MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO SERVIDOR

TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Generación dinámica de páginas web: Lógica de negocio

07

ÍNDICE

OBJETIVOS

Identificar las ventajas de separar la lógica de negocio de los aspectos de presentación de la aplicación.

Analizar las tecnologías y mecanismos que permiten realizar esta separación y sus características principales.

Utilizar objetos y controles en el servidor para generar el aspecto visual de la aplicación web en el cliente.

Aplicar los principios de la programación orientada a objetos.

/ 1. Introducción y contextualización práctica

El desarrollo de aplicaciones web puede estar formado por diferentes partes que a simple vista podemos diferenciar. Por un lado, quedaría la parte más visual, correspondiente a la interfaz, y por otro, encontramos el código encargado de la implementación del funcionamiento interno, es decir, el tratamiento que se realiza de las peticiones enviadas por los clientes para ofrecerles una respuesta.

En este tema, analizaremos las diferentes capas en las que se divide este tipo de desarrollo, normalmente la capa de presentación, la capa encargada de la lógica de negocio, es decir, la parte más funcional, y finalmente, los controladores. Estos últimos desempeñan un papel fundamental, puesto que son los encargados de derivar la petición para ser respondida adecuadamente.

Para llevar a la práctica la división anteriormente descrita, es común utilizar patrones de diseño, como el método modelo-vista-controlador, que se verá en este capítulo.

Existen, además, aplicaciones que permiten automatizar la creación de proyectos que implementan una estructura separada en capas. Se analizará el funcionamiento de algunas de estas herramientas a lo largo de este tema, en concreto, Symfony y Smarty.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y resolución del caso práctico.

Fig. 1. Imagen de ilustración de servidores.

Audio intro. "¿Es útil la separación de lógicas en un proyecto?"

https://bit.ly/32UdvYp

/ 2. Mecanismos de separación de la lógica de negocio

En programación, la **lógica de negocio** se refiere a aquella parte del sistema encargada de codificar todas las funciones necesarias para que cualquier aplicación o herramienta funcione, y cuya implementación determine, de forma inequívoca, la manera en que la información va a ser creada, almacenada y modificada.

Por lo tanto, cuando hablamos de la separación entre la lógica de negocio y la capa de presentación, nos referimos a la división entre el diseño de las funciones que determinan el funcionamiento "interior" del software y el diseño de la propia página o aplicación, estamos diferenciando entre interior del exterior.

El método de separación de software presenta multitud de **ventajas:**

- Mejora del rendimiento de una aplicación.
- Permite la especialización del software.
- Se consigue el desarrollo de aplicaciones cada vez más robustas.
- Presenta mayor flexibilidad y escalabilidad.
- Proporciona un mantenimiento más sencillo, lo que contribuye a la disminución de costes.
- Uso de distintas tecnologías adecuadas a cada una de las aplicaciones.
- Permite la reusabilidad del código.

Podemos encontrar varios métodos que permiten definir una separación entre la lógica de negocio y el diseño de la presentación: **método cliente-servidor, arquitecturas multicapa y arquitecturas modelo-vista-controlador.**

Uno de los esquemas comunes es el que se muestra en la siguiente imagen. En este, el usuario realizará la petición de una página dinámica que requiere de varios niveles para su construcción: servidor web, servidor de aplicaciones y un servidor de base de datos. Esta es una de las posibles configuraciones que vamos a encontrar hoy en día.

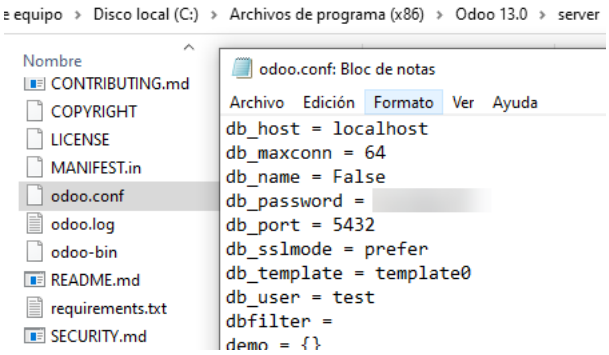

Fig. 2. Modelo de separación de niveles.

