symfony2023>
```

Figura 10: Migración de las tablas a la base de datos.

Antes de continuar con la siguiente sección, crearemos unas tuplas de ejemplo en la tabla `articulos` de nuestra base de datos para poder mostrarlos en la vista `articulos`.

## 4. Creación Controlador `ArticuloController`

Para crear nuestro **Controlador** para los artículos, el cual llamaremos `ArticuloController`, abriremos una terminal en VISUAL STUDIO e introduciremos el siguiente comando:

```
$ php bin/console make:controller ArticuloController
```

SYMFONY orquesta toda creación que queramos hacer y predispone al desarrollador bases sobre qué construir la aplicación web de manera rápida y cómoda. La ruta de dicha página será `/articulos`.

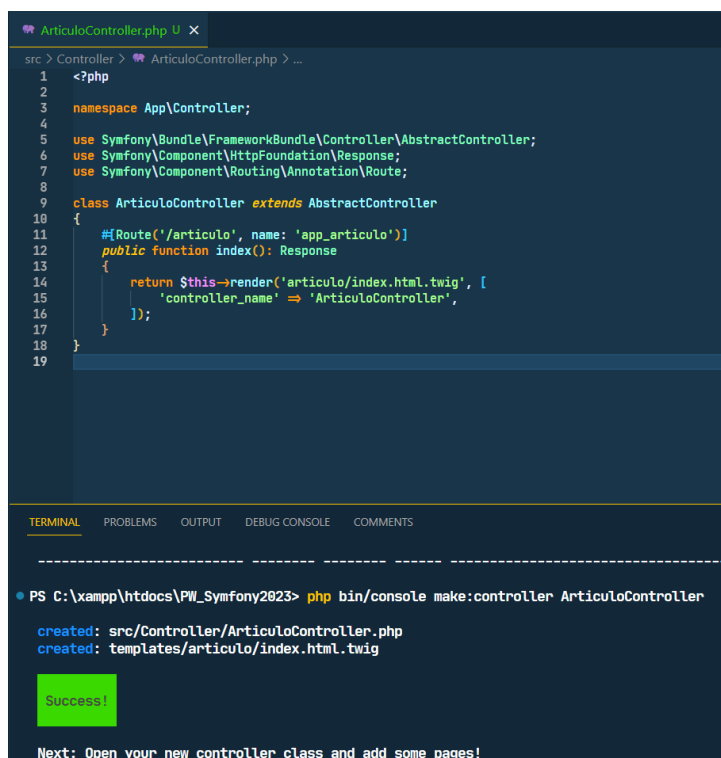


Figura 11: Creación de `ArticuloController`

### 4.1. Vista `articulos`

Queremos que la vista principal de `\articulos` tenga una opción para **añadir artículos**. Para ello, hemos de modificar los siguientes archivos:

NOTA: Véase que caracteres como la “ñ” o las tildes no se imprimen correctamente por el paquete `lstlistings` de `LATEX`.

- `base.html.twig`: Modificamos el código incluido para personalizarlo a nuestra manera. En este caso, se ha dejado como se muestra a continuación:

```
1 // ...
2
3 <title>{% block title %}Titulo App Symfony 2023{% endblock %}</title>
4
```

```
5 // ...
6
7 <body>
8     <h1> App Symfony 2023 </h1>
9     {% block body %}{% endblock %}
10 </body>
11 </html>
```

- index.html.twig: Al igual que el .twig previo, modificamos a nuestro gusto:

```
1 {% extends 'base.html.twig' %}
2
3 {% block title %}Articulos{% endblock %}
4
5 {% block body %}
6 <style>
7     .example-wrapper { margin: 1em auto; font: 16px/1.5 sans-serif; }
8     .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
9 </style>
10
11 <div class="example-wrapper">
12     <h2>Bienvenido a la web sobre Articulos.</h2>
13     <a href="{{ path('insertar_articulo') }}" title="Annadir articulo">Annadir
14         articulo</a>
15     <p>Estos son los articulos publicados.</p>
16 </div>
17 {% endblock %}
```

- routes.yaml: Para añadir la ruta de la vista que queremos crear:

```
1 insertar_articulo:
2     path: /articulos/nuevo
3     controller: App\Controller\ArticuloController::nuevo
```

- services.yaml: Configuramos el controlador y lo definimos como un servicio para que SYMFONY pueda inyectar las dependencias necesarias en el controlador:

```
1 // ...
2
3 App\Controller\ArticuloController:
4     arguments: [ '@doctrine.orm.entity_manager ']
```

Por lo que tendríamos la siguiente vista si accedemos a la ruta: `http://127.0.0.1:8000/articulos:`

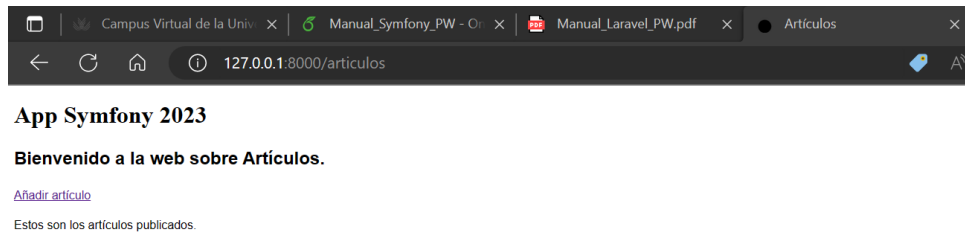


Figura 12: Vista web de articulos

## 4.2. Funcionalidad para mostrar articulos en `articulos`

Para mostrar todos los artículos de la base de datos en `articulos`, modificaremos el método `index` –que es el que venía por defecto creado– añadiendo el siguiente código:

```

1      #[Route('/articulos', name: 'app_articulo')]
2      public function index(ManagerRegistry $doctrine): Response
3      {
4          $articulos = $doctrine->getRepository(Articulos::class)->findAll();
5          return $this->render('articulo/index.html.twig', [
6              'articulos' => $articulos,
7          ]);
8      }

```

Y modificamos su correspondiente vista `index.html.twig`:

```

1      // ...
2
3      {% for articulo in articulos %}
4          <h3><a href="{{ path('mostrar_articulo', {id:articulo.id}) }}">{{ articulo.
5              titulo }}</a></h3>
6          <h6>{{ articulo.descripcion}}</h6>
7          <h7>{{ articulo.cuerpo}}</h7>
8      {% endfor %}
9
10     </div>
11     {% endblock %}

```

Si accedemos a la vista desde la web, veremos lo siguiente:



Figura 13: Vista web Artículos.

Para mostrar cada artículo individualmente, necesitaremos crearnos un método que muestre un artículo identificado por su `id` y una vista que corresponda con dicho método. Entonces, nos crearemos el método `mostrar_articulo` con el siguiente código incluido en nuestro **controlador**:

```

1      #[Route('/articulos/{id}', name: 'mostrar_articulo')]
2      public function mostrar_articulo(ManagerRegistry $doctrine, int $id): Response
3      {
4          $articulo = $doctrine->getRepository(Articulos::class)->find($id);
5          return $this->render('articulo/mostrar.html.twig', [
6              'articulo' => $articulo,
7          ]);
8      }

```

A continuación, creamos la vista `mostrar.html.twig` con el siguiente código incluido:

```

1      {% extends 'base.html.twig' %}
2
3      {% block title %} Artículo {{ articulo.id }}{% endblock %}
4
5      {% block body %}
6
7      <div>

```

```
8      <a href="{{ path('articulos') }}">Ver todos los articulos</a>
9
10     <p>Este es el articulo {{ articulo.id }}.</p>
11
12     <h3>{{ articulo.titulo }}</h3>
13     <h5>{{ articulo.descripcion }}</h5>
14     <h6>{{ articulo.cuerpo }}</h6>
15
16 </div>
17 {% endblock %}
```

Y actualizamos el fichero `routes.yaml` asignándole la ruta correspondiente al método creado:

```
1  mostrar_articulo :
2    path:  '/articulos/{id}'
3    controller: 'App\Controller\ArticuloController::mostrar_articulo'
4    methods: GET
```

Si accedemos, por ejemplo, al `articulo2`, se nos mostrará la siguiente vista:

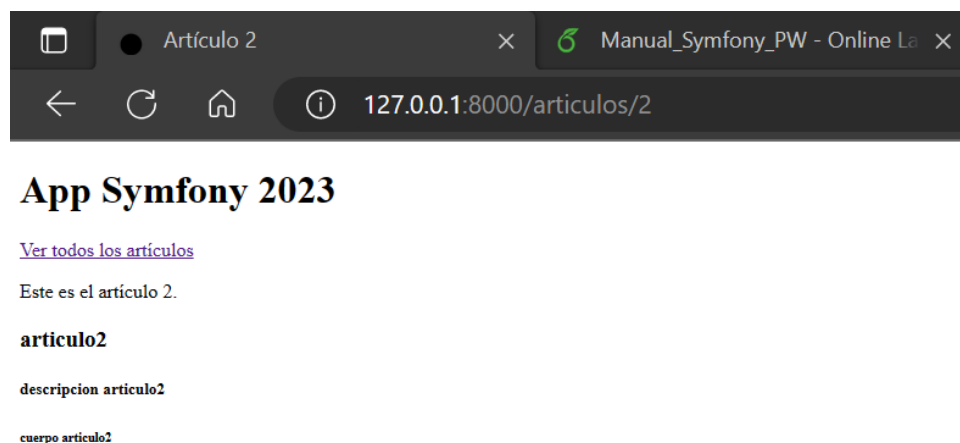


Figura 14: Vista web `mostrar_articulo`.

Véase que en la URL se muestra el identificador correspondiente del objeto leído de la base de datos, en este caso 2. Si pulsamos en `Ver todos los artículos`, volveremos a la vista anterior.

### 4.3. Formulario para añadir artículos

Ahora, para poder añadir artículos a la base de datos y que se muestren en la vista `articulos`, nos creamos un formulario introduciendo el siguiente comando en la terminal de Visual Studio:

```
$ php bin/console make:form ArtículoAddType
```

Deberemos introducir el mismo nombre que utilizamos para la **Entidad**. En nuestro caso, **Articulos**.

