

# Desarrollo Web con Symfony

v.20220719

<b>Introducción</b>	<b>3</b>
<b>Algunos conceptos previos</b>	<b>3</b>
<b>Iteración 0. Instalación e Inicio del proyecto</b>	<b>4</b>
1. Instalar y Configurar Symfony	4
1.1. Instalación en Windows	4
1.2. Instalación en Linux (Ubuntu/Debian)	5
1.3. Instalar herramientas de desarrollo: IDE, Git	5
2. Crear Proyecto Symfony	7
3. Primeras páginas. Controladores y Vistas	9
4. Control de Versiones: Git	12
4.1. Crear repositorio y Confirmar cambios	12
4.2. Repositorio remoto en GitHub	12
4.3. Clonar proyecto para desarrollo	12
4.4. Desplegar proyecto en producción	13
5. Documentación	14
Rutas y URLs	14
<b>Iteración 1. Gestión de usuarios</b>	<b>15</b>
1. Modelo. Configurar y Crear Base de Datos	16
2. Security. User, Login y Logout	18
3. Enviar email	22
4. Formulario de registro	24
5. Resetear Password	26
6. Documentación	28
Modelo	28
Rutas y URLs	28
Actualizar aplicación	29
<b>Iteración 2. Interfaz de usuario</b>	<b>30</b>
1. Preparar plantilla base	30
2. Crear menú principal de usuario	31
3. Añadir un Favicon	32
4. Arreglar formulario de registro y otras plantillas	33
5. Condiciones de Uso	33
6. Documentación	35
Rutas y URLs	35
Vistas y plantillas	35
<b>Iteración 3. Perfil de usuario.</b>	<b>36</b>
1. Modelo. Añadir campos a User	37
2. Actualizar lastLogin y updatedAt	38

3. Perfil de usuario: mostrar, editar y borrar	40
4. Modificar correo electrónico	41
5. Añadir foto de perfil	42
6. Documentación	45
Modelo	45
Rutas y URLs	45
Vistas y plantillas	45

# Introducción

La finalidad de este proyecto es **compartir** una experiencia de desarrollo de una aplicación web, aprendiendo todo lo posible en el camino, con el objetivo de desarrollar alguna herramienta que tenga un uso real y significativo.

A lo largo de esta documentación se mostrarán las bases para desarrollar *paso a paso* una aplicación web desde su instalación en el entorno de desarrollo hasta su despliegue en producción.

Se utilizará el framework **Symfony de PHP** como herramienta central de desarrollo, junto con otras muchas tecnologías, lenguajes, etc: HTML, CSS, JavaScript, SQL, MariaDB, etc.

Inicialmente se desarrollará una aplicación genérica con algunas funcionalidades básicas:

1. Gestión de usuarios, con registro e inicio de sesión basado en nombre de usuario y contraseña, almacenados en una base de datos.
2. Configuración y envío de correo electrónico.
3. Exportar contenido en PDF

Una vez implementadas estas funcionalidades la aplicación se podrá adaptar a distintos objetivos y podrán desarrollarse otras funcionalidades más específicas.

## Algunos conceptos previos

### Java vs PHP

Conceptos	Base de Datos
<a href="#">Framework</a> <a href="#">Modelo-vista-controlador</a> <a href="#">UML</a>	SQL MariaDB ORM
Programación	Herramientas de desarrollo
HTML CSS JavaScript PHP (Composer, <b>Symfony</b> )	<a href="#">Visual Studio Code</a> <a href="#">GitHub</a> <a href="#">Diagrams.net</a>

# Iteración 0. Instalación e Inicio del proyecto

Objetivos:

1. Instalar Symfony y el entorno de desarrollo.
2. Crear proyecto.
3. Crear primeras páginas.
4. Crear repositorio remoto y confirmar cambios.

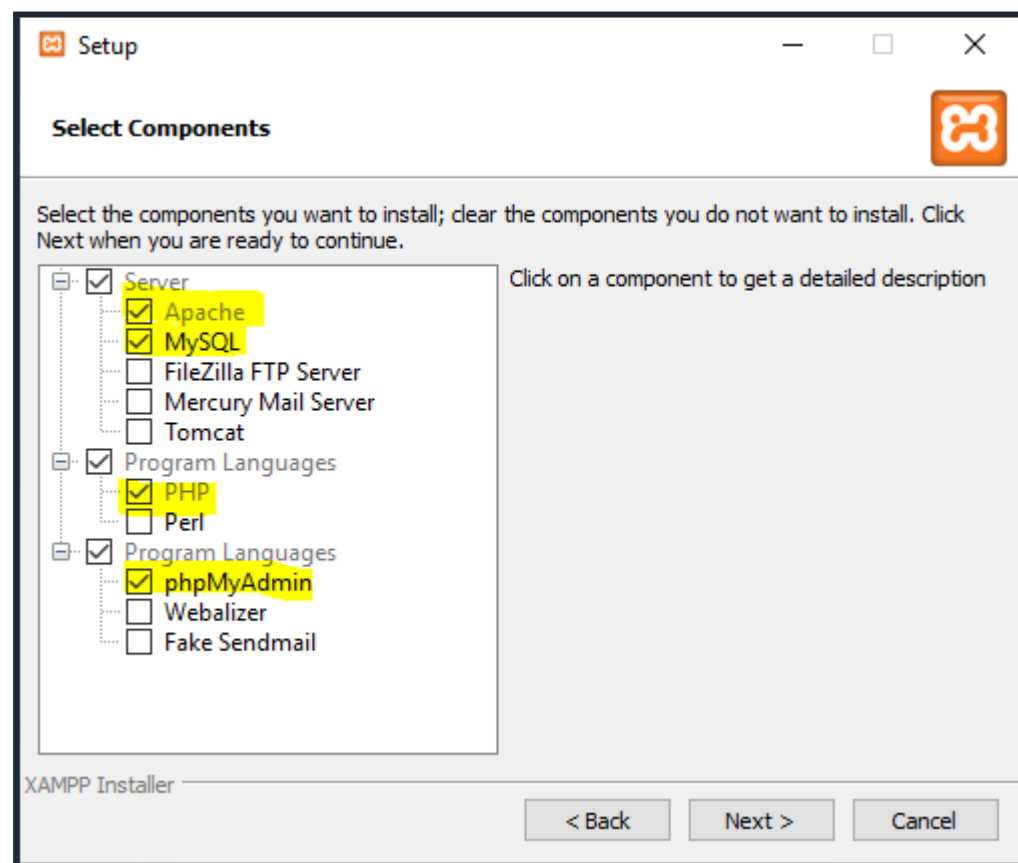
## 1. Instalar y Configurar Symfony

- [Installing & Setting up the Symfony Framework](#)

### 1.1. Instalación en Windows

1. Instalar PHP y extensiones

Instalando [Xampp](#) podemos instalar, además de PHP, el gestor de BD MariaDB y, aunque no son imprescindibles para desarrollar, el servidor web Apache y la aplicación web phpMyAdmin para gestionar las bases de datos.



2. Instalar [Composer](#) para gestionar las dependencias de paquetes y librerías que usa el proyecto.
3. [Instalar Symfony CLI](#)

- a. Añadir ruta del comando symfony a la variable de entorno PATH. Iniciar un nuevo terminal para que se apliquen los cambios.

#### 4. Comprobar requisitos del sistema

```
symfony check:requirements
```

5. Opcional. Seguir recomendaciones para mejorar la configuración. Por ejemplo:
  - a. Editar C:\xampp\php\php.ini y descomentar o editar las siguientes líneas:
    - i. realpath\_cache\_size = 5M
    - ii. post\_max\_size = 100M
    - iii. extension=intl
    - iv. zend\_extension=opcache

```
[OK]
Your system is ready to run Symfony projects
```

Para ampliar:

- [Scoop](#), [winget](#)
- [PHP accelerator](#)

## 1.2. Instalación en Linux (Ubuntu/Debian)

También podemos instalar Symfony y lo necesario para el desarrollo del proyecto en Linux.

```
# Instala PHP, Composer, Git, BD, driver php-mysql, etc.
sudo apt install php php-xml composer curl git mariadb-server
php-mysql

# Instala Symfony-CLI
curl -sLf
'https://dl.cloudsmith.io/public/symfony/stable/setup.deb.sh' |
sudo -E bash
sudo apt install symfony-cli
```

Opcionalmente se puede instalar también Apache y phpMyAdmin.

## 1.3. Instalar herramientas de desarrollo: IDE, Git

En esta documentación se utilizará el IDE [Visual Studio Code](#), aunque se puede usar cualquier otro perfectamente.

