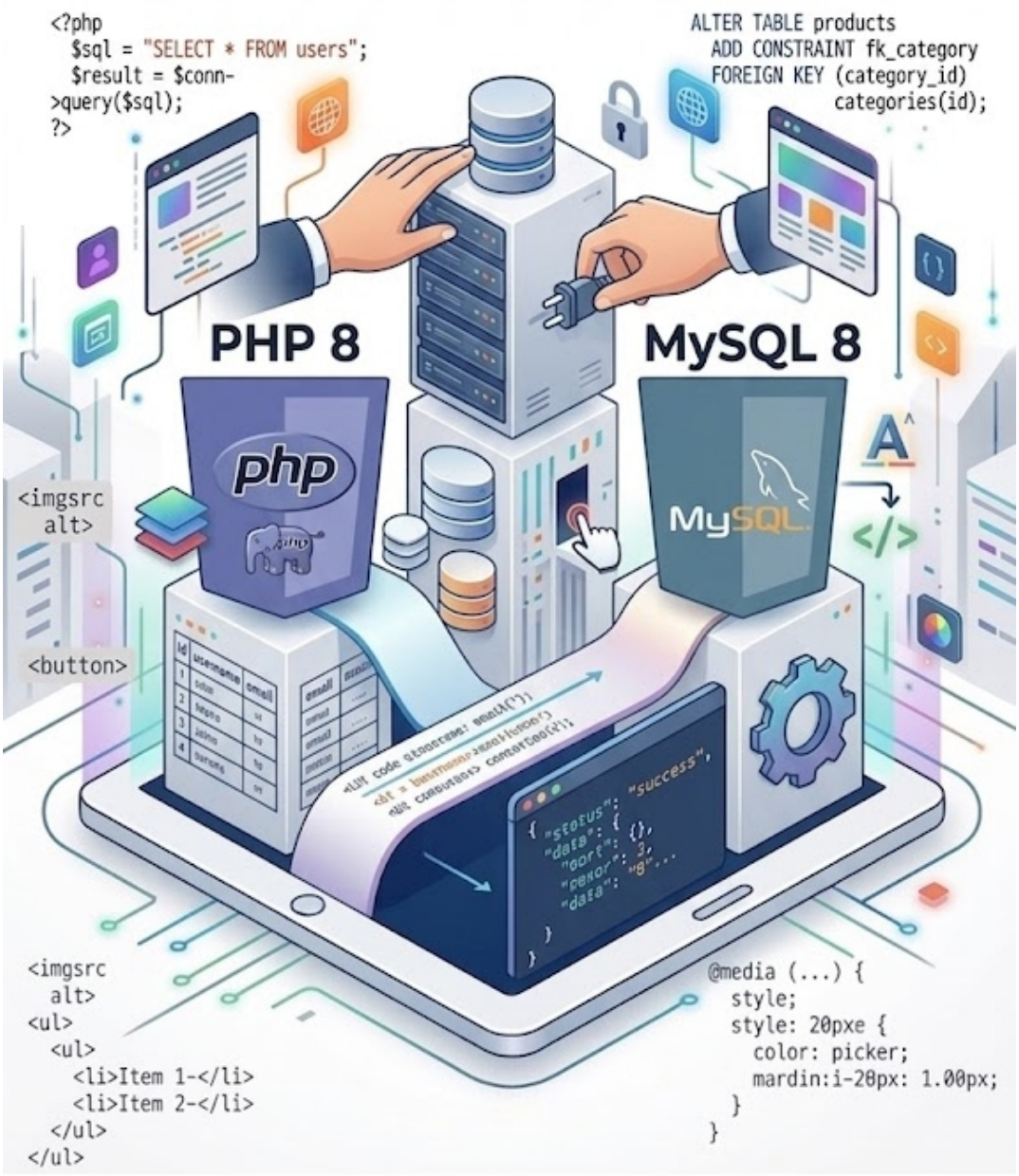
PHP & MySQL

GUÍA DE DESARROLLO WEB BACK-END

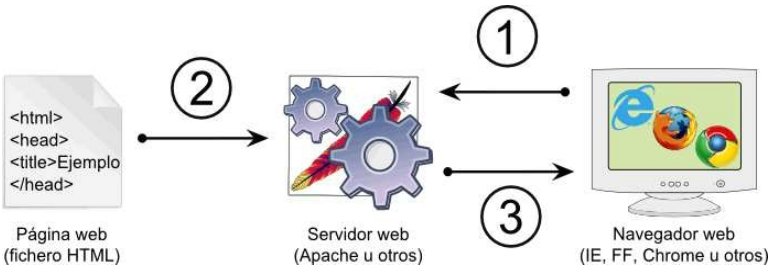


IES Fco. Grande Covián
José Antonio Sallán Arasanz
Informática II - Versión 1.0.0

1 Arquitectura Cliente-Servidor y el Entorno de Ejecución

Hasta ahora, el desarrollo web que hemos realizado con HTML y CSS se ha centrado en el **Frontend** (el lado del cliente). El propio navegador web (Chrome, Firefox) es el encargado de leer el archivo, interpretar las etiquetas y dibujar la página en pantalla.

Ese tipo de sitios se conocen como ***sitios web estáticos***. Muestran información permanente. En ellos, el navegante se limita a obtener dicha información, sin poder interactuar con el sitio web visitado. En esquema:



Sin embargo, en un sitio web como una tienda online o cualquier red social, es necesario procesar información, tomar decisiones, guardar datos. No se muestra siempre la misma información, el resultado depende de los datos que el usuario envíe con la petición al servidor. En ese caso estaremos hablando de **sitios web dinámicos**.

Para construir este tipo de sitios se utiliza una arquitectura conocida como **Cliente-Servidor** y lenguajes de **Backend** como **PHP**.

¿Qué es un "Servidor"?

En informática, el término "servidor" tiene un doble significado que es fundamental distinguir:

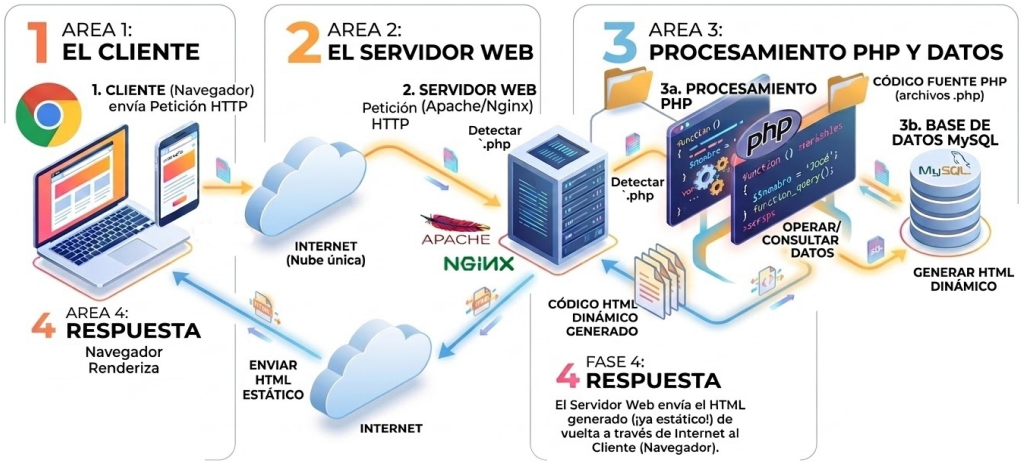
- **Servidor Físico (Hardware):** Ordenador de alto rendimiento, diseñado para estar encendido ininterrumpidamente (24/7) y conectado a Internet con un gran ancho de banda. Se almacenan en grandes centros de datos (Data Centers).
- **Servidor Web (Software)**: Programa informático que se ejecuta dentro de ese ordenador físico (los más utilizados son Apache o Nginx).

Su función es "escuchar" a través de la red, esperando a que un usuario pida una página web, para procesarla y enviarla de vuelta.

