

**INSTITUTO DE EDUCACIÓN SECUNDARIA  
I.E.S. CONSELLERIA -**

**FAMILIA PROFESIONAL DE INFORMÁTICA Y  
COMUNICACIONES**

**Desarrollo web en Symfony**

**PROYECTO DE DESARROLLO DE APLICACIONES WEB  
NÚMERO INF-14-21**

Autor: Cristina Esteban López

Tutor individual: Pilar Gimeno

## Resumen

Desarrollo de una aplicación web de recetas de cocina donde el usuario pueda acceder a recetas de distintas categorías, incluidas algunas relacionadas con dietas especiales donde no se puedan consumir algunos alimentos específicos como pueden ser: “celiacos”, “vegetarianos” o “veganos”.

Las recetas serán subidas a la aplicación por el propio usuario siempre y cuando se haya registrado e iniciado sesión. Una vez subida la receta, esta tendrá que ser validada antes por el administrador y una vez que este la verifique podrá ser publicada y visualizada en la web. Por lo tanto, existe también un panel de administrador accesible para los usuarios que tienen este rol.

Además los usuarios una vez inicien sesión podrán interactuar con ellas comentándolas y puntuándolas.

## **Abstract**

Development of a web application for cooking recipes where the user can access recipes from different categories, including some related to special diets where some specific foods cannot be consumed, such as: “celiac”, “vegetarian” or “vegan”.

The recipes will be uploaded to the application by the user himself as long as he has registered and logged in. Once the recipe is uploaded, it will have to be validated before by the administrator and once he verifies it, it can be published and viewed on the web. Therefore, there is also an administrator panel accessible to users who have this role.

In addition, users once they log in will be able to interact with them by commenting on them and rating them.

# ÍNDICE

1.	Introducción.....	1
1.1.	Justificación .....	1
1.2.	Contexto.....	1
1.3.	Objetivos.....	2
1.4.	Organización .....	2
2.	Planificación .....	3
2.1.	Symfony.....	3
2.1.1.	Estructura de directorios .....	4
2.2.	Tecnologías complementarias a Symfony .....	6
2.2.1.	ORM .....	6
2.2.2.	Doctrine .....	6
3.	Desarrollo del proyecto .....	7
3.1.	Desarrollo del back end .....	7
3.1.1.	Doctrine .....	7
3.1.2.	Symfony .....	9
3.1.3.	Sonata Admin Bundle .....	12
3.2.	Desarrollo del front end .....	14
3.2.1.	Motor de plantillas TWIG .....	14
3.3.	Aspectos técnicos.....	16
3.4.	Explicación del proyecto .....	17
3.4.1.	Base de datos .....	17
3.4.2.	Inicio.....	18
3.4.3.	Registro.....	19
3.4.4.	Login.....	19
3.4.5.	Explora .....	20
3.4.6.	Recetas de una categoría.....	20
3.4.7.	Detalle de la receta .....	21
3.4.8.	Panel de administrador .....	22
4.	Evaluación y conclusiones finales .....	23
5.	Bibliografía y referencias .....	24

# 1. Introducción

## 1.1. Justificación

Es una web que tiene como finalidad crear una plataforma donde poder acceder a multitud de recetas de cocina organizadas por categorías. Los usuarios son los que subirán sus propias recetas y podrán interactuar con las recetas mediante calificaciones y comentarios. El sistema de comentarios permitirá responder a comentarios ya registrados. Además existe un panel de administración donde únicamente podrán acceder los usuarios con rol de administrador. De esta forma, los administradores tendrán que revisar y verificar que las recetas son aptas para poder ser visualizadas en la web. Esta plataforma permite facilitar a cualquier persona descubrir nuevas recetas que introducir en su día a día, incluidas aquellas personas que siguen dietas especiales como pueden ser las personas celiacas, vegetarianas o veganas.

Entre las tecnologías utilizadas en el proyecto destacaría Symfony, tecnología investigada y usada en el desarrollo back end.

## 1.2. Contexto

Este proyecto está relacionado con varios de los módulos impartidos durante los dos cursos que forman el ciclo superior.

El módulo “Desarrollo en Entorno Servidor” tiene un gran papel en el proyecto. Ya que la mayor dificultad de este es la investigación y creación del back end mediante uno de los frameworks de PHP más populares en el mundo laboral del desarrollo web, Symfony. También al crear varias entidades y conseguir mediante el ORM Doctrine convertirlas en tablas conseguimos crear una base de datos relacional. Utilizando MySQL y PhpMyAdmin para administrar la base de datos, relacionamos el proyecto con el módulo “Gestión de Bases de Datos”. En relación al módulo de “Diseño de Interfaces Web”, el diseño de la web esta realizado mediante CSS y uno de sus frameworks, Bootstrap. Además también se usa JQuery haciendo referencia al módulo “Desarrollo en Entorno Cliente”.

Además, el framework Symfony fue la tecnología principal que aprendí y use en el modulo de FCT. La utilización de esta tecnología en el proyecto me ha servido para afianzar los conocimientos obtenidos en las prácticas y ponerlos en práctica.

### 1.3. Objetivos

- Creación de una web donde poder subir y acceder a recetas de cocina.
- Investigación y creación del back end mediante Symfony.
- Investigación y utilización de Doctrine en la creación de la base de datos y MySQL.
- Investigación y utilización de motor de plantillas Twig.
- Utilización de JavaScript y JQuery para conseguir cierta interactividad.
- Utilización de Git y Github para guardar las instantáneas del proyecto.