Se instalan algunos plugins para ampliar las funcionalidades del IDE, orientados a desarrollo web, PHP y Symfony:

- [PHP Intelephense](#)
- [PHP Namespace Resolver](#)
- [Symfony for VS Code](#)
- [Twig](#)
- [Twig Language 2](#)

Otros *plugins* útiles:

- [GitGraph](#)

Para ampliar:

- [Symfony code snippets And Twig Support & Yaml - Visual Studio Marketplace](#)
- [Symfony Extension Pack - Visual Studio Marketplace](#)
- [How to Setup VS Code for Symfony | Symfony 6 for Beginners | Learn Symf...](#)
  - [00:00](#) - Introduction
  - [02:01](#) - Community Material Theme
  - [03:05](#) - [PHP Intelephense](#)
  - [04:01](#) - PHP Doc Comment
  - [05:05](#) - [PHP Namespace Resolver](#)
  - [06:13](#) - HTML/JS Snippets
  - [07:03](#) - Symfony code snippets and Twig Support
  - [08:06](#) - [Symfony for VS Code](#)
  - [08:22](#) - [Twig & Twig Language 2](#)
  - [08:52](#) - VS Code Helpers

Instalar [Git](#) (si no lo está ya)

1. Configurar identidad en Git

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

## 2. Crear Proyecto Symfony

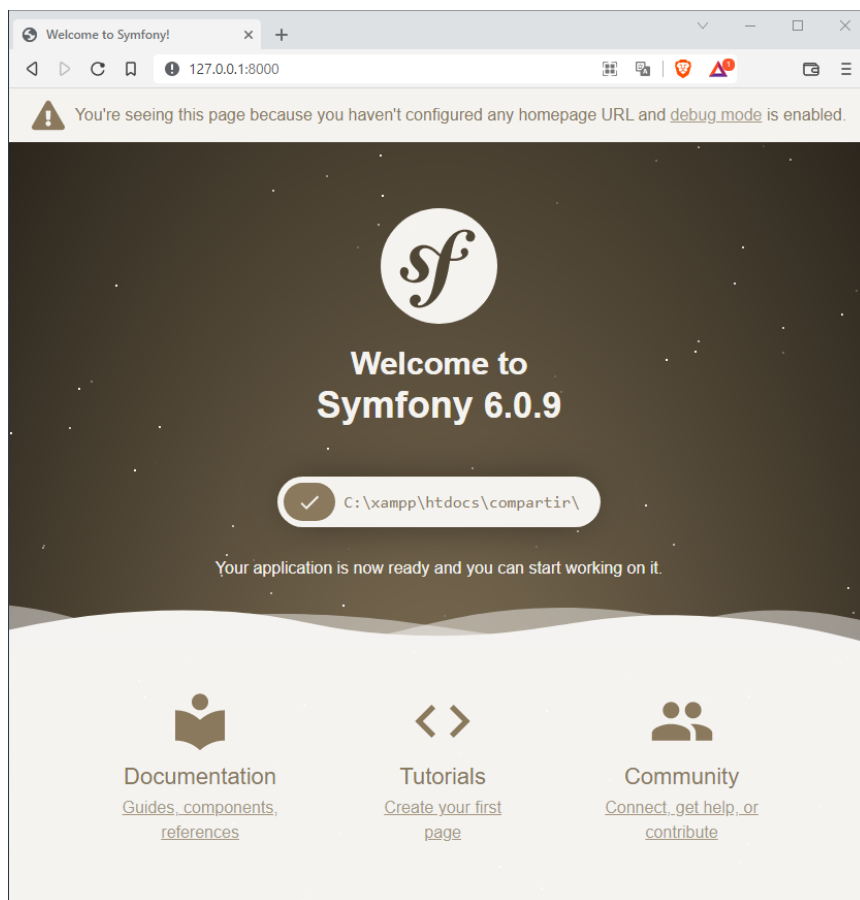
1. Crear proyecto para una aplicación web (`--webapp`) usando el comando symfony:  
(Por ahora el nombre clave que se usará para el proyecto será “compartir”)

```
symfony new compartir --webapp
```

2. Ejecutar el servidor web de Symfony y acceder a la aplicación desde el navegador:

```
cd compartir  
symfony server:start
```

Acceder a <http://127.0.0.1:8000/>



3. Instalar primeras librerías o paquetes (*bundles*) de Symfony:

```
# Depurador: añade una barra inferior en el navegador  
composer require symfony/profiler-pack --dev  
  
# Maker: instala el comando make que ayuda a generar código  
composer require symfony/maker-bundle --dev  
  
# Anotaciones  
composer require annotations  
  
# Plantillas Twig  
composer require symfony/twig-bundle
```

El parámetro `--dev` indica que este paquete sólo se usará en desarrollo (`--dev` = development)





### 3. Primeras páginas. Controladores y Vistas

- [Create your First Page in Symfony](#)

Symfony utiliza el patrón en el diseño Modelo-Vista-Controlador, o MVC, que separa la implementación de los interfaces de usuario (vista), los datos (modelo) y la lógica de control (controlador) de la aplicación. En las primeras páginas no necesitaremos base de datos así que nos centraremos en los controladores y vistas.

Vamos a crear la primera página de la aplicación que será la raíz o página de inicio (/ o *index*).

1. Crear el primer controlador y la primera vista.

```
# El comando make nos ayuda a generar el código del primer controlador y la primera vista.
```

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

```
Choose a name for your controller class (e.g. GentleChefController):
```

```
> Index
```

```
created: src/Controller/IndexController.php
```

```
created: templates/index/index.html.twig
```

```
Success!
```

2. Editamos la URL de la ruta en el controlador IndexController.php para asociarlo a la raíz del sitio web.

```
class IndexController extends AbstractController
{
    #[Route('/', name: 'app_index')]
    public function index(): Response
    {
        return $this->render('index/index.html.twig', [
            'controller_name' => 'IndexController',
        ]);
    }
}
```


3. Podemos editar a nuestro gusto la plantilla index.html.twig de la vista que mostrará el navegador. El contenido twig se expresa en bloques `{{ ... }}` `{% ... %}` y `{# ... #}` (aprende más en [Creating and Using Templates \(Symfony Docs\)](#)). El resto es

HTML, CSS, u otro contenido ejecutable por el navegador web. Por ejemplo:

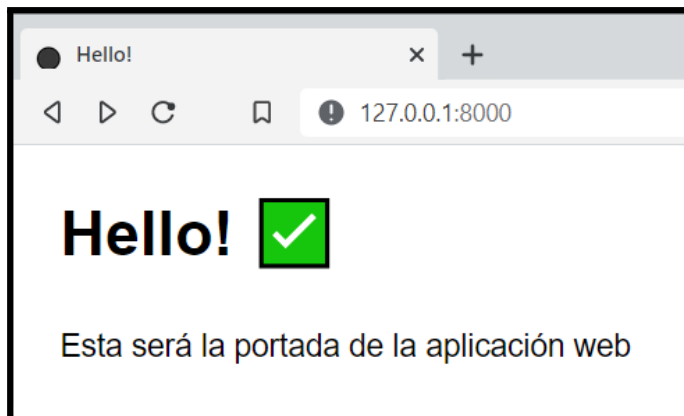
```
{% extends 'base.html.twig' %}

{% block title %}Hello!{% 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>Hello! 
```

4. Ahora la página de inicio se mostrará así:



5. Podemos añadir una nueva página a la web, por ejemplo para describir el proyecto, simplemente añadiendo una nueva función al controlador

```
#[Route('/about', name: 'app_about')]
public function about(): Response
{
    return $this->render('index/about.html.twig');
}
```

Y creando la correspondiente plantilla en la carpeta templates:

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

{% block title %}Sobre el Proyecto{% 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>Sobre el Proyecto</h1>

  <p>¿De qué trata el proyecto? ¿Cuáles son sus objetivos? ¿Qué ofrece? ¿Quiénes forman parte? ¿Qué lo hace diferente?</p>
  <p><a href="{{ path('app_index') }}">Volver a la Página de Inicio</a></p>
</div>
{% endblock %}
```

La vista incluye un enlace a la página inicial usando la función `path()` de Twig e indicando el nombre de la ruta de la página de inicio (`app_index`).

El resultado es otra página web accesible en la URL indicada en el controlador:



6. Así, añadiendo nuevas rutas y métodos en los controladores y creando plantillas twig, podríamos crear y enlazar tantas páginas simples como necesitemos.

## 4. Control de Versiones: Git

### 4.1. Crear repositorio y Confirmar cambios

Al [crear el proyecto symfony](#) ya se inició un repositorio Git y se establecieron las credenciales del desarrollador.

Ahora se añadirán al repositorio los ficheros de la versión actual de la aplicación y se confirmarán los cambios. Estas operaciones se pueden hacer desde el IDE o desde el terminal:

```
git add .  
git commit -m "Inicio del Proyecto"
```

#### Para ampliar:

GitHub en la metodología de desarrollo ([GitHub flow](#))

1. Create a branch
2. Make changes
3. Create a pull request
4. Address review comments
5. Merge your pull request
6. Delete your branch

### 4.2. Repositorio remoto en GitHub

Vamos a crear también un repositorio público en GitHub para compartir la aplicación.

- [Crear repositorio GitHub](#)

Configuramos el repositorio remoto y subimos a él los cambios.

