Desarrollo de aplicaciones web en entorno servidor

CAPÍTULO 8 : DESARROLLO DE APLICACIONES EN SYMFONY

Patrón MVC

- El patrón MVC divide la aplicación en tres capas, el Modelo, la Vista y el Controlador.
- Al desacoplar los elementos de la aplicación se consigue código reusable.
- Además, permite el desarrollo en paralelo, con equipos independientes para cada capa.

Patrón MVC (2)

- Modelo: En el modelo está la lógica de negocio. Se encarga de manejar la base de datos de la aplicación.
- Vista: La interfaz gráfica. Una vista muestra una parte del modelo al usuario.
- Controlador: El controlador se encarga de recoger las acciones del usuario y, en función de estas interactúa con el modelo.

Frameworks para MVC

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

Symfony: Introducción

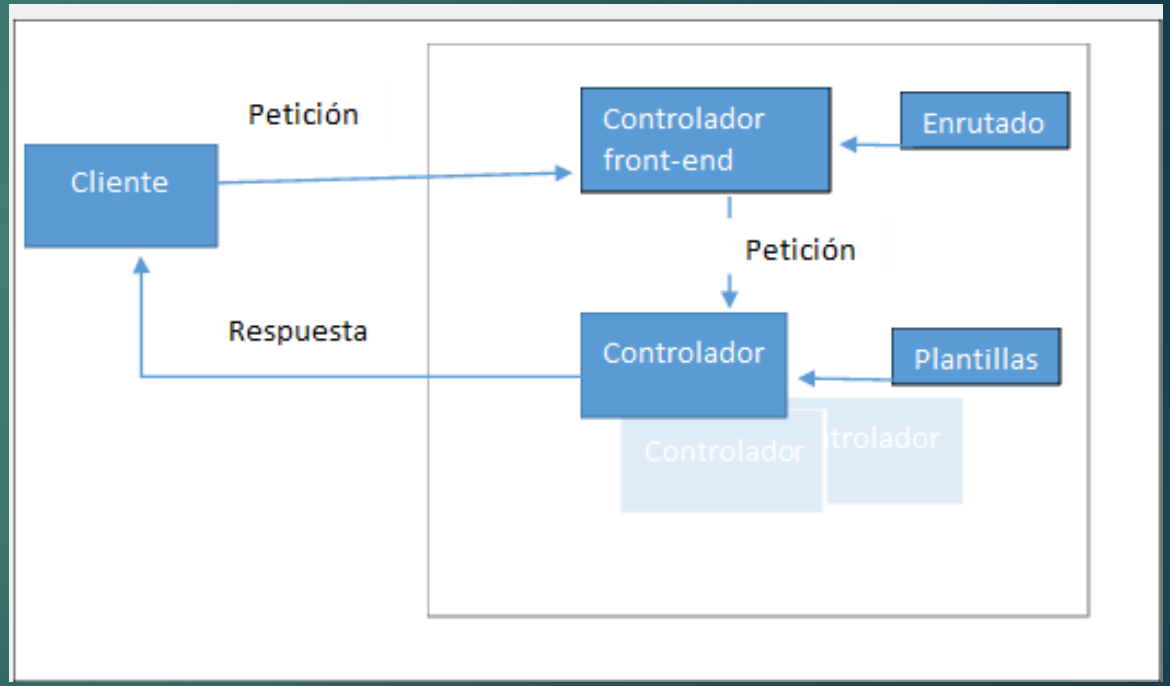
- Symfony es un framework para desarrollo de aplicaciones web en PHP utilizando el patrón MVC.
- Symfony plantea las aplicaciones web de una forma determinada a la que el desarrollador debe adaptarse.
- Incluye componentes para muchas de las tareas habituales, como formularios o seguridad.
- Código abierto.

Symfony: URLs

- La diferencia principal entre las aplicaciones realizadas hasta ahora y Symfony está en cómo se interpretan las URL.
- Por ejemplo, al acceder a localhost/holamundo.php se ejecuta el script **holamundo.php**. Los ficheros son la interfaz entre cliente y servidor.
- En Symfony: Las URL son procesadas por un controlador de *front-end* que analiza las solicitudes para redirigir la petición al controlador adecuado.

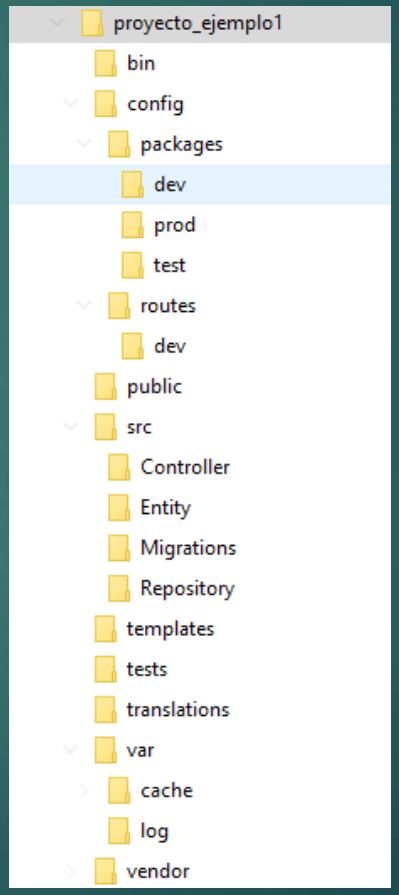
Symfony: enrutado

- Un controlador es un método que recibe la petición del cliente, la procesa y genera una respuesta o un reenvío.
- Los controladares están asociados a una o más rutas, las URLs en las que están disponibles.



