



Programación Web

Tema 2: Frameworks PHP

Curso 2024/25

1. ¿Qué es Symfony?

- Framework PHP profesional y modular.
- Basado en componentes independientes reutilizables.
- Muy usado en grandes plataformas (Drupal, Prestashop, APIs).

Ventajas:

- Alta flexibilidad y control.
- Estándares de calidad y buenas prácticas.
- Gran ecosistema de componentes.

Symfony y el MVC

- Symfony adopta la idea de MVC, pero no la impone de manera estricta como Laravel o CodeIgniter.
- El controlador en Symfony puede hacer más lógica de negocio.
- El modelo no está limitado a bases de datos: puede ser cualquier estructura de datos o servicio.
- Symfony permite respuestas flexibles: HTML, JSON, archivos, etc

2. Comparación

Característica	Symfony	Laravel	CodeIgniter
Curva de aprendizaje	Alta	Media	Baja
Modularidad	Máxima	Media	Baja
Filosofía	Configuración total	Productividad inmediata	Simplicidad
Componentes reutilizables	Sí	No (todo integrado)	No

3. Requisitos e instalación

- **Composer**
 - Tener Composer instalado.
 - PHP \geq 8.2
 - Symfony CLI (opcional)
-
- <https://symfony.com/doc/current/setup.html>

Instalación rápida:

```
composer create-project symfony/skeleton mi-proyecto
```

o

```
symfony new mi-proyecto --webapp
```

Skeleton: Proyecto base minimalista.

Webapp: Proyecto completo con todo lo necesario preinstalado.

4. Estructura de un proyecto

/src/ → Controladores, entidades, servicios

/templates/ → Vistas con Twig

/config/ → Configuraciones de rutas, bases de datos, bundles

/public/ → Archivos accesibles vía navegador

/vendor/ → Librerías instaladas con Composer

5. Rutas en Symfony

- Las rutas se pueden definir **directamente en los controladores** usando attributes (`#[Route]`).
- O también en archivos externos YAML, XML o PHP.

Ventajas de definir rutas en el controlador:

- Mayor claridad: la ruta está junto a su lógica.
 - Facilita la lectura y el mantenimiento.
 - Ideal para proyectos pequeños o medianos.
-
- A pesar de todo, en proyectos grandes y complejos, es mejor mantener las rutas centralizadas.

6. Controladores

```
namespace App\Controller;

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

class SaludoController extends AbstractController
{
    #[Route('/saludo', name: 'saludo')]
    public function index(): Response
    {
        return new Response('Hola desde Symfony');
    }
}
```


7. Introducción a Twig

- Motor de plantillas usado por defecto en Symfony.
- Sintaxis muy limpia y segura.

```
<h1>Hola {{ nombre }}</h1>
```

Ventajas:

- Evita mezclar PHP y HTML directamente.
- Protege automáticamente contra inyecciones XSS.

8. Bases de datos

- Hasta ahora, cuando hemos trabajado con PHP puro, la conexión a base de datos se hacía manualmente usando mysqli o PDO. Symfony gestiona esto de forma mucho más avanzada gracias a una herramienta que integra de serie: Doctrine ORM.
- ORM significa 'Object Relational Mapping', o mapeo objeto-relacional. En otras palabras, en lugar de escribir consultas SQL directamente, trabajamos con objetos de PHP que Doctrine se encarga de traducir a operaciones de base de datos.

8. Bases de datos

- Symfony utiliza el fichero .env para configurar todo lo que depende del entorno, incluida la conexión a base de datos.
- En ese archivo veremos una línea parecida a esta:
- `DATABASE_URL="mysql://usuario:contraseña@127.0.0.1:3306/nombre_base_de_datos"`

8. Bases de datos

- En Doctrine, una 'Entidad' es simplemente una clase de PHP normal que representa una tabla en la base de datos.
- Si pensamos en un sistema de productos, tendríamos una clase Producto, que corresponde a una tabla productos.
- Symfony nos permite declarar esa relación con una sintaxis especial. Veréis que encima de las propiedades hay unas líneas que empiezan con #. Eso se llama **attribute** en PHP 8.

8. Bases de datos

- ¿Qué es ese `#[ORM\Id]`, `#[ORM\Column]`...?
- Son instrucciones para Doctrine.
 - Esos attributes le dicen a Symfony cómo construir la tabla o cómo debe leerla: qué campos tiene, qué tipo de datos son, cuál es la clave primaria, etc.
 - No escribimos SQL manualmente: Doctrine entiende cómo debe comportarse solo leyendo esa clase.

8. Bases de datos

- Aquí estamos diciendo que Producto es una tabla, que id es la clave primaria, que nombre es un string y precio es un número decimal.
- Doctrine sabrá cómo guardar o leer productos directamente usando objetos de esta clase.
- Desde el controlador, podemos pedirle a Symfony que nos traiga todos los productos.

```
$productos = $em->getRepository(Producto::class)->findAll();
```

- Symfony se conecta a MySQL, hace un SELECT * FROM productos internamente, pero nosotros nunca escribimos la consulta SQL directamente.