```

src > Form > ArtículoAddType.php > ...
1  <?php
2
3  namespace App\Form;
4
5  use App\Entity\Articulos;
6  use Symfony\Component\Form\AbstractType;
7  use Symfony\Component\Form\FormBuilderInterface;
8  use Symfony\Component\OptionsResolver\OptionsResolver;
9
10 class ArtículoAddType extends AbstractType
11 {
12     public function buildForm(FormBuilderInterface $builder, array $options): void
13     {
14         $builder
15             →add('titulo')
16             →add('descripcion')
17             →add('cuerpo')
18         ;
19     }
20
21     public function configureOptions(OptionsResolver $resolver): void
22     {
23         $resolver→setDefaults([
24             'data_class' => Articulos::class,
25         ]);
26     }
27 }

```

```

PS C:\xampp\htdocs\PW_Symfony2023> php bin/console make:form ArtículoAdd

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> Articulos

created: src/Form/ArticuloAddType.php

Success!

Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html
PS C:\xampp\htdocs\PW_Symfony2023>

```

Figura 15: Creación de formulario `ArticuloAddType`.

Modificaremos los siguientes archivos:

- `ArticuloAddType.php`: incluiremos el siguiente código en el método `buildForm`:

```

1  use Symfony\Component\Form\Extension\Core\Type\TextType;
2  use Symfony\Component\Form\Extension\Core\Type\TextareaType;
3  use Symfony\Component\Form\Extension\Core\Type\SubmitType;
4
5  // ...
6
7  public function buildForm(FormBuilderInterface $builder, array $options):
8      void
9  {
10     $builder
11         →add('Titulo', TextType::class, ['required'=>true])

```

```
11         ->add('Descripcion', TextAreaType::class)
12         ->add('Cuerpo', TextAreaType::class)
13         ->add('Crear articulo', SubmitType::class)
14     ;
15 }
```

- ArticuloController.php: Debemos añadir tres herramientas y modificar el método nuevo creado anteriormente con el siguiente código:

```
1     use Symfony\Component\HttpFoundation\Request;
2     use App\Form\ArticuloAddType;
3     use Doctrine\Persistence\ManagerRegistry;
4
5     // ...
6
7     #[Route('/articulos/nuevo', name: 'app_articulo_nuevo')]
8     public function nuevo(ManagerRegistry $doctrine, Request $request):
9         Response
10    {
11        $entityManager = $doctrine->getManager();
12        $articulo = new Articulos();
13        $form = $this->createForm(ArticuloAddType::class, $articulo);
14        $form->handleRequest($request);
15        if($form->isSubmitted() && $form->isValid())
16        {
17            $articulo = $form->getData();
18            $entityManager->persist($articulo);
19            $entityManager->flush();
20            return $this->redirectToRoute('app_articulo', [
21                'articulo'=>$articulo,
22            ]);
23        }
24        return $this->render('articulo/nuevo.html.twig', [
25            'form' => $form->createView(),
26        ]);
27    }
```

Y creamos la vista nuevo.html.twig incluyendo el siguiente código:

```
1 { % extends 'base.html.twig' %}
2
3 {% block title %}Crear articulo{% endblock %}
4
5 {% block body %}
6
7 <div>
```

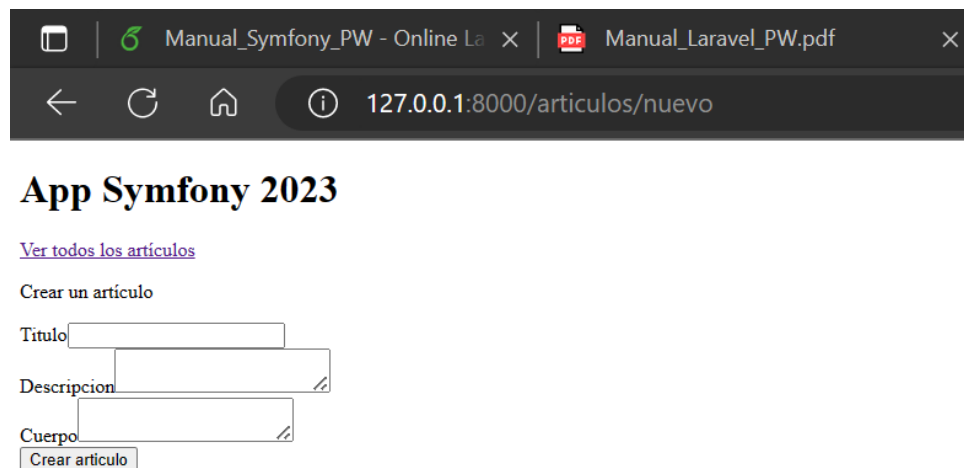