El Ciclo de Vida de una Petición Web (Backend)

Sabiendo esto, el ciclo de ejecución de PHP funciona de la siguiente manera:

- **Petición**: El Cliente (navegador) envía una petición a través de Internet solicitando un archivo, por ejemplo, index.php.
- **Recepción**: El Servidor Web (el programa Apache) recibe la petición. Al ver que la extensión es .php, sabe que no puede enviar el archivo directamente como haría con un HTML.
- **Ejecución**: El servidor delega el archivo al intérprete de PHP. Este ejecuta la lógica programada y, si es necesario, consulta datos en la base de datos (MySQL).
- **Respuesta**: Una vez que PHP termina de procesar la información, devuelve al cliente un documento en formato HTML puro.



Concepto Clave: El navegador del usuario jamás recibe ni ve el código fuente PHP original, solo recibe el resultado final estructurado en HTML.

2 Desarrollo Local: XAMPP y VS Code

Dado que el código PHP requiere de un Servidor Web para ser interpretado, **ya no podemos ejecutar los archivos haciendo doble clic sobre ellos**. Si lo intentamos, el navegador no entenderá el código Backend y mostrará el texto fuente tal cual lo escribimos.

De momento vamos a trabajar sin salir a Internet (offline). Para poder programar y probar nuestro código, vamos a necesitar dos herramientas que compondrán nuestro entorno de desarrollo:

- **XAMPP (El Servidor)**: Este paquete de software instala en nuestro ordenador de forma local un **servidor web** (Apache) y un motor de **bases de datos** (MariaDB/MySQL). Gracias a XAMPP, nuestro equipo de clase actuará simultáneamente como Cliente (el navegador) y como Servidor Local (procesando PHP internamente bajo la dirección de red conocida como localhost).
- **Visual Studio Code (El Editor)**: Editor profesional donde escribiremos nuestros scripts combinando HTML y PHP, aprovechando sus herramientas de autocompletado y detección de errores.

> 💡 **Nota**: En realidad, una vez que el servidor local está activo, cualquier equipo de la red local del instituto podrá visitar tu web si conoce la dirección IP de tu ordenador y la carpeta donde has guardado el proyecto.

PRÁCTICA 0: Preparación del Entorno

Dado que somos los administradores de nuestros propios equipos con Linux Mint, la primera tarea práctica de este bloque será preparar nuestras máquinas instalando el software necesario.

Por lo tanto antes de continuar leyendo la teoría y empezar a programar, debes dirigirte a los Anexos ubicados al final de estos apuntes. Completaremos paso a paso los siguientes procesos:

- **Anexo I:** Instalación y configuración de XAMPP en Linux Mint (resolución de permisos de administrador).
- **Anexo II:** Instalación de Visual Studio Code y creación de la carpeta de trabajo (htdocs).

Fujo de trabajo diario (Una vez instalado)

Para que nuestro ordenador entienda PHP, la organización de los archivos cambia drásticamente. A partir de ahora, no podemos guardar las prácticas en el Escritorio o en Documentos.

A. Arrancando el motor (XAMPP):

Una vez que instalado XAMPP, nuestra rutina diaria al llegar a clase será:

1. Abrir el panel de control de XAMPP.
2. Iniciar los servicios Apache y MySQL.
3. Verificar que ambos servicios están "Running" (con luz verde).

B. El directorio raíz htdocs:

XAMPP crea una carpeta especial en el sistema llamada htdocs (ubicada en /opt/lampp/htdocs en Linux). El servidor Apache solo tiene permisos para leer los archivos que estén dentro de esta carpeta. Cualquier script PHP debe guardarse aquí.

> **Paso previo**: Crea una carpeta con tu nombre dentro de htdocs (Ej: htdocs/jose_blog/).

C. Configurando Visual Studio Code:

1. Abre Visual Studio Code.
2. Ve a Archivo > Abrir Carpeta y selecciona tu subcarpeta recién creada (htdocs/mi_blog/).
3. Todo lo que programemos se guardará ahora en la ruta correcta del servidor.

D. Nuestro primer script PHP:

Crea un archivo prueba.php en VS Code y escribe el siguiente código:

```php
<?php
  echo "¡El servidor Apache está funcionando
  correctamente!";
?>
```

Guarda el archivo. Para ejecutarlo, abre Chrome o Firefox y escribe en la barra de direcciones: http://localhost/mi_blog/prueba.php.

3 de Python a PHP: Sintaxis y Estructura

Gracias a los conocimientos de programación adquiridos previamente con Python, la curva de aprendizaje de PHP se reduce drásticamente. Los conceptos lógicos son idénticos; únicamente debemos aprender su nueva sintaxis.

Ten en cuenta los siguientes aspectos:

Scripts php

Normalmente los programas en php estarán incrustados dentro del código html de una página web. A estos bloques de código se les llama ***scripts***.

> **Script PHP**: Bloque de código escrito en lenguaje PHP dentro de una página web.

Para que el servidor interprete un bloque de código como un script PHP dentro de un documento, este debe estar encapsulado estrictamente entre las etiquetas <?php (apertura) y ?> (cierre).

```
<?php
  "código php"
?>
```

Declaración de Variables y Finalización de Sentencias

En PHP todas las variables deben ir precedidas obligatoriamente por el símbolo del dólar ($).

Además, a diferencia de Python, PHP no basa la separación de instrucciones en saltos de línea. Exige finalizar cada instrucción con un punto y coma (;). La omisión de este carácter detendrá la ejecución del servidor devolviendo un error fatal (Parse error).

Concepto	Sintaxis en Python	Sintaxis en PHP
Declaración de variable	nombre = "José"	$nombre = "José";
Asignación numérica	edad = 18	$edad = 18;

Salida de datos por pantalla

En Python se utiliza la función print() para enviar datos a la consola. En PHP, la instrucción utilizada para inyectar texto, etiquetas o variables en el flujo HTML que se enviará al navegador es el constructor echo.

Concepto	Sintaxis en Python	Sintaxis en PHP
Salida de texto plano	print("Hola Mundo")	echo "Hola Mundo";
Impresión de variable	print(nombre)	echo $nombre;

Operador de Concatenación

En Python, el operador + tiene dos usos (suma números o concatena textos).

PHP separa estrictamente estas operaciones: el + es exclusivo para operaciones matemáticas, mientras que el punto (.) es el operador designado para la concatenación de cadenas.

Concepto	Sintaxis en Python	Sintaxis en PHP
Concatenación mixta	print("Hola" + nombre)	echo "Hola" . $nombre;

Alcance y Bloques de Código: Indentación vs. Llaves

Python define el alcance de las estructuras de control mediante una indentación (tabulación) estricta.

PHP, por el contrario, ignora los espacios en blanco. En su lugar, utiliza llaves { } para delimitar de forma explícita dónde empieza y dónde acaba un bloque de código.

En PHP limitaremos las condiciones con paréntesis.

Ejemplo de estructura condicional (if/else) en Python:

```
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

Ejemplo equivalente en PHP:

```
<?php
  if ($edad >= 18) {
    echo "Mayor de edad";
  } else {
```

```
    echo "Menor de edad";
  }
?>
```

4 Estructuras de Control y Arrays

Las estructuras lógicas que vamos a ver son las mismas que estudiamos en Python, pero adaptadas a la sintaxis de PHP.

> **Recuerda**: condiciones entre paréntesis () y bloques de código entre llaves {}.

4.1 Condicionales Múltiples: elseif

En el punto anterior vimos cómo hacer un if / else básico. Pero, ¿qué pasa si tenemos más de dos opciones? En Python utilizábamos la palabra elif. En PHP, la traducción literal es elseif.

```php
<?php
$nota = 7.5;

if ($nota >= 9) {
  echo "¡Sobresaliente!";
} elseif ($nota >= 5) { // Equivale a 'elif' Python
  echo "Has aprobado.";
} else {
  echo "Toca estudiar más.";
}
?>
```

4.2 El Bucle while (Mientras):

Funciona exactamente igual que en Python. Solo recuerda declarar la variable contador y sumarle 1 en cada vuelta usando ++.