2.1. La arquitectura de las tres capas

En la actualidad, para el desarrollo de sitios web, uno de los modelos más utilizados es la arquitectura de las tres capas.

Este tipo de diseño es una de las posibles combinaciones que podemos encontrar dentro de las denominadas arquitecturas multicapa. En este caso concreto, el sistema quedaría estructurado en los siguientes bloques: capa de presentación, capa de negocio y capa de persistencia.

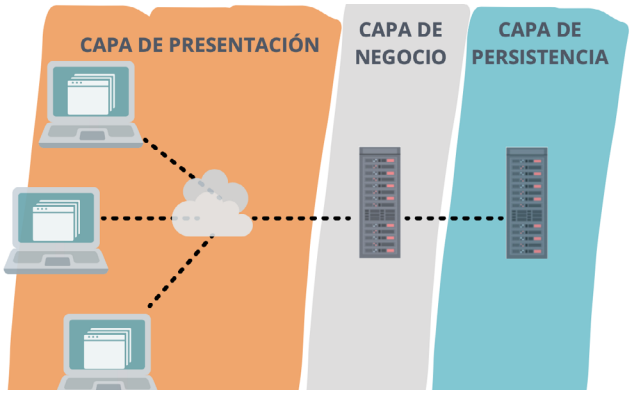

Fig. 3. Identificador LDAP.

- **Capa de presentación:** La capa de presentación es la que se encarga de presentar la parte "visual" del sitio web al usuario, es decir, modela la interfaz gráfica.

  Esta capa también se encuentra conectada con la capa bajo la que se define la lógica de negocio, por lo que también tendrá como función principal la captación y envío de la información para continuar con el proceso definido para el sitio web.

- **Capa de negocio:** Esta segunda capa es la encargada de recepcionar las peticiones enviadas por el cliente y, por lo tanto, será la encargada de determinar la respuesta que se le va a dar a dicha petición.

  Queda comunicada con la capa de presentación, hacia la que redirecciona la respuesta que debe ser mostrada. Por otro lado, también queda conectada a la capa de persistencia u otras que fueran necesarias para la implementación del servicio.

- **Capa de persistencia:** Finalmente, está última capa es la encargada del acceso a datos. Permite mantener los datos necesarios para el correcto funcionamiento de la aplicación. Permite leer, escribir, modificar o borrar los datos en el denominado almacenamiento persistente.

Audio 1. "¿Para qué sirve un WebService?"
https://bit.ly/2EScQyI

/ 3. Tecnologías asociadas a páginas webs dinámicas

Para conseguir el desarrollo de páginas webs dinámicas basadas en sistemas multicapa, es necesario utilizar lenguajes y tecnologías que permiten la programación orientada a objetos entre otras:

- **Programación orientada a objetos en PHP:** Este lenguaje de programación permite organizar el código en diferentes objetos contenedores de datos y funciones.

- **Java:** El lenguaje de programación Java permite trabajar con múltiples tecnologías desarrolladas sobre este, y que posibilita la creación de entornos con este tipo de arquitectura multicapa. Algunas de las tecnologías más utilizadas son: JavaServer Faces, JavaServer Pages, o Enterprise Beans, entre otras.

- **ASP.Net:** Esta tecnología permite el desarrollo de arquitecturas multicapa a través de la herramienta de desarrollo Visual Studio.

Una de las ventajas principales que implica la separación en capas en el desarrollo software es que cada una de estas capas podrá estar implementada en una tecnología diferente, ajustándonos así de forma más óptima a los requisitos de cada una de las partes.

- **Capa de controlador:** Esta capa está implementada habitualmente en el entorno servidor con los lenguajes de programación descritos más arriba (PHP, Java, ASP.NET). Si bien es cierto que, en la actualidad, se utiliza JavaScript para el modelado de páginas web dinámicas. Esto se verá con más en detalle en el siguiente capítulo.

- **Capa modelo de datos:** Habitualmente, se utilizan tecnologías como MySql para el modelado de la capa de datos.

- **Capa vista/presentación:** Finalmente, para la implementación de la interfaz de presentación, van a ser muchas las opciones que vamos a encontrar, desde HTML para entornos web hasta Android para entornos de desarrollo móvil.