```
8      <a href="{{ path('articulos') }}">Ver todos los articulos</a>
9
10     <p>Crear un articulo</p>
11     {{ form(form) }}
12
13 </div>
14 {% endblock %}
```

En el fichero `routes.yaml`, agregamos la ruta del método nuevo:

```
1 insertar_articulo :
2   path: /articulos/nuevo
3   controller: App\Controller\ArticuloController::nuevo
4   methods: GET
```

Si pinchamos en `añadir artículo`, nos redireccionará a la siguiente vista:



Manual\_Symfony\_PW - Online La x | Manual\_Laravel\_PW.pdf x

← ↻ 🏠 ⓘ 127.0.0.1:8000/articulos/nuevo

## App Symfony 2023

[Ver todos los artículos](#)

Crear un artículo

Titulo

Descripcion

Cuerpo

Figura 16: Vista web formulario nuevo.

Rellenamos el formulario con el siguiente artículo, `articulo4`, y comprobamos el funcionamiento del formulario al pinchar el botón `crear articulo`.

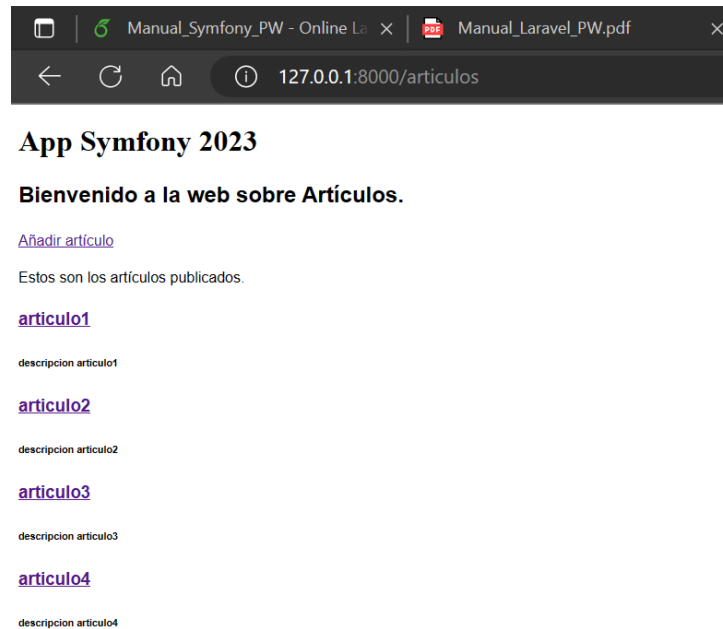


Figura 17: Vista web articulos actualizada.

Comprobamos como en la base de datos, desde phpMyAdmin, se ha actualizado la tabla `articulos`.



proyectopw	
pw	
pw_laravel2023	
pw_symfony2023	
Nueva	
articulos	
doctrine_migration_versions	
messenger_messages	
test	

				id	titulo	descripcion	cuerpo
<input type="checkbox"/>	✎ Editar	📋 Copiar	🗑 Borrar	1	articulo1	descripcion articulo1	cuerpo articulo1
<input type="checkbox"/>	✎ Editar	📋 Copiar	🗑 Borrar	2	articulo2	descripcion articulo2	cuerpo articulo2
<input type="checkbox"/>	✎ Editar	📋 Copiar	🗑 Borrar	3	articulo3	descripcion articulo3	cuerpo articulo3
<input type="checkbox"/>	✎ Editar	📋 Copiar	🗑 Borrar	4	articulo4	descripcion articulo4	cuerpo articulo4

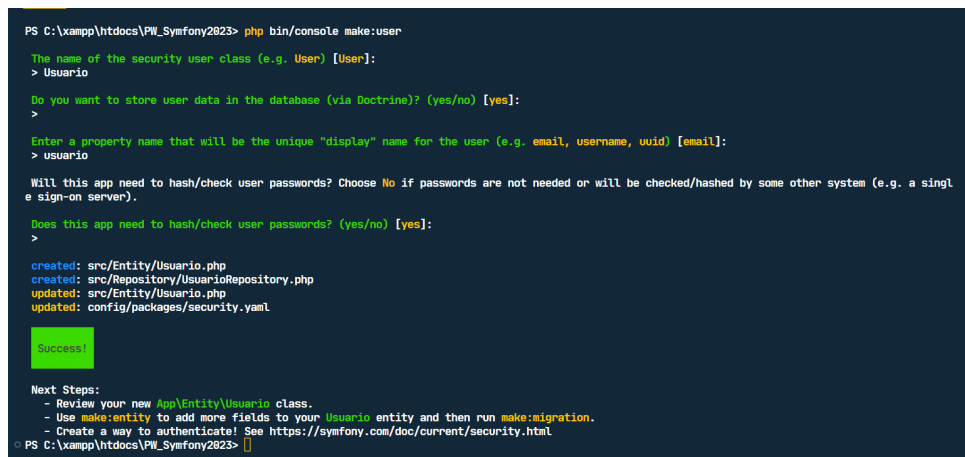
Figura 18: Tabla articulos actualizada.

## 5. Login y Registro de Usuario

### 5.1. Creación de la Entidad Usuario

Procederemos a crear una **Entidad** llamada `Usuario` con el comando:

```
$ php bin/console make:user
```



```
PS C:\xampp\htdocs\PW_Symfony2823> php bin/console make:user

The name of the security user class (e.g. User) [User]:
> Usuario

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
> usuario

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).
Does this app need to hash/check user passwords? (yes/no) [yes]:
>

created: src/Entity/Usuario.php
created: src/Repository/UsuarioRepository.php
updated: src/Entity/Usuario.php
updated: config/packages/security.yaml

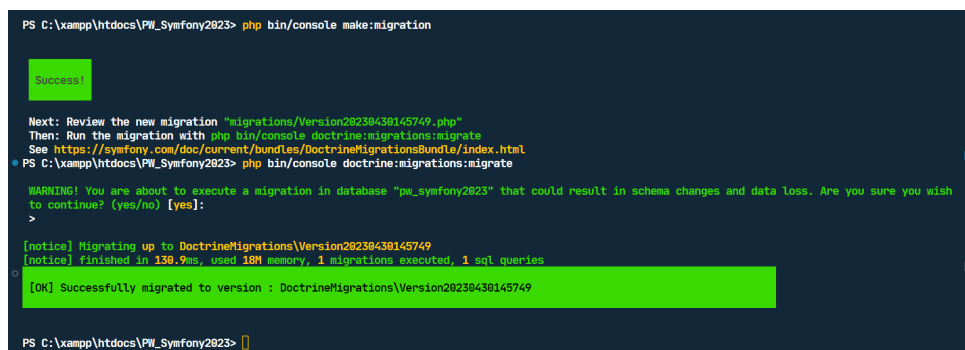
Success!

Next Steps:
- Review your new App\Entity\Usuario class.
- Use make:entity to add more fields to your Usuario entity and then run make:migration.
- Create a way to authenticate! See https://symfony.com/doc/current/security.html

PS C:\xampp\htdocs\PW_Symfony2823>
```

Figura 19: Creación de la Entidad Usuario.

A continuación, migramos a la base de datos dicha **Entidad**.



```
PS C:\xampp\htdocs\PW_Symfony2823> php bin/console make:migration

Success!

Next: Review the new migration "migrations/Version28238438145749.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

PS C:\xampp\htdocs\PW_Symfony2823> php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "pw_symfony2823" that could result in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
>

[notice] Migrating up to DoctrineMigrations\Version28238438145749
[notice] finished in 138.9ms, used 18M memory, 1 migrations executed, 1 sql queries

[OK] Successfully migrated to version : DoctrineMigrations\Version28238438145749

PS C:\xampp\htdocs\PW_Symfony2823>
```

Figura 20: Migración de la Entidad Usuario.

## 5.2. Creación de la vista de registro

Nuevamente, en la terminal, introducimos un comando para que Symfony nos cree un registro por defecto:

```
$ php bin/console make:registration-form
```

```
PS C:\xampp\htdocs\PW_Symfony2023> php bin/console make:registration-form

Creating a registration form for App\Entity\Usuario

Do you want to add a @UniqueEntity validation annotation on your Usuario class to make sure duplicate accounts aren't created? (yes/no) [yes]:
>

Do you want to send an email to verify the user's email address after registration? (yes/no) [yes]:
> no

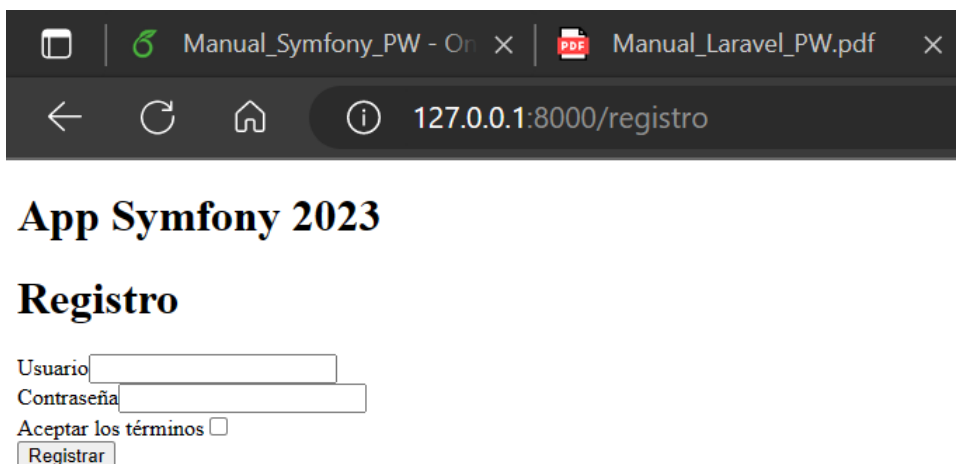
Do you want to automatically authenticate the user after registration? (yes/no) [yes]:
>

