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
- Generar una primera base de datos para el almacenamiento de las obras del museo

Inicio: Semana del 19/03/2018
Entrega: Semana del 16/04/2018
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 contenidos (obras, 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 y preparar la página para mostrar los contenidos que se encuentren almacenados en dicha BD.

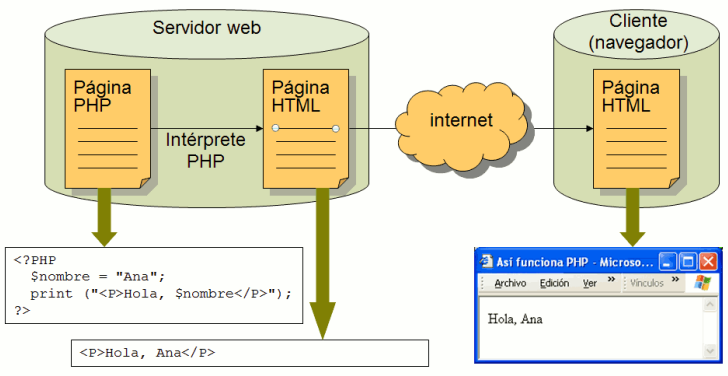

Figura 1: Esquema de funcionamiento de servidor web con PHP

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 los script generan.

3. Desarrollo de la práctica

En las páginas webs 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...).

La mejor forma de generar contenido de webs complejas sin tener que repetir código, es hacer que nuestro archivo index.php delegue la generación del contenido HTML en distintos scripts, es decir, que index.php u otro script en quien este delegue, actúe como *CONTROLADOR*. Por ejemplo:

- En index.php no se genera ningún código HTML, pero sí se centralizan las actividades de backend (consultas a BD, lógica del programa, cálculos...);
- En plantillaObra.php generaremos el HTML para la antigua página obra.html (probablemente incluyendo algunos o todos los ficheros siguientes);
- En header.php generaremos el HTML para la cabecera y el menú;
- En footer.php generaremos el HTML para el pie de página;
- En sidebar.php generaremos el HTML para los elementos de la barra lateral;
- En obra.php generaremos el HTML para mostrar el contenido central de las páginas de las obras.

Evidentemente esta estructura es flexible y cada uno deberá adaptarla a su caso particular añadiendo los scripts y funciones que considere necesarias para, por ejemplo, envío de e-mails, control de los banners, gestión de la BD, etc.

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

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 plantillaObra.php:

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

- Para una correcta separación entre la **Vista** y el **Controlador** lo normal es usar algún *motor de plantillas*. Para `PHP` exiten varios, como por ejemplo `Twig` (`https://twig.symfony.com/`), pero en esta práctica no vamos a usarlo.
- 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/?obra=12` generase internamente una consulta a la BD del estilo `SELECT * FROM obras WHERE id=12` tenemos que asegurarnos que el valor de la variable `obra` es un número correcto. Si no hacemos dicha validación y alguien solicita la siguiente URL maliciosa: `http://localhost/?obra=;DELETE FROM obras` 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).

- Para evitar errores comunes cuando generamos los scripts de plantillas no es conveniente separar la apertura de una etiqueta y su cierre en distintos archivos. Por ejemplo:

Fichero `plantilla.php`

```
<?php
  include("cabecera.php");

  include("cuerpo.php");
?>
```

Fichero `cabecera.php`

```
<html>
  <head>
    <title>LALALA</title>
  </head>
```

Fichero `cuerpo.php`

```
  <body>
    <p>LOLOLO</p>
  </body>
</html>
```

En este caso la etiqueta `<html>` se abre en `cabecera.php` y se cierra en `cuerpo.php`. Una solución mucho mejor sería:

Fichero `plantilla.php`

```
<html>
  <?php
    include("cabecera.php");

    include("cuerpo.php");
  ?>
</html>
```

Fichero `cabecera.php`

```
  <head>
    <title>LALALA</title>
  </head>
```

Fichero `cuerpo.php`

```
  <body>
    <p>LOLOLO</p>
  </body>
```

6. Desarrollo de las prácticas

- **Sesión 1:** Análisis de la estructura arquitectónica
- **Sesión 2:** Resolución de dudas en clase
- **Sesión 3:** Proposición de mejoras. Resolución de dudas.