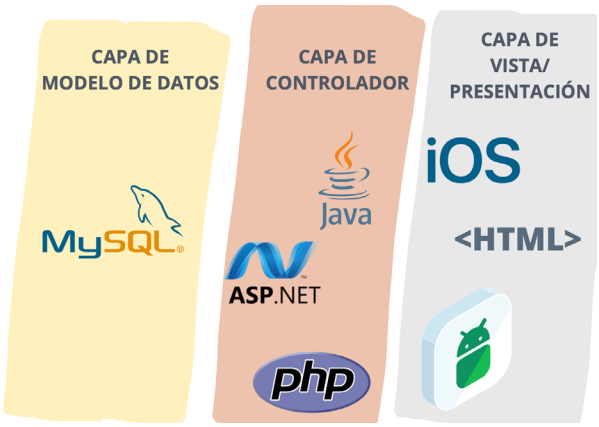

Fig. 4. Tecnologías asociadas por capas.

/ 4. Patrones de diseño

Los patrones de diseño de software son unos "esqueletos" ya probados y validados que permiten el diseño y desarrollo de nuevo software esquematizado de una forma más ágil y eficaz.

Uno de los patones de diseño más extendidos es el llamado Modelo-Vista-Controlador (MVC), el cual divide el código en tres partes claramente diferenciadas que tendrán una función específica.

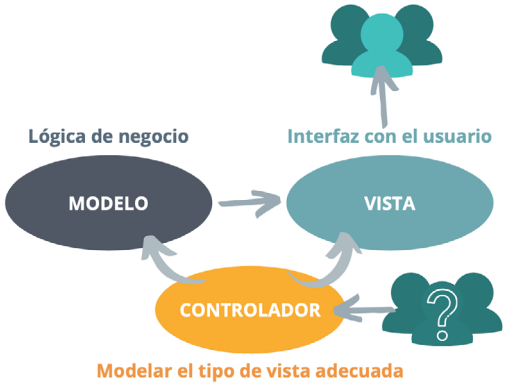

Fig. 5. Modelo-Vista-Controlador.

- **Modelo:** El modelo es la parte que se encarga del manejo de todos los datos, por lo tanto, proporcionará los mecanismos oportunos para la recuperación y modificación de los datos. El modelo representa la lógica de negocio.

- **Vista:** Estos componentes son los que permiten al usuario una interacción directa con la aplicación o sitio web, por lo tanto, las vistas son las interfaces en las que se tiene lugar la interacción entre el usuario y el programa o aplicación.

- **Controlador:** Finalmente, los controladores son los elementos que permiten seleccionar el tipo de vista adecuada para cada usuario. Este se encarga de analizar la petición del usuario y redirigirla para poder generar una repuesta adecuada, que será enviada de nuevo al usuario. Podemos decir que este elemento está entre el modelo y la vista y se encarga de modelar que la interacción y extracción de la información sea la correcta para cada ocasión.

En la actualidad, existen diferentes motores de plantillas web, que no son otra cosa que aplicaciones que permiten la generación de páginas web a partir de un fichero contenedor de la información de presentación (capa vista) y otro fichero con la lógica de la aplicación (capa modelo).

Algunos de los motores de plantillas más conocidos son: Smarty o Symfony.

/ 5. Smarty

Smarty es un motor de plantillas web de código abierto utilizado para entornos de desarrollo en PHP. Se encarga de separar la lógica de negocio (todo lo relativo a la programación y codificación del programa) de la presentación (de la vista de interfaz).

El uso de estas plantillas nos va a permitir separar la parte lógica de la parte “física”. Si hablamos en términos de código, va a separar la parte de PHP de la parte de HTML. Aunque en un primer momento podemos considerar que es algo más tedioso que a lo que estamos acostumbrados, su uso mejora el posterior mantenimiento y actualización de cualquier aplicación que utilice este tipo de tecnología, además, de esta manera, se podrá trabajar de forma paralela entre los desarrolladores de código y los de interfaz.

