

# UD8. Desarrollo de aplicaciones en Symfony

---

DESARROLLO WEB EN ENTORNO SERVIDOR

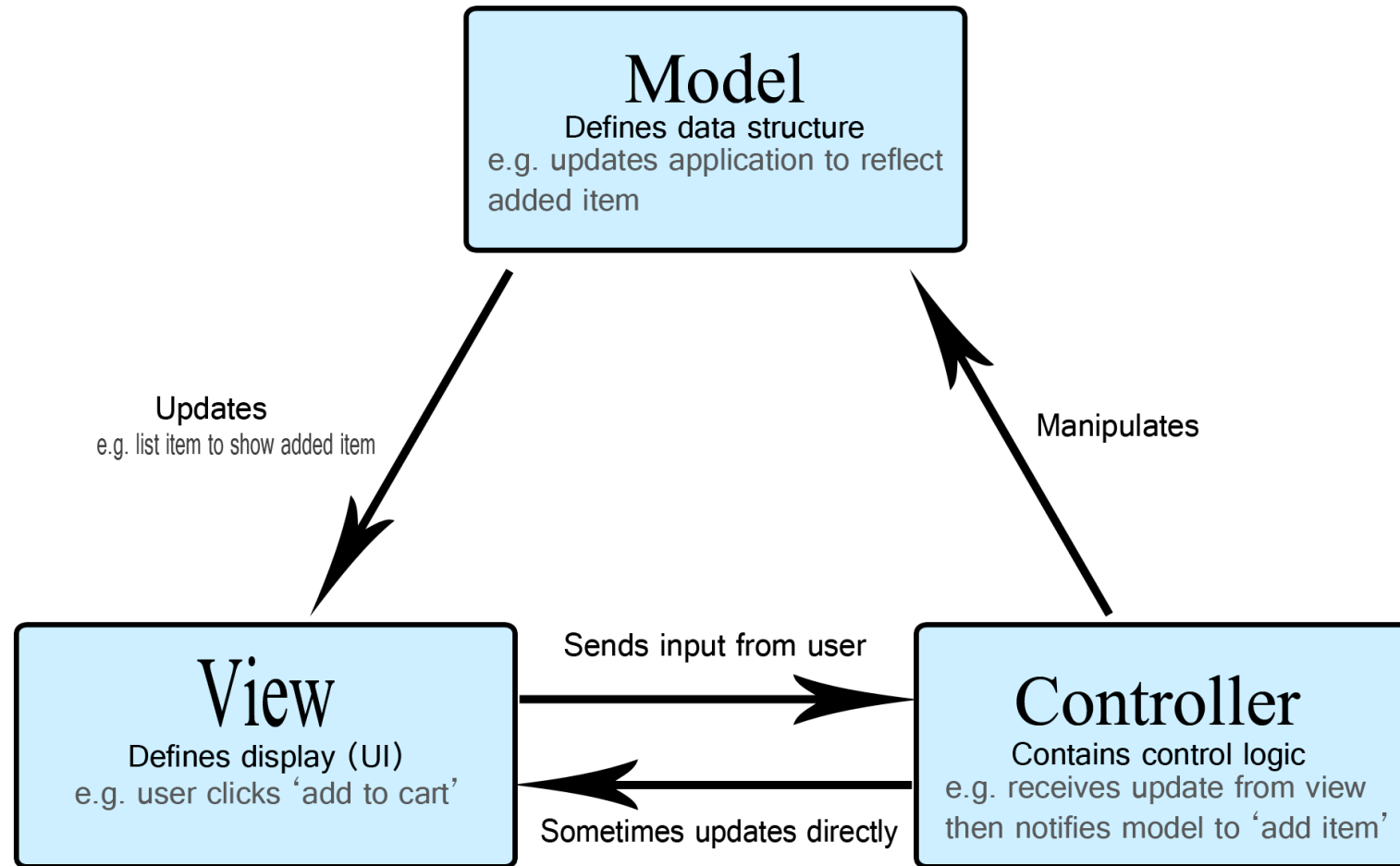
# Patrón MVC

# MVC

---

- Patrón que divide la aplicación en tres capas: modelo, vista y controlador.
- Al desacoplar de este modo:
  - Código mas reutilizable.
  - Equipos trabajando en paralelo en cada capa.

