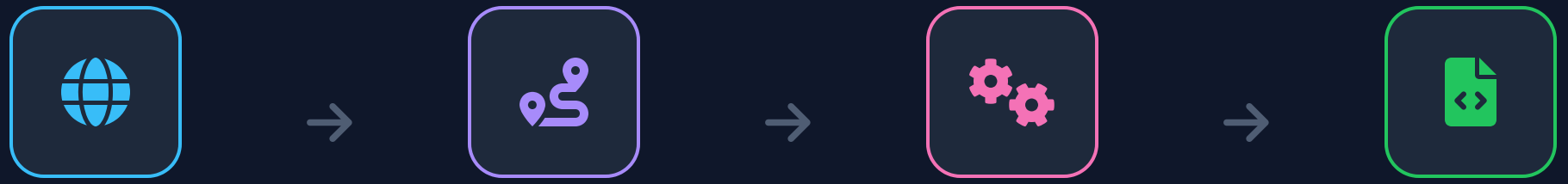
TEMA 7

El Flujo MVC en Symfony

Arquitectura Request-Response



Symfony 8

Enrutamiento con Atributos

src/Controller/BlogController.php

```php
namespace App\Controller;

use Symfony\Component\Routing\Attribute\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractControlle

class BlogController extends AbstractController
{
    // Definición de la ruta con atributos PHP 8
    #[Route('/blog/{slug}', name: 'blog_show', methods: ['GET']
    public function show(string $slug): Response
    {
        // ... lógica para buscar el post
        return $this->render('blog/show.html.twig', [
            'slug' => $slug
        ]);
    }
}
```

path

El patrón de URL pública. Puede ser estático (`/blog`) o dinámico (`/blog/{slug}`).

name

ID único interno. **Obligatorio** para buenas prácticas. Permite cambiar la URL sin romper enlaces en la app.

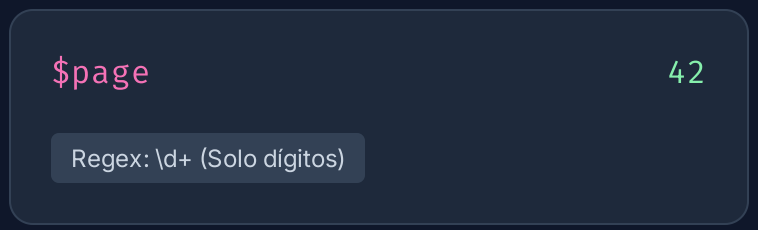
methods

Array de verbos HTTP permitidos (GET, POST, PUT...). Si no coincide, Symfony devuelve `405 Method Not Allowed`.

Parámetros y Requisitos

src/Controller/BlogController.php

```php
// 1. Parámetro obligatorio con requisito (Regex)
#[Route('/blog/{page}', name: 'blog_list', requirements: ['page' => '\d+'])]
public function list(int $page = 1): Response
{
    // Solo coincide si {page} son dígitos.
    // Si la URL es /blog/hola → 404 Not Found
}

// 2. Parámetro opcional (con valor por defecto en PHP)
#[Route('/search/{query}', name: 'blog_search')]
public function search(string $query = ''): Response
{
    // Coincide con /search/symfony y /search/
}

// 3. Prioridad: Rutas estáticas ganan a dinámicas
#[Route('/blog/contact', name: 'blog_contact')] // Gana a /blog/{slug}
```



✕ 404 Not Found

Enrutamiento Avanzado

src/Controller/AdvancedController.php

```php
// 1. Prioridad: Definir rutas específicas ANTES de las dinámicas
#[Route('/blog/new', name: 'blog_new')]
public function new() { ... }

#[Route('/blog/{slug}', name: 'blog_show')]
public function show($slug) { ... }


// 2. Host Matching: Solo responde en subdominios específicos
#[Route('/', host: 'api.example.com')]
public function apiIndex() { ... }


// 3. Condiciones: Lógica compleja con Expression Language
#[Route('/contact', condition: "request.headers.get('User-Agent') matches '/fi
public function firefoxOnly() { ... }
```

Prioridad 1

Symfony evalúa las rutas de arriba a abajo. Si pones `/{slug}` antes de `/new`, la URL `/blog/new` será capturada por la ruta dinámica erróneamente.