### 1.4. Organización

A continuación vamos a presentar y explicar las secciones principales que contiene la web:

- **Inicio:** la página de inicio será la presentación a la web donde se podrá visualizar un video de fondo con el eslogan de la plataforma “Disfruta de la cocina”. Además existirá una cabecera en la parte superior que enlazará con las páginas de registro e iniciar sesión. También podremos observar el logo de la web.
- **Registro:** en esta página el usuario podrá registrarse introduciendo los datos indicados en el formulario de registro.
- **Inicio de sesión:** en esta página podremos encontrar el formulario de inicio de sesión donde validará que las credenciales del usuario coinciden con las registradas en la base de datos.
- **Categorías:** en esta sección podremos visualizar todas las categorías alimentarias contempladas en la web. Una vez seleccionemos una de ellas, podremos acceder a las recetas asociadas a esa categoría.
- **Recetas de una categoría:** en esta página podremos ver todas las recetas asociadas a la categoría que hayamos seleccionado anteriormente. Tendrá enlaces a los detalles de cada receta Además tendremos un buscador de recetas para poder obtener las recetas que en el titulo tengan resultados parecidos a nuestra búsqueda, incluso si pertenecen a una categoría distinta.
- **Receta:** en esta página encontraras más detalles sobre la receta: ingredientes y pasos para cocinarla. También se podrá ver los comentarios de las recetas.

## 2. Planificación

A continuación vamos a hablar de Symfony y sus tecnologías complementarias usadas en este proyecto, las cuales serían las principales y más destacables.

### 2.1. Symfony

Symfony es un conjunto de librerías que se utilizan para el desarrollo de aplicaciones web PHP. Actualmente, Symfony es uno de los frameworks de PHP más utilizados por las empresas de desarrollo web de todo el mundo. Se utiliza bajo licencia MIT y por lo tanto se trata de software libre.

Además cuenta con una gran comunidad activa en el desarrollo y mantenimiento del framework. Es por eso que grandes proyectos como Drupal han sido creados a partir de Symfony.

Como ventajas principales destacarían las siguientes:

- Sistema flexible que permite usar las dependencias que quieras usar en tu proyecto.
- Sistema rápido que consume la mitad de memoria comparado con otros frameworks.
- Estabilidad: Symfony por cada versión estable da soporte y proporciona corrección de errores, de tal forma que nos da una gran seguridad a la hora de crear un proyecto profesional.
- Gran comunidad y documentación oficial que permite aprender Symfony con mayor rapidez.

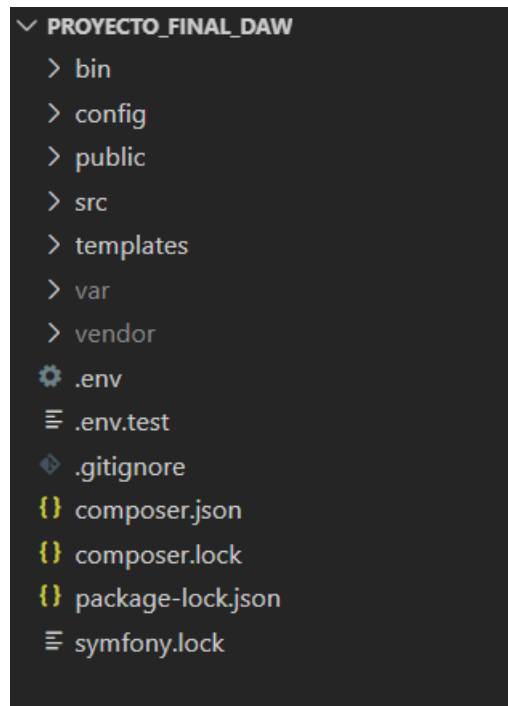
Como principales desventajas destacarían las siguientes:

- La curva de aprendizaje de Symfony es mayor frente a la de otros frameworks como Laravel.
- Constantes cambios entre versiones como pueden ser cambios de estructura de directorios y comandos de automatización de tareas.

### 2.1.1. Estructura de directorios

Como hemos hablado anteriormente una de las grandes desventajas de Symfony son los cambios que hacen en la estructura de directorios entre distintas versiones. El mayor cambio se presentó entre el salto de Symfony 3 a Symfony 4.

Este proyecto esta creado en Symfony 4.4 y tiene la siguiente estructura.



*Figura 1: Estructura de directorios del proyecto*

Una vez creamos un proyecto en Symfony 4 por defecto se crean en la ruta raíz del proyecto las carpetas bin, config, public, src, templates, tests, var y vendor.

- **Bin:** directorio utilizado para la entrada de comandos desde la línea de comandos. Dentro de él se encuentra el archivo “console”.
- **Config:** directorio en el que podemos encontrar una multitud de archivos de configuración necesarios enter otras cosas para poder conectar a la base de datos, encontrar rutas, importar bundles (plugins o paquetes externos a nuestro proyecto) a nuestro proyecto o añadir servicios.
- **Public:** es el directorio raíz de la web donde se almacenan todos los recursos HTTP dinámicos, como son las imágenes y los ficheros de JavaScript y CSS.



- **Src:** este sería el directorio más importante donde se ubica todo el código de nuestra aplicación. En esta carpeta encontraremos las clases, los controladores, los repositorios y los formularios.
- **Templates:** directorio donde se encuentran las plantillas de Twig del proyecto.
- **Var:** Contiene los archivos logs y cache generados en tiempo de ejecución.
- **Vendor:** Directorio donde se ubican los paquetes y librerías instaladas, incluso el mismo Symfony. El directorio es administrado con Composer, herramienta que nos permite importar e eliminar dependencias de nuestro proyecto.

Sin embargo la estructura de Symfony 3 presenta grandes diferencias:

- Existe un nuevo directorio llamado “app” donde se encuentran dos directorios importantes, Resources y config. En Resources se encuentran los recursos (las vistas) y en config los archivos de configuración.
- Existe otro nuevo directorio, web. El cual en la versión 4 de Symfony sería similar al directorio public, donde los archivos son accesibles. Aquí se encontrarías los archivos media (videos e imágenes) y archivos JavaScript y CSS.