! [NOTE] No Guard authenticators found - so your user won't be automatically authenticated after registering.

What route should the user be redirected to after registration?:
[0] _preview_error
[1] _wdt
[2] _profiler_home
[3] _profiler_search
[4] _profiler_search_bar
[5] _profiler_phpinfo
[6] _profiler_xdebug
[7] _profiler_search_results
[8] _profiler_open_file
[9] _profiler
[10] _profiler_router
[11] _profiler_exception
[12] _profiler_exception_css
[13] app_articulo
[14] app_articulo_nuevo
[15] articulos
[16] insertar_articulo
[17] mostrar_articulo
> 15
```

Figura 21: Creación del formulario de registro.

En los ficheros que se han creado `RegistrationController.php`, `register.html.twig`, podemos modificar a nuestro antojo los nombres de las rutas y demás para que salga en español, tal y como se ve en la siguiente imagen de la vista web:



Manual\_Symfony\_PW - On x | Manual\_Laravel\_PW.pdf x

← ↻ 🏠 ⓘ 127.0.0.1:8000/registro

# App Symfony 2023

## Registro

Usuario

Contraseña

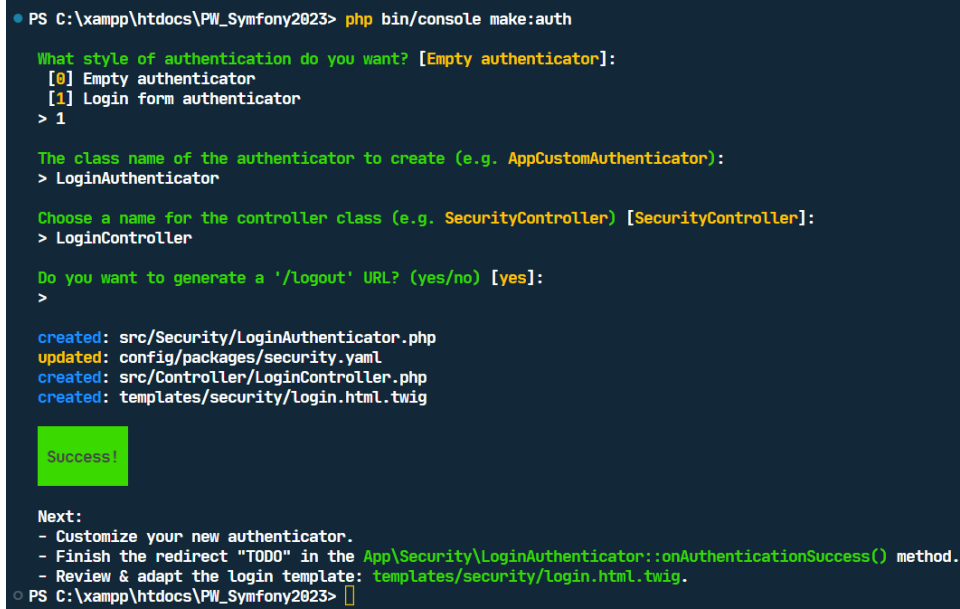
Aceptar los términos ☐

Figura 22: Vista web Registro.

### 5.3. Creacion de la vista de inicio de sesión

Para la vista del *login*, usaremos el siguiente comando:

```
$ php bin/console make:auth
```



```
PS C:\xampp\htdocs\PW_Symfony2023> php bin/console make:auth

What style of authentication do you want? [Empty authenticator]:
  [0] Empty authenticator
  [1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginAuthenticator

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
> LoginController

Do you want to generate a '/logout' URL? (yes/no) [yes]:
>

created: src/Security/LoginAuthenticator.php
updated: config/packages/security.yaml
created: src/Controller/LoginController.php
created: templates/security/login.html.twig

Success!

Next:
- Customize your new authenticator.
- Finish the redirect "TODO" in the App\Security>LoginAuthenticator::onAuthenticationSuccess() method.
- Review & adapt the login template: templates/security/login.html.twig.
PS C:\xampp\htdocs\PW_Symfony2023>
```

Figura 23: Creación de Login.

Al igual que hicimos con los archivos del registro, modificamos a nuestro antojo, pero si hay algo importante en esta sección, entonces esto sería modificar la ruta del archivo `LoginAuthenticator.php` para que a la hora de iniciar sesión nos redirija a la página de artículos.