```php
<?php
  $contador = 1;
  while ($contador <= 5) {
    echo "<p>Párrafo número: " . $contador . "</p>";
    $contador++; //  $contador = $contador + 1;
  }
?>
```

4.3 El Bucle for (Para):

Aquí hay un cambio importante respecto a Python. En PHP, el bucle for clásico tiene tres partes separadas por punto y coma dentro de su paréntesis: (1. Inicio ; 2. Condición límite ; 3. Incremento).

```php
<?php
  // Bucle que cuenta del 1 al 10
  for ($i = 1; $i <= 10; $i++) {
    echo "<li>Elemento de lista " . $i . "</li>";
  }
?>
```

4.4 Arrays (Las Listas y Diccionarios de PHP)

En Python teníamos "Listas" [] y "Diccionarios" {}. En PHP todo se agrupa bajo un mismo concepto llamado Array.

1. Arrays Indexados (Las Listas de Python):

Sirven para guardar varios valores ordenados. Igual que en Python, la primera posición siempre es la cero (0).

```php
<?php
  // Creamos la lista usando corchetes
  $alumnos = ["Ana", "Luis", "Carlos", "Marta"];

  // Para mostrar a "Luis" (posición 1):
  echo "El segundo alumno es: " . $alumnos[1];
?>
```

2. Arrays Asociativos (Los Diccionarios de Python):

En lugar de usar posiciones numéricas (0, 1, 2...), usamos una "palabra clave" para guardar el dato. Usamos una flecha => para asignar el valor.

```php
<?php
  $edades = [
    "Ana" => 17,
    "Luis" => 18,
    "Carlos" => 17
  ];
  echo "Luis tiene " . $edades["Luis"] . " años.";
?>
```

4.5 El bucle foreach

En Python, la forma más cómoda de recorrer una lista era la forma:

```
for alumno in alumnos:
```

En PHP existe una instrucción equivalente, y se llama foreach (por cada).

Concepto	Sintaxis en Python	Sintaxis en PHP
Recorrer una lista	for alumno in alumnos:	foreach($alumnos as $alumno){

```php
<?php
  $alumnos = ["Ana", "Luis", "Carlos"];

  // "Por cada elemento en $alumnos, guárdalo
  temporalmente en $alumno"
  foreach ($alumnos as $alumno) {
    echo "<p>Hola, " . $alumno . "</p>";
  }
?>
```

4.6 Práctica: Tu primera lista dinámica

Vamos a mezclar todo lo aprendido. Abre tu archivo index.php en Visual Studio Code. Vamos a crear una lista <ul> en HTML, pero los elementos <li> los va a escribir PHP automáticamente recorriendo un Array.

```php
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
  <title>Mi lista dinámica</title>
</head>

<body>
  <h1>Lista de la Compra</h1>
  <ul>
  <?php
    $compra = ["Manzanas", "Leche", "Pan", "Huevos"];
    foreach ($compra as $articulo) {
      echo "<li>" . $articulo . "</li>";
    }
  ?>
  </ul>
```

```
</body>
</html>
```

Guarda el archivo y actualiza tu navegador (http://localhost/tu_carpeta/). Si inspeccionas el código fuente en el navegador (Clic derecho > Ver código fuente), verás que han desaparecido las etiquetas de PHP y la lista está perfectamente escrita en HTML.

5 Formularios: Conecta Usuario Servidor

Hasta este momento, nuestro servidor web solo "hablaba" (imprimía datos por pantalla usando echo). Ahora vamos a enseñarle a "escuchar".

Para que un usuario pueda enviar información a nuestro servidor (por ejemplo, para hacer login, registrarse o buscar un producto), utilizamos los Formularios HTML.

¿Qué es un formulario y cómo se dibuja en HTML?

> **Formulario**: Zona de la página web que contiene cajas de texto, desplegables o casillas donde el usuario puede escribir.

Todo formulario se envuelve dentro de la etiqueta <form> e incluye diferentes elementos (etiquetas <input>) y un botón para enviar los datos.

Fíjate en esta estructura básica (solo HTML):

```
<form>
  <label>Dime tu nombre:</label>
  <input type="text">
  <button type="submit">Enviar</button>
</form>
```

Tres atributos para conectar con PHP

El formulario anterior se ve muy bonito, pero al pulsarlo, no hace nada. Para que ese HTML pueda "hablar" con nuestro servidor PHP, necesitamos añadirle tres atributos obligatorios:

- **name** (En el <input>): Es el nombre o "etiqueta" que le ponemos a la caja de texto. Si no le pones un name al input, PHP será incapaz de saber qué dato le estás enviando.
- **action** (En el <form>): Indica el nombre del archivo PHP que va a recibir y procesar los datos cuando el usuario pulse el botón.
- **method** (En el <form>): Indica cómo viajarán los datos por la red. Hay dos formas:

- **GET**: Los datos viajan visibles en la URL del navegador (Ej: google.com/search?q=gatos).
- **POST**: Los datos viajan ocultos y protegidos "por debajo" de la petición. Es el que usaremos casi siempre por seguridad.

Ejemplo del formulario completo

A modo de ejemplo vamos a añadir el siguiente formulario a una página web nueva (deberás crear el resto de la estructura e incluir este código en la sección body).

```
<h2>Formulario de Registro</h2>
<form action="procesar.php" method="POST">
  <label>Nombre:</label>
  <input type="text" name="nombre_usuario"> <br><br>
  <label>Edad:</label>
  <input type="number" name="edad_usuario"> <br><br>
  <button type="submit">Enviar al Servidor</button>
</form>
```

Guarda el resultado como formulario.php (lo utilizaremos más adelante).

Recibiendo los datos en PHP: Las Superglobales

Cuando el usuario pulsa "Enviar", el navegador empaqueta los datos y abre automáticamente el archivo procesar.php. Pero, ¿cómo lee PHP esos datos?

El servidor crea de forma automática un Array Asociativo (el "diccionario" del punto anterior) llamado $_POST (o $_GET si usaste ese método).

Dentro de este array, las "claves" para extraer la información son exactamente los atributos name que escribiste en el HTML.

Ejemplo del receptor

Incluye el siguiente código php en un archivo al que llamarás procesar.php:

```
<?php
  // 1. Recoger datos con el atributo 'name' del HTML
  $nombre = $_POST["nombre_usuario"];
  $edad = $_POST["edad_usuario"];

  // 2. Usamos la lógica de programación
  echo "<h1>Hola, " . $nombre . "</h1>";