Symfony 4 cambia la estructura de directorios con la finalidad de organizar mejor el código y hacerlo más accesible. Una de las diferencias que más destacable es la ubicación de la carpeta config. Esta se hace más visible y se le da más importancia ubicándola en la ruta raíz del proyecto. Lo mismo sucede con las vistas de la aplicación, que pasa a tener un primer plano al ubicarse en la ruta raíz.

## 2.2. Tecnologías complementarias a Symfony

### 2.2.1. ORM

Un ORM (Object Relational Mapper) es un modelo de programación que nos permite convertir los datos de los objetos en un formato correcto que nos permita guardarlos en una base de datos relacional. A esta acción comúnmente se le llama “mapeo”. De esta forma, las clases serían las tablas y los registros serían los objetos. El resultado es una base de datos orientada a objetos. De esta manera resulta más sencillo trabajar con los registros y las tablas.

### 2.2.2. Doctrine

Doctrine es una librería PHP que tiene su propio ORM y proporciona una capa de persistencia de objetos. Este es el ORM utilizado internamente por Symfony por defecto y el utilizado en este proyecto. Doctrine puede conectarse a las bases de datos más usadas como pueden MySQL, PostgreSQL y Microsoft SQL.

El uso de Doctrine es muy recomendable ya que al permitirnos trabajar con clases y objetos conseguimos desarrollar un código escalable y mantenible. Además acceder a la base de datos y leerla directamente es considerado una mala práctica ya que es posible tener problemas de SQL Injection. Por lo tanto usar Doctrine nos ayudaría a prevenir este tipo de ataques.

Una de las grandes ventajas de Doctrine es la reutilización. Doctrine permite llamar a los métodos de un objeto de datos desde varios lugares de la aplicación, incluso desde varias aplicaciones.

Doctrine tiene definidas varias funciones que nos facilitan obtener los datos de la base de datos:

- **findAll():** obtiene todos los registros de una tabla.
- **find():** obtiene un registro pasándole por parámetro una clave primaria.
- **findBy():** obtiene los registros pasándole como parámetros los valores de la cláusula WHERE.
- **findOneBy():** obtiene un único registro pasándole como parámetros los valores de la cláusula WHERE.

## 3. Desarrollo del proyecto

### 3.1. Desarrollo del back end

#### 3.1.1. Doctrine

Como anteriormente hemos comentado nuestro proyecto usa como tecnología principal Symfony para el desarrollo del back end. Además nos ayudaremos de Doctrine para poder nuestro modelo, crear la base de datos y trabajar con los datos.

Lo primero de todo es indicar en el archivo **.env** donde está ubicada la base de datos.

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

*Figura 2: Configuración de parámetros para acceder a la BD.*

Las clases de las entidades están acompañadas de anotaciones. Las anotaciones son metadatos que preprocesa Doctrine antes de trasladar los datos a la base de datos. Estas anotaciones nos aportan información más detallada sobre una entidad, su clase y propiedades.

```
/**
 * @ORM\Column(type="string", length=255)
 */
private $username;
```

*Figura 3: Anotación para la propiedad username.*

A continuación podemos observar una de las entidades del proyecto con sus respectivas anotaciones. Todas las entidades comienzan indicando un *namespace*. Un *namespace* es nos indica el directorio de una clase. Son obligatorios en todos los ficheros de Symfony donde exista una clase.

```

<?php

namespace App\Entity;

use App\Repository\CategoryRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=CategoryRepository::class)
 */
class Category
{
    public function __construct() {
        $this->recipe = new \Doctrine\Common\Collections\ArrayCollection();
    }

    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=false)
     */
    private $name;

    /**
     * @ORM\ManyToMany(targetEntity="Recipe", mappedBy="categories")
     */
    private $recipe;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getName(): ?string
    {
        return $this->name;
    }

    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }
}

```

*Figura 4: Consulta DQL que obtiene las recetas visibles de una categoría.*

Como podemos observar mediante las anotaciones no solo indicamos datos relacionados con las propiedades, también podemos crear las relaciones entre las futuras tablas. Además, todas las entidades tienen su propio repositorio desde donde se realizarán las consultas DQL (Doctrine Query Language), lenguaje propio de Doctrine para hacer consultas a la base de datos.

```

public function findRecipesByCategory($category)
{
    return $this->createQueryBuilder('r')
        ->innerJoin('r.categories', 'categories')
        ->andWhere('r.visible = 1')
        ->andWhere('categories.name = :cat')
        ->setParameter('cat', $category)
        ->getQuery()
        ->getResult();
}

```

*Figura 5: Consulta DQL que obtiene las recetas visibles de una categoría.*

Una vez configuradas las entidades es el momento de trasladar toda esa información a la base de datos para poder crear las tablas con sus respectivas propiedades y relaciones. Para ello Doctrine tiene el siguiente comando:

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

*Figura 6: Comando para trasladar a la BD los datos.*

Una vez que queramos modificar algún dato en las entidades únicamente tendremos que actualizar mediante el siguiente comando:

```
php bin/console doctrine:schema:update --force
```

*Figura 7: Comando para actualizar entidades a la BD.*

### 3.1.2. Symfony

Una vez tengamos el modelo con las entidades y la base de datos creada, podremos empezar a crear los controladores y formularios propios de Symfony.

Los controladores estarán compuestos por una clase la cual contendrá los métodos que hacen referencia al controlador de esa entidad.

```

namespace App\Controller;

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

class ExplorerController extends AbstractController
{
    /**
     * @Route("/explora", name="explorer")
     */
    public function index(CategoryRepository $categoryRepository): Response
    {
        return $this->render('explorer/explorer.html.twig', [
            'categories' => $categoryRepository->findAll()
        ]);
    }
}

```

*Figura 8: Controlador Explorer*

En el caso de la imagen podemos observar el controlador Explorer con su función index, la cual pasándole como parámetro el repositorio de las categorías es capaz de obtener todas las categorías y mandárselas a la vista *explorer.html.twig*.