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity]
class Producto
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

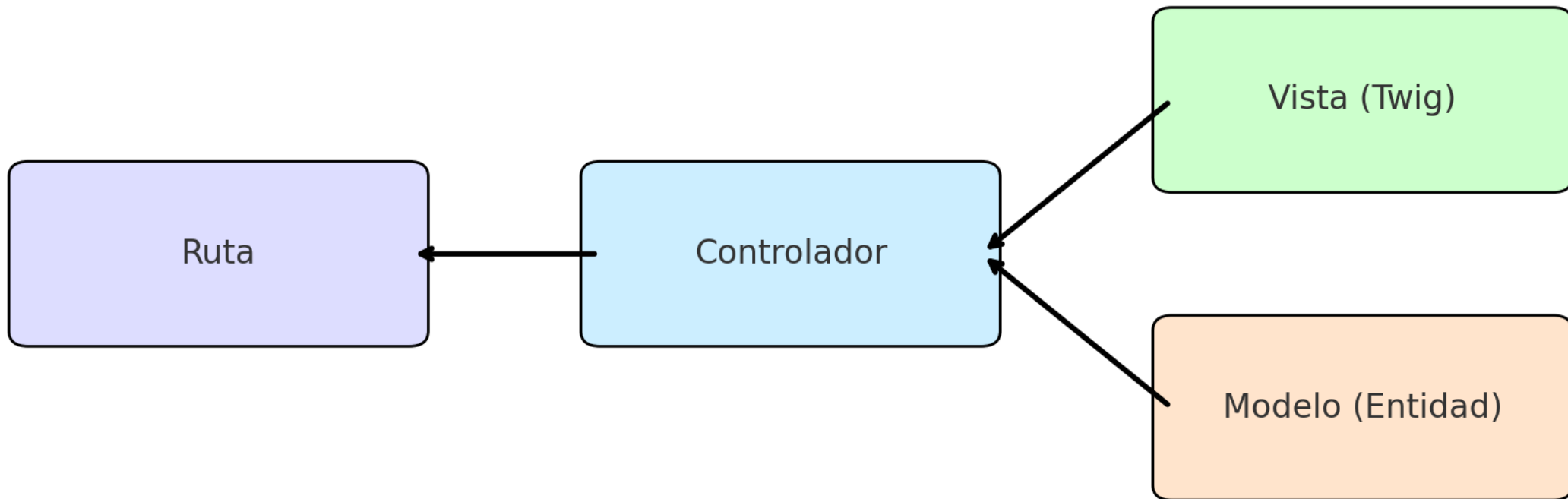
    #[ORM\Column(type: 'string')]
    private $nombre;