# MVC



<https://developer.mozilla.org/es/docs/Glossary/MVC>

# MVC

---

- Frameworks.
  - Cada uno difiere en la implementación particular del MVC.

Framework	Lenguaje
Spring MVC	Java (JEE)
Symfony	PHP
ASP.NET	MVC ASP
Ruby-on-Rails	Ruby
Angular	Javascript
TreeFrog	C++

# Symfony

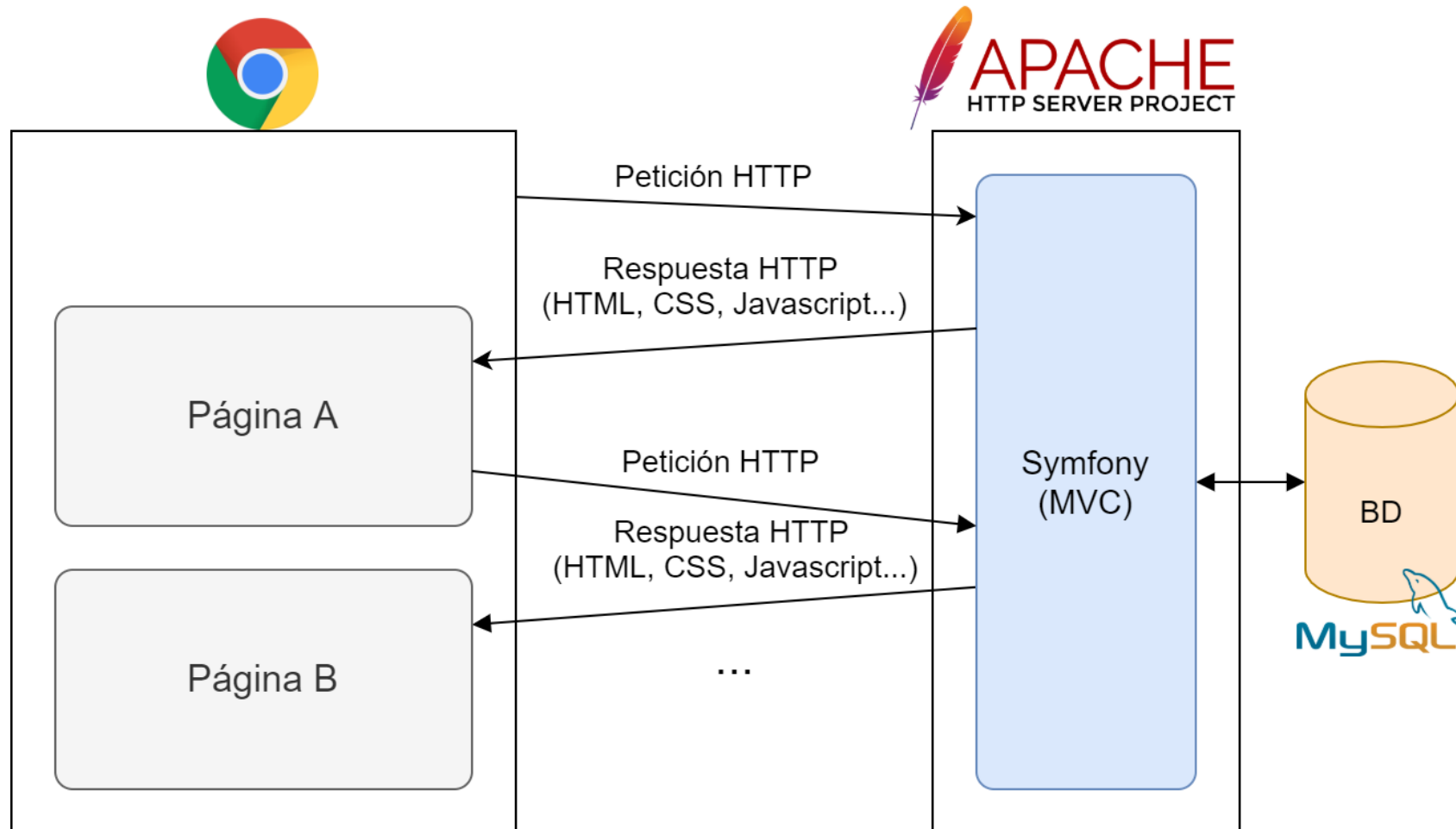
# Symfony

---

- Framework para desarrollar aplicaciones web en PHP.
- Siguiendo el patron MVC.
- Plantea las aplicaciones de una manera determinada, a la que temenos que adaptarnos.
- Incluye componentes para tareas comunes.
- Es software libre.