Las funciones como podemos observar en la imagen pueden tener anotaciones en las cuales se indica la ruta desde la cual podemos acceder a la función y el nombre de la ruta para poder llamarla desde otras partes del código.

```

<a href="{{path('recipe_show', {'title': recipe.title })}}" class="d-flex justify-content-center text-decoration-none">
    <button type="button" class="button-steelblue btn-sm">Ver más</button>
</a>

```

*Figura 9: Ejemplo de llamada a ruta de un controlador*

En el ejemplo podemos observar como llamando al nombre de la ruta *recipe\_show*, que necesita como argumento el titulo de la receta, podemos acceder a la función del controlador.

Por último, indicar que Symfony tiene sus propios formularios. Los formularios tienen su propia clase que siempre debe terminar en Type. Aquí podemos observar un ejemplo de uno de los creados en la aplicación.

```
class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('email', EmailType::class, [
                'constraints' => [
                    new NotBlank([
                        'message' => 'Por favor introduce un email',
                    ]),
                ],
            ])
            ->add('username', TextType::class, [
                'label' => 'Nombre de usuario',
                'constraints' => [
                    new NotBlank([
                        'message' => 'Ya existe ese nombre de usuario',
                    ]),
                ],
            ])
            ->add('password', RepeatedType::class, [
                'type' => PasswordType::class,
                'invalid_message' => 'Las contraseñas no coinciden',
                'options' => ['attr' => ['class' => 'password-field']],
                'required' => true,
                'first_options' => ['label' => 'Contraseña'],
                'second_options' => ['label' => 'Repita contraseña'],
                'constraints' => [
                    new NotBlank([
                        'message' => 'Por favor introduce una contraseña',
                    ]),
                    new Length([
                        'min' => 6,
                        'minMessage' => 'Tu contraseña debe tener al menos {{ limit }} caracteres',
                        'max' => 4096,
                    ]),
                ],
            ])
        ];
    }
}
```

*Figura 10: Ejemplo de formulario*

Como se puede observar los formularios Symfony nos permiten hacer comprobaciones antes de validar un formulario. De tal forma que nos permite comprobar si el campo esta vacío o tiene una longitud mínima entre otras comprobaciones.

En este proyecto se usa Bootstrap, que ya por defecto puede comprobar si un campo esta vacío o no. Sin embargo, es recomendable hacer igualmente estas comprobaciones en los formularios Symfony y no fiarse únicamente de las validaciones que puede hacer Bootstrap en la parte de front end. Además Symfony ofrece un listado de comprobaciones que pueden ser muy interesantes como puede ser comprobar que el campo contraseña y repetir contraseña coinciden.

Una vez tengamos el formulario creado hay que llamarlo. Para ello nos ubicaremos en la función del controlador que estemos interesados y lo crearemos de la siguiente manera.

```
$user = new User();  
$form = $this->createForm(RegistrationFormType::class, $user);
```

*Figura 11: Creando un formulario en el controlador.*

Una vez creado comprobaremos si se ha pulsado el botón del formulario y si este es válido. En caso de ser así, el objeto al que haga referencia el formulario se rellenará o editará con nuevos datos o se eliminará. Finalmente, usaremos el método `persist()` para guardar el objeto y `flush()` para que se ejecuten los cambios.

```
if ($form->isSubmitted() && $form->isValid()) {  
    $user->setPassword(  
        $passwordEncoder->encodePassword(  
            $user,  
            $form->get('password')->getData()  
        )  
    );  
  
    $user->setUsername($user->getUsername());  
    $user->setRoles(["ROLE_USER"]);  
  
    $entityManager = $this->getDoctrine()->getManager();  
    $entityManager->persist($user);  
    $entityManager->flush();  
  
    $this->addFlash('success', '¡Usuario registrado correctamente!');  
}
```

*Figura 12: Comprobando si un formulario es válido.*

### 3.1.3. Sonata Admin Bundle

Sonata Admin Bundle es uno de los bundles de Symfony más populares que te permiten crear un panel de administración de forma rápida y eficaz. Un bundle es un conjunto de archivos estructurados que implementan una o un conjunto de funcionalidades. Sonata funciona mediante el ORM de Doctrine. De forma que mediante las entidades consigue listar los registros y crear varios CRUDs que nos permiten eliminar, crear y editar registros de nuestra base de datos. Además nos permite filtrar mediante las propiedades y buscar registros. También nos permite exportar los datos en varios formatos como pueden ser JSON o CSV.



Id	Email	Roles	Contraseña	Nombre de usuario	Acciones
1	cristina@gmail.com	[0 => ROLE_USER]	\$2y\$13SF22W8dv4564T7333nkYPOUk070GFhoQ2rj/mlXcUMnclA9Yv/BJ2	Cristina	Mostrar Editar Borrar
2	admin@gmail.com	[0 => ROLE_ADMIN]	\$2y\$13SF22W8dv4564T7333nkYPOUk070GFhoQ2rj/mlXcUMnclA9Yv/BJ2	Cristina	Mostrar Editar Borrar
3	juan@gmail.com	[0 => ROLE_USER]	\$2y\$13SWKIQYXzbAl5NWWct7auuWhr1pGU2Fqb3UdDk0SQmZoduUD.f/u	Juan	Mostrar Editar Borrar

Figura 13: Listado de registros de la entidad Usuarios

Editar "admin@gmail.com"

Acciones

Usuarios

Email\*

admin@gmail.com

Roles\*

ROLE\_ADMIN

Contraseña\*

Repite contraseña\*

Nombre de usuario\*

Cristina

Actualizar Actualizar y cerrar Borrar

Figura 14: Edición de un registro de la tabla usuario

Mostrar "cristina@gmail.com"