Host Matching 2

Permite tener la misma ruta (ej: `/`) apuntando a controladores diferentes según el subdominio (`api.` vs `www.`).

Condiciones 3

Usa **Expression Language** para lógica avanzada: verificar headers, IPs, o cualquier dato del objeto Request antes de entrar.

Depuración de Rutas

`debug:router`

Muestra una tabla con todas las rutas configuradas en la aplicación. Útil para verificar nombres, paths y métodos HTTP permitidos.

`router:match`

Simula una petición a una URL específica y muestra qué ruta coincide (o por qué falla). Muestra los parámetros extraídos.

```
user@symfony:~$ php bin/console debug:router

Name              Method   Scheme   Host   Path
app_home          ANY      ANY      ANY    /
blog_list         GET      ANY      ANY    /blog/{page}
blog_show         GET      ANY      ANY    /blog/post/{slug}
_preview_error    ANY      ANY      ANY    /_error/{code}

user@symfony:~$ php bin/console router:match /blog/post/symfony-8

 [OK] Route "blog_show" matches

Route Name    blog_show
Path          /blog/post/{slug}
Path Regex    #^/blog/post/(?P<slug>[^/]++)$#sD
Host          ANY
Scheme        ANY
Method        ANY
Class         App\Controller\BlogController::show
slug          "symfony-8"
```

El Controlador Moderno

src/Controller/HomeController.php

```php
namespace App\Controller;

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

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

AbstractController

Clase base recomendada. Proporciona acceso al contenedor de servicios y métodos atajo para tareas comunes.

Objeto Response

Todo método de controlador **debe** devolver un objeto `Response`. Symfony se encarga de enviar las cabeceras y el contenido al navegador.

Métodos Auxiliares

Atajos para no instanciar servicios manualmente:

- `$this->render()`
- `$this->json()`
- `$this->redirectToRoute()`
- `$this->createNotFoundException()`

Tipos de Respuesta

JsonResponse

Convierte arrays automáticamente a JSON y establece el header `Content-Type: application/json`.

```
return$this→json([ 'status' ⇒ 'ok'' 'data' ⇒ $user]);
```

RedirectResponse

Redirige al usuario a otra URL. Usar siempre nombres de ruta en lugar de URLs hardcodeadas.

```
return$this→redirectToRoute( 'blog_show'' ['slug' ⇒ 'sy
```

BinaryFileResponse

Optimizado para servir archivos estáticos (imágenes, PDFs) con soporte para rangos y cache.

```
return$this→file( '/path/to/invoice.pdf'' 'factura.pdf')
```

StreamedResponse

Para generar contenido grande (CSV, video) poco a poco sin cargar todo en memoria RAM.

```
return new StreamedResponse(function() {echo'data chunk 1 ... '
```

El Objeto Request

src/Controller/BlogController.php

```php
use Symfony\Component\HttpFoundation\Request;

public function index(Request $request): Response
{
    // 1. Query Params (GET) /blog?page=2
    $page = $request->query->get('page', 1);

    // 2. Form Data (POST)
    $title = $request->request->get('title');

    // 3. Archivos subidos (multipart/form-data)
    $file = $request->files->get('attachment');

    // 4. Headers y Cookies
    $ua = $request->headers->get('User-Agent');
    $session = $request->cookies->get('PHPSESSID');

    // ...
```

`$_GET`	→	`$request->query`
`$_POST`	→	`$request->request`
`$_FILES`	→	`$request->files`
`$_COOKIE`	→	`$request->cookies`
`$_SERVER`	→	`$request->server`

INYECCIÓN DE DEPENDENCIAS

Nunca uses las superglobales ($_GET) directamente. Inyecta el objeto `Request` en el método del controlador para facilitar el testing y la seguridad.

Sesiones y Mensajes Flash

src/Controller/UserController.php

