UD1. Selección de arquitecturas y herramientas de programación

1 .Arquitectura Cliente-Servidor:

- La arquitectura cliente/servidor está basada en la idea de **servicios**
- El cliente solicita servicios (**REQUEST**)
- El servidor es un proceso proveedor de dichos servicios (**RESPONSE**)
- La comunicación entre ambos se realiza mediante el intercambio de mensajes.
- El cliente, a través de un navegador, inicia el intercambio de información, solicitando datos al servidor.
- El servidor responde enviando uno o más flujos de datos al cliente.

La arquitectura Cliente servidor contempla varias **capas**:

- **Presentación:** Esta capa se comunica únicamente con la capa de negocio
  - Es la capa que interactúa con el usuario
  - Conocida como interfaz gráfica
  - Debe ser «amigable» para el usuario
  - Esta capa se comunica únicamente con la capa de negocio

- **Lógica o de Negocio:** Es la capa intermedia, Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, y con la capa de persistencia, para solicitar al gestor de base de datos almacenar o recuperar datos de él.
  - Es donde residen los programas que se ejecutan.
  - Recibe las peticiones del usuario y se envían las respuestas tras el proceso
  - Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse.

- **Persistencia:** Es la capa donde se encuentran los datos almacenados para generar información.
  - Donde residen los datos.
  - Encargada de acceder a los mismos.
  - Formada por uno o más SGBD que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Ejemplo 3 capas:

UD2. Inserción de codigo PHP en paginas web.

1. Tecnologias de servidor:

Según el modo de procesar las peticiones del cliente, podemos clasificar los servidores web en:

Servidor basado en Procesos:

1. Un proceso principal escucha posibles peticiones de clientes
2. Cuando llega una petición, el proceso se duplica creando una copia exacta (*fork*)
3. La copia atiende la petición mientras el proceso principal sigue escuchando nuevas peticiones

Servidor basado en Hilos

1. Un proceso principal escucha posibles peticiones de clientes
2. Cuando llega una petición, el proceso crea un Hilo de ejecución (*thread*) que comparten el mismo espacio de memoria (interbloqueo)
3. El hilo atiende la petición mientras el proceso principal sigue escuchando nuevas peticiones

Servidor dirigido por eventos

1. Se basa en *sockets* no bloqueantes.
2. Socket: espacio de memoria para compartir entre procesos de dos aplicaciones en dos máquinas distintas (cliente/servidor)
3. Las lecturas/escrituras entre sockets son asíncronas y bidireccionales
4. Se identifican mediante IP:puerto
5. La concurrencia de procesamiento es simulada: hay un único proceso y un sólo hilo atendiendo las conexiones gestionadas por sockets.

Servidor implementado en el Kernel

1. Se usa un espacio de trabajo perteneciente al SO y no en el área de usuario
2. En la práctica o mundo real, tiene muchos problemas e inconvenientes
3. Cualquier problema que se produce a nivel de Kernel puede inutilizar el SO

2. Etiquetas para inserción de código

<?php // apertura
php **?> // cierre**

https://www.php.net/manual/es/ini.list.php

3. Constantes

Las constantes no se asignan con el operador =, sino con la función define :
define(nombre_constante_entre_comillas, dato_constante);
define (“PI”, 3.1416);
print PI;

- No llevan $ delante
- La función defined(“PI”) devuelve TRUE si existe la constante.
- Son siempre globales por defecto.
- Sólo se pueden definir constantes de los tipos escalares (boolean, integer, double, string)

4. Constantes Predefinidas

- **PHP_VERSION: Indica la versión de PHP que se está** utilizando.
- PHP_OS: Nombre del sistema operativo que ejecuta PHP.
- TRUE
- FALSE
- E_ERROR: Indica los errores de interpretación que no se pueden recuperar.

● E_PARSE: Indica errores de sintaxis que no se pueden recuperar.
● E_ALL: Representa a todas las constantes que empiezan **por E_.**

5. Variables Superglobales

● **$_GET**: lleva los datos de forma "visible" al cliente (navegador web). El **medio de envío es la URL**. Para recoger los datos que llegan en la url se usa $_GET. Ejemplo: www.midominio.com/action.php?nombre=paco&apellidos1= gomez

● **$_POST**: consiste en datos "ocultos" (porque el cliente no los ve) **enviados por un formulario** cuyo método
de envío es post. Ideal para formularios. Para recoger los datos que llegan por este método se usa $_POST.