# Symfony





# Hasta ahora

---

1. El cliente solicita explícitamente al servidor qué fichero necesita.
  - Ejemplo: `http://localhost/holamundo.php`
2. El módulo PHP ejecuta el script y genera la salida.
  - Ejemplo: fichero `holamundo.php`

```
<?php  
  
    echo "<html><body><h1>Hola</h1></body></html>";
```

3. Apache devuelve la respuesta al cliente

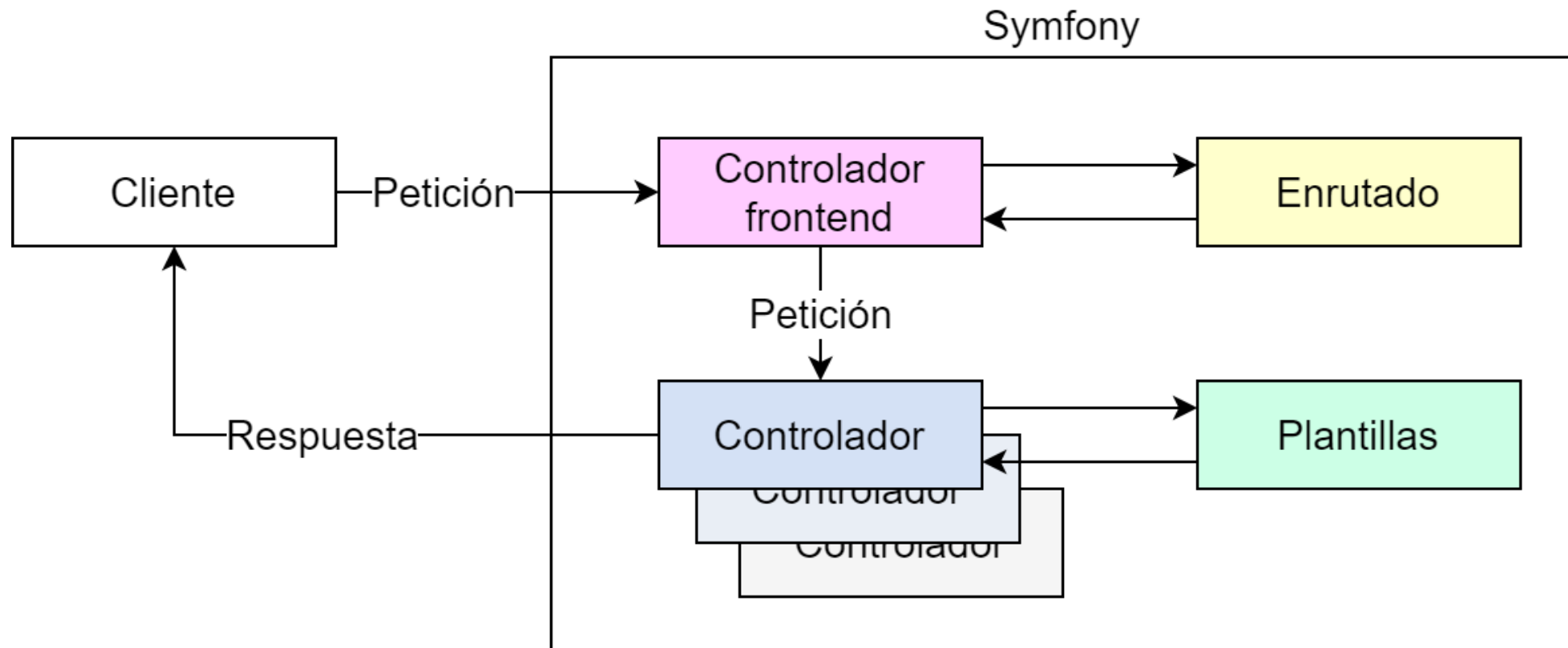
# Con Symfony

---

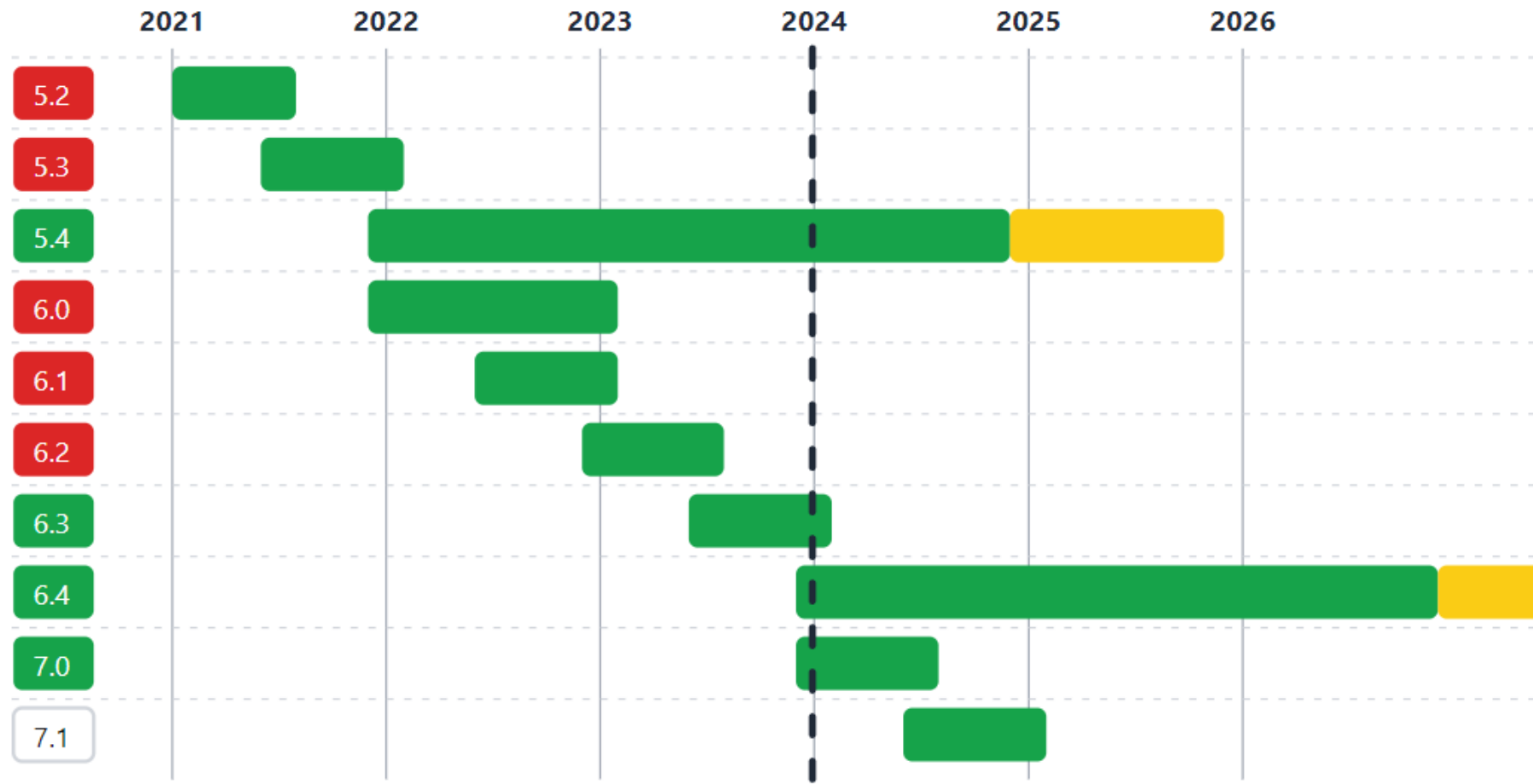
1. El cliente accede a una URL.
  - Ejemplo: `http://localhost/HolaMundo`
2. Un controlador de front-end redirige la petición a la parte de la aplicación que toque (**controlador**).
3. El **controlador** procesa la petición y genera una respuesta.
  - Ejemplo: controlador `hola`:

```
# [Route('/HolaMundo', name:'hola')]
public function hola() {
    return new Response("<html><body>Hola mundo!</body></html>");
}
```
4. Apache envía la respuesta al cliente.