  if ($edad >= 18) {
```

```
    echo "<p>Eres mayor de edad. Tienes acceso
  completo.</p>";
  } else {
    echo "<p>Eres menor de edad. Acceso
  restringido.</p>";
  }
?>
```

Acabamos de programar nuestra primera interacción real Cliente-Servidor:

1. El navegador empaquetó tus datos.
2. Los datos viajaron al servidor Apache.
3. El servidor Apache se los entregó a PHP.
4. A través del código PHP se evaluó la edad del usuario con un condicional y fabricó una respuesta en html solo para el usuario que envío la petición.

6 Del "Localhost" a Internet

Hasta ahora, nuestra página web interactiva solo existe dentro de nuestro ordenador (en el servidor local XAMPP). Si utilizamos el enlace http://localhost/... desde otro ordenador, no verá nada.

> 💡 **Recuerda:** Si estamos en una red local, todos los usuarios de la misma **sí podrán acceder a nuestro servidor** y probar nuestro formulario. Para ello, solo tienen que escribir en sus navegadores la **dirección IP** de nuestro ordenador en lugar de la palabra "localhost" (por ejemplo: http://192.168.1.50/tu_proyecto/).

Para que nuestra aplicación esté verdaderamente en Internet necesitamos subirla a un servidor público. Para ello, utilizaremos nuestra cuenta de alojamiento gratuito en **Byethost**.

Recuerda el procedimiento:

- Establece la conexión con tu servidor a través de ftp-sync.
- Sube los archivos formulario.php y procesar.php.

Coge tu teléfono móvil y escribe la dirección de tu formulario.

Rellena los datos y pulsa enviar. Si el servidor te responde saludándotem acabas de publicar tu primera aplicación web dinámica en Internet.

Un problema para reflexionar...

Nuestra aplicación ya funciona en internet y procesa los datos de los usuarios. Pero hay un pequeño inconveniente. Si un usuario se registra en tu formulario... ¿Dónde se han guardado sus datos?

La respuesta es: En ningún sitio.

PHP lee los datos, genera el HTML de respuesta y, en cuanto la página termina de cargar, olvida toda la información. Si queremos crear un sistema de login, una tienda o un foro, necesitamos un "almacén" que guarde a nuestros usuarios de forma permanente.

Es hora de dar el siguiente salto del desarrollo web: Las Bases de Datos.

7 Guardando la información de forma permanente

Como vimos en el punto anterior, una vez que PHP procesa un formulario y dibuja la página, olvida los datos. Para crear sistemas reales (usuarios, tiendas, foros), necesitamos guardar esa información de forma permanente.

Para ello utilizaremos el estándar de bases de datos relacionales MySQL (o MariaDB).

Conceptos Básicos: La analogía con Excel

Una base de datos es, en esencia, muy similar a un documento de Excel.

- **Base de datos**: Es el archivo completo.
- **Tabla**: Es la "hoja" del Excel donde guardamos elementos del mismo tipo (ej: una tabla para usuarios, otra para productos).
- **Campos** (Columnas): Son las propiedades que vamos a guardar (Nombre, Apellidos, Edad).
- **Registros** (Filas): Es el conjunto de datos de un usuario concreto.

La Clave Primaria (Índice):

Hay una norma fundamental en las bases de datos: no puede haber dos filas idénticas. Para evitar que dos usuarios llamados "Ana" se confundan, obligatoriamente debemos crear un campo llamado id (identificador). Este campo funcionará como el DNI o la matrícula de cada registro.

7.1 MySQL en Byethost y phpMyAdmin

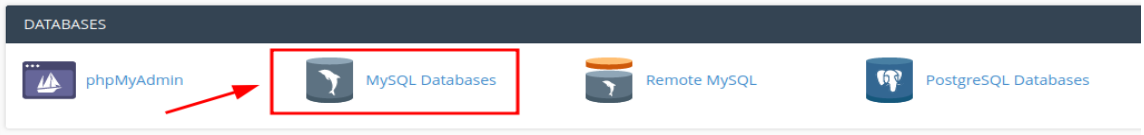
En el desarrollo web profesional, las bases de datos viven dentro del servidor remoto.

> Los pasos que vamos a ver se podrían realizar de manera sunukar sobre el servidor local de XAMPP (sin tener que definir usuario y contraseña).

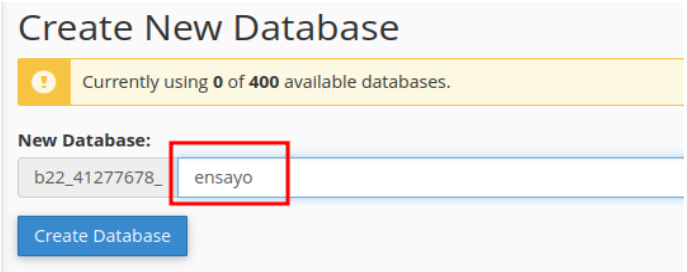
Para crear nuestra base de datos en Byethost, el proceso consta de dos partes: primero "declaramos" la base de datos en el panel de control, y luego crearemos las tablas usando la herramienta visual phpMyAdmin.

Paso 1: Crear la Base de Datos en el Panel

- Entra a tu panel de control de Byethost (cPanel o VistaPanel).
- Busca la sección llamada "MySQL Databases" (Bases de datos MySQL) y haz clic en ella.



- En la casilla para crear una nueva base de datos, escribe proyecto y dale a "Create".



⚠️ ¡Atención! Por seguridad, Byethost añade tu nombre de usuario al principio de la base de datos. Si tu usuario es b12_123 456, tu base de datos se llamará realmente b12_123 456_proyecto. Apunta este nombre completo, lo vas a necesitar.

Paso 2: Crear la Tabla con phpMyAdmin

Desde el menú principal de Byethost, haz clic en la herramienta phpMyAdmin.

En la barra lateral izquierda, selecciona la base de datos que acabamos de crear (b22_41 277 678_ensayo).

La aplicación te pedirá crear una nueva tabla. Llámala usuarios y dile que tendrá 3 columnas (campos).

Configura los campos tal y como hacíamos en LibreOffice:

- **Campo 1**: **Nombre: id** | Tipo: INT | Busca la casilla A_I (Auto Incremental) y márcala. Esto la convertirá en tu Clave Primaria.
- **Campo 2**: **Nombre: nombre** | Tipo: VARCHAR | Longitud: 50.
- **Campo 3**: **Nombre: edad** | Tipo: INT.

Haz clic en Guardar. Nuestra base de datos online ya está lista para trabajar.

Paso 3: Conectando PHP con MySQL

Tu página web (PHP) y tu base de datos (MySQL) ya están en Byethost, pero todavía no se comunican. Tenemos que tender un "puente" entre ellos.

A diferencia de XAMPP (donde el usuario era root y no había contraseña), en Internet necesitamos usar las credenciales reales de nuestro servidor.

Para establecer la conexión crearemos un archivo específico. Lo llamaremos conexion.php.

> En realidad el código de este equipo podría estar incluido directamente en cada página que quisiera establecer la conexión. Simplemente estamos centralizando el proceso.

Los datos de conexión aparecen en la sección "MySQL Databases" de tu panel de Byethost o en la parte derecha del menú principal.

Archivo: conexion.php

```php
<?php
  // Datos de configuración del servidor Byethost
  // (¡Sustituye estos datos por los tuyos propios!)
  $servidor = "sqlXXX.byethost.com";
  $usuario = "b12_123456";
  $password = "tu_contraseña_secreta";
  $base_datos = "b12_123456_proyecto";

  // Creamos la conexión orientada a objetos (MySQLi)
  $conexion = new mysqli($servidor, $usuario,
  $password, $base_datos);

  // Comprobamos si el puente ha fallado
  if ($conexion->connect_error) {
    die("Error crítico: No se puede conectar a la
    base de datos en Byethost.");
  }
?>
```

Paso 4: Guardar los datos del Formulario en la base de datos

En primer lugar debemos abrir el archivo procesar.php (el que recibe los datos cuando el usuario pulsa "Enviar") para modificarlo. Nuestro objetivo es que PHP lea esos datos, construya una orden de guardado y se la envíe a la base de datos.

Para que este archivo funcione, debemos seguir cinco pasos:

1. Incluir el archivo de conexión

En lugar de escribir las credenciales del servidor una y otra vez, usamos la función include("conexion.php"). Esto le dice a PHP: "antes de seguir, copia y pega aquí dentro todo el código que hay en conexion.php". Gracias a esto, ya tenemos disponible la variable puente $conexion.

2. Recoger los datos del HTML

Cuando el navegador envía el formulario, el servidor guarda esa información en un "buzón" llamado $_POST. Para que el código sea más limpio, sacamos esos datos del buzón y los guardamos en dos variables de PHP: $nombre_recibido y $edad_recibida.

3. Preparar la orden en lenguaje SQL

PHP no sabe cómo guardar datos en tablas, eso es trabajo de MySQL. Por tanto, PHP debe construir una frase en lenguaje SQL (el idioma que habla MySQL) y guardarla en una variable de texto que llamamos $orden_sql.

La estructura de la frase es:

```
INSERT INTO nombre_tabla (columnas) VALUES (valores).
```

> **Regla estricta de SQL**: Si un valor es texto (como el nombre), debe ir obligatoriamente entre comillas simples ' '. Si es un número (como la edad), va sin comillas. Si te saltas esta regla, MySQL rechazará la orden.

4. Ejecutar la orden (El envío)

Hasta ahora solo teníamos una frase guardada en una variable. En este paso usamos el puente ($conexion) y utilizamos su función query().

Esto es el equivalente a darle al botón de enviar:

1. enviamos el texto de la orden SQL al servidor MySQL.
2. MySQL la lee, intenta añadir la fila en la tabla usuarios.
3. MySQL devuelve una respuesta: TRUE (éxito) o FALSE (fracaso).
4. Guardamos esa respuesta en la variable $resultado.

5. Comprobar la respuesta

Por último, usaremos un condicional if para leer la respuesta de MySQL. Si $resultado es TRUE, mostramos un mensaje de confirmación al usuario. Si es FALSE, utilizamos $conexion->error para pedirle a MySQL que nos explique exactamente por qué ha fallado (por ejemplo: la tabla no existe o nos hemos olvidado una comilla).

El código completo de procesar.php quedaría así:

```
<?php
// 1. Cargamos el puente de conexión
include("conexion.php");

// 2. Recogemos los datos del formulario HTML
$nombre_recibido = $_POST["nombre_usuario"];
$edad_recibida = $_POST["edad_usuario"];

// 3. Preparamos la orden SQL para insertar los datos
// Los textos ($nombre_recibido) DEBEN ir entre
comillas simples, los números no.
$orden_sql = "INSERT INTO usuarios (nombre, edad)
VALUES ('$nombre_recibido', $edad_recibida)";

// 4. Enviamos la orden a la base de datos para que
la ejecute
$resultado = $conexion->query($orden_sql);

// 5. Evaluamos la respuesta que nos ha devuelto la
base de datos
if ($resultado == TRUE) {
echo "¡Hola " . $nombre_recibido . "! Tus datos se
han guardado para siempre.";
} else {
echo "Vaya, ha ocurrido un error al guardar: " .
$conexion->error;
}
?>
```

Paso 5 : Comprobación

Ve a tu formulario en el navegador web y rellénalo con los datos de un par de amigos.

Abre de nuevo phpMyAdmin.

Haz clic en tu tabla usuarios y ve a la pestaña Examinar. Verás una tabla exactamente igual con los datos de los usuarios guardados perfectamente. Observa que el campo id se ha rellenado de forma automática.

8 Leyendo información de la base de datos

El objetivo final de nuestro proyecto va ser crear una página nueva que se conecte a MySQL, lea todos los usuarios que se han registrado y los dibuje en una tabla HTML.

Para hacer esto, necesitamos dos herramientas nuevas: la orden SQL SELECT y el bucle while de PHP.

La orden SELECT

En lenguaje SQL, para leer datos de una tabla usamos la instrucción SELECT.

Esta orden tiene muchas opciones y parámetros que nos permiten tener un control total sobre la información a leer.

Si simplemente queremos pedir a MySQL que lea todos los campos de todos los registros de la tabla usuarios", la orden es así de sencilla:

```
SELECT * FROM usuarios
```

El asterisco * significa "todo".

El bucle while y la función fetch_assoc()

Cuando PHP envía la orden SELECT a MySQL, la base de datos no le devuelve un texto normal, sino un "paquete" con un montón de filas. PHP no puede imprimir todas las filas de golpe con un simple echo.

Tenemos que usar un bucle while. La lógica es esta: "Mientras queden filas por leer en el paquete, saca una fila, guárdala en una variable llamada $fila, e imprime su HTML".

Para sacar esa fila del paquete usamos la función:

```
$resultado->fetch_assoc()
```

Creando la página ver_usuarios.php

Crea un archivo nuevo llamado ver_usuarios.php. Fíjate en cómo mezclamos el HTML normal (para crear la estructura de la tabla) con PHP (para rellenar las filas dinámicamente).

```
<!DOCTYPE html>
<html lang="es">

<head>
<meta charset="UTF-8">
<title>Lista de Usuarios</title>
<style>
  table {
    border-collapse: collapse;
    width: 50%;
  margin: 20px auto;
  }

  th, td {
    border: 1px solid black;
    padding: 10px;
    text-align: center;
  }

  th {
    background-color: #f2f2f2;
  }
</style>
</head>

<body>
  <h1 style="text-align: center;">Usuarios
Registrados en el Sistema</h1>
  <table>
    <tr>
      <th>ID</th>
      <th>Nombre</th>
      <th>Edad</th>
    </tr>
    <?php
    // 1. Cargamos el archivo de conexión
    include("conexion.php");

    // 2. Preparamos la orden SQL para leer datos
    $orden_sql = "SELECT * FROM usuarios";

    // 3. Enviamos la orden y guardamos el paquete de
    datos que nos devuelve MySQL
    $resultado = $conexion->query($orden_sql);
```

```
    // 4. Comprobamos si el paquete tiene más de 0
    filas (si hay alguien registrado)
    if ($resultado->num_rows > 0) {

      // 5. Bucle: Mientras haya filas por extraer
      del paquete...
      while ($fila = $resultado->fetch_assoc()) {
        // Dibujamos una fila de tabla HTML (<tr>)
        echo "<tr>";
        // Rellenamos las celdas (<td>) sacando los
        datos exactos por el nombre de su columna
        echo "<td>" . $fila["id"] . "</td>";
        echo "<td>" . $fila["nombre"] . "</td>";
        echo "<td>" . $fila["edad"] . "</td>";
        echo "</tr>";
      }
    } else {
      // Si la tabla de la base de datos está vacía,
      mostramos este mensaje
      echo "<tr><td colspan='3'>Aún no hay usuarios
      registrados.</td></tr>";
    }
    // 6. Cerramos la conexión por buena educación
    $conexion->close();
    ?>
  </table>
</body>
</html>
```

Notas importantes sobre este código:

HTML y PHP, trabajando en equipo:

Hemos estructurado el código PHP justo en el medio de las etiquetas <table> del HTML. Así puedes ver visualmente cómo PHP se encarga exclusivamente de fabricar las filas (<tr>) y las celdas (<td>) repetitivas, integrándose a la perfección con tu maqueta HTML.

¿CSS sin archivo externo?:

Hemos escrito el CSS directamente en la cabecera (dentro de la etiqueta <style>), en lugar de enlazando un archivo externo .css como hemos hecho durante todo el curso. En este caso lo hacemos así por pura comodidad, para que tengas todo el ejemplo a la vista en un solo archivo.

Prevención de tablas vacías (num_rows):

> Hemos incluido la comprobación $resultado->num_rows > 0. Esta es una práctica profesional excelente. Si por algún motivo la tabla de la base de datos se queda vacía (por ejemplo, si borras todo desde phpMyAdmin), en lugar de romperse la página web, tu código lo detectará y mostrará el mensaje amigable: "Aún no hay usuarios registrados".

Comprobación

Sube este nuevo archivo ver_usuarios.php a tu carpeta htdocs de Byethost.

Abre la página en tu navegador. Verás a todos tus amigos registrados.

Abre en otra pestaña tu formulario.php, registra a una persona nueva y dale a enviar.

Vuelve a la pestaña de ver_usuarios.php y pulsa F5 para recargar la página.

El nuevo usuario aparecerá instantáneamente en la tabla. Acabas de programar tu primer sistema web dinámico completo.

9 Anexo I: Instalación y Configuración de XAMPP en Linux Mint

Descarga e Instalación del Software

1. Descarga la última versión de XAMPP para Linux (archivo .run) desde su sitio web oficial (https://www.apachefriends.org/es/index.html)
2. Abre un terminal y desplázate al directorio donde se haya descargado el archivo.
3. Otorga permisos de ejecución al archivo descargado ejecutando el siguiente comando (sustituye el nombre del archivo por la versión exacta que hayas descargado):

```
chmod +x xampp-linux-x64-8.2.4-0-installer.run
```

4. Ejecuta el instalador con privilegios de administrador:

```
sudo ./xampp-linux-x64-8.2.4-0-installer.run
```

5. Se abrirá el asistente gráfico (Setup Wizard). Acepta las sucesivas ventanas. Por defecto, XAMPP se instalará en el directorio /opt/lampp.

Paso 2: Solución de permisos

Por defecto, la carpeta donde debemos guardar nuestros proyectos web (htdocs) pertenece al administrador del sistema (root). Para más adelante poder guardar archivos desde nuestro editor de código sin problemas, debemos hacernos propietarios de esa carpeta.

En la terminal, ejecuta este comando:

```
sudo chown -R $USER:$USER /opt/lampp/htdocs
```

Paso 3: Iniciar XAMPP desde la Terminal y Comprobar Servicios

XAMPP requiere el paquete de herramientas de red net-tools. Si no lo tienes, instálalo con:

```
sudo apt install net-tools
```

Para arrancar los servicios manualmente, utiliza el comando:

```
sudo /opt/lampp/lampp start
```

Deberías ver un mensaje indicando que Apache y MySQL se han iniciado con un "ok". Para detenerlos, el comando es sudo /opt/lampp/lampp stop.

Prueba de funcionamiento:

Abre tu navegador web y visita la dirección http://localhost/. Si ves la página de bienvenida de XAMPP, el servidor web funciona.

Para verificar el funcionamiento dela base de datos, visita la aplicación phpmyadmin: http://localhost/phpmyadmin/.

Paso 4: Crear un Lanzador (Acceso Directo) en el Menú de Mint

Para no tener que usar la terminal cada vez que queramos encender el servidor, vamos a crear un acceso directo con su icono oficial en nuestro menú de aplicaciones:

1. Asegúrate de que la interfaz gráfica tiene permisos de ejecución:

```
sudo chmod 755 /opt/lampp/manager-linux-x64.run
```

2. Hacer una copia de seguridad del icono: Vamos a copiar el logo vectorial que trae el programa y guardarlo de forma segura en la raíz de XAMPP para que no se borre por accidente.

```
sudo cp /opt/lampp/htdocs/dashboard/images/xampp-logo.svg
/opt/lampp/xampp-icon.svg
```

3. Abre un editor de texto desde la terminal para crear el acceso directo:

```
xed ~/.local/share/applications/xampp.desktop
```

4. Pega el siguiente contenido exactamente como aparece:

```
[Desktop Entry]
Type=Application
Name=XAMPP Manager
```

```
Exec=x-terminal-emulator -e "sudo /opt/lampp/manager-linux-
x64.run"
Icon=/opt/lampp/xampp-icon.svg
Comment=Administrador de XAMPP
Categories=Development;
Terminal=false
```

5. Guarda el archivo (Ctrl + S) y ciérralo.

¡Listo! Ahora encontrarás "XAMPP Manager" (con su logo naranja) en tu menú de aplicaciones de Linux Mint. Al abrirlo, te pedirá tu contraseña y mostrará el panel de control gráfico para encender y apagar los servidores con un clic.

Mantenimiento: Cómo Desinstalar XAMPP

(Utiliza estos comandos solo si necesitas eliminar XAMPP de tu sistema).

Accede al directorio de instalación (cd /opt/lampp) y ejecuta el desinstalador:

```
sudo ./uninstall
```

Para eliminar cualquier rastro o archivo residual, borra la carpeta:

```
sudo rm -r /opt/lampp
```

10 Anexo II: Configuración de Visual Studio Code para PHP

Para programar de forma cómoda y evitar errores tipográficos, vamos a preparar nuestro editor para que entienda el lenguaje PHP y nos ayude mientras escribimos.

Paso 1: Abrir nuestra carpeta de trabajo (htdocs)

Como instalamos XAMPP en el Anexo I, el servidor Apache solo leerá los archivos que estén dentro de su carpeta pública.

1. Abre Visual Studio Code.
2. Ve a Archivo > Abrir carpeta (File > Open Folder).
3. Navega hasta Sistema de archivos y entra en la ruta: /opt/lampp/htdocs.
4. Crea un directorio dentro de la carpeta htdocs. Es buena idea darle un nombre significativo que describa a tu sitio web.
5. Haz clic en Aceptar. (Si VS Code te pregunta si confías en los autores de esta carpeta, dile que sí).

A partir de ahora, todo lo que programes lo crearás dentro.de la carpeta que has definido para tu sitio web.

Paso 2: Instalar la extensión "PHP Intelephense"

Esta herramienta ofrece un sistema avanzado de autocompletado y detecta errores de sintaxis en tiempo real.

1. En la barra lateral izquierda de VS Code, haz clic en el icono de Extensiones (son cuatro cuadrados).
2. En el buscador, teclea PHP Intelephense (del creador Ben Mewburn).
3. Haz clic en Instalar.

Paso 3: Conectar VS Code con el motor PHP de Linux

Para que la extensión anterior funcione perfectamente y compruebe nuestro código , debemos decirle a VS Code dónde está instalado PHP dentro de nuestro Linux Mint.

1. Haz clic en el engranaje de Administrar (abajo a la izquierda) y selecciona Configuración (Settings).
2. En la esquina superior derecha de la pestaña que se abre, busca un pequeño icono que parece una hoja de papel con una flechita curva (Open Settings (JSON)) y haz clic en él.
3. Esto abrirá un archivo de texto llamado settings.json.
4. Justo antes de la última llave de cierre }, añade una coma a la línea anterior (si no la tiene) y pega este código:

```
"php.validate.executablePath": "/opt/lampp/bin/php"
```

(Esta es la ruta exacta donde XAMPP guarda el ejecutable de PHP en tu equipo Linux).

5. Guarda el archivo pulsando Ctrl + S y reinicia Visual Studio Code para que los cambios surtan efecto.

11 Anexo III: Guía Avanzada de Formularios y Subida de Archivos

Aquí encontrarás cómo utilizar los campos de formulario más avanzados y cómo resolver problemas comunes, como procesar opciones múltiples o permitir que los usuarios suban imágenes a tu servidor.

Campos de Formulario más habituales

Campo texto de Contraseña (password)

Sólo se diferencia del texto normal en que, al rellenarlo, los caracteres se ocultan visualmente (se muestran como puntos o asteriscos) por seguridad.

```
<input type="password" name="clave">
```

Botón de opción (radio)

Permite elegir solo uno de los varios valores posibles. Todos los inputs correspondientes al mismo grupo deben tener el mismo nombre (name). El atributo value es lo que se enviará a PHP. Para marcar una opción por defecto, se incluye la palabra checked.

```
<input type="radio" name="color" value="Rojo">
Rojo<br>
<input type="radio" name="color" value="Verde"
checked> Verde<br>
```

Casilla de verificación (checkbox)

Permite activar varias opciones simultáneamente. Cada <input type="checkbox"> requiere un nombre distinto y un valor.

```
<input type="checkbox" name="tapiceria" value="Cuero"
checked> Tapicería de Cuero<br>
<input type="checkbox" name="gps" value="Si">
Navegador GPS<br>
```

Lista o Menú Desplegable (select)

Muestra un menú con opciones. El atributo name va en la etiqueta <select>, y las opciones van dentro de etiquetas <option>. Si incluyes la palabra selected en una opción, aparecerá por defecto.

```
<select name="provincia">
  <option>(Elige una provincia)</option>
  <option value="Zaragoza" selected>Zaragoza</option>
  <option value="Huesca">Huesca</option>
  <option value="Teruel">Teruel</option>
</select>
```

Área de texto (textarea)

Similar al campo de texto, pero permite escribir párrafos largos. Se puede definir el número de filas (rows) y columnas (cols) visibles.

```
<textarea rows="5" cols="50" name="comentario">
Escribe aquí tu opinión...</textarea>
```

Campo oculto (hidden)

Permite insertar en un formulario un valor que no es visible para el usuario, pero que se enviará al servidor. Muy útil para enviar IDs o configuraciones internas.