Acciones

Usuarios

Email

cristina@gmail.com

Roles

0 => ROLE\_USER

Contraseña

\$2y\$13SF22W8dv4564T7333nkYPOUk070GFhoQ2rj/mlXcUMnclA9Yv/BJ2

Nombre de usuario

Cristina

Figura 15: Mostrando los datos de un usuario

## 3.2. Desarrollo del front end

### 3.2.1. Motor de plantillas TWIG

Twig es un potente motor de plantillas diseñado para PHP que nos ayuda a mantener las vistas mas ordenadas y limpias. Al crear un proyecto en Symfony viene integrado Twig, sin embargo no tienes porque trabajar con este motor de plantillas. Sin embargo son tantas las ventajas de trabajar con TWIG que realmente es muy recomendable implementar su uso en proyectos con Symfony. Entre estas ventajas podemos destacar las siguientes:

- Extender de plantillas base. Estas plantillas contienen la estructura de las páginas web dividida en bloques, de tal forma que al extender de estas plantillas podamos rellenar los bloques con la información que queramos visualizar en cada página. En caso de no indicar en la plantilla uno de los bloques definidos en la plantilla base, ese bloque se implanta automáticamente de la plantilla base de la que se extiende.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Test Application{% endblock %}</title>
  </head>
  <body>
    <div id="sidebar">
      {% block sidebar %}
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/blog">Blog</a></li>
        </ul>
      {% endblock %}
    </div>
    <div id="content">
      {% block body %}{% endblock %}
    </div>
  </body>
</html>
```

*Figura 16: Ejemplo de plantilla base.*

- Rapidez: Twig es un sistema muy rápido ya que compila las plantillas a código PHP optimizado.
- Flexibilidad: Twig tiene por defecto ya definidas multitud de expresiones y etiquetas que nos ayudan a visualizar los datos que queremos en una plantilla.

Para poder visualizar una variable en Twig simplemente necesitaremos pasársela como parámetro a la plantilla y visualizarla con la siguiente sintaxis:

```
{{ recipe.ingredients }}
```

*Figura 17: Visualización de variable en Twig.*

En caso de no existir la variable se recibirá un valor nulo y Twig lanzará un error a la hora de intentar mostrar la plantilla. Twig permite definir variables y asignarle valores mediante la etiqueta “set”.

```
{% set foo = 'foo' %}  
{% set foo = [1, 2] %}  
{% set foo = {'foo': 'bar'} %}
```

*Figura 18: Asignación de valores en variable definida en Twig.*

Twig también cuenta con filtros que nos permiten modificar variables mediante el símbolo “|”.

```
{{comment.createdAt | date("d/m/Y H:i")}}
```

*Figura 19: Filtro de Twig para formatear una fecha.*

Por si todo esto fuera poco para recomendar Twig, este permite estructuras de control y comparaciones en sus plantillas.