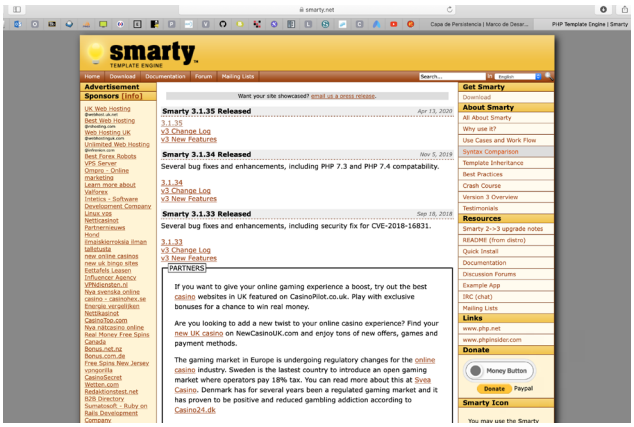

Fig. 6. Sitio web Smarty.

Descarga y configuración de Smarty

El proceso de descarga y configuración de Smarty es común para todos los sistemas operativos, basta con realizar la descarga y colocarlo en carpeta htdocs de XAMPP, la única diferencia será la creación de la variable include_path en php.ini.

Para instalar Smarty en un entorno de desarrollo como el utilizado en esta asignatura, debemos seguir los siguientes pasos:

- En primer lugar, desde la página web de Smarty se realiza la descarga del software. En la pestaña de Download, seleccionamos la última versión y esta nos llevará al repositorio GitHub desde el que concluiremos la descarga. Actualmente, está disponible en este enlace.
- Tras realizar la descarga, se descomprime el contenido y se guarda en la carpeta htdocs.
- Finalmente, desde el fichero de configuración php.ini, se modifica la variable include_path añadiendo la ruta en la que se ha colocado la carpeta de ficheros de Smarty en htdocs.

Cuando se concluye el proceso de instalación, es imprescindible volver a reiniciar el entorno de desarrollo de PHP con el que se está trabajando para que los cambios tengan efecto. Una de las particularidades de Smarty es que utiliza su propio sistema de etiquetas, cuyas equivalencias con el código se verán más adelante.

5.1. Primeros pasos. Parte lógica

En primer lugar, vamos a **crear la estructura de carpetas** necesarias para la organización del código, estas se situarán dentro de la carpeta htdocs. Siempre van a ser creadas las siguientes carpetas, que posteriormente serán referenciadas desde el código, y en caso contrario, se produciría algún error de compilación. Las carpetas utilizadas son: templates, templates_c, configs y cache. También es importante tener en cuenta que se les ha de dotar de permisos de escritura.

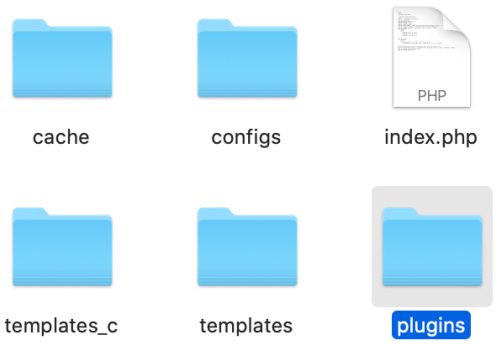

Fig. 7. Organización de carpetas Smarty.

A continuación, vamos a implementar el fichero index.php. El primer paso es vincular cada nuevo proyecto con la librería Smarty descargada. Para ello, siempre hay que incluir la siguiente línea al inicio del fichero de configuración de Smarty (en función de la ubicación de la carpeta libs contenida en Smarty, puede cambiar la ruta).

```
require('../libs/Smarty.class.php');
```

Código 1. Inclusión de librería Smarty.

Si todo este proceso se ha realizado de forma correcta, se creará una nueva instancia del objeto Smarty. En el fichero index.php, se incluye la siguiente instrucción:

```
$newSmarty = new Smarty();
```

Código 2. Instanciación objeto Smarty.