```php
public function update(Request $request): Response
{
    // 1. Acceder a la sesión
    $session = $request->getSession();
    $session->set('last_action', 'update');

    // 2. Mensaje Flash (vive solo 1 petición)
    // Ideal para notificaciones tras redirección
    $this->addFlash(
        'success',
        '¡Perfil actualizado correctamente!'
    );

    $this->addFlash('warning', 'Revisa tu email.');

    return $this->redirectToRoute('user_profile');
}
```

Sesión

Almacenamiento persistente por usuario. Útil para carritos de compra, preferencias o datos de autenticación.

Flash Bag

Almacenamiento temporal especial. Los mensajes se guardan en sesión pero **se eliminan automáticamente** en cuanto se leen.

TEMPLATES/BASE.HTML.TWIG

```twig
{% for message in app.flashes('success') %}
    <div class="alert alert-success">
        {{ message }}
    </div>
{% endfor %}
```

Inyección de Dependencias

src/Controller/ReportController.php

```php
class ReportController extends AbstractController
{
    private LoggerInterface $logger;

    // 1. Inyección por Constructor (Recomendado)
    // Disponible en TODOS los métodos de la clase
    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    #[Route('/report/send')]
    // 2. Inyección en el Método (Action Injection)
    // Solo disponible en ESTE método
    public function send(MailerInterface $mailer): Response
    {
        $this->logger->info('Enviando reporte...');

        $mailer->send(/* ... */);
```

Autowiring

Symfony detecta automáticamente el tipo de la variable (ej: `LoggerInterface`) y te pasa el servicio correcto. ¡Sin configuración manual!

Constructor Injection

La forma estándar. Ideal para dependencias que la clase necesita obligatoriamente para funcionar.

Action Injection

Exclusivo de los Controladores. Útil si un servicio pesado solo se usa en una ruta específica, ahorrando recursos en las demás.

Manejo de Errores

src/Controller/ProductController.php

```php
public function show(int $id, ProductRepository $repo): Response
{
    $product = $repo->find($id);

    if (!$product) {
        // Lanza una NotFoundHttpException (404)
        // Symfony captura esto y muestra la página de error
        throw $this->createNotFoundException(
            'El producto no existe'
        );
    }

    // También puedes lanzar excepciones manualmente
    if (!$this->isGranted('ROLE_ADMIN')) {
        throw new AccessDeniedHttpException('Acceso denegado');
    }

    return $this->render('product/show.html.twig', ...);
```

Excepciones HTTP

No devuelvas una respuesta de error manualmente. Lanza excepciones para que Symfony maneje el código de estado HTTP correcto (404, 403, 500).

Personalización

Para personalizar las páginas de error en producción, crea plantillas en esta ruta específica:

```
templates/bundles/TwigBundle/Exception/
├── error404.html.twig
├── error403.html.twig
└── error500.html.twig
```

Desarrollo vs Prod

En **dev**, verás la traza de la excepción completa. En **prod**, el usuario verá tu plantilla personalizada "amigable".

Introducción a Twig

{{ output }}

Imprime el contenido de una variable. Aplica **auto-escaping** por seguridad contra XSS.

```
Hola {{ name }}
```

{% logic %}

Ejecuta comandos: bucles, condicionales, definición de variables o herencia.

```
{% if user.isAdmin %}
```

{# comment #}

Comentarios que **no se renderizan** en el HTML final enviado al navegador.

```
{# TODO: Fix this #}
```

Controller

```php
return $this->render('user/profile.html.twig', [
    'name' => 'Mario',
    'points' => 150
]);
```

→

user/profile.html.twig

```twig
<h1>Hola {{ name }}</h1>

{% if points > 100 %}
    <p>¡Eres un usuario VIP!</p>
{% endif %}
```

Estructuras de Control

Bucles (For)

```
<ul>
{% for user in users %}
    <li>
        {{ loop.index }} - {{ user.name }}
        {% if loop.first %} (Líder) {% endif %}
    </li>
{% else %}
```

Variable `loop`

`loop.index`	Índice actual (empieza en 1)
`loop.index0`	Índice actual (empieza en 0)
`loop.first`	True si es la primera iteración
`loop.last`	True si es la última iteración
`loop.length`	Total de elementos

Condicionales (If)