```
git remote add origin https://github.com/aguadodev/compartir.git  
git push -u origin main
```

### 4.3. Clonar proyecto para desarrollo

- [Setting up an Existing Symfony Project](#)

1. Clona o descarga y descomprime el repositorio  
<https://github.com/aguadodev/compartir.git>

```
git clone https://github.com/aguadodev/compartir.git
```

2. Utiliza Composer para instalar las dependencias de paquetes y librerías

```
cd carpeta_de_proyecto  
composer install
```

3. Arranca el servidor web de desarrollo

```
symfony server:start
```

4. Accede a la aplicación web desde el navegador: <http://127.0.0.1:8000/>

## 4.4. Desplegar proyecto en producción

Cuando el proyecto esté un poco más avanzado se desplegará en producción, en un servidor web *real* accesible desde Internet.

Es un proceso similar al del apartado anterior, pero con algunos requisitos añadidos:

- Un **hosting** en Internet con un servidor web de producción: Apache, nGinx, ..
- Un nombre de **dominio** para acceder a la aplicación web.
- En producción no se deberían instalar ciertas librerías de desarrollo. Puede hacerse con la opción `--no-dev` de composer

```
composer install --no-dev
```

- En un futuro se necesitará también una base de datos y otros recursos cuya configuración puede ser un poco específica dependiendo de las características (sistema operativo) del servidor de producción.

## 5. Documentación

Además de este mismo tutorial se irán documentando también algunos resultados de la aplicación desarrollada, como un diagrama de clases o entidades, etc..

Por ahora, simplemente un resumen de las páginas implementadas en la aplicación y sus rutas.

### Rutas y URLs

Nombre de la ruta	Método (GET/POST/ANY/...)	URL
app_index	ANY	/
app_about	ANY	/about

Esta y otra información se puede obtener con el comando:

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

# Iteración 1. Gestión de usuarios

En esta iteración se implementará una gestión de usuarios básica que se almacenarán en una base de datos y que podrán iniciar y cerrar sesión, registrarse verificando el correo electrónico o resetear la contraseña.

1. Crear y configurar base de datos.
2. Crear tabla User y funcionalidades de login y logout de usuarios.
3. Configurar envío de correo electrónico.
4. Formulario de registro.
5. Resetear password.

Creamos en el repositorio git una nueva rama principal de nombre "development".

Antes de empezar esta iteración creamos una nueva rama de nombre "iteracion1-gestion-usuarios" desde la rama "development" y cambiamos a ella.

Ahora, desde la rama "iteracion1-gestion-usuarios" crearemos una nueva rama por cada apartado/ funcionalidad básica

# 1. Modelo. Configurar y Crear Base de Datos

- [Databases and the Doctrine ORM \(Symfony Docs\)](#)

Para trabajar con bases de datos en Symfony instalaremos [Doctrine](#), un ORM o mapeador objeto relacional para PHP (una herramienta equivalente en cierto modo a Hibernate para Java).

```
composer require symfony/orm-pack
```

Como no usaremos Docker en este proyecto podemos responder con una “x” a la siguiente pregunta:

```
The recipe for this package contains some Docker configuration.

This may create/update docker-compose.yml or update Dockerfile (if it exists).

Do you want to include Docker configuration from recipes?
[y] Yes
[n] No
[p] Yes permanently, never ask again for this project
[x] No permanently, never ask again for this project
(defaults to y): x
```

La receta (*recipe*) de instalación de Doctrine, además de descargar los nuevos ficheros necesarios, habrá añadido unas cuantas líneas en el fichero de configuración .env.

- [Symfony. Configuration Environments](#)

Entre otras cosas, en el fichero .env se almacenan en forma de variables de entorno datos de configuración como los necesarios para acceder a la base de datos, para enviar correos electrónicos, librerías para generar PDF, etc.

Son datos que pueden ser diferentes en el entorno de producción que en aquel que usamos para desarrollar la aplicación. Incluso cada desarrollador del proyecto podría tener una configuración diferente.

Además éstos son datos sensibles y no deberían guardarse en un repositorio público, por ejemplo en GitHub. Por eso, para trabajar en desarrollo haremos una copia del fichero .env con el nombre .env.local que tendrá la configuración que se utilizará localmente y estará excluido del control de cambios del repositorio como se puede comprobar en .gitignore.

La línea importante que debemos configurar en .env.local es la que contiene la variable DATABASE\_URL, donde indicaremos el driver del gestor de base de datos que usaremos, el usuario, contraseña, nombre o IP del servidor, puerto, nombre de la base de datos. En el caso de usar MariaDB tendremos que indicar también la versión instalada.

Por ejemplo, con una instalación local de MariaDB, utilizando el usuario root sin contraseña y una base de datos de nombre “compartir”, la línea de configuración quedaría así:



```
DATABASE_URL="mysql://root:@127.0.0.1:3306/compartir?serverVersion=mariadb-10.4.24&charset=utf8mb4"
```

Si los datos de configuración son correctos, y si el servidor de base de datos está encendido y funcionando correctamente, ahora podremos crear la base de datos de la aplicación con el siguiente comando:

```
php bin/console doctrine:database:create
```

Si todo fue bien, se habrá creado una base de datos vacía con el nombre indicado y recibiremos el siguiente mensaje:

```
Created database `compartir` for connection named default
```

Sin embargo, si hay algún error con los datos o con el servidor recibiremos un mensaje como el siguiente:

```
SQLSTATE[HY000] [2002] No se puede establecer una conexión ya que el equipo de destino denegó expresamente dicha conexión
```

## 2. Security. User, Login y Logout

- [Security \(Symfony Docs\)](#)

1. Instalamos el paquete de seguridad de Symfony:

```
composer require symfony/security-bundle
```

2. Generamos la clase/entidad User usando el comando make, respondiendo a las preguntas con los valores por defecto:

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

```
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]:
>

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 password server).
Does this app need to hash/check user passwords? (yes/no) [yes]:
>

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!

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

Tal y como indica la salida del comando make, se habrá creado la clase User en el fichero src/Entity/User.php. En el siguiente punto continuamos los pasos siguientes que nos indica el propio comando make.

3. Creamos la tabla User en la base de datos a partir de la entidad creada. El primer comando (make) crea una migración, es decir, un fichero con las sentencias SQL necesarias para generar en la base de datos los cambios haya habido en el modelo de datos de la aplicación. En este caso la nueva entidad User con todos sus atributos. El segundo comando (doctrine) ejecuta la migración, es decir, las sentencias SQL en la base de datos.

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

```
[notice] Migrating up to DoctrineMigrations\Version20220702223136
[notice] finished in 44.9ms, used 18M memory, 1 migrations executed, 1 sql queries
```

4. Generamos un formulario de [Login](#) con el comando make que se encarga de crear un nuevo controlador y una plantilla twig adecuadas.

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

```
created: src/Controller/LoginController.php  
created: templates/login/index.html.twig
```

5. Para activar el formulario de login seguimos las instrucciones de la [documentación de Symfony](#) e incluimos lo siguiente en el fichero config/packages/security.yaml bajo la clave “main”

```
form_login:  
    login_path: login  
    check_path: login
```

6. Actualizamos el controlador y plantilla de login generados siguiendo de nuevo las instrucciones de la [documentación de Symfony](#), por ejemplo así:

src/Controller/LoginController.php

```
<?php  
  
namespace App\Controller;  
  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;  
  
class LoginController extends AbstractController  
{  
    #[Route('/login', name: 'app_login')]  
    public function index(AuthenticationUtils $authenticationUtils): Response  
    {  
        // get the login error if there is one  
        $error = $authenticationUtils->getLastAuthenticationError();  
  
        // last username entered by the user  
        $lastUsername = $authenticationUtils->getLastUsername();  
  
        return $this->render('login/index.html.twig', [  
            'last_username' => $lastUsername,  
            'error'         => $error,  
        ]);  
    }  
}
```

templates/login/index.html.twig

```

{# templates/login/index.html.twig #}
{% extends 'base.html.twig' %}

{% block title %}Hello LoginController!{% endblock %}

{% block body %}
    {% if error %}
        <div>{{ error.messageKey|trans(error.messageData, 'security')
    }}</div>
    {% endif %}

    <form action="{{ { path('login') } }}" method="post">
        <label for="username">Email:</label>
        <input type="text" id="username" name="_username" value="{{
last_username }}" />

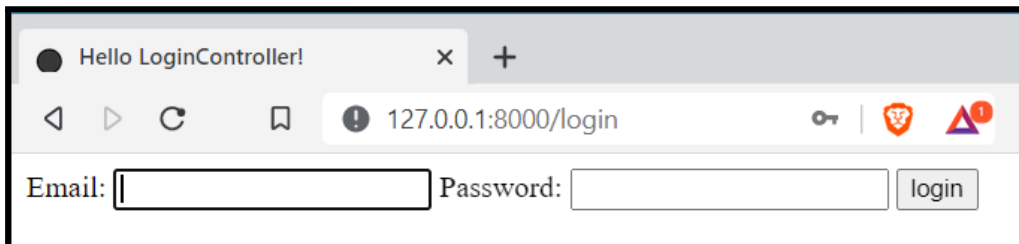
        <label for="password">Password:</label>
        <input type="password" id="password" name="_password" />

        {# If you want to control the URL the user is redirected to on
success
        <input type="hidden" name="_target_path" value="/account" /> #}

        <button type="submit">login</button>
    </form>
{% endblock %}

```

Con lo hecho hasta ahora ya tenemos operativo de login en una nueva ruta cuyo resultado en el navegador será el siguiente:



Hello LoginController!
   