# Con Symfony



# Instalación



# Instalación

---

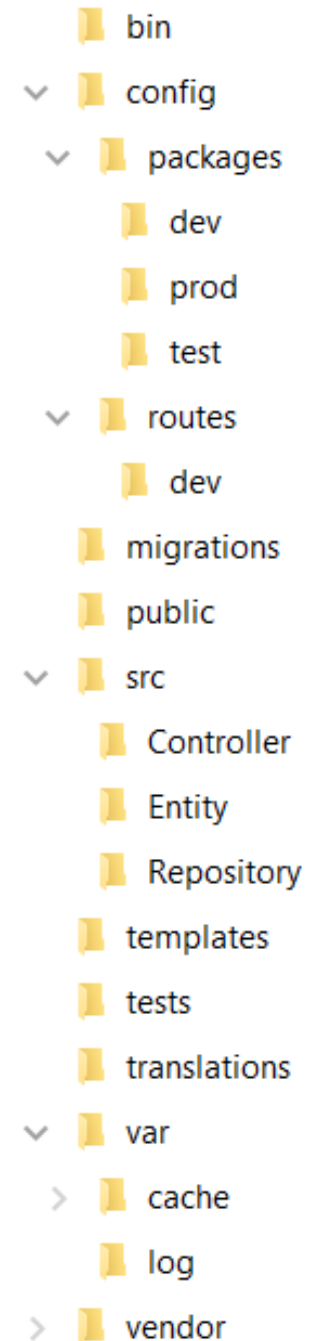
Sigue los pasos que están en el aula virtual:

1. Instalación del entorno
2. Crear nuevo proyecto en Symfony
3. Arrancar y parar el servidor web de Symfony

# Estructura de directorios

Cada proyecto de Symfony:

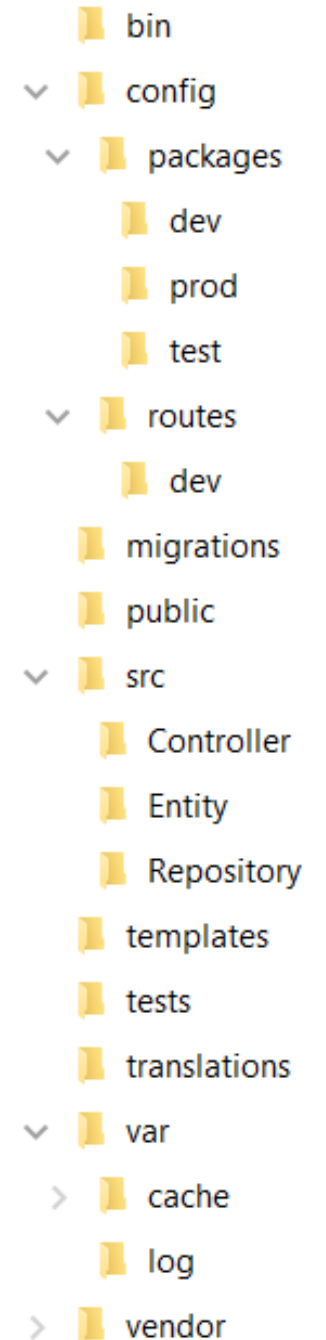
- Tiene una jerarquía de directorios.
- Cada tipo de componente va en un sitio.
- A priori es pesado, pero a la larga compensa.



# Estructura de directorios

## Directorios más importantes:

- `.env`: configuración de la BB.DD. y servidor SMTP.
- `/config/packages`: configuración de componentes.
- `/src/Controller`: controladores.
- `/src/Entity`: entidades de Doctrine.
- `/templates`: plantillas.
- `/bin`: ficheros ejecutables que usa el comando `symfony console`



# Controladores y rutas



# Controladores

---

El controlador es elemento principal de Symfony:

- Método que recibe una petición, la procesa y genera una respuesta.
- Está dentro una clase, llamada clase controladora.
- Estas clases se guardan en `/src/Controller` (puedes crear subdirectorios).
- Suelen heredar de `AbstractController`, que aporta funcionalidad extra.