Tras haber realizado la instanciación, se configura el sistema de carpetas de trabajo (con permisos de escritura) requerido para el correcto funcionamiento de Smarty:

```
$newSmarty->setTemplateDir('templates/');
$newSmarty->setCompileDir('templates_c/');
$newSmarty->setConfigDir('configs/');
$newSmarty->setCacheDir('cache/');
```

Código 3. Asignación de carpetas

Finalmente, siempre se incluye la llamada a la parte física donde se modela el código HTML de la capa de presentación:

```
$newSmarty -> display('index.tpl');
```

Código 4. Llamada a parte física

5.2. Configuración de la parte física

Como se veía en el último paso del apartado anterior, para realizar la llamada a través de la función display de la parte física, se incluye el nombre del fichero entrecomillado en el cual se implementa el código HTML.

Los ficheros desde los que se modela la vista del proyecto tomarán la extensión '.tpl', y se van a colocar dentro de la carpeta templates.

La función **assign** permite crear una variable y asignarle un valor. Después, desde el fichero .tpl, o lo que es lo mismo, desde la parte física, se podrá acceder a estas variables utilizando las llaves {} y realizando el llamamiento habitual de una variable en PHP, $nombrevariable.

Vídeo1. "Instalación, configuración y prueba de Smarty"

https://bit.ly/2Z2SVnO

5.3. Pautas de programación con Smarty

En las plantillas desarrolladas con Smarty, en concreto en los ficheros encargados de implementar la interfaz de usuario (con extensión .tpl), existen ciertas diferencias en cuanto al lenguaje de programación utilizado hasta ahora.

- **Variables:** Se sigue utilizando una sintaxis similar para el llamamiento de las variables, se utiliza el símbolo de dólar ($) y, a continuación, el nombre de la variable. Ahora bien, se incluye el uso de las llaves {}, tanto para extraer un valor como para modificar el contenido.

```
{$valor} // Extraer el contenido de la variable
{$valor->otroValor} // Modifica el contenido de la variable
```

Código 5. Variables con Smarty.

- **Estructuras de control:** La sintaxis para la inclusión de este tipo de sentencias es ligeramente distinta. Hablamos de las estructuras de control: if, elseif y else.

```
{if condicion} {*Comprueba la condición*}
acciones
{else} {*Si no se cumple if*}
acciones
{/if}
```

Código 6. Estructuras de control con Smarty.

- **Comentarios:** Los comentarios se introducen entre los delimitadores {* *}.

```
{* Aquí se introduce el comentario*}
```

Código 7. Comentarios con Smarty.

- **Bucle foreach:** Al igual que ocurre con las estructuras de control, este bucle también modifica su sintaxis. Bajo la propiedad from, colocamos el array sobre el que va a iterar el bucle, mientras que se va almacenando cada uno de los valores en item.

```
{foreach from=$arraInicial item=iterador} {*Comprueba la condición*}
acciones
{/foreach}
```

Código 8. Bucle foreach con Smarty.

- **Inclusión de otros ficheros:** El diseño de la interfaz visual también se podrá dividir en varios ficheros. Para ello, basta con incluir todos los fragmentos en el fichero principal, para lo que se utiliza la etiqueta include.

```
{include file="nombreOtroArchivo.tpl"}
```

Código 9. Inclusión de otros ficheros con Smarty.

/ 6. Symfony

Symfony es un framework que **permite el desarrollo en PHP utilizando el patrón de diseño MVC**. Este framework permite optimizar el desarrollo de aplicaciones web separando la lógica de negocio, la de presentación y la lógica de servidor.

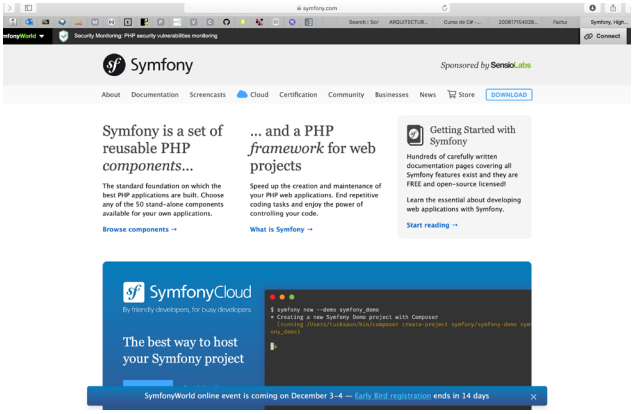

Fig. 8. Sitio web oficial Symfony.