    #[ORM\Column(type: 'float')]
    private $precio;
}
```

8.1. ¿Y por qué ORM?

Nos permite trabajar a más alto nivel: pensamos en objetos, no en tablas.

Y además hace el código más limpio, más seguro frente a inyecciones de SQL, y más mantenible a largo plazo.

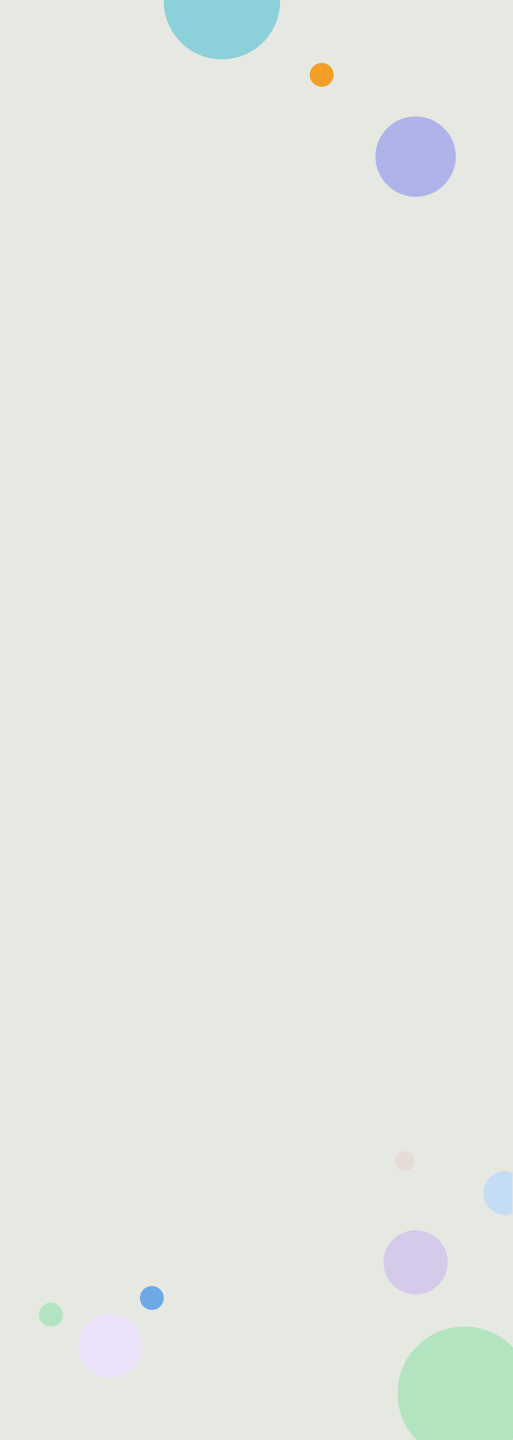


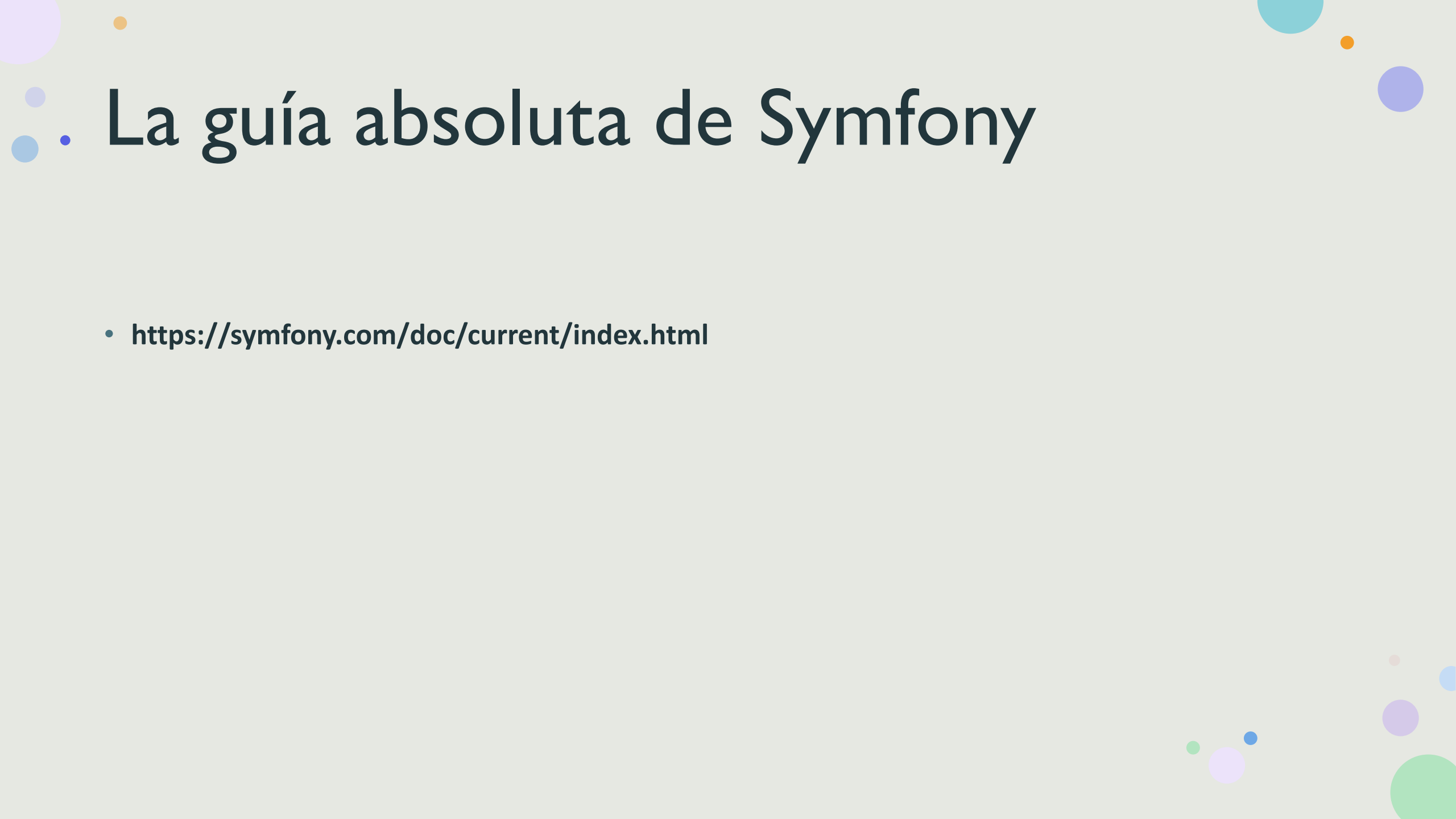
Symfony gestiona las rutas automáticamente
Las acciones del controlador pueden generar respuestas HTML (Twig)
o acceder a entidades de base de datos mediante Doctrine ORM



Reflexión Final

Comparativa breve:

- Symfony: Control absoluto, configuración detallada, alta escalabilidad.
 - Laravel: Rápido para empezar, estructura amigable.
 - CodeIgniter: Simple, ideal para proyectos pequeños.
- 



• La guía absoluta de Symfony

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