# Controladores

---

¿Cómo se crea una nueva clase controladora?

- Escribiéndolo de cero.
- Copiando el código de otro controlador.
- De manera perezosa, con symfony-cli:  
`symfony console make:controller Controladorcito`

# Enrutado

---

Un controlador tiene asociado una o varias rutas.

Existen diferentes modos de indicar rutas en Symfony:

- Atributos PHP8
- Anotaciones
- Fichero XML.
- Fichero YAML.
- Dinámicamente en el código.

# Enrutado

## Indicación de ruta con atributos PHP8:

```
#[Route('/deporte/esqui', name: 'd_esqui', methods: ['GET', 'POST'])  
public function mostrarEsqui() {  
...  
}
```

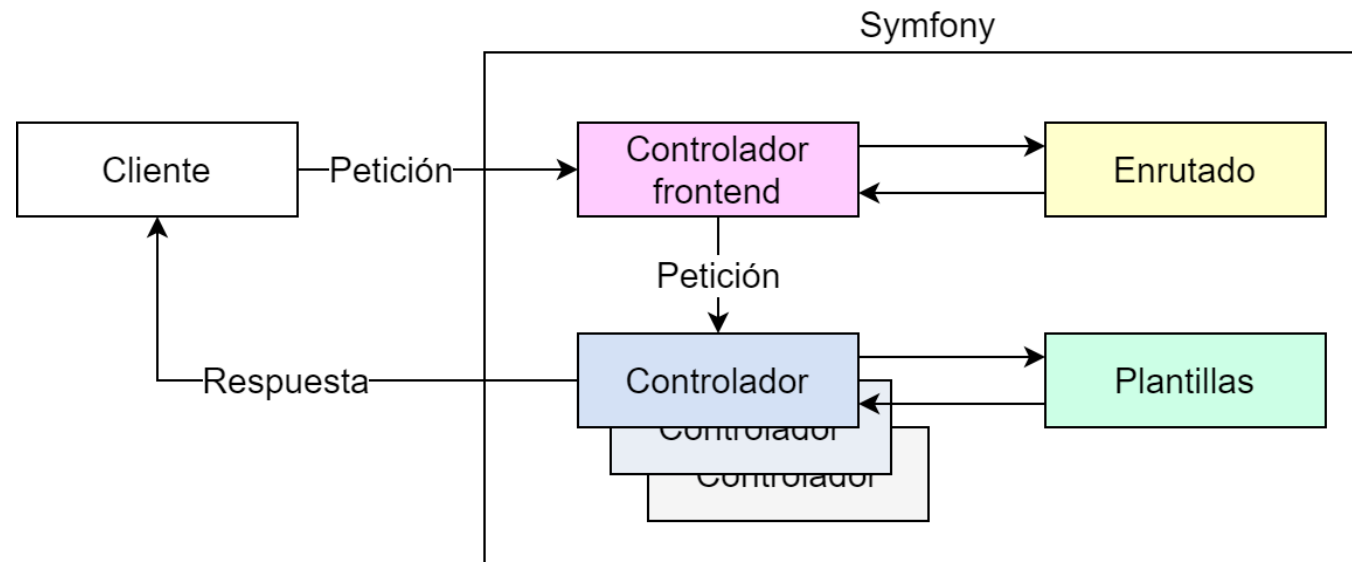
The diagram illustrates the mapping between the route attribute and the controller function. A blue arrow points from the `mostrarEsqui()` function to the `name: 'd_esqui'` attribute in the `#[Route]` attribute. Another blue arrow points from the `methods: ['GET', 'POST']` attribute to the `Optional` box. The `Optional` box is a blue-bordered rectangle containing the word "Optional".