Los controladores son elementos esenciales en este entorno, puesto que se trata de funciones que se encargan de la recepción de las peticiones por parte del cliente, a continuación, las procesan, y, finalmente, devuelven una respuesta o redireccionan la petición al proceso encargado.

HTTP a través de un navegador realiza una petición, esta petición es procesada por el código servidor, que prepara una respuesta y se la envía de nuevo al cliente a través del navegador. Symfony se encargará de la preparación de la respuesta. Una de las características más importantes de este framework es, como hemos adelantado, que proporciona un modelo MVC.

El funcionamiento se basa en un **conjunto de rutas que van a desembocar a un controlador frontal**. Cada una de esas rutas serán asignadas a una acción del controlador concreta. El controlador, a través de los servicios, va a realizar las operaciones oportunas con los datos necesarios para modelar la respuesta final, que se devuelve al cliente.

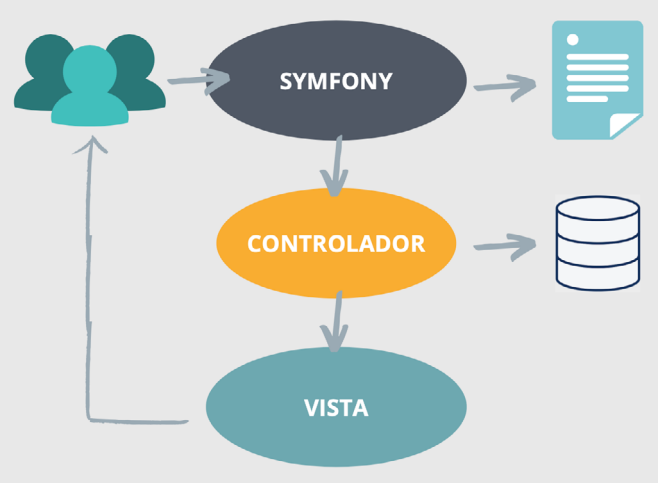

Fig. 9. Diagrama del modelo controlador con Symfony.

6.1. Descarga e instalación de Symfony

Para realizar la descarga del framework de Symfony en cualquiera de los sistemas operativos conocidos (Windows, Linux y Mac OS X), se utiliza la siguiente secuencia de comandos desde el terminal correspondiente o se descarga el ejecutable.

- **Windows:** Desde este sistema operativo, se debe descargar el fichero .exe y ejecutarlo. Este archivo está disponible desde la pestaña Download, aquí.

- **Mac OS X:**

```
$ sudo curl -sS https://get.symfony.com/cli/installer | bash
$ sudo chmod a+x /usr/local/bin/symfony
```

Código 10. Descarga de Symfony en Mac,

- **Linux:**

```
$ sudo wget https://get.symfony.com/cli/installer -O - | bash
$ sudo chmod a+x /usr/local/bin/symfony
```

Código 11. Descarga de Symfony en Linux,

Desde el terminal o cmd, la creación de un proyecto utiliza la secuencia de comandos:

```
$ symfony new --full my_project
```

Código 12. Creación de un nuevo proyecto de Symfony.

Si la instalación se ha realizado con éxito, la creación de un nuevo proyecto utilizando este framework nos devolvería la siguiente salida desde el terminal.

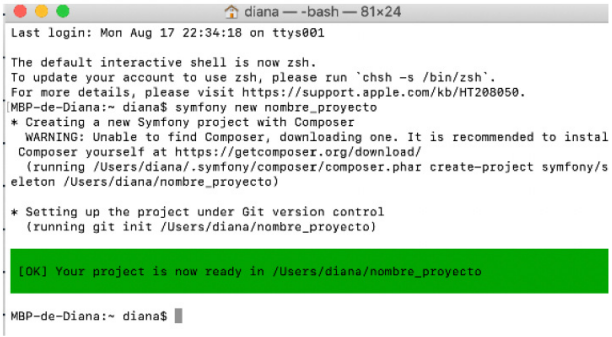

Fig. 10. Terminal con instrucciones para Symfony

A continuación, se inicia el servidor. Para ello se utiliza la siguiente instrucción desde el terminal. Como se puede ver, está escuchando en la dirección 127.0.0.1 y en el puerto 8000. Importante, esta instrucción debe ejecutarse desde la ruta en la que se ha creado el nuevo proyecto.

```
$ symfony server:start
```

Código 13. Arranque de Symfony.