● **$_REQUEST**: Con la variable $_REQUEST **recuperaremos los datos** de los formularios enviados tanto por **GET como por POST**.

6. Tipos de datos

● La función **gettype()** devuelve el tipo de una variable.

● Las funciones **is_type** comprueban si una variable es de un tipo dado: is_array(), is_bool(), is_float(), is_integer(), is_null(),
is_numeric(), is_object(), is_resource(),
is_scalar(),is_string()

● La función **var_dump()** muestra el tipo y el valor de una variable. Es especialmente interesante con los arrays.

7. Expresiones y Operadores

● **Operador de identidad === :** Compara también el tipo de las variables.

UD4. Procesamiento de Formularios

1. Protocolo HTTP

El protocolo HTTP es un protocolo sin estado (no guarda informacion o estado del cliente)→[usar coockies o variables ocultas en formularios]. La comunicacion dura lo necesario para resolver la peticion del cliente.

2. Formularios

Puesto que se utiliza el protocolo HTTP, estamos limitados por su interfaz, solo se puede utilizar algunos de los comandos del protocolo para establecer la comunicacion: **GET o POST**

● Dos tipos diferentes de peticiones, segun atributo method del <FORM>:
✔ Peticiones GET (metodo GET de HTTP)
✔ Peticiones POST (metodo POST de HTTP)
<FORM ACTION="nombreFichero.php" METHOD="post/get">

3. Formularios - Peticiones GET

Los parametros se indican en la URL tras el signo “?” y se concatenan con & indicando variable=valor.
● En el servidor, los valores se guardan en el array asociativo **$_GET**. La URL que se genera es similar a:
http://site/procesa.php?name1=value1&name2=value2

● Reglas de codificacion URL:
✔ RFC 3986
✔ Los caracteres especiales se envian codifican con el formato %NN (NN: valor hexadecimal de caracter). El servidor se encarga de decodificarlo
● Son caracteres especiales:
✔ N, n, a, etc. (no tienen un caracter US ASCII asociado)
✔ Los peligrosos: “**:**” “**/**” “**#**” “**?**” “**[**“ “**]**” “**@**”
✔ Los reservados con significado especial: “**!**” “**$**” “**&**” “**‘**” “**(**“ “**)**” “**-**” “**+**” “**,**” “**;**” “**=**”

4. Formularios - Peticiones POST

Los parametros se envian en el cuerpo del mensaje (no url).
● El servidor, almacena los valores en el array asociativo **$_POST**. Los caracteres especiales se traducen a ASCII.
● Es necesario indicar en el <form> el tipo de codificacion para enviar datos al servidor con el atributo ***enctype:***

✔ **application/x-www-form-urlencoded** (Por defecto). Los caracteres se codifican antes de ser enviados (espacios se convierten en “+” y caracteres especiales a %NN). NO PERMITE ENVIAR ARCHIVOS

✔ **multipart/form-data** No se codifican caracteres, Se requiere en forms que envian ficheros. PERMITE ENVIAR ARCHIVOS

✔ **text/plain** Los espacios se convierten a “+” pero los caracteres especiales no se codifican

5. Información del cliente Web

Se obtiene información de distintos tipos de controles en un formulario:
● **Elementos de tipo INPUT**
✔ TEXT (caja de texto)
✔ RADIO (boton seleccion)
✔ CHECKBOX (check de seleccion)
✔ BUTTON
✔ FILE (para seleccion de archivos a enviar)
✔ HIDDEN (control oculto para obtener datos)
✔ PASSWORD
✔ SUBMIT / RESET (boton enviar/limpiar formulario)
● Elemento SELECT (desplegable)
✔ Simple (solo un elemento)
✔ Multiple (varios elementos a la vez)
● Elemento TEXTAREA (caja texto multiples lineas

TEXT (caja de texto)

Introduzca la cadena a buscar:
<INPUT TYPE="text" NAME="cadena" VALUE="valor por defecto" SIZE="20">

```
<?php
$cadena = $_REQUEST['cadena'];
echo $cadena;
?>
```

RADIO (boton selección)

Sexo:
<INPUT TYPE="radio" NAME="sexo" VALUE="M"
CHECKED>Mujer
<INPUT TYPE="radio" NAME="sexo" VALUE="H">Hombre
<?PHP
$sexo = $_REQUEST['sexo'];
echo ($sexo);
?>
Los radiobutton se llaman igual para que si se elige uno, se desmarque el resto