7. En el aula de prácticas

En el aula de prácticas, iniciando la imagen `lsiweb`, tendréis funcionando el servidor `Apache`, que es el encargado de interpretar las órdenes de los scripts `PHP`.

En vuestra unidad `U:` tenéis el directorio `u:\xampp\htdocs` donde se colgarán los archivos que posteriormente podrán verse en el navegador utilizando la url `http://localhost`

8. En vuestro ordenador

Deberéis instalar tanto `Apache` como `PHP`, y pensando en prácticas posteriores, también `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`).

9. Bibliografia básica

- `http://www.w3schools.com/php/`

10. Evaluación de la práctica

Se tendrán en cuenta los siguientes aspectos:

- Arquitectura del sistema: **MVC** (15 %)
- Identificamos al menos 3 tipos de elementos que pueden ser accesibles: (15 %)
  - **Obra**: Con la información que ya se ha comentado en prácticas anteriores. No hace falta añadir más de 5 o 6 obras: las necesarias para que los listados que se muestren tengan un aspecto razonable.
  - **Página de información general**: Son páginas con información general sobre el museo, como por ejemplo, la página de contacto, página de localización del museo, precios de entradas, etc. El contenido de dichas páginas debe estar guardado en la BD.
  - **Colecciones**: Cada obra estará asignada a una colección, de manera que podremos mostrar un listado de obras que pertenecen a una colección concreta.
- Usa `GET` con un `id` de **obra**, **páginas** o **colección**.
- Respecto a los comentarios, en la BD se guardan los siguientes datos: (5 %)
  - Dirección `IP` utilizada
  - Nombre
  - Correo electrónico del usuario que hace el comentario
  - Fecha y hora del comentario
  - Texto del comentario
- En la BD se guarda la lista de palabras prohibidas y se usa para la funcionalidad correspondiente (2 %)
- Usa `POST` para datos del formulario (3 %)
- Validación en servidor de TODAS las variables `GET` y `POST` (independientemente de las validaciones `Javascript`, que un atacante se puede saltar fácilmente). (10 %)
- En botones de `TW` y `FB` se muestra una ventana emergente de `Javascript` con el siguiente mensaje: (5 %)

``Se publicará en Twitter (o Facebook, según corresponda) el siguiente mensaje:
TEXTO DEL MENSAJE''

La ventana se cerrará pulsando dentro de ella en una opción denominada `Aceptar`.

- El `TEXTO DEL MENSAJE` anterior contiene:
  - El título de la obra
  - Vía @elnombredevuestromuseo
  - La foto de la obra
- En la página `obra_imprimir`: (5 %)
  - Arriba a la izquierda aparece el logotipo
  - La obra se muestra con foto grande a todo lo ancho
  - El texto está a dos columnas
  - Si hay vídeos, se pone su URL
- En cada obra se ve la fecha de publicación y la fecha de última modificación (3 %)
- Se cuida la estética, con contenido y texto limpio (sólo se permite negrita), y se evitan otros aspectos de formato en el texto (2 %)
- Se cuida la seguridad del sistema: (10 %)
  - 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 sitios
  - 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)
- Hay galería de fotos (en al menos una obra) (5 %)
- Usa clases y métodos de clase (2 %)
- La información de la BD está en tablas bien estructuradas (3 %)
- Uso de sesiones/`HTML5` storage (5 %)
- Menú dinámico (a partir de datos en la BD) (5 %)
- Hay posibilidad de incluir vídeos (5 %)
- Extras (10 %)