Fig. 11. Terminal con arranque de servidor Symfony.

6.2. Funcionamiento de Symfony

El funcionamiento de Symfony se basa en un conjunto de rutas que van a ser asignadas a una acción de controlador. Por lo tanto, cuando se quiere crear una nueva página, será necesario crear una nueva ruta y, consecuentemente, crear un nuevo controlador.

- En primer lugar, nos situamos en la ruta en la que se ha creado el proyecto utilizando 'symfony new --full my_project'. Recuerda que debes abrir un nuevo terminal, puesto que en el que está el servidor escuchando no lo puedes cerrar ni parar.

- A continuación, se crea el controlador utilizando la secuencia de comandos que sigue. Cuando pida el nombre, basta con introducir el nombre que le queramos asignar a nuestro nuevo controlador.

```
$ sudo php bin/console make:controller
```

Código 14. Creación de controladores.

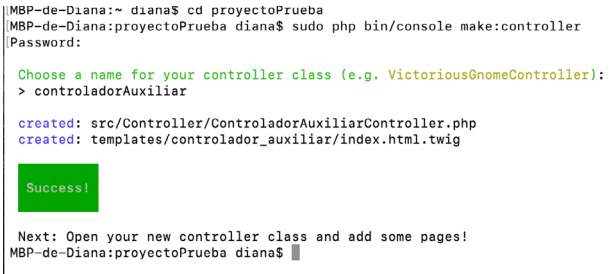

Fig. 12. Terminal con instrucciones para crear controladores Symfony.

- En este punto, ya se habría creado un nuevo controlador con el nombre asignado. Desde la carpeta en la que se almacenan todos los controladores (src/Controller), podremos ver todos los que han sido creados. Se trata de archivos con extensión PHP. Estos ficheros son controladores que contienen funciones.

- Por otro lado, en la carpeta templates, se debe haber creado un fichero con extensión html.twig. Este fichero es el encargado de la capa de presentación, es decir, de la vista.

```
<!DOCTYPE html>
<html>
  <head>
<meta charset="UTF-8">
<title>{% block title %}
Welcome!
{% endblock %}</title>
{% block stylesheets %}{% endblock %}
  </head>
  <body>
    {% block body %}{% endblock %}
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

Código 15. Esqueleto código hmtl.twig controladores.

Vídeo2. "Uso de Symfony"
https://bit.ly/2F15Cbs

/ 7. Caso práctico 1: "Proyecto con Smarty"

Planteamiento: Utilizando el motor de plantillas Smarty, vamos a crear desde cero un nuevo proyecto en el cual se muestre por pantalla "Hola Mundo". Para la implementación, se necesitan dos ficheros. Por un lado, el encargado del modelar la lógica de negocio (PHP), y por otro lado, el que modela la parte de presentación (escrito utilizando la sintaxis propia de Smarty).

Fig. 13. Resultado proyecto con Smarty.

Nudo: Como se ha expuesto en el enunciado, se van a crear dos ficheros que van a estar ubicados en las siguientes rutas:

- interfazPresentacion.tpl en el directorio templates.
- index.php en el directorio principal del proyecto.

Desenlace: La implementación de los ficheros es la siguiente:

- **interfazPresentacion.tpl**

```
{$cadenaTexto}
```

Código 16. Código capa presentación.

- **Index.php**

```
<?
require_once("Smarty/libs/Smarty.class.php");

$miSmarty = new Smarty();

$miSmarty->template_dir = 'templates';
$miSmarty->config_dir = 'config';
$miSmarty->cache_dir = 'cache';
$miSmarty->compile_dir = 'templates_c';