```
1 public function onAuthenticationSuccess(Request $request, TokenInterface $token
2     , string $firewallName): ?Response
3     {
4         if ($targetPath = $this->getTargetPath($request->getSession(),
5             $firewallName)) {
6             return new RedirectResponse($targetPath);
7         }
8
9         // For example:
10        return new RedirectResponse($this->urlGenerator->generate('articulos'));
11        //throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
12    }
```

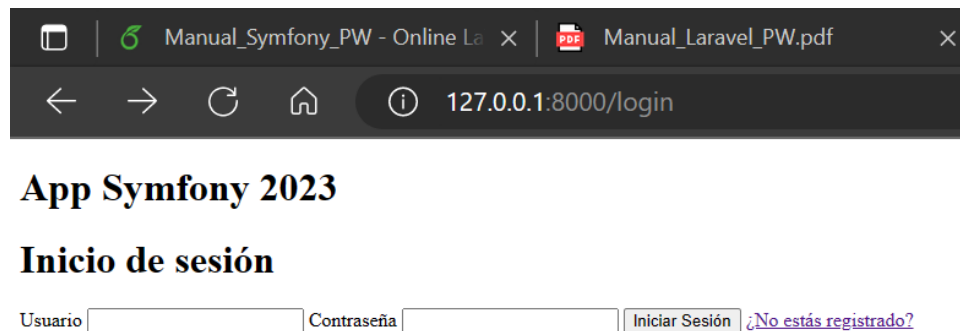


Figura 24: Vista web de Login.

## 5.4. Botón de cerrar sesión

Ahora, en la vista de `articulos` queremos mostrar quién ha iniciado sesión y un botón que cierre la sesión del actual usuario *logueado*. Para ello, vamos a modificar los siguientes archivos:

- `index.html.twig`: Añadimos los botones de inicio y cierre de sesión –mediante una imagen insertada en la ruta `public/uploads/-`, todo ello mediante un condicional que compruebe si el usuario está *logueado*:

```

1  {% extends 'base.html.twig' %}
2
3  {% block title %}Articulos{% endblock %}
4
5  {% block body %}
6      <style>
7          .example-wrapper { margin: 1em auto; font: 16px/1.5 sans-serif; }
8          .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
9          .logout-btn { background-image: url('/uploads/as_meme.jpg'); }
10     </style>
11
12     <div class="example-wrapper">
13         <h2>Bienvenido a la web sobre Articulos.</h2>
14         {% if user %}
15             <p>Usuario: {{ user.getUserIdentifier() }}</p>
16             <a href="{{ path('app_logout') }}" title="Cerrar sesion" class="logout-
17                 btn"></a>
18         {% else %}
19             <a href="{{ path('login') }}" title="Iniciar sesion">Iniciar sesion</a>
20         {% endif %}
21
22         <a href="{{ path('insertar_articulo') }}" title="Annadir articulo">Annadir
23             articulo</a>
24         <p>Estos son los articulos publicados.</p>
25
26         {% for articulo in articulos %}

```

```
25         <h3><a href="{{ path('mostrar_articulo', {id:articulo.id}) }}">{{
            articulo.titulo }}</a></h3>
26         <h6>{{ articulo.descripcion}}</h6>
27     {% endfor %}
28
29 </div>
30 {% endblock %}
```

- LoginController.php: Añadiremos una variable privada tokenStorage, un constructor y modificaremos el método logout:

```
1 use Symfony\Component\Security\Core\Authentication\Token\Storage\
    TokenStorageInterface;
2
3 // ...
4
5 class LoginController extends AbstractController
6 {
7     private $tokenStorage;
8
9     public function __construct(TokenStorageInterface $tokenStorage)
10    {
11        $this->tokenStorage = $tokenStorage;
12    }
13
14    // ...
15
16    public function logout(): Response
17    {
18        $this->tokenStorage->setToken(null);
19
20        return $this->redirectToRoute('app_articulo');
21    }
22 }
```

- routes.yaml: Añadimos las rutas que hagan falta:

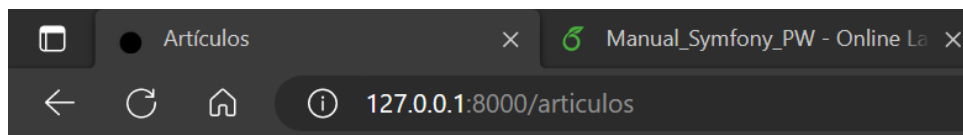
```
1 registro :
2     path: '/registro '
3     controller: 'App\Controller\RegistrationController::register '
4     methods: GET
5
6 login :
7     path: '/login '
8     controller: 'App\Controller\LoginController::login '
9     methods: GET|POST
10
11 app_logout:
12     path: '/logout '
```

```
13 controller: 'App\Controller\LoginController::logout'
14 methods: GET
```

- `services.yaml`: tenemos que agregar la siguiente línea. Esto permite que el controlador `LoginController` pueda hacer referencia a `security.logout_url_generator` en lugar de `Symfony\Component\Security\Http\Logout\LogoutUrlGenerator`, lo que hace que el código sea más legible y fácil de mantener.