 127.0.0.1:8000/login
   
 Email:  Password:

Llegados a este punto, para probar el proceso de login podríamos crear algunos usuarios directamente en la BD, por ejemplo, usando phpMyAdmin:

Las contraseñas deben estar “encriptadas”. Disponemos de un comando security para generarlas:

```
php bin/console security:hash-password
```

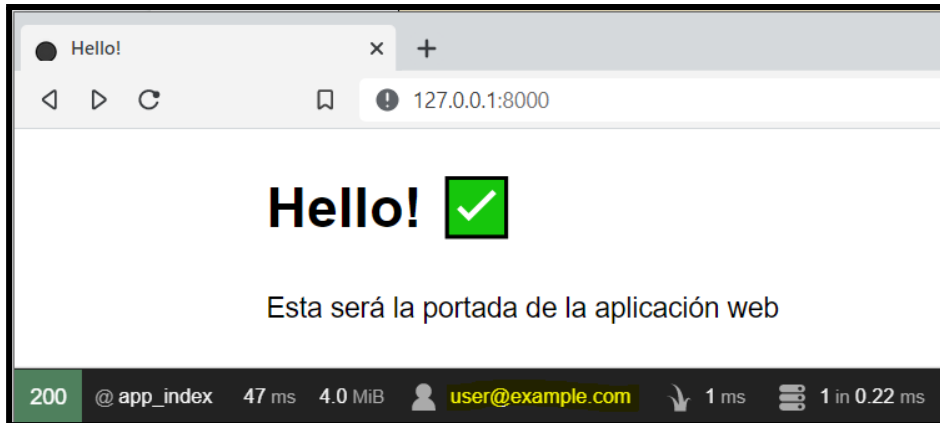
Por ejemplo, hash de “abc123.”:

\$2y\$13\$d0x2nZHmPG60Bm7DmIG8AOc4V56UA9FNuIqzkAu0HpFuxhkj7IjX2

Ejemplo de usuario [user@example.com](mailto:user@example.com) creado en phpMyAdmin:

id	email	roles (DC2Type:json)	password
1	user@example.com		\$2y\$13\$d0x2nZHmPG60Bm7DmIG8AOc4V56UA9FNuIqzkAu0HpF...

Una vez iniciada sesión con un usuario en el entorno de desarrollo podremos comprobarlo en la barra inferior del depurador de Symfony:



7. Por último, para activar el cierre de sesión de usuario o logout ([Logging Out](#)) tendremos que:

- a. Editar de nuevo el archivo security

```
logout:
    path: logout

    # where to redirect after logout
    # target: app_any_route
```

- b. Actualizar de nuevo el controlador LoginController.php, añadiendo el siguiente método:

```
#[Route('/logout', name: 'logout')]
public function logout(): void
{
    // controller can be blank: it will never be called!
    throw new \Exception('Don\'t forget to activate logout in
security.yaml');
}
```

8. Ahora ya podemos cerrar sesión de usuario con la URL <http://127.0.0.1:8000/logout>

Para ampliar:

- [Customizing the Form Login Authenticator Responses \(Symfony Docs\)](#)

### 3. Enviar email

- [Sending Emails with Mailer \(Symfony Docs\)](#)

1. Instalamos las librerías necesarias para enviar correo electrónico.

```
composer require symfony/mailer
```

2. Instalamos las librerías necesarias para usar una cuenta existente de Gmail para enviar los mensajes.

```
composer require symfony/google-mailer
```

3. Configuramos el correo electrónico en el entorno de desarrollo en el archivo de configuración local: .env.local

```
...  
MAILER_DSN=gmail://USERNAME:PASSWORD@default  
...
```

Para utilizar una cuenta de Gmail para enviar mensajes desde la aplicación web habrá que sustituir USERNAME por el nombre de usuario de Google y PASSWORD por una contraseña. Para mayor seguridad, y en caso de tener activada la verificación en dos pasos, podemos generar una [contraseña de aplicaciones desde la sección de Seguridad de la cuenta de Google](#).

4. Por último, para [comprobar que podemos enviar correos](#) desde la aplicación podemos añadir una nueva ruta a un controlador (o crear uno nuevo) para enviar un correo de prueba. Por ejemplo, con el siguiente código, sustituyendo las direcciones de origen y destino:

```
// Importamos las clases necesarias para enviar correos  
use Symfony\Component\Mime\Email;  
use Symfony\Component\Mailer\MailerInterface;  
//...  
  
#[Route('/mail', name: 'app_mail')]  
public function sendEmail(MailerInterface $mailer): Response  
{  
    $email = (new Email())  
        ->from('hello@example.com')  
        ->to('you@example.com')  
        //->cc('cc@example.com')  
        //->bcc('bcc@example.com')  
        //->replyTo('fabien@example.com')
```

```
//->priority(Email::PRIORITY_HIGH)
->subject('Enviando correo de prueba desde la aplicación!')
->text('Contenido en texto plano!')
->html('<p>Contenido en formato HTML. Se pueden aplicar plantillas Twig para
mayor flexibilidad y reutilización</p>');

$mailer->send($email);

dd($email);
}
```

## 4. Formulario de registro

- [How to Implement a Registration Form \(Symfony Docs\)](#)

En la documentación de Symfony se explica como crear fácilmente un formulario de registro de usuarios, incluyendo el envío de correo electrónico para verificar la dirección de correo introducida.

1. Instalamos paquetes de las librerías que podemos necesitar:
  - a. “form” y “validator”, si no las tenemos instaladas ya, para crear y procesar formularios, entre otras cosas.
  - b. “symfonycasts/verify-email-bundle” para validar el correo electrónico introducido por el usuario enviando un mensaje a la dirección introducida.

```
composer require form validator
composer require symfonycasts/verify-email-bundle
```

2. Creamos el formulario de registro con el comando make:

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

```
Creating a registration form for App\Entity\User

Do you want to add a @UniqueEntity validation annotation on your User 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]:
>

By default, users are required to be authenticated when they click the verification link that is emailed to them.
This prevents the user from registering on their laptop, then clicking the link on their phone, without
having to log in. To allow multi device email verification, we can embed a user id in the verification link.

Would you like to include the user id in the verification link to allow anonymous email verification? (yes/no) [no]:
> yes

What email address will be used to send registration confirmations? e.g. mailer@your-domain.com:
> from@example.com

What "name" should be associated with that email address? e.g. "Acme Mail Bot":
> Proyecto Compartir Symfony

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

En el asistente, respondemos en casi todas las respuestas con los valores por defecto. Indicamos que tras el registro se redirija al usuario a la portada de la web.

Tal como nos indica el asistente al terminar, se crean una serie de ficheros con controlador, formulario y vistas que podemos ajustar y personalizar.

```
updated: src/Entity/User.php
created: src/Security/EmailVerifier.php
created: templates/registration/confirmation_email.html.twig
created: src/Form/RegistrationFormType.php
created: src/Controller/RegistrationController.php
created: templates/registration/register.html.twig
```

Success!



3. Por último, creamos y ejecutamos la migración requerida para añadir a la tabla User de la base de datos un nuevo campo que indicará si el usuario creado ha verificado o no su correo electrónico.

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

4. Ahora el registro de usuario ya es funcional y podemos acceder en la URL <http://127.0.0.1:8000/register>

#### **Algunos detalles y cuestiones para ampliar:**

1. Una vez creado el usuario se puede iniciar sesión aunque aún no se haya verificado el email.
  2. Se podría usar el nuevo campo is\_verified para distinguir usuarios con el email verificado de los que no lo tienen y mostrar contenidos o funcionalidades diferentes.
  3. Para la contraseña se incluye una validación por defecto que obliga a una longitud mínima de 6 caracteres.
  4. Ojo! Los correos electrónicos pueden ir a la bandeja de SPAM.
  5. Los enlaces de verificación caducan al cabo de un tiempo (por defecto 1 hora).
    - a. ¿Cómo solicitar otro enlace?
  6. Por defecto, al verificar el correo electrónico NO se inicia sesión. Sobre cómo iniciar sesión automáticamente: [Manual Authentication > Symfony 5 Security](#)
  7. Por defecto, al verificar el correo electrónico se redirige a /register.
    - a. En RegistrationController::verifyUserEmail() podemos personalizar la ruta
    - b. del último redirectToRoute() a la que se redirigirá al usuario después de verificar con éxito el correo electrónico.
  8. Por defecto, al verificar el correo electrónico no se muestra el mensaje flash.
  9. Al introducir un correo electrónico existente en el formulario de registro se muestra el mensaje "There is already an account with this email", lo que puede ser inseguro ya que permitiría a un atacante saber qué usuarios tienen cuenta en la aplicación.
    - a. ¿Cómo arreglarlo? Eliminando el mensaje?
  10. Se puede registrar un usuario aunque se esté logueado con otro.
- Para ampliar más: [Registration Form > Symfony 5 Security: Authenticators | SymfonyCasts](#)

## 5. Resetear Password

- [Reset Password](#)

Vamos a añadir la funcionalidad de que los usuarios puedan resetear una contraseña, ya sea por olvido o porque quieran cambiarla.

1. Instalamos el paquete “symfonycasts/reset-password-bundle” que implementa esta funcionalidad.