```
{% if users|length > 0 %}  
  <ul>  
    {% for user in users %}  
      <li>{{ user.username|e }}</li>  
    {% endfor %}  
  </ul>  
{% endif %}
```

*Figura 20: Ejemplo de bucle “for” y comparación con “if”.*

### 3.3. Aspectos técnicos

En este apartado encontraremos las herramientas utilizadas en el proyecto explicándolas brevemente.

Herramientas	Descripciones	Enlaces
Symfony	Framework de PHP para desarrollar aplicaciones web.	<a href="https://symfony.com">https://symfony.com</a>
Doctrine	ORM que usa Symfony por defecto para trabajar con bases de datos MySQL.	<a href="https://www.doctrine-project.org">https://www.doctrine-project.org</a>
Sonata Admin Bundle	Herramienta que nos permite crear un panel de administrador.	<a href="https://sonata-project.org">https://sonata-project.org</a>
CKEditor	Editor de texto para campos de formulario.	<a href="https://ckeditor.com/">https://ckeditor.com/</a>
JavaScript/ Query	Lenguaje de programación que nos permite dar interactividad a la web.	<a href="https://jquery.com">https://jquery.com</a>
CSS	Lenguaje de estilos que permite crear el diseño de la web.	<a href="https://developer.mozilla.org/es/docs/Web/CSS">https://developer.mozilla.org/es/docs/Web/CSS</a>
Twig	Motor de plantillas para el lenguaje de programación PHP.	<a href="https://twig.symfony.com">https://twig.symfony.com</a>
Bootstrap	Framework de CSS que permite aplicar a nuestro código clases ya diseñadas.	<a href="https://getbootstrap.com">https://getbootstrap.com</a>
Git	Software de control de versiones que permite guardar instantáneas de los proyectos.	<a href="https://git-scm.com">https://git-scm.com</a>

Github	Plataforma web que permite gestionar proyectos basándose en el control de versiones de Git.	<a href="https://github.com">https://github.com</a>
MySQL	Sistema de gestión de base de datos relacionales.	<a href="https://www.mysql.com">https://www.mysql.com</a>
PhpMyAdmin	Herramienta para administrar la administración de MySQL.	<a href="https://www.phpmyadmin.net">https://www.phpmyadmin.net</a>
Visual Studio Code	Editor de código de código abierto y gratuito.	<a href="https://code.visualstudio.com">https://code.visualstudio.com</a>

### 3.4. Explicación del proyecto

Lo primero que tenemos que tener claro antes de empezar el proyecto es la organización de la estructura que va a tener. Para ello es necesario tener una idea clara de que entidades tendrá el software y que relaciones habrá entre ellas. Una vez tengamos eso claro crearemos la base de datos, la cual tendrá un papel fundamental ya que creará la estructura del proyecto.

