Programación Web - Grado Superior

Tema 5: Formularios en PHP 8.4

Procesamiento, Validación, Sanitización y Seguridad

Integración completa HTML + PHP

HTML5

PHP 8.4

Seguridad

Curso 2024/2025

Índice de Contenidos

Estructura del Formulario HTML

`<form>`

Contenedor principal. Define el inicio y fin del área interactiva. Sin él, los inputs no se agrupan para el envío.

`action=" ... "`

URL de destino. Si se omite o está vacío, el formulario se envía a la **misma página** (self-processing).

`method=" ... "`

Protocolo de transporte:

- **GET**: Datos en URL (búsquedas).
- **POST**: Datos en cuerpo (altas, login).

registro.php

HTML5

```php
<!-- Formulario de auto-procesamiento -->
<form action="" method="POST">

    <!-- Campo de texto -->
    <label for="nombre">Nombre:</label>
    <input type="text"
           id="nombre"
           name="usuario_nombre">

    <!-- Botón de envío -->
    <button type="submit">
        Registrar
    </button>

</form>
```

Inputs de Texto Básico

`<input type=" ... ">`

Texto

HTML

```
<input type="text" name="usuario"
placeholder="Tu nombre">
```

PHP (RECEPCIÓN)

```
$user = $_POST['usuario'];
```

Entrada estándar de una línea. Visible y editable por el usuario.

Password

HTML

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

PHP (RECEPCIÓN)

```
$pass = $_POST['clave'];
```

Oculta los caracteres visualmente (••••). **No cifra** los datos en el envío (requiere HTTPS).

Hidden

HTML

```
<input type="hidden"
name="token_id" value="XYZ-123">
```

PHP (RECEPCIÓN)

```
$token = $_POST['token_id'];
```

Invisible para el usuario. Útil para enviar IDs, tokens CSRF o estados sin mostrar inputs.

El atributo **name** es obligatorio. Es la **clave** (key) que usará PHP para identificar el valor en el array superglobal ($_GET o $_POST).

Inputs de Contacto y URL

Validación Cliente (HTML5) vs Servidor (PHP)

Frontend: Tipos Semánticos

formulario.html

```
<!-- Email: Valida formato @ --> <input type="email" name="correo" required> <!-- URL: Valida protocolo http:// --> <input type="url" name="sitio_web"> <!-- Tel: Teclado numérico en móvil --> <input type="tel" name="telefono" pattern="[0-9]{9}" title="9 dígitos requeridos">
```

Nota: HTML5 solo mejora la UX y previene errores accidentales. ¡Es fácilmente manipulable!

Backend: Validación Real

procesar.php

```
$email = $_POST['correo'] ?? ''; $web = $_POST['sitio_web'] ?? ''; $tel = $_POST['telefono'] ?? ''; // 1. Validar Email if (!filter_var($email, FILTER_VALIDATE_EMAIL)) { $errores[] = "Email incorrecto"; } // 2. Validar URL if (!filter_var($web, FILTER_VALIDATE_URL)) { $errores[] = "URL inválida"; } // 3. Validar Teléfono (Regex) if (!preg_match('/^[0-9]{9}$/', $tel)) { $errores[] = "Teléfono debe ser 9 dígitos"; }
```

Inputs Numéricos y Rangos

Cliente (Interfaz)

HTML5

```html
<!-- Input numérico (spinbox) -->
<input type="number" name="edad" min="18"
max="99">
<!-- Slider visual -->
<input type="range" name="nivel" min="0" max="10"
step="0.5">
```

min	max	step
Valor mínimo permitido	Valor máximo permitido	Incremento (1, 0.5, etc.)

Servidor (Seguridad)

PHP 8.4

```php
$edad = $_POST['edad'] ?? 0;
$opciones = ['options' => ['min_range' => 18,
'max_range' => 99]];
if (filter_var($edad, FILTER_VALIDATE_INT,
$opciones)) {
echo "Edad válida";
} else {
echo "Error: Edad fuera de rango";
}
```

¡Importante! Los atributos HTML `min` y `max` son solo sugerencias de interfaz. Un usuario malintencionado puede enviar cualquier número editando el HTML. **Siempre valida el rango en PHP.**

Inputs de Fecha y Hora

Frontend (HTML5)

```html
<label>Fecha de Nacimiento:</label>
<input type="date" name="nacimiento" min="1900-01-01" max="
<label>Hora de Cita:</label>
<input type="time" name="hora_cita">
<input type="datetime-local" name="evento">
```

FORMATO DE ENVÍO (ISO 8601)

```
$_POST['nacimiento'] = "2023-10-25"
```

Backend (PHP 8.4)

```php
<?php
$raw = $_POST['nacimiento'] ?? null;
if ($raw) {
    try {
        $fecha = new DateTime($raw);
        echo "Fecha: " . $fecha->format('d/m/Y');
        $hoy  = new DateTime();
        $edad = $hoy->diff($fecha)->y;
    } catch (Exception $e) {
        echo "Fecha inválida";
    }
}
?>
```

VENTAJA DE DATETIME

Permite cálculos (diff), comparaciones y formateo local sin manipular strings manualmente.