```
composer require symfonycasts/reset-password-bundle
```

2. Usamos el comando make para generar el formulario y todo lo necesario.

```
php bin/console make:reset-password
```

```
Let's make a password reset feature!
=====

Implementing reset password for App\Entity\User

- ResetPasswordController -
-----

A named route is used for redirecting after a successful reset. Even a route that does not exist yet can be used here.

What route should users be redirected to after their password has been successfully reset? [app_home]:
> app_index

- Email -
-----

These are used to generate the email code. Don't worry, you can change them in the code later!

What email address will be used to send reset confirmations? e.g. mailer@your-domain.com:
> from@example.com

What "name" should be associated with that email address? e.g. "Acme Mail Bot":
> Proyecto Compartir Symfony
```

Tras el asistente se generarán una serie de ficheros: controlador, entidad, formularios, vistas Twig, etc.

```
created: src/Controller/ResetPasswordController.php
created: src/Entity/ResetPasswordRequest.php
updated: src/Entity/ResetPasswordRequest.php
created: src/Repository/ResetPasswordRequestRepository.php
updated: src/Repository/ResetPasswordRequestRepository.php
updated: config/packages/reset_password.yaml
created: src/Form/ResetPasswordRequestFormType.php
created: src/Form/ChangePasswordFormType.php
created: templates/reset_password/check_email.html.twig
created: templates/reset_password/email.html.twig
created: templates/reset_password/request.html.twig
created: templates/reset_password/reset.html.twig
```

3. Creamos y ejecutamos la migración requerida para actualizar la base de datos. A partir de la nueva entidad ResetPasswordRequest, se generará una nueva tabla para almacenar las solicitudes de reseteo de contraseña

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

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

4. Y ya podemos solicitar reseteo de contraseña en la ruta “app\_forgot\_password\_request”), en la URL <http://127.0.0.1:8000/reset-password>.

Se usará la configuración del correo electrónico ya establecida para enviar un enlace al usuario para que resetee su contraseña. Podemos añadir el enlace al formulario de login. Y podemos también personalizar a nuestro gusto las plantillas, etiquetas y mensajes de formularios y vistas.

#### **Algunos detalles y cuestiones para ampliar:**

1. Al solicitar el reseteo de contraseña no se indica si el usuario tiene o no cuenta en el sistema. Bien!
  2. Tras solicitar el reseteo se muestra en pantalla el mensaje: “This link will expire in 0 minutes.” y también se incluye en el correo electrónico enviado.
    - a. ¿Cómo se configura el tiempo y se muestra correctamente? ¿O eliminamos el mensaje?
  3. Se puede solicitar el reseteo de contraseña de un usuario aunque se esté logueado con otro.
- Para ampliar más: [Registration Form > Symfony 5 Security: Authenticators | SymfonyCasts](#)

## 6. Documentación

Para documentar la aplicación se crea en el proyecto la carpeta /doc en la que se irán añadiendo archivos variados que se incluirán en el repositorio.

Terminada la iteración actualizaremos el repositorio.

### Modelo

En esta iteración se ha añadido una base de datos para almacenar los usuarios de la aplicación. A continuación se muestra un diagrama de clases resumido con la entidad User y sus atributos y métodos más interesantes, sin añadir por ahora tipos de datos ni getters y setters para mantenerlo simple. Tampoco se añade la entidad creada para resetear la contraseña.



Diagrama UML de clases/entidades

Este diagrama ha sido creado con <https://app.diagrams.net/>. Se añade a la carpeta de documentación tanto el archivo fuente como una imagen PNG exportada, resultados de este iteración.

### Rutas y URLs

En esta iteración se han añadido las siguientes rutas a las que ya tenía la aplicación.

Ruta	URL
login	/login
logout	/logout
app_register	/register
app_verify_email	/verify/email
app_forgot_password_request	/reset-password
app_check_email	/reset-password/check-email
app_reset_password	/reset-password/reset/{token}

## Actualizar aplicación

Para actualizar los cambios de esta iteración en otro entorno de desarrollo ya configurado haremos *pull* para actualizar los cambios del repositorio y a continuación usaremos *composer* para instalar y/o actualizar las librerías que se han instalado en los apartados anteriores.

```
git pull
composer update
```

Ahora que la aplicación también dispone de base de datos tendremos que instalarla y configurarla.

También habrá que configurar el correo electrónico. Se recomienda hacerlo en una nueva copia local del archivo `.env.local`.

Por ejemplo, para crear y configurar base de datos y mail en Linux podría hacerse así:

```
# Creamos la BD
sudo mysql -u root -p

create database NOMBRE_BD;
CREATE USER USER_BD@'localhost' IDENTIFIED BY 'PASSWORD_BD';
GRANT ALL ON NOMBRE_BD.* TO USER_BD@'localhost';
exit;

# Importamos contenido de la BD del fichero .sql incluido en la carpeta
/doc
sudo mysql -u root -p NOMBRE_BD < doc/compartir01.sql

# Hacemos una copia local del archivo de configuración de la aplicación
cp .env .env.local

# Editamos DATABASE_URL (usuario, contraseña, base de datos y, si es
necesario, versión del gestor de BD instalado) y MAILER_DSN (usuario y
contraseña)
# Editamos
nano .env

# Editamos la configuración de base de datos y mail según el entorno
donde se realiza la instalación.
DATABASE_URL="mysql://USER_BD:PASSWORD_BD@127.0.0.1:3306/NOMBRE_BD?server
Version=mariadb-10.6.7&charset=utf8mb4"

MAILER_DSN=gmail://USERNAME:PASSWORD@default
```

# Iteración 2. Interfaz de usuario

En este apartado trabajaremos la interfaz de usuario buscando y aplicando al sitio web una plantilla HTML5 y CSS que se adapte a distintas resoluciones de pantalla (*responsive*). Adaptaremos los contenidos para obtener un sitio web presentable y agradable en el que colocar tanto las funcionalidades ya implementadas como las que se desarrollen en un futuro de manera fácilmente escalable.

## 1. Preparar plantilla base

- [Creating and Using Templates \(Symfony Docs\)](#)

1. Descargamos una plantilla gratuita. Por ejemplo: <https://html5up.net/editorial>

Para definir las vistas en nuestro proyecto usamos el motor de plantillas Twig que ya fue instalado anteriormente. Twig permite estructurar una página en bloques (title, body, stylesheets, javascripts, etc.) y definir aquellos comunes en una plantilla común (base.html.twig) de la que las demás pueden heredar, con `{% extends 'base.html.twig' %}`, y sobrescribir solo los bloques específicos.

Una primera tarea será por tanto utilizar el contenido de la plantilla descargada (por ejemplo el fichero index.html) para generar una nueva plantilla base.html.twig a nuestro gusto.

2. Copiar .index.html y editar base.html.twig
  - a. Eliminar bloques de código sobrantes, es decir, aquellos que incluye la plantilla original pero que no nos interesan para nuestro proyecto.
  - b. Crear bloques Twig básicos: title, stylesheets, javascripts, body
  - c. Editar textos y contenido básico a nuestro gusto.

La plantilla descargada incluye una carpeta “assets” con recursos que forman parte del diseño de la web como imágenes, CSS o JavaScript. Tenemos que copiar esta carpeta de **assets** en la carpeta **public** de nuestro proyecto. Para hacer referencia a estos recursos desde el código de las plantillas Twig utilizamos la función **asset()** que nos facilitará la generación de URLs adecuadas y que requiere la instalación del componente Symfony [Asset](#).

3. Copiar carpeta de assets de la plantilla descargada (e images si es necesario) a la carpeta public del proyecto.
4. Instalar el componente Symfony [Asset](#)

```
composer require symfony/asset
```

5. Editar los enlaces de assets en el fichero base.html.twig

```
{# Ejemplos de bloques y assets en base.html.twig #}  
...  
<title>{% block title %}Editorial by HTML5 UP{% endblock %}</title>
```

```

...
{% block stylesheets %}
{{ encore_entry_link_tags('app') }}
<link rel="stylesheet" href="{{ asset('assets/css/main.css') }}" />
{% endblock %}
...
{% block body %}{% endblock %}
...
{% block javascripts %}
{{ encore_entry_script_tags('app') }}
<!-- Scripts -->
<script src="{{ asset('assets/js/jquery.min.js') }}"></script>
<script src="{{ asset('assets/js/browser.min.js') }}"></script>
<script src="{{ asset('assets/js/breakpoints.min.js') }}"></script>
<script src="{{ asset('assets/js/util.js') }}"></script>
<script src="{{ asset('assets/js/main.js') }}"></script>
{% endblock %}

```

Tras estos pasos iniciales ya podríamos tener una plantilla funcional. Sin entrar en muchos detalles seguiremos haciendo algunos ajustes y mejoras en nuestro sitio web

## 2. Crear menú principal de usuario

Vamos a crear un menú que enlace a las páginas y funciones de usuario implementadas hasta ahora y que se presentará en todas las páginas del sitio.