```
1 services:
2     Symfony\Component\Security\Http\Logout\LogoutUrlGenerator:
3         alias: security.logout_url_generator
```

Por lo que, si volvemos a la vista de artículos, veremos lo siguiente:



## App Symfony 2023

### Bienvenido a la web sobre Artículos.

Usuario: Alejandro



[Añadir artículo](#)

Estos son los artículos publicados.

[articulo1](#)

descripcion articulo1

[articulo2](#)

descripcion articulo2

[articulo3](#)

descripcion articulo3

[articulo4](#)

descripcion articulo4

Figura 25: Vista web de Artículos logueado.



Si pinchamos en la foto agregada, cerraremos sesión y veremos la siguiente vista de `articulos` actualizada:

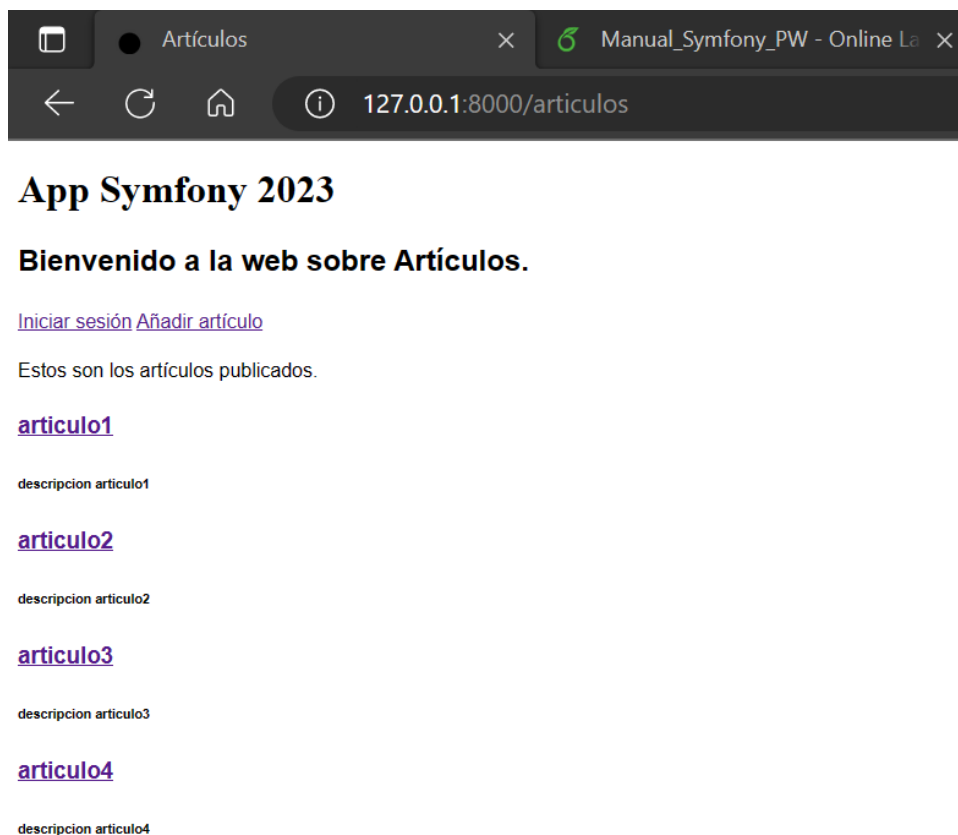


Figura 26: Vista web de `Articulos` sin loguear.

## 6. Conclusiones

El desarrollo de este manual ha sido sin duda el más rápido de los tres realizados durante el curso de PROGRAMACIÓN WEB –un aproximado de tres días–, y es que SYMFONY, dada su facilidad de proporcionar las Entidades, Controladores y escaso código ha implementar, invitaba a desarrollarlo en poco tiempo.

- Ciertamente es que la ausencia de un manual de guía como fue en LARAVEL se ha notado, pero esto tiene un punto primordial a diferencia de los otros entregables, y es que se trabaja más en la **disciplina** de enfrentarse al problema tú mismo. Sin contar el desarrollo de esta competencia transversal, SYMFONY ha sido el *framework* más fácil de usar –aunque sus *templates* no me han agradado tanto como el `blade` de LARAVEL–.
- También encontré un poco repentino el cambio de definición de rutas en los propios métodos de los **Controladores**, y el uso de `.yaml` a los cuales no he estado tan acostumbrado.