```
<input type="hidden" name="id_usuario" value="4598">
```

El problema de los Checkbox: La función isset()

Cuando un formulario con checkboxes envía los datos al servidor, ocurre algo peculiar: si el usuario no activa un checkbox, este no envía absolutamente nada.

Eso hace que si en nuestro PHP intentamos leer $_POST['tapiceria'] y el usuario no la marcó, la página nos devolverá un error avisando de que esa variable no existe.

Para evitar este problema utilizamos la función isset(variable). Esta función evalúa si una variable existe y devuelve TRUE o FALSE.

Ejemplo de solución en PHP:

```
<?php
```

```php
// Preguntamos: ¿Existe el dato 'tapiceria' en el
formulario que nos ha llegado?
if (isset($_POST['tapiceria']) == TRUE) {
  $extra = $_POST['tapiceria']; // Si existe,
  guardamos su valor
} else {
  $extra = "Sin tapicería extra"; // Si no existe, le
  damos un valor por defecto
}
?>
```

El Nivel Pro: Subir archivos e imágenes al servidor

Subir un archivo (como una foto de perfil) requiere dos preparativos esenciales.

Primero, en el formulario HTML debemos usar obligatoriamente el método POST y añadir el atributo enctype="multipart/form-data". Sin esto, el archivo no viajará por la red.

```html
<form action="procesar_archivo.php" method="POST"
enctype="multipart/form-data">
  <label>Selecciona tu imagen:</label>
  <input type="file" name="mi_archivo">
  <button type="submit">Subir Imagen</button>
</form>
```

¿Cómo procesa PHP los archivos? La variable $_FILES

Cuando envías un archivo, PHP no lo guarda en $_POST, sino en una variable global especial llamada $_FILES. El archivo se guarda temporalmente en una carpeta oculta del servidor. Nuestro trabajo es usar PHP para coger ese archivo de la zona temporal y moverlo a la carpeta de nuestro proyecto (por ejemplo, a una carpeta llamada images).

$_FILES['mi_archivo'] es un array que contiene toda esta información:

- name: El nombre original del archivo (ej: foto.png).
- type: El tipo de archivo (ej: image/png).
- tmp_name: La ruta temporal donde el servidor lo ha guardado provisionalmente.
- size: El tamaño del archivo en bytes.

El Script definitivo para mover el archivo (procesar_archivo.php)

Para mover el archivo a su ubicación definitiva utilizamos la función move_uploaded_file().

⚠️ ¡Atención! Si estás en XAMPP (Linux), asegúrate de que la carpeta de destino (ej: htdocs/tu_proyecto/images/) tenga permisos de escritura. Puedes dárselos abriendo una terminal y ejecutando: chmod 777 /opt/lampp/htdocs/tu_proyecto/images/.

```php
<?php
  // Comprobamos si el formu se ha enviado por POST
  if ($_SERVER["REQUEST_METHOD"] === "POST") {

  // 1. Definimos la carpeta destino
  $carpeta_destino = $_SERVER['DOCUMENT_ROOT'] .
  '/tu_proyecto/images/';

  // 2. Verificamos si se recibió el archivo sin
  errores (error === 0)
  if (isset($_FILES["mi_archivo"]) &&
  $_FILES["mi_archivo"]["error"] === 0) {

    // Extraemos la información útil
    $nombre_archivo = $_FILES["mi_archivo"]["name"];
    $ruta_temporal = $_FILES["mi_archivo"]
  ["tmp_name"];

    // Fabricamos la ruta final completa (ej:
  .../images/foto.png)
    $ruta_destino_completa = $carpeta_destino .
  $nombre_archivo;

    // 3. Movemos el archivo de la zona temporal a la
    carpeta destino
    if (move_uploaded_file($ruta_temporal,
  $ruta_destino_completa)) {
      echo "¡Éxito! El archivo se ha guardado
  correctamente.";
      echo "<br>Puedes verlo aquí: <img src='images/"
. $nombre_archivo . "' width='200'>";
    } else {
      echo "Error: No se pudo mover el archivo.
(Revisa los permisos de la carpeta).";
    }
  } else {
    echo "Error en la carga o no se seleccionó ningún
archivo.";
  }
}
?>
```

12 Anexo IV: Arrays avanzados

Arrays multidimensionales

Son arrays que utilizan dos (o más) índices. Permiten crear tablas de doble (o más) entrada:

Array escalar

- $a[xx][yy] = valor
- $a[][] = valor
- $a[xx][] = valor
- $a[][xx] = valor

Array asociativo

- $a[“nombre”] [“nombre2”] = valor

Array con índice tipo variable

- $a[$variable][$variable] = valor

Array mixtos

- Un índice es un escalar y el otro un número.

Trabajando con arrays

Supongamos que queremos crear un array en el que guardar el nombre, cuenta de correo y número de teléfono de un usuario.

Parece interesante definir los nombres de los índices con valores significativos (bases de datos). Una estructura muy eficaz para crear este ***array unidimensional*** es:

```php
<?php
  $data = [
    'nombre' => 'José Antonio',
    'email' => 'josesallan@iesfgc.net',
    'telefono' => '654987321'
  ];
  echo $data['nombre'];
?>
```

Mostrará como resultado:

```
José Antonio
```

Si trabajamos con más de un usuario sería muy útil repetir la estructura anterior, pero el ***array tendrá dos índices***, el primero identificará al usuario y el segundo hará referencia a la propiedad almacenada de ese usuario.

En nuestro caso, añado un segundo usuario a la definición y accedo a su número de teléfono:

```php
<?php
  $data = [
    [
      'nombre' => 'José Antonio',
      'email' => 'josesallan@iesfgc.net',
      'telefono' => '654987321'
    ],
    [
      'nombre' => 'María',
      'email' => 'mariaperez@iesfgc.net',
      'telefono' => '666983733'
    ]

  ];
  echo $data[1]['telefono'];
?>
```

Dando por resultado:

```
666 983 733
```

Como de costumbre existen infinidad de funciones que nos permiten hacer el trabajo con array mucho más sencillo. Siempre es una buena idea consultar la documentación de referencia.

13 Anexo V: Ampliación de SQL y Gestión de Errores

A lo largo del tema hemos visto cómo insertar datos (INSERT) y cómo leer todos los registros de una tabla (SELECT *). Sin embargo, en un proyecto real (como una tienda online o un foro) necesitarás realizar búsquedas específicas, ordenar la información, actualizar perfiles o borrar usuarios.

A continuación, tienes una guía rápida de las operaciones SQL más habituales para potenciar tu aplicación.

1. Consultas Avanzadas (El poder del SELECT)

1.1. Consultando sólo algunos campos

Cuando no necesitas obtener absolutamente todas las columnas de una tabla, puedes seleccionar solo las que realmente te interesan. Esto hace que tu web cargue más rápido y consuma menos memoria.

Sintaxis SQL:

```
SELECT campo1, campo2 FROM tabla;
```

```php
$orden_sql = "SELECT nombre, edad FROM usuarios";
$resultado = $conexion->query($orden_sql);
```

1.2. Ordenando la salida (ORDER BY)

Si quieres que tu lista de usuarios salga ordenada alfabéticamente o por edad, debes añadir la orden ORDER BY al final de tu consulta.

Sintaxis SQL:

```
SELECT campo1, campo2 FROM tabla ORDER BY campo_n [ASC|DESC];
```

- ASC: Ordena de forma ascendente (de menor a mayor o de la A a la Z). Es el orden por defecto.
- DESC: Ordena de forma descendente (de mayor a menor o de la Z a la A).

Ejemplo: Mostrar todos los usuarios, ordenados del más mayor al más joven.