```
{% if user.isActive and not user.isBanned %}

    <div class="alert alert-success">
        Bienvenido, {{ user.name }}
    </div>

{% elseif user.isBanned %}

    <div class="alert alert-danger">
        Tu cuenta está suspendida.
    </div>
```

Operadores Lógicos

and, or, not
is defined (existe la variable)
is empty (array vacío o null)
is null

Filtros y Funciones

Filtros (Pipe |)

Modifican el contenido de una variable antes de mostrarlo.

```
{{ 'hola'|upper }}                    →  HOLA
{{ createdAt|date('d/m/Y') }}         →  12/01/2026
{{ phone|default('N/A') }}            →  N/A
{{ users|length }}                    →  42
{{ html|raw }}                        →  <b>Bold</b>
```

Funciones

Generan contenido o realizan acciones lógicas.

```
{{ dump(user) }}
{{ random(1, 10) }}
{{ max(a, b) }}
```

Encadenamiento

Los filtros se aplican de izquierda a derecha.

```
{{ name|trim|upper }}
```

1. Elimina espacios (trim)
2. Convierte a mayúsculas (upper)

Herencia de Plantillas

templates/base.html.twig — LAYOUT BASE

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            {% block title %}Welcome!{% endblock %}
        </title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        <nav> ... Menu ... </nav>

        <div class="container">
            {% block body %}{% endblock %}

        </div>
        <footer> ... Footer ... </footer>
```

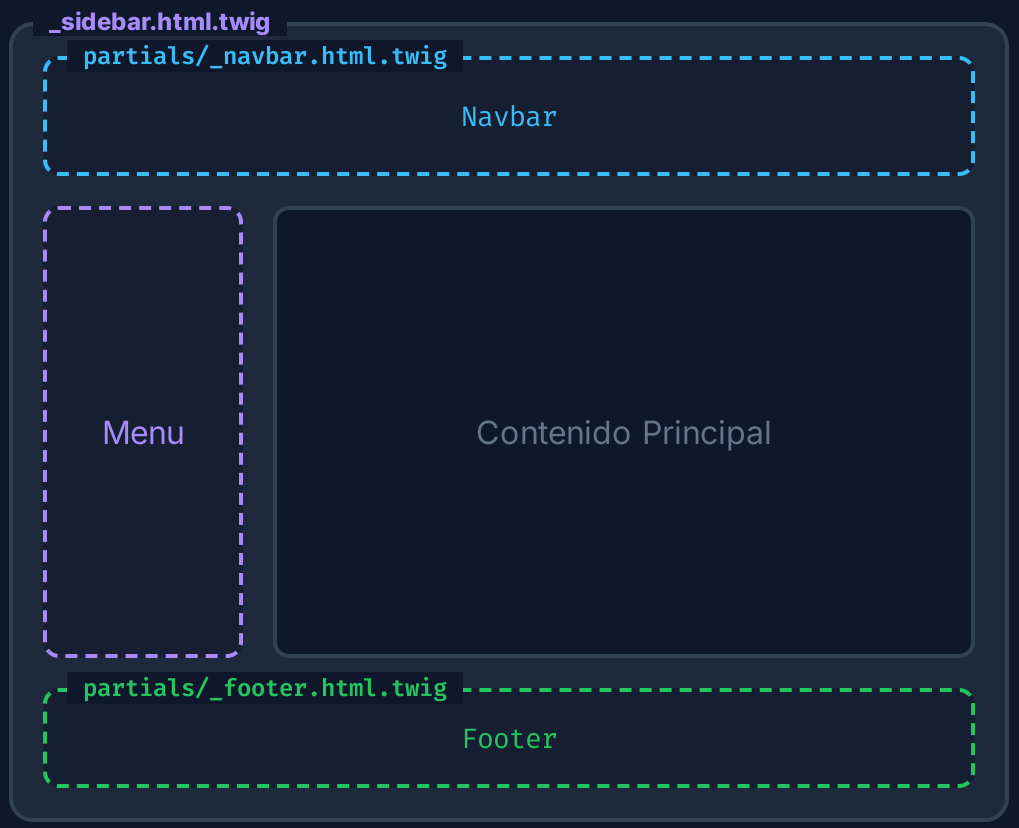
Define la estructura común y los "huecos" (blocks) a rellenar.