CHECKBOX (check de selección)

<INPUT TYPE="checkbox" NAME="extras[]" VALUE="garaje"
CHECKED>Garaje
<INPUT TYPE="checkbox" NAME="extras[]"
VALUE="piscina">Piscina
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="jardin">Jardin
<?PHP
$extras = $_REQUEST['extras'];
foreach ($extras as $extra)
echo "$extra
\n";
?>
Los check tienen el mismo nombre y se almacenan en arrays

BUTTON

<INPUT TYPE="button" NAME="actualizar"
VALUE="Actualizar datos">
<?PHP
$actualizar = $_REQUEST['actualizar'];
if ($actualizar)
print ("Se han actualizado los datos");
?>

FILE (para seleccion de archivos a enviar)

<FORM ACTION="procesa.php" METHOD="post"
ENCTYPE="multipart/form-data">
<INPUT TYPE="file" NAME="fichero">
</FORM>
Este tipo se detalla mas adelante en la subida de archivos al servidor

PASSWORD

Contrasena: <INPUT TYPE="password"
NAME="clave">
<?PHP
$clave = $_REQUEST['clave'];
echo $clave;
?>
Este control no muestra los caracteres introducidos
sino que los oculta mostrando puntos

SUBMIT / RESET (boton enviar/limpiar formulario)

<INPUT TYPE=submit NAME="enviar" VALUE="Enviar datos">
<?PHP
$enviar = $_REQUEST['enviar'];
if ($enviar)
echo "Se ha pulsado el boton de enviar";

?>
<INPUT TYPE=reset NAME="borrar" VALUE="Limpiar datos">
<?PHP
$borrar = $_REQUEST['borrar'];
if ($borrar)
echo "Se ha pulsado el boton de limpiar datos";
?>

SELECT SIMPLE

Color:
<SELECT NAME="color">
<OPTION VALUE="rojo" SELECTED>Rojo</OPTION>
<OPTION VALUE="verde">Verde</OPTION>
<OPTION VALUE="azul">Azul</OPTION>
</SELECT>
<?PHP
$color = $_REQUEST['color'];
echo $color;
?>

SELECT MULTIPLE

Idiomas:
<SELECT MULTIPLE SIZE="3" NAME="idiomas[]">
<OPTION VALUE="ingles" SELECTED>Ingles</OPTION>
<OPTION VALUE="frances" SELECTED>Frances</OPTION>
<OPTION VALUE="aleman">Aleman</OPTION>
<OPTION VALUE="holandes">Holandes</OPTION>
</SELECT>
<?PHP
$idiomas = $_REQUEST['idiomas'];
foreach ($idiomas as $idioma)
echo "$idioma
\n";?>
Por defecto solo se accede a un valor

Elementos de tipo TEXTAREA

Comentario:
<TEXTAREA COLS="50" ROWS="4" NAME="comentario">
Este libro me parece ...
</TEXTAREA>
<?PHP
$comentario = $_REQUEST['comentario'];
echo $comentario;
?>

UD4B. Procesamiento de Formularios

1. Validar envío de formulario

Para saber si se ha enviado el formulario se acude a la variable correspondiente al boton de envio.
Si este boton aparece de la siguiente forma en el formulario HTML:
<INPUT TYPE=SUBMIT NAME="enviar" VALUE="procesar">
entonces la condicion anterior se transforma en:
if (isset($_POST['enviar'])
o bien

if ($_POST['enviar'] == "procesar")

2. Validación formulario sin etiquetas html

Función **strip_tags($variable)**. *Elimina las etiquetas HTML y PHP de la variable*
Ante la entrada de texto Paco<Ruiz>
El código:

```
<?php
print "<p>Su nombre es " .
strip_tags($_REQUEST["nombre"]) . "</p>\n";
?>
```

Obtendría el resultado
Su nombre es Paco //Ruiz no lo mostraría por identificarlo como etiqueta

3. Validar que no hay espacios en blanco en el texto

Función
trim($variable). *Elimina los espacios en blanco antes y después del texto*.
Ante una entrada de usuario con espacios en blanco
El código

```
<?php
if (trim($_REQUEST["nombre"]) == "") {
print "<p>No ha escrito ningún nombre</p>\n";
} else {
print "<p>Su nombre es ". trim($_REQUEST["nombre"]) . "</p>\n";
}
?>
```

Devolvería: No ha escrito ningún nombre