```
$orden_sql = "SELECT * FROM usuarios ORDER BY edad DESC";
```

Nota: Puedes ordenar por varios campos a la vez. Ej: ORDER BY edad DESC, nombre ASC (ordena por edad y, si hay empates, los desempata alfabéticamente).

1.3. Filtrando registros (La cláusula WHERE)

La cláusula WHERE (donde) es tu mejor amiga. Se usa para filtrar los registros y obtener solo aquellos que cumplen una condición.

Sintaxis SQL:

```
SELECT * FROM tabla WHERE condición;
```

Operadores de comparación comunes:

Operador	Tipo	Sintaxis	Descripción
=	Numérico / Cadena	WHERE edad = 18 o WHERE nombre = 'Ana'	Igualdad exacta. (Recuerda: las cadenas van entre comillas simples '').
<	Numérico	WHERE edad < 18	Menor que.
<=	Numérico	WHERE edad <= 18	Menor o igual que.
>	Numérico	WHERE edad > 18	Mayor que.
>=	Numérico	WHERE edad >= 18	Mayor o igual que.

Operadores lógicos (Combinar condiciones):

Operador	Descripción	Ejemplo
AND (&&) =	Ambas condiciones deben cumplirse.	WHERE edad >= 18 AND ciudad = 'Madrid'
OR (||)<	Al menos una debe cumplirse.	WHERE edad < 18 OR ciudad = 'Barcelona'
<=NOT (!)	La condición no debe cumplirse.	WHERE NOT ciudad = 'Valencia'

1.4. El buscador flexible (Comodín LIKE)

Si quieres hacer un buscador en tu web, el operador = no te sirve (porque busca coincidencias exactas). El operador LIKE te permite buscar fragmentos de texto usando el comodín % (que representa "cualquier texto").

Sintaxis	¿Qué busca?
LIKE '%cadena%'	Contiene "cadena" en cualquier parte.
LIKE 'cadena%'	Empieza exactamente por "cadena".
LIKE '%cadena'	Termina exactamente en "cadena".

2. Modificar y Eliminar Datos

⚠️ **NUNCA** ejecutes una orden UPDATE o DELETE sin ponerle un WHERE al final. Si se te olvida la condición, modificarás o borrarás TODOS los registros de tu tabla de golpe y sin posibilidad de deshacerlo.

2.1. Modificar registros (UPDATE)

Permite cambiar el valor de uno o varios campos en un registro que ya existe.

Sintaxis SQL:

```
UPDATE tabla SET campo1 = nuevo_valor WHERE condicion;
```

Igual que vimos en el INSERT, cuando pases variables de PHP a tu sentencia SQL, recuerda que los textos van entre comillas simples.

```php
$nuevo_nombre = "Pedro";
$id_usuario = 5;
// Correcto: '$nuevo_nombre' lleva comillas simples,
$id_usuario no por ser número.
$orden_sql = "UPDATE usuarios SET nombre =
'$nuevo_nombre' WHERE id = $id_usuario";
$conexion->query($orden_sql);
```

2.2. Eliminar registros (DELETE)

Sirve para borrar permanentemente una fila entera de tu base de datos.

Sintaxis SQL:

```
DELETE FROM tabla WHERE condicion;
```

Ejemplo: Borrar al usuario que tenga el ID número 3.

```php
$orden_sql = "DELETE FROM usuarios WHERE id = 3";
$conexion->query($orden_sql);
```

3. Comprendiendo los Errores de MySQLi

Cuando trabajamos con bases de datos desde PHP, a veces las consultas fallan (porque nos falta una comilla, porque la tabla no existe o porque hemos intentado guardar un dato duplicado).

Para no programar "a ciegas", utilizamos las propiedades de nuestro objeto $conexion:

```
$conexion->errno:
```

Devuelve el número de código del error. Si devuelve 0, es que todo ha ido bien.

```
$conexion->error:
```

Devuelve un texto en inglés explicando el motivo del fallo.

Ejemplo práctico de diagnóstico:

```php
$sql = "INSERT INTO usuarios (id, nombre) VALUES (1,
'Ana')";
$resultado = $conexion->query($sql);

if (!$resultado) {
  // Si la orden ha fallado, imprimimos el chivato
  echo "<b>Error número:</b> " . $conexion->errno .
  "<br>";
  echo "<b>Descripción:</b> " . $conexion->error;
} else {
  echo "Operación realizada correctamente.";
}
```

Errores frecuentes que te puedes encontrar:

- Error 1050: Se ha intentado crear una tabla que ya existe.
- Error 1062: Se ha intentado insertar un valor duplicado en un campo con restricción UNIQUE o en la Clave Primaria. (Por ejemplo, si intentas hacer un INSERT forzando que el id sea 1, y ya existe un usuario con el id 1).

- Error 1146: La tabla que has escrito en la consulta no existe en la base de datos (revisa si le pusiste mayúsculas o te comiste una 's' al final).

14 Anexo V: Funciones

Función: Conjunto de órdenes predefinidas que se ejecutan al incluir en el código del script un nombre que la define.

Distinguiremos entre:

- Funciones preestablecidas: Vienen con el propio lenguaje
- Funciones diseñadas por el programador.

Funciones preestablecidas

Existen muchas funciones preestablecidas, podemos investigar sobre ellas consultando la documentación oficial de php: https://www.php.net/docs.php

Por ejemplo si queremos que nuestro programa genere un número entero aleatorio entre 1 y 6 utilizaríamos rand con la siguiente sintaxis:

```
<?php
  echo "Número al azar: " . rand(1,6);
?>
```

Nos daría como resultado:

```
Número al azar: 4
```

Funciones diseñadas por el programador

Sintaxis:

```
function nombrefuncion(){
  ordenes
}
```

Para llamar a una función desde el script basta con invocar su nombre:

```
nombrefuncion();
```

Una función no lee valores de variables definidas fuera de la función salvo que dentro de la propia función se definan como globales.

La función puede escribirse:

- En cualquier script y ser invocada desde el mismo o cualquier otro.

- En un documento aparte, cuando se invoque hay que indicar dónde buscarlas.

Asignación de valores a variables

A las variables no globales se les pueden asignar sus valores iniciales de varias formas:

- Incluyéndolas en una línea de instrucciones contenida en la propia función.
- Insertando los nombres y valores de las variables dentro del paréntesis:

```
function nom ($a=v1,$b=v2)
```

- Definiendo los nombres de la variable en la definición de la función y sus valores:

```
function nom ($a,$b)
```

En este caso al llamar la función utilizaremos una sintaxis del tipo:

```
nom(valor1, valor2,...);
```

Orden return

Al igual que ocurría en Python si queremos que la función devuelva al código principal uno o varios valores tendremos que utilizar la orden return, con la forma:

```
return $nombrevariable;
```

Al llamar la función tendremos que incluir el nombre de la variable que almacenará el valor devuelto en la forma:

```
$resultado = nombreFuncion(parametros);
```

Cuando la función devuelve más de un valor, podremos recoger los mismos a través de un array, con una sintaxis un poquito más complicada.

Ejemplo: Envíamos dos números a una función y recogemos en un array el valor máximo y le valor mínimo,Las variables v1, v2, etc. recogerán los valores de los elementos del array devuelto por la función.

```php
<?php
  $valor1 = 6;
  $valor2 = 10;
  function calculo($num1, $num2){
    if ($num1 >= $num2){
      $max = $num1;
      $min = $num2;
    }else{
      $max = $num2;
      $min = $num1;
    }
    return array($max, $min);
  }

  $resultado = calculo($valor1, $valor2);
  echo "Máximo: " . $resultado[0] . "<br>"; //
Imprime "Máximo: 10"
  echo "Mínimo: " . $resultado[1]; // Imprime
"Mínimo: 6"
?>
```

Recuerda:

- Una variable definida en una función no puede utilizarse en el script (toma valor cero o “”).
- ***Variable global:*** EXCEPCION a lo anterior. La función puede utilizar valores de variables externas si se ha incluido en la función la orden ***“global nombrevariable;”***, se busca el valor de la variable en el resto del script).

Pasar un número variable de parámetros

Sintaxis (tres puntos antes del nombre)

```php
function nombre(...$parametros){
  ……..
}
```

La función recibirá todos los parámetros que le enviemos y los almacenará en un array de nombre $parametros.

Sumario