```

<ul>
  <li><a href="{{ path('app_index') }}">Inicio</a></li>
  {% if is_granted('IS_AUTHENTICATED_FULLY') %}
    <li><a href="#"><strong>{{ app.user.email }}</strong></a></li>
    <li><a href="{{ path('logout') }}">Logout</a></li>
  {% else %}
    <li><a href="{{ path('login') }}">Login</a></li>
    <li><a href="{{ path('app_register') }}">Registro</a></li>
  {% endif %}
  <li><a href="{{ path('app_about') }}">About</a></li>
</ul>

```

Como se puede ver, Twig permite utilizar sentencias condicionales para mostrar un contenido u otro en función de si se cumple o no una condición. En nuestro caso, si el usuario ha iniciado sesión se muestra su email y un enlace para cerrar sesión (logout). Si aún no ha iniciado sesión se muestra el enlace para hacerlo o para registrarse. Fuera del condicional se muestra un enlace a la página de inicio y otro a la de About.

Para generar las URL a rutas de la propia aplicación usamos la función **path()** indicando el nombre de la ruta.

Para termina, el código de este menú podemos colocarlo en el lugar adecuado de la plantilla base.html.twig o, mejor aún, podemos colocarlo en un fichero aparte, por ejemplo **templates/\_menu.html.twig**, e incluirlo en la plantilla en el lugar deseado con la función **include()**:

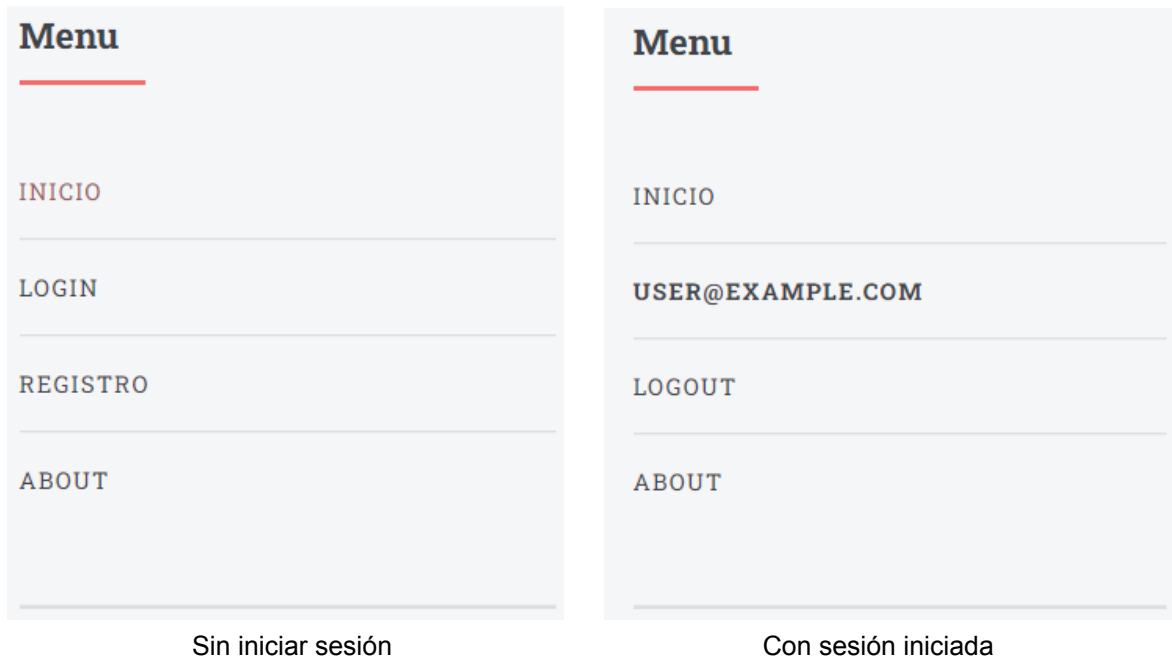
```

{{ include('_menu.html.twig') }}

```

De este modo será más reutilizable y nos será más sencillo modificarlo en el futuro. En Twig se recomienda que los nombres de ficheros que no sean plantillas completas sino solo fragmentos comiencen por el carácter subrayado bajo (“\_”).

Nuestro menú se verá así:



### 3. Añadir un Favicon

- [Adding a Favicon](#)

Los navegadores modernos suelen pedir un favicon.ico al servidor al tiempo que se carga una web. Este icono se utiliza para ilustrar la barra de título del navegador y la entrada del sitio web cuando lo guardamos en los marcadores.

Además de la cuestión estética, si no existe un favicon el servidor devolverá un error 404 que también registrará en el log del sitio web.

```
URL de la solicitud: http://127.0.0.1:8000/favicon.ico
Método de la solicitud: GET
Código de estado: 404 Not Found
Dirección remota: 127.0.0.1:8000
```

Para evitar esto añadiremos un favicon a la carpeta public del proyecto. Por ahora utilizaremos el favicon de Symfony accesible en <https://symfony.com/favicon.ico> aunque luego podemos cambiarlo por algo más apropiado a nuestro proyecto.





## 4. Arreglar formulario de registro y otras plantillas

Una vez adaptada la plantilla base, revisamos las plantillas de las páginas individuales, aquellas otras de la carpeta templates, para asegurarnos de que heredan la plantilla base cuando sea necesario (no lo es, por ejemplo, en las plantillas que generan contenido para enviar por correo electrónico: registro y resetear contraseña) y que se muestran correctamente, eliminando código innecesario y haciendo los ajustes que correspondan para que el sitio web luzca presentable.

Un detalle especial que podemos percibir es que en el formulario de registro no aparece el recuadro del checkbox para aceptar los términos de servicio de la web. Si miramos el código fuente de la página podemos ver que el código HTML está ahí, pero no se muestra. Si el usuario no puede marcar la casilla tampoco podría registrarse.

Una manera de arreglar esto es modificando la configuración de Twig para que genere la salida de los formularios en un formato “Bootstrap 5-friendly”. Para ello añadimos el código resaltado en la configuración de Twig:

config/packages/twig.yaml

```
twig:
    form_themes:
        - bootstrap_5_layout.html.twig
```

Más detalles en:

- [Registration Form > Symfony 5 Security: Authenticators | SymfonyCasts](#)

## 5. Condiciones de Uso

En el anterior apartado arreglamos el checkbox para que el usuario acepte los términos de uso al registrarse pero... aún no disponemos de esa página, así que vamos a aprovechar para crearla aquí:

1. Añadimos un nuevo método en **src\Controller\IndexController.php**

```
#[Route('/terms', name: 'app_terms')]
public function terms(): Response
{
    return $this->render('index/terms.html.twig');
}
```

Indicamos la URL, el nombre de la ruta y la plantilla Twig que se renderizará.

2. Creamos la plantilla twig correspondiente **templates/index/terms.html.twig**

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

{% block title %}Términos de uso{% endblock %}

{% block body %}

    <h1>Política de privacidad</h1>
    <ul>
        <li>¿Quién es el responsable del tratamiento de datos?</li>
        <li>¿Con qué finalidades tratamos tus datos?</li>
        <li>¿Qué datos personales tratamos?</li>
        <li>¿Qué hacemos para proteger los datos?</li>
        <li>¿Qué derechos puedes ejercer sobre tus datos?</li>
    </ul>

    <h1>Aviso legal</h1>
    <p>...<p>

{% endblock %}
```

En una página sencilla como ésta tan sólo heredamos la plantilla base y redefinimos el contenido de los bloques `title` y `body`.

3. Comprobamos el resultado en el navegador:



4. Por último, colocamos un enlace a la página en el menú o donde consideremos apropiado

```
<a href="{{ path('app_terms') }}">Términos de uso</a>
```

## 6. Documentación

### Rutas y URLs

En esta iteración se ha añadido la siguiente ruta a las que ya tenía la aplicación.