$miSmarty->assign("cadenaTexto", "Hola mundo con Smarty");
$miSmarty->display("interfazPresentacion.tpl");
?>
```

Código 17. Código lógica negocio

/ 8. Caso práctico 2: "Creación de un nuevo controlador"

Planteamiento: Gracias al framework Symfony, es posible crear proyectos con un eficiente modelo-vista-controlador. Uno de los elementos clave es la creación de componentes que permiten enlazar las peticiones hacia aquel servicio que va a resolver el caso y devolver la respuesta. Se diseña un nuevo controlador y se pide que el mensaje que este muestre cuando es invocado sea "Hola, soy un controlador y funciono con éxito".

Nudo: Cada nuevo controlador se crea utilizando la instrucción "sudo php bin/console make:controller" y, a continuación, se especifica el nombre de dicho controlador, en este caso, ControlPractico1, por lo que los ficheros que se generan son:

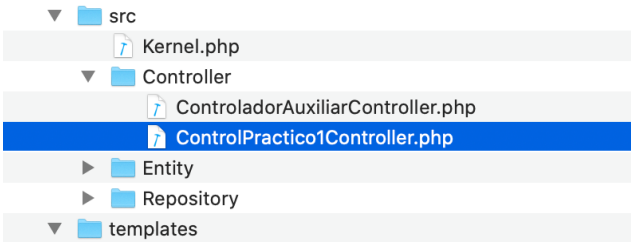

Fig. 14. Fichero controlador html.twig.

Se crea un fichero en PHP con el nombre ControlPractico1Controller.php con la lógica de negocio, y para la vista de diseño, se crea una nueva carpeta con el nombre asignado al controlador "ControlPractico1", dentro de la cual se sitúa el fichero HTML.

Desenlace: El código modificado en HTML, para que se muestre el mensaje indicado en el planteamiento, es:

```
<style>
.example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
.example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
<h1>Hola, soy un controlador y funciono con éxito ! </h1>

This friendly message is coming from:
<ul>
    <li>Your controller at <code><a href="{{ '/Users/diana/proyectoPrueba/src/Controller/ControlPractico1Controller.php'|file_link(0) }}">src/Controller/ControlController.php</a></code></li>
    <li>Your template at <code><a href="{{ '/Users/diana/proyectoPrueba/templates/controlPractico1/index.html.twig'|file_link(0) }}">templates/control/index.html.twig</a></code></li>
</ul>
</div>
```

Código 18. Código controlador Caso Práctico 2

/ 9. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, se han puesto en práctica varios patrones y framework de diseño que permiten la separación de la lógica de negocio y la capa de presentación, analizando las distribuciones más comunes en entornos de desarrollo web.

Hemos conocido a Symfony, que supone un potente framework de diseño que permite la creación de proyectos basados en un patrón MVC. Una de las características principales de Symfony es la creación de controladores que permiten el tratamiento de peticiones en el servicio adecuado y su posterior envío de respuesta.

También hemos usado Smarty, un patrón de plantillas de diseño que permite, a través de la generación de una estructura de carpetas, organizar el código.

Resolución del caso práctico inicial

Como hemos visto en este tema, la separación de la lógica de negocio del resto de capas resulta clave en el desarrollo actual de sitios web. Como repuesta a las preguntas planteadas en el audio inicial:

¿Si no se utilizan los modelos que permiten separar la lógica de negocio de la de presentación qué ocurre?

Si no se utiliza este tipo de modelos, lo habitual será encontrar toda la lógica de la aplicación en un único fichero PHP, desde la parte correspondiente a la lógica de negocio hasta la encargada de la capa de presentación.

¿Cuál será una de las principales desventajas de colocar todo el código en un mismo fichero o módulo?

Una de las principales desventajas es que el error en una de estas capas puede suponer el fallo en el resto, además, este tipo de modelos solo va a funcionar si la base de datos es MySql. Por ello, se propone su división en base al patrón MVC.

¿Cómo se puede implementar esta separación?

Por un lado, podemos crear un nuevo fichero en el cual se modela la capa de presentación de la aplicación. Esta capa muestra al cliente la respuesta a una petición. Esta vista queda almacenada en un fichero “resultado.php”, el cual es llamado por el fichero que modela al controlador de la aplicación, este se encarga de manejar la lógica de negocio y de devolver el resultado mediante la capa de presentación.

/ 10. Bibliografía

Ganzabal, X. (2019). *Desarrollo web en entorno servidor* (1.a.ed.). Madrid, España: Síntesis

Vara, J.M. (2012). *Desarrollo en entorno servidor* (1.a ed.). Madrid, España: Rama.