#### 3.4.1. Base de datos

La base de datos está compuesta por seis tablas:

- category: hace referencia a la entidad Category
- comment: hace referencia a la entidad Comment
- recipe: hace referencia a la entidad Recipe
- score: hace referencia a la entidad Score
- user : hace referencia a la entidad User
- recipes\_categories: hace referencia a la relación M:M Recipe - Category.

A continuación podemos observar el diagrama de la base de datos.

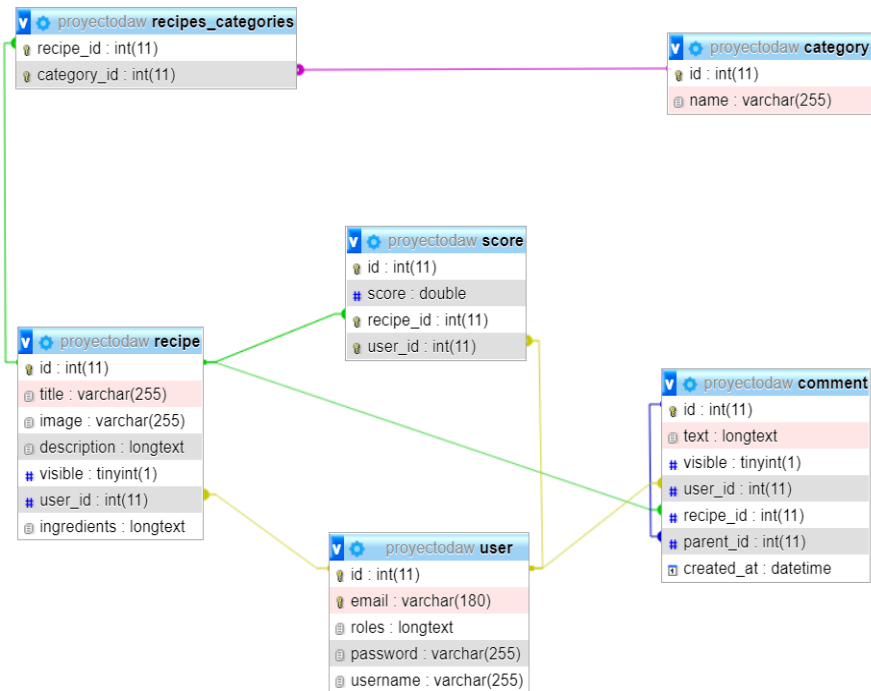


Figura 21: Diagrama de la base de datos.

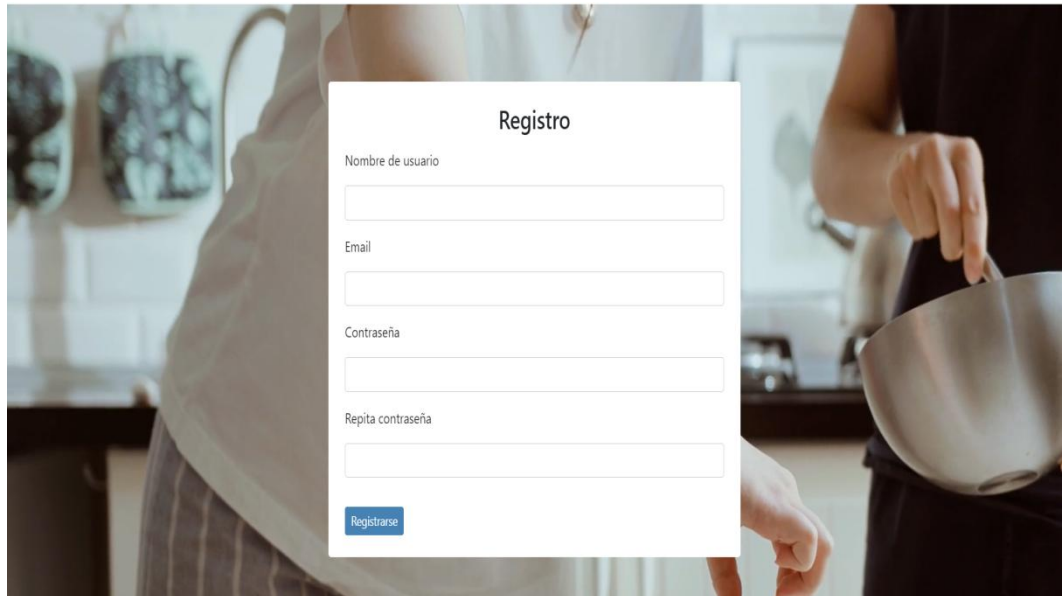
### 3.4.2. Inicio

Es la página de presentación de la web. En la parte superior encontramos un navbar. Este navbar estará presente en todas las páginas del sitio web, excepto en el panel de administrador. En la parte izquierda de este podemos observar el logo de la web y los enlaces a las secciones “Subir receta” y “Explora”. En la parte derecha podemos observar los botones “Iniciar sesión” y “Registrarse”. Debajo del navbar hay un video de fondo de dos chicas cocinando y encima de este se sitúa el eslogan, “Disfruta cocinando”. De esta forma hacemos ver a los usuarios la temática de la web.

IMAGEN DEL LOGO

### 3.4.3. Registro

En esta sección de la web será donde los usuarios podrán registrarse. La pagina tiene un formulario Symfony, en el cual debemos de introducir un nombre de usuario e email que no existan previamente en la base de datos y una contraseña que tendrá que tener mínimo 6 caracteres.

The image shows a web registration form titled "Registro" centered on the screen. The form is white with a thin border and contains four input fields: "Nombre de usuario", "Email", "Contraseña", and "Repita contraseña". Below the last field is a blue button labeled "Registrarse". The background is a blurred photograph of a person in a kitchen, wearing a white apron and holding a metal pot.

*Figura 23: Formulario de registro de usuarios.*

### 3.4.4. Login

Esta sección de la web cuenta con un formulario que permite a los usuarios iniciar sesión. Comprueba que el email exista y si la contraseña es correcta para ese email. Una vez introducidas unas credenciales correctas comprueba que rol tiene ese usuario. En caso de ser un usuario con rol de administrador, lo redirige al panel de administración de Sonata. En caso de ser un usuario con sol de user, se le redirigirá a la sección de explora.

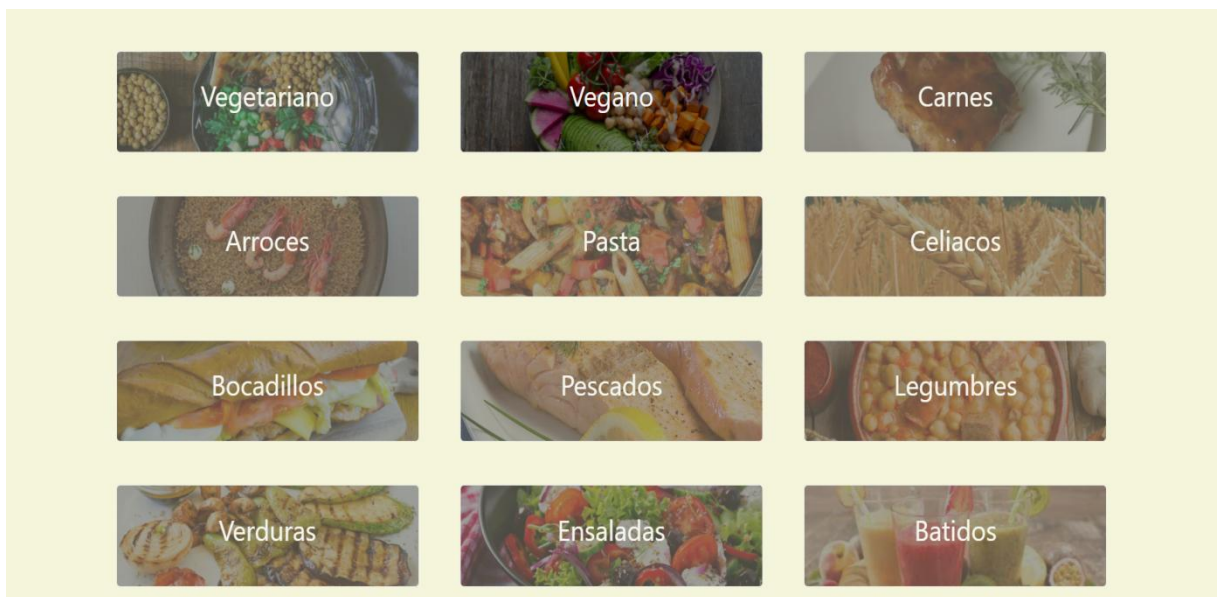
```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {
        return new RedirectResponse($targetPath);
    }