Ruta	URL
<b>app_terms</b>	<b>/terms</b>

### Vistas y plantillas

En esta iteración se han creado o revisado las siguientes plantillas en la carpeta /templates:

Plantilla	Uso
base.html.twig	Plantilla base que contiene la vista general de la web y de la que heredan todas las plantillas que muestran páginas.
_menu.html.twig	Menú de usuario incluido en la plantilla base.
index\index.html.twig	Portada de la web.
index\about.html.twig	Descripción del proyecto y del sitio web.
index\terms.html.twig	Términos de uso.
login\index.html.twig	Formulario de login.
registration\register.html.twig	Formulario de registro de nuevo usuario.
registration\confirmation_email.html.twig	Email de confirmación de registro enviado al usuario.
reset_password\request.html.twig	Formulario de solicitud de reseteo de contraseña.
reset_password\check_email.html.twig	Aviso para consultar el mail para resetear la contraseña.
reset_password\email.html.twig	Email con el enlace para resetear la contraseña.
reset_password\reset.html.twig	Formulario para establecer una nueva contraseña.

## Iteración 3. Perfil de usuario.

En esta iteración ampliaremos la entidad User para implementar un perfil de usuario básico añadiendo los siguientes atributos:

- **username**: Nombre para mostrar. Debe ser único. Si no existe se mostrará el correo electrónico
- **createdAt**: fecha/hora de creación del usuario.
- **updatedAt**: fecha/hora de última modificación.
- **lastLogin**: fecha/hora del último inicio de sesión del usuario.
- **enabled**: usuario habilitado o no.
- **photoFilename**: Imagen de perfil que el usuario puede subir

Un usuario podrá consultar la información de su perfil de usuario, modificar algunos de sus atributos como username, email, photoFilename (otros atributos como createdAt, updatedAt y lastLogin se modificarán automáticamente cuando corresponda), e incluso borrar el usuario completamente.

Además, a lo largo de esta iteración:

1. se creará un “escuchador/suscriptor” de eventos para detectar cuando un usuario inicia sesión con éxito y escribir código en respuesta.
2. Se reutilizará la funcionalidad de verificación de correo electrónico incluida en el formulario de registro.
3. Se habilitará la aplicación para subir ficheros de los usuarios.

Un perfil de usuario podría necesitar almacenar muchos otros datos según las necesidades del proyecto concreto a desarrollar. Por ejemplo, posteriormente se podrían añadir otros datos personales:

- Nombre y apellidos
- DNI
- Fecha Nacimiento
- Dirección
- Código Postal
- Concello
- Provincia
- Carnet de conducir
- etc.

# 1. Modelo. Añadir campos a User

Para ampliar la entidad User usamos el comando make de Symfony.

```
# Instalamos el componente Maker si no está instalado ya
composer require --dev symfony/maker-bundle

# Ampliamos la entidad User con el comando make
symfony console make:entity User // Añadir campos..
```

username	string (50)	null
createdAt	datetime	not null
updatedAt	datetime	null
lastLogin	datetime	null
enabled	boolean	not null

En el apartado 5 se añadirá otro campo para implementar la foto de perfil de usuario.

En el caso del campo **username** queremos que tenga un valor único, lo que podemos expresar en la entidad User modificada src\Entity\User.php:

```
#[UniqueEntity(fields: ['username'], message: 'There is already an
account with this username')]
class User implements UserInterface, PasswordAuthenticatedUserInterface
//...

#[ORM\Column(type: 'string', length: 50, nullable: true, unique: true)]
private $username;
```

Podemos aprovechar y añadir un constructor a la clase **User** para inicializar algunos de estos campos:

```
public function __construct()
{
    // Inicializa la fecha creación al momento actual
    $this->createdAt = new \DateTimeImmutable();
    // El nuevo usuario está activo por defecto
    $this->enabled = true;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';
}
```

Para ampliar:

- [UniqueEntity \(Symfony Docs\)](#)
- [Unique \(Symfony Docs\)](#)

- [Access Control \(Authorization\)](#)

Una vez completada la entidad, generamos la migración para actualizar la estructura de la base de datos:

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

Si ejecutamos la migración ahora obtendremos un error, en caso de que la tabla **User** contenga datos anteriores, ya que el campo **createdAt** que no puede ser nulo en esos registros.

Lo podemos arreglar editando el SQL de la migración generada (último archivo generado en la carpeta migrations) añadiendo valores por defecto en los nuevos campos NOT NULL. Así quedaría la consulta SQL editada:

```
ALTER TABLE user
ADD username VARCHAR(50) DEFAULT NULL,
ADD created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT
\'(DC2Type:datetime_immutable)\',
ADD updated_at DATETIME DEFAULT NULL COMMENT
\'(DC2Type:datetime_immutable)\',
ADD last_login DATETIME DEFAULT NULL COMMENT
\'(DC2Type:datetime_immutable)\',
ADD enabled TINYINT(1) NOT NULL DEFAULT 1,

CREATE UNIQUE INDEX UNIQ_8D93D649F85E0677 ON user (username)
```

En el otro campo NOT NULL, **enabled**, se asignará un uno (1) como valor por defecto.

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

## 2. Actualizar lastLogin y updatedAt

- [Events and Event Listeners \(Symfony Docs\)](#)
- [Security \(Symfony Docs\) Security Events](#)
- [Creating a Security Event Subscriber > Symfony 5 Security: Authenticators](#)

Para actualizar la fecha y hora del último acceso de un usuario es necesario capturar el evento que se produce cuando el *login* es exitoso y escribir el código necesario en respuesta.

1. Para ello decidimos crear una clase EventSubscriber con el comando make

```
symfony console make:subscriber
```

2. Le damos nombre y elegimos el evento que queremos escuchar

```

Choose a class name for your event subscriber (e.g. ExceptionSubscriber):
> LoginSubscriber

Suggested Events:
* Symfony\Component\Mailer\Event\MessageEvent (Symfony\Component\Mailer\Event\MessageEvent)
* Symfony\Component\Security\Http\Event\CheckPassportEvent (Symfony\Component\Security\Http\Event\CheckPassportEvent)
* Symfony\Component\Security\Http\Event>LoginSuccessEvent (Symfony\Component\Security\Http\Event>LoginSuccessEvent)
* Symfony\Component\Security\Http\Event\LogoutEvent (Symfony\Component\Security\Http\Event\LogoutEvent)
* console.command (Symfony\Component\Console\Event\ConsoleCommandEvent)
* console.error (Symfony\Component\Console\Event\ConsoleErrorEvent)
* console.terminate (Symfony\Component\Console\Event\ConsoleTerminateEvent)
* debug.security.authorization.vote (Symfony\Component\Security\Core\Event\VoteEvent)
* kernel.controller (Symfony\Component\HttpKernel\Event\ControllerEvent)
* kernel.controller_arguments (Symfony\Component\HttpKernel\Event\ControllerArgumentsEvent)
* kernel.exception (Symfony\Component\HttpKernel\Event\ExceptionEvent)
* kernel.finish_request (Symfony\Component\HttpKernel\Event\FinishRequestEvent)
* kernel.request (Symfony\Component\HttpKernel\Event\RequestEvent)
* kernel.response (Symfony\Component\HttpKernel\Event\ResponseEvent)
* kernel.terminate (Symfony\Component\HttpKernel\Event\TerminateEvent)
* kernel.view (Symfony\Component\HttpKernel\Event\ViewEvent)
* security.authentication.success (Symfony\Component\Security\Core\Event\AuthenticationSuccessEvent)
* security.interactive_login (Symfony\Component\Security\Http\Event\InteractiveLoginEvent)
* security.switch_user (Symfony\Component\Security\Http\Event\SwitchUserEvent)

What event do you want to subscribe to?:
> Symfony\Component\Security\Http\Event>LoginSuccessEvent

created: src/EventSubscriber/LoginSubscriber.php

Success!

```

- Como se indica, se habrá creado **src/EventSubscriber/LoginSubscriber.php** en el que ahora deberemos programar la respuesta al evento que consiste en actualizar el valor del atributo **lastLogin** del usuario actual:

```

public function onLoginSuccessEvent(LoginSuccessEvent $event): void
{
    // Obtiene el usuario que ha iniciado sesión
    $user = $event->getAuthenticatedToken()->getUser();
    // Actualiza la fecha de último acceso
    $user->setLastLogin(new \DateTimeImmutable());
    // Persiste el usuario en la base de datos
    $this->entityManager->flush();
}

```

Para el caso de actualizar el atributo **updatedAt** cuando se resetea la contraseña, basta con actualizar su valor en la función **reset()** del controlador **ResetPasswordController.php** justo antes de guardar el valor de la nueva contraseña en la base de datos con **flush()**.

```

$user->setPassword($encodedPassword);

// Actualiza la fecha de modificación
$user->setUpdatedAt(new \DateTimeImmutable());

$this->entityManager->flush();

```

Para impedir el inicio de sesión a los usuarios deshabilitados (`enabled = 0`) podemos ampliar el suscriptor de eventos anterior para capturar el momento en el que el usuario ya ha enviado el formulario de login pero aún no se ha autenticado (`CheckPassportEvent`). Si el usuario no está habilitado podemos lanzar una excepción

```
public function onCheckPassportEvent(CheckPassportEvent $event): void
{
    // Obtiene el usuario que intenta iniciar sesión
    $user = $event->getPassport()->getUser();
    // Si el usuario no está habilitado lanzamos una excepción
    if (!$user->isEnabled()){
        throw new AuthenticationException('Usuario deshabilitado');
    }
}
```