Directorios

- Al crear un nuevo proyecto se crea una jerarquía de directorios.
- La ubicación de los ficheros es muy importante.
- Se espera que cada tipo de componente esté en un directorio concreto



Directorios (2)

- Directorio raíz del proyecto. Aquí está el fichero **.env**, dónde se almacena la configuración de base de datos y correo.
- **/config/packages**. Aquí está el fichero **security.yaml**, que contiene la configuración de seguridad.
- **/src/Controller**. Para los controladores.
- **/src/Entity**. Para guardar las entidades.
- **/src/Templates**. Para las plantillas.

Controladores

- Son el elemento principal del desarrollo en Symfony.
- Son métodos que reciben las peticiones del cliente, las procesan y generan una salida o redirección.
- Los controladores suelen ser métodos de una clase. Estas clases se denominan clases controladoras.
- Pueden heredar de la clase **AbstractController**, que tiene varios métodos prácticos.

Rutas

- Asocian las URLs solicitadas con un controlador.
- Se utilizan anotaciones en bloques de comentarios.
- La anotación @route sirve para establecer la ruta en la que estará disponible el controlador.
- Se añade a la ruta base del servidor. Por lo tanto, para acceder al controlador se usaría:

localhost:8000/hola

```
/**
 * @Route("/hola", name="hola")
*/
public function hola(){
    return new
     Response('<html><body>Hola</bo
     dy></html>');
}
```

Parámetros

- En lugar de:

nombre_fichero.php?param1=value1
&param2=value2

- Symfony usa

  nombre_ruta/valor1/valor2

```
/**
 * @Route("/producto/{num1}/{num2}",
     name="producto")
*/
public function producto($num1, $num2){
 $producto = $num1 * $num2;
 return new Response("<html><body>". $producto.
     "</body></html>");
}
```

Valores por defecto

- Poniendo el valor por defecto en la lista de argumentos de la función.
- En la anotación, añadiendo el símbolo '?' y el valor después del argumento.

```
/**
 * @Route("/producto/{num1}/{num2}", name="producto")
*/
public function producto($num1, $num2){
 $producto = $num1 * $num2;
 return new Response("<html><body>". $producto. "</body></html>");
}
/**
 * @Route("/defecto1/{num}", name="defecto1")
 */
public function defecto1($num = 3){
   return new Response("<html><body>" .$num ."</body></html>");
}
```

Plantillas

- En general, la salida de los controladores en Symfony se realiza a través de plantillas.
- Estás plantillas contienen la parte estática de la página y también la lógica de presentación.
- De esta manera, se desacopla la lógica de negocio de la de presentación.
- Pueden recibir argumentos, que son los datos que hay que mostrar.

TWIG: Ejemplo

- La única parte dinámica es **{{ nombre }}**.
- Hace referencia a un parámetro de la plantilla.
- Al mostrar la plantilla, esa parte se sustituirá por el valor del parámetro nombre.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Saludo</title>
    </head>
    <body>
        Hola {{ nombre }}
    </body>
</html>
```

Plantillas: Uso desde un controlador

```
/**
 * @Route("/saludo/{nombre}", name="saludo")
*/
public function saludo($nombre){
  return $this->render('saludo.html.twig', array ( 'nombre' =>  $nombre));
}
```

Rutas en plantillas

- Para introducir una ruta dentro de la plantilla, por ejemplo, en el atributo **href** de un vínculo, se usa la función **path()**.
- Si no tiene parámetros es simplemente:

  {{ path('nombre_ruta') }}
- Si hay parámetros se añaden en un array.

  {{ path('nombre_ruta', {'param1' : valor1, 'param2' : valor2) }}
- **url()** devuelve una ruta absoluta y se usa de la misma forma.

Correo electrónico

- Symfony utiliza la librería SwiftMailer para enviar correos.
- El servidor de correo se configura con la variable MAILER_URL, que está en fichero .env.
- Para usar una cuenta de Gmail se utiliza:

  MAILER_URL=gmail://<usuario>:<contraseña>@localhost

Seguridad

- Symfony cuenta con un componente de seguridad basado en la idea de usuarios y roles.
- Aunque no es obligatorio, resulta muy útil que las aplicaciones lo utilicen.
- El componente de seguridad se configura a través del fichero **/config/packages/security.yaml**.

Usuarios

La entidad que representa a los usuarios debe implementar la interfaz **UserInterface**, con los métodos:

- **getUsername().**
- **getPassword().**
- **getRoles().**
- **getSalt().**
- **eraseCredentials().**