Cada vez que se acceda a la ruta <http://localhost:8000/deporte/esqui> con un método GET o POST, se ejecutará `mostrarEsqui()`

# Enrutado

## Symfony:

- Analiza dinámicamente todos los archivos para saber qué rutas existen.
- Crea un mapa y así decide qué URL corresponde a qué controlador.
- Si incluimos una nueva ruta, es posible que tarde en responder.



# Enrutado

---

## Symfony:

- Analiza dinámicamente todos los archivos para saber qué rutas existen.
- Crea un mapa y así decide qué URL corresponde a qué controlador.
- Si incluimos una nueva ruta, es posible que tarde en responder.

## ¿Cómo puedo ver el mapa de rutas?

```
symfony console debug:router
```

# Enrutado: paso de parámetros EjemploControladores.php

---

En Symfony, en lugar de usar:

```
http://localhost/controlador?param1=val1&param2=val2
```

Usaremos:

```
http://localhost/controlador/param1/param2
```

# Enrutado: paso de parámetros

`http://localhost/controlador/param1/param2`

El controlador asociado será:

```
#[Route('/producto/{num1}/{num2}', name:'prod')]
public function producto($num1, $num2) {
    $producto = $num1 * $num2;
    return new Response("<html><body> " . $producto . "</body></html>");
}
```

Es posible poner valores por defecto a los parámetros.

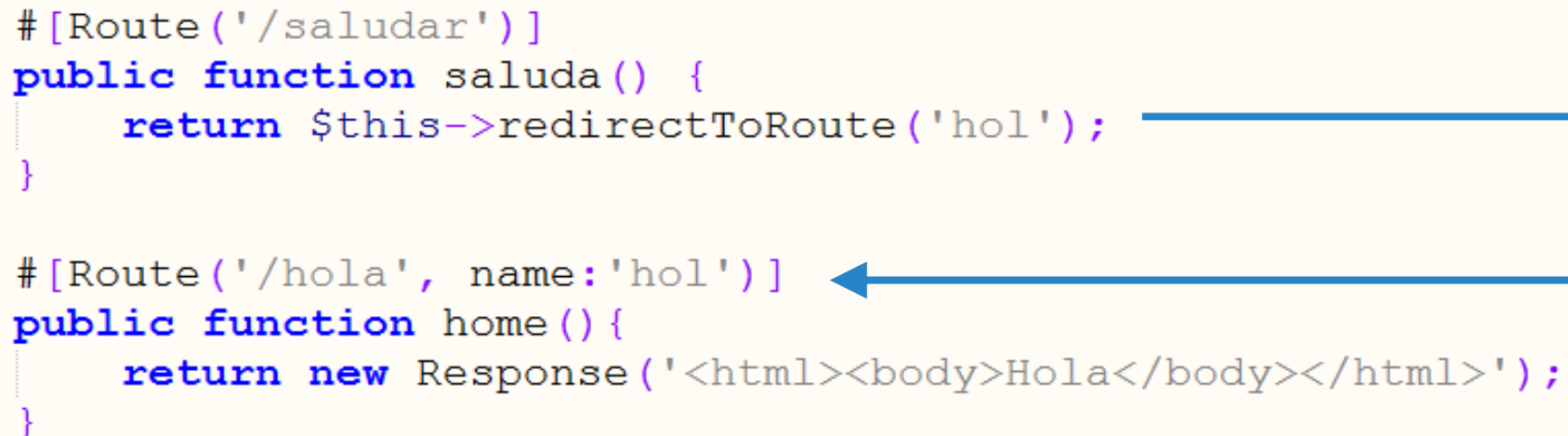


# Enrutado: redirecciones

Los controladores pueden hacer redirecciones.

```
#[Route('/saludar')]
public function saluda() {
    return $this->redirectToRoute('hol');
}

#[Route('/hola', name:'hol')]
public function home() {
    return new Response('<html><body>Hola</body></html>');
}
```

A blue box highlights the 'return \$this->redirectToRoute('hol');' line in the 'saluda' method. A blue arrow points from this box to the 'name:' parameter in the 'home' method's route definition, illustrating the redirect logic.

...y es posible indicar parámetros al otro controlador.

# Enrutado: rutas de clase

---

Las anotaciones `# [Route]` también pueden ir encima de una clase.  
En este caso, la ruta se antepone a las rutas de los controladores.