### 3. Perfil de usuario: mostrar, editar y borrar

Usando el comando `make` de Symfony podemos generar una [interfaz CRUD](#) básica para una entidad Doctrine dada. Es útil para implementar rápidamente algunas funcionalidades.

```
# Usamos el comando make:crud para generar el código necesario
(controlador, formularios y plantillas) para listar, crear, mostrar, editar
y borrar objetos de la entidad especificada (User en este caso)
```

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

```
The class name of the entity to create CRUD (e.g. OrangeGnome):
> User

Choose a name for your controller class (e.g. UserController) [UserController]:
>

Do you want to generate tests for the controller?. [Experimental] (yes/no) [no]:
>

created: src/Controller/UserController.php
created: src/Form/UserType.php
created: templates/user/_delete_form.html.twig
created: templates/user/_form.html.twig
created: templates/user/edit.html.twig
created: templates/user/index.html.twig
created: templates/user/new.html.twig
created: templates/user/show.html.twig
```

```
Success!
```

Basándonos en el código generado implementamos las funcionalidades de mostrar, editar y borrar perfil de usuario en las siguientes nuevas rutas (el resto de métodos generados podemos descartarlos por ahora):



Ruta	Método	URL	Control Acceso
app_user_profile_show	GET	/profile	Usuario actual
app_user_profile_edit	GET POST	/profile/edit	Usuario actual
app_user_profile_delete	POST	/profile/delete	Usuario actual

#### 1. Problemas con Borrado de usuario

- Error (no borra) si el usuario tiene un solicitud de reseteo pendiente (claves foránea) => Solución: Retocar la clave foránea de la entidad generada para el reseteo de contraseña para que Borre en cascada si se elimina el usuario.

```
#[ORM\ManyToOne(targetEntity: User::class)]
#[ORM\JoinColumn(nullable: false, onDelete: 'CASCADE')]
private $user;
```

- [Attributes Reference - Doctrine Object Relational Mapper \(ORM\)](#)

- Tras borrar el usuario, surge un error en el menú de la plantilla base al preguntar si está autenticado. => Solución: antes de borrar el usuario hay que cerrar la sesión con el siguiente código:

```
// Cerrar sesión (Logout) => invalidate session
$request->getSession()->invalidate();
$tokenStorage->setToken();
```

- [How can one force logout a user in Symfony? - Stack Overflow](#)
- [Symfony authentication \(You cannot refresh a user from the EntityUserProvider\) - Stack Overflow](#)

2. Actualizamos el atributo **updatedAt** tras editar con éxito el usuario.
3. Editamos el menú lateral de la plantilla base para mostrar el apodo (**username**), si existe, en lugar del correo electrónico.

## 4. Modificar correo electrónico

- [Forms \(Symfony Docs\) -> Unmapped Fields](#)
- [Controller \(Symfony Docs\) -> Flash Messages](#)

Al modificar el correo electrónico el usuario debería verificarlo de nuevo.

1. Para comprobar si se ha modificado el correo electrónico introducimos en el formulario de editar perfil un [campo oculto no mapeado](#) que contendrá el valor original del correo electrónico.

```
$builder
    ->add('email')
```

```
->add('oldEmail', HiddenType::class, ['mapped' => false])
```

2. Una vez que se detecta el cambio se pone a falso el atributo de usuario verificado y se envía un nuevo mail de verificación usando el método `sendEmailConfirmation()` ya implementado con el formulario de registro.

```
if ($form->isSubmitted() && $form->isValid()) {  
    // Comprueba si se ha modificado el email  
    if ($user->getEmail() != $form->get('oldEmail')->getData()) {  
        // Cambia el estado de verificado a falso  
        $user->setIsVerified(false);  
        // Envía un email de confirmación al nuevo email  
        $rc->sendEmailConfirmation($user);  
        // e Informa al usuario con un mensaje flash  
        $this->addFlash('success', 'Se ha enviado un email de confirmación a  
tu nuevo email');  
    }  
}
```

3. Para enviar un nuevo mail de verificación se utiliza el código utilizado en el controlador de registro que se extrae a una nueva función para reutilizarlo sin duplicar código.

```
public function sendEmailConfirmation(User $user){  
    // generate a signed url and email it to the user  
    $this->emailVerifier->sendEmailConfirmation('app_verify_email', $user,  
    (new TemplatedEmail())  
        ->from(new Address('from@example.com', 'Proyecto Compartir Symfony'))  
        ->to($user->getEmail())  
        ->subject('Please Confirm your Email')  
        ->htmlTemplate('registration/confirmation_email.html.twig')  
    );  
}
```

4. Una vez enviado el mail utilizamos un [mensaje Flash](#) para informar al usuario de que se le ha enviado un mensaje para verificar el correo electrónico.

```
{# read and display success flash messages #}  
{% for message in app.flashes('success') %}  
    <div class="flash-success">  
        {{ message }}  
    </div>  
{% endfor %}
```

## 5. Añadir foto de perfil

- [How to Upload Files \(Symfony Docs\)](#)

- [Image \(Symfony Docs\) - Constraints](#)
- [The String Component \(Symfony Docs\) - Slugger](#)

Para implementar esta funcionalidad seguiremos, en líneas generales, la [documentación de Symfony sobre como subir archivos](#):

1. Añadimos un nuevo campo a la entidad **User** para almacenar el nombre del archivo de imagen. Creamos y ejecutamos la migración en la base de datos.

```
symfony console make:entity

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

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

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

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

Field length [255]:
>

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

php bin/console make:migration

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

2. Añadimos un nuevo campo al formulario src/Form/ProfileType.php

```
->add('photo', FileType::class, [
    'label' => 'Foto de perfil (gif, jpeg, png)',
    ...
])
```

3. Modificamos el controlador src/Controller/ProfileController.php para procesar la imagen.

```
if ($form->isSubmitted() && $form->isValid()) {
    $photoFile = $form->get('photo')->getData();
    // this condition is needed because field is not required
    // so file must be processed only when a file is uploaded
    if ($photoFile) {
        ...
    }
}
```

4. Para borrar el fichero de la foto de perfil anterior si existiese añadimos el siguiente código:

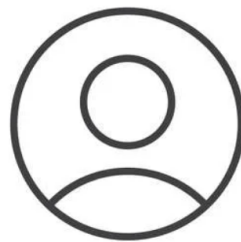
```
// Borra el fichero de imagen de perfil anterior si existe
if ($user->getPhotoFilename()) {
    $oldPhoto = $user->getPhotoFilename();
    $oldPhotoPath =
```

```
$this->getParameter('profile_photos_directory').'/'.$oldPhoto;
    if (file_exists($oldPhotoPath)) {
        unlink($oldPhotoPath);
    }
}
```

5. Añadimos el directorio al que se subirán los archivos (profile\_photos\_directory = public/images/profile\_photos) como un parámetro de la aplicación en el archivo de configuración **services.yaml**. Deberemos crear esa carpeta en la estructura del proyecto.

```
parameters:
    profile_photos_directory: '%kernel.project_dir%/public/images/profile_photos'
```

6. Añadimos en esa carpeta una imagen (default\_profile\_photo.PNG) que se mostrará en el perfil de los usuarios que no hayan subido su propia foto.



7. Actualizamos la plantilla de la página de perfil para mostrar la foto, si existe, o la imagen por defecto en caso contrario.

```
{% if user.photoFilename %}
    <p><a href="{{ asset('images/profile_photos/' ~ user.photoFilename) }}">
    
    </a></p>
{% else %}
    <p><a href="{{ path('app_profile_edit') }}">
    
    </a></p>
{% endif %}
```

8. Editamos .gitignore para que no se incluya en el repositorio ninguna de las fotos de perfil que se puedan usar en desarrollo, excepto la foto por defecto.

```
/public/images/profile_photos/*
!/public/images/profile_photos/default_profile_photo.PNG
```

## 6. Documentación

En esta iteración se ha modificado el modelo ampliando la entidad User y se han añadido nuevas rutas y plantillas.

### Modelo

User
- id
- email
- roles
- password
- isVerified
- username
- createdAt
- updatedAt
- lastLogin
- enabled
- photoFilename
+ getUserIdentifier()
+ eraseCredentials()
+ isVerified()

### Rutas y URLs

Ruta	Método	URL	Control Acceso
app_user_profile_show	GET	/profile	Usuario actual
app_user_profile_edit	GET POST	/profile/edit	Usuario actual
app_user_profile_delete	POST	/profile/delete	Usuario actual

### Vistas y plantillas

Plantilla	Uso
profile\_delete_form.html.twig	Formulario con el botón para borrar el usuario
profile\_form.html.twig	Formulario de perfil de usuario
profile\edit.html.twig	Editar perfil de usuario
profile\show.html.twig	Mostrar perfil de usuario