templates/blog/index.html.twig — VISTA HIJA

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

{% block title %}
    Blog - Symfony 8
{% endblock %}

{% block body %}

    <h1>Últimos Artículos</h1>

    <ul>
        {% for post in posts %}
            <li>{{ post.title }}</li>
        {% endfor %}
    </ul>
```

Hereda del base y sobrescribe SOLO los bloques necesarios.

Reutilización con Include



templates/base.html.twig

```twig
{{ include('partials/_navbar.html.twig') }}

<main>
    {% block body %}{% endblock %}
</main>

{{ include('partials/_footer.html.twig', {
    year: 2024,
    show_socials: true
}) }}
```

Convención de Nombres

Es una buena práctica prefijar los archivos parciales con un guion bajo (_). Esto indica visualmente que el archivo es un fragmento y no una página completa renderizable por sí misma.

Generación de URLs

En el Controlador

```
$url = $this->generateUrl(    'blog_show',    ['slug'
```

En la Plantilla (Twig)

```
<a href="{{path('blog_show', { slug: 'symfony-8' })
```

¿Por qué usar nombres de ruta?

Desacoplamiento total. Si mañana decides cambiar la URL de `/blog/{slug}` a `/noticias/{slug}`, solo tienes que editar el atributo `#[Route]`. **Todos los enlaces de tu aplicación se actualizarán automáticamente.**

NEVER DO THIS

```
<a href="/blog/{{ slug }}">
```

Gestión de Assets

Archivos Estáticos

Usa la función `asset()` para referenciar imágenes, PDFs o fuentes. Symfony gestiona automáticamente la ruta base y el versionado.

templates/home.html.twig

```twig
<img src="{{ asset('images/logo.png') }}">

<a href="{{ asset('doc/manual.pdf') }}">
    Descargar
</a>
```

HTML GENERADO

```
<img src="/images/logo.png?v=2">
```

AssetMapper

El estándar moderno para CSS y JS. Sin Node.js ni Webpack. Utiliza **Import Maps** nativos del navegador para cargar módulos JavaScript.

templates/base.html.twig

```twig
{% block stylesheets %}
    <link rel="stylesheet" href="{{ asset('styles/app.css') }}">
{% endblock %}

{% block javascripts %}
    {{ importmap('app') }}
{% endblock %}
```

MAPEO AUTOMÁTICO

```
assets/app.js → public/assets/app-[hash].js
```

Productividad con Maker

`make:controller`

Crea una clase controladora y su plantilla Twig asociada. Configura automáticamente la ruta básica.

`make:entity`

Crea o actualiza una entidad Doctrine. Te guía paso a paso para añadir campos (nombre, tipo, longitud).

`make:crud`

Genera un sistema completo (Controller, Form, Templates) para Listar, Crear, Editar y Borrar una entidad.

```
user@symfony:~$ php bin/console make:controller

 Choose a name for your controller class (e.g. OrangeGnomeC
 > BlogController

 created: src/Controller/BlogController.php
 created: templates/blog/index.html.twig


  Success!


 Next: Open your new controller class and add some pages!
user@symfony:~$
```

Symfony Profiler

Request / Response

Inspecciona variables GET/POST, headers, cookies, atributos de sesión y configuración del servidor.

Doctrine

Ver todas las consultas SQL ejecutadas, parámetros, tiempo de ejecución y filas afectadas.

Performance

Línea de tiempo detallada. Descubre qué servicios o listeners están ralentizando tu aplicación.

Logs & Events

Mensajes de log (info, error, debug), eventos disparados y notificaciones de deprecación.

↓ La **Web Debug Toolbar** aparece en la parte inferior de tu navegador en modo desarrollo.