    if ($this->security->isGranted('ROLE_ADMIN')) {
        return new RedirectResponse($this->urlGenerator->generate('sonata_admin_redirect'));
    }
    if ($this->security->isGranted('ROLE_USER')) {
        return new RedirectResponse($this->urlGenerator->generate('explorer'));
    }
}
```

*Figura 24: Método que redirige según rol al iniciar sesión.*

### 3.4.5. Explora

En esta sección los usuarios podrán observar con una interfaz atractiva todas las categorías que tienen la web y elegir aquella en la que estén interesados para comenzar a buscar y leer recetas. Se ha agregado la propiedad hover sobre los elementos de las categorías para llamar más la atención de las categorías por las que se mueve el ratón.



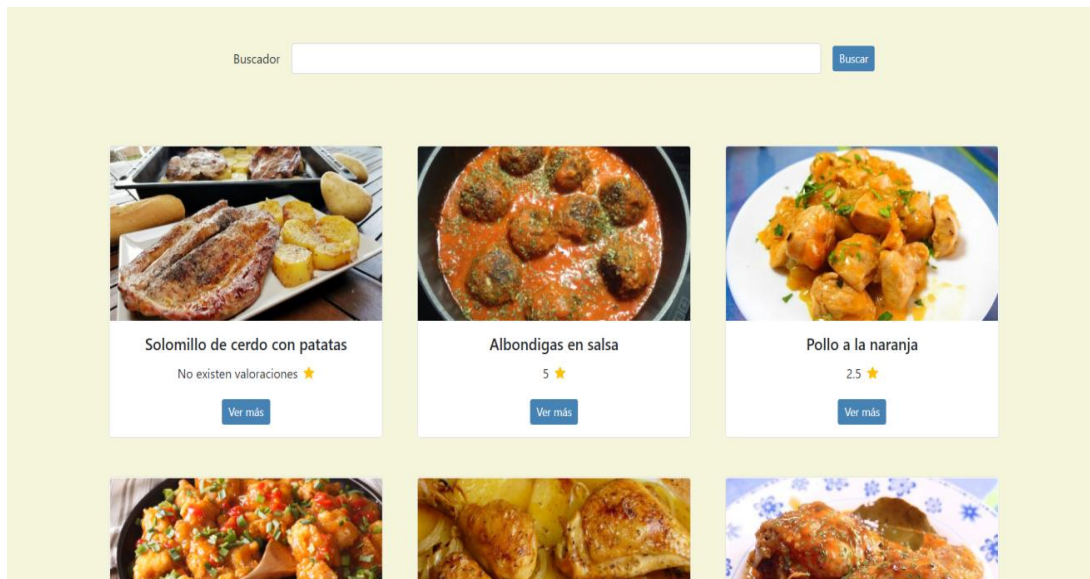
*Figura 25: Vista de la sección Explora*

### 3.4.6. Recetas de una categoría

Una vez el usuario se encuentre en esta página, podrá ver todas las recetas visibles de esa categoría junto con su puntuación media. En caso de no haber registradas puntuaciones para esa receta se mostrará un mensaje indicándolo. En caso de no existir recetas para una categoría se mostrará un mensaje indicando que no hay recetas para esa categoría. También estará el botón “Ver más”, que enlazará con la página de cada receta.

Además se ha añadido un buscador de recetas por título que facilita al usuario encontrar su receta deseada.





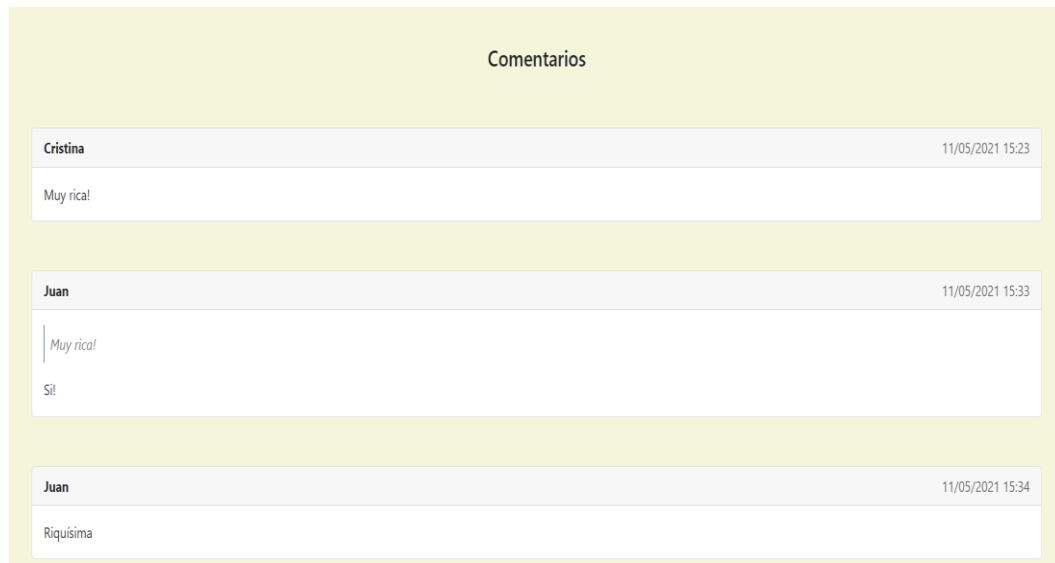
*Figura 26: Vista de las recetas de una categoría.*

### 3.4.7. Detalle de la receta

El usuario podrá acceder a toda la información de la receta (título, ingredientes y pasos a seguir). Además desde aquí también podrá puntuar y comentar una receta en caso de que haya iniciado sesión. El usuario solo podrá puntuar la receta una vez. En caso de haber puntuado ya la receta no permitirá puntuarla de nuevo y mostrara un mensaje indicándolo.



*Figura 27: Vista de una receta*



*Figura 28: Comentarios de una receta*

#### 3.4.8. Panel de administrador

Una vez explicada la interfaz diseñada al usuario con rol de usuario solo nos faltaría por ver el panel de administración. Este CRUD está enfocado a los usuarios con rol de administrador.

Desde esta sección los administradores podrán hacer no visibles aquellos comentarios o recetas que no sean indicadas para estar en la plataforma. De esta forma podrán controlar el contenido de la misma. Además podrá editar y eliminar recetas o añadir, editar y eliminar usuarios entre sus funcionalidades más destacables.

A screenshot of the 'Comentarios' form in the administrator panel. The form has a title 'Comentarios' and a label 'Texto\*' above a text input field containing 'Riquísima'. Below the input field is a checkbox labeled 'Visible' which is checked. At the bottom of the form are three buttons: 'Actualizar' (green), 'Actualizar y cerrar' (green), and 'Borrar' (red).

*Figura 29: Haciendo visible un comentario.*

## 4. Evaluación y conclusiones finales

Actualmente se ha conseguido desarrollar una aplicación web de recetas de cocina donde el usuario pueda acceder a recetas de distintas categorías, subir sus propias recetas, evaluarlas e incluso comentarlas.

Los objetivos planteados se han cumplido exitosamente, realizando todas las tareas planteadas. Estos objetivos han ido variando durante el proceso de desarrollo. Al principio el panel de administrador implementado con Sonata Bundle, se planteó con el bundle Easy Admin. Pero a pesar de este último tener una interfaz más atractiva, presentaba distintas funcionalidades que no me acabaron de convencer ya que es un bundle que no tiene flexibilidad y en caso de querer modificar cualquier cosa establecida por defecto no se puede o resulta mucho más complejo. Otra desventaja es que Easy Admin no ofrece rutas amigables. Sin embargo Sonata sí que ofrece esa flexibilidad y URL's amigables que Easy Admin no tiene. Haciendo balanza y pensando en los usuarios en los que está enfocado el panel de administración vi más adecuado apostar por Sonata.

Por otro lado, se han añadido nuevas funcionalidades que al principio no estaban contempladas, como es el caso de que los usuarios puedan comentar comentarios ya creados previamente.

Como funcionalidades que se podrían implementar en el proyecto sería que los usuarios tuvieran una sección llamada "Mi perfil", donde pudieran editar sus recetas y sus datos de registro. También como mejora se podría añadir que los usuarios pudieran editar sus puntuaciones en las recetas.

Por último, concluir diciendo que considero que se ha desarrollado una aplicación web completa, implementando los conocimientos obtenidos en clase y en las prácticas. Además, se ha investigado y desarrollado con el framework Symfony y sus tecnologías complementarias, implementado sus funcionalidades y cumpliendo los objetivos del proyecto.

## 5. Bibliografía y referencias

### Back end

- <https://symfony.com/doc/4.4/index.html>
- <https://openwebinars.net/blog/que-es-symfony/>
- <https://openwebinars.net/blog/ventajas-de-usar-twig-en-symfony/>
- <https://diego.com.es/creacion-de-formularios-en-symfony>
- <https://www.doctrine-project.org/index.html>
- <https://www.doctrine-project.org/projects/doctrine-orm/en/2.8/reference/association-mapping.html>

### Front-end

- <https://twig.symfony.com/doc/3.x/>
- <http://gitnacho.github.io/Twig/templates.html>
- <https://openwebinars.net/blog/ventajas-de-usar-twig-en-symfony/>

### Sonata Admin Bundle

- <https://symfony.com/doc/current/bundles/SonataAdminBundle/index.html>

### CKEditor Bundle

- <https://symfony.com/doc/current/bundles/FOSCKEditorBundle/installation.html>