Selección Única: Radio Buttons

Frontend (HTML)

Agrupación

Para que los radios sean excluyentes (solo uno seleccionable), deben compartir exactamente el mismo atributo **name**.

```html
<!-- Grupo: 'suscripcion' -->
<label>
  <input type="radio"
         name="suscripcion"
         value="free" checked>
  Gratis
</label>

<label>
  <input type="radio"
         name="suscripcion"
         value="pro">
  Profesional
</label>
```

Backend (PHP)

Lógica Match

Recibimos el **value** seleccionado. Usamos `match` para asignar lógica limpia basada en la selección.

```php
<?php
// Null coalescing por si no llega nada
$plan = $_POST['suscripcion'] ?? 'free';

// Lógica limpia con match (PHP 8.0+)
$precio = match ($plan) {
    'free'  => 0.00,
    'pro'   => 9.99,
    'corp'  => 49.99,
    default => 0.00,
};

echo "Precio seleccionado: " . $precio;
?>
```

Selección Booleana: Checkbox Simple

Frontend (HTML)

```html
<!-- Checkbox único (Términos) -->
<label>
  <input type="checkbox"
         name="aceptar_terminos"
         value="si">
  Acepto las condiciones
</label>
```

Marcado	Desmarcado
Envia: "aceptar_terminos"="si"	NO envía nada

Backend (PHP)

```php
<?php
// Opción 1: isset() (Clásico)
if (isset($_POST['aceptar_terminos'])) {
    $aceptado = true;
} else {
    $aceptado = false;
}

// Opción 2: Null Coalescing (Moderno)
// Si no existe, asigna false
$aceptado = $_POST['aceptar_terminos'] ?? false;

if (!$aceptado) {
    echo "Debes aceptar los términos.";
}
?>
```

¡Cuidado!

Si el checkbox **no está marcado**, el navegador **no envía** la clave al servidor. Intentar acceder a `$_POST['aceptar_terminos']` directamente generará un *Warning: Undefined array key*.

Selección Múltiple: Checkbox Array

Frontend (HTML)

```html
<label>Intereses:</label>

<!-- Todos tienen name="tags[]" -->
<input type="checkbox" name="tags[]" value="php"> PHP
<input type="checkbox" name="tags[]" value="sql"> SQL
<input type="checkbox" name="tags[]" value="js"> JS
```

⚠ Si olvidas los `[]`, PHP solo recibirá el **último** valor seleccionado, sobrescribiendo los anteriores.

Backend (PHP)

```php
<?php
// Verificar si se envió el array
$tags = $_POST['tags'] ?? [];

if (is_array($tags)) {
    // Recorrer valores seleccionados
    foreach ($tags as $tag) {
        echo "Interés: " . htmlspecialchars($tag);
    }

    // Verificar valor específico
    if (in_array('php', $tags)) {
        echo "¡Excelente elección!";
    }
}
?>
```

```
[ "php" "sql" ]
```

Listas Desplegables: Select

Selección Simple

HTML — STRING

```html
<select name="pais">
    <option value="">Elige...</option>
    <option value="es">España</option>
    <option value="mx">México</option>
</select>
```

PHP — $_POST['PAIS']

```php
<?php
$pais = $_POST['pais'] ?? '';

// Recibe un string simple
// Ejemplo: "es"
?>
```

Selección Múltiple

HTML — ARRAY []

```html
<select name="idiomas[]" multiple size="4">
    <optgroup label="Europa">
        <option value="es">Español</option>
        <option value="en">Inglés</option>
    </optgroup>
</select>
```

PHP — $_POST['IDIOMAS']

```php
<?php
$langs = $_POST['idiomas'] ?? [];

// Recibe un array indexado
// Ejemplo: ["es", "en"]
?>
```

Inputs Especiales: Color y Search

Color Picker

HEX

```html
<!-- Selector visual de color -->
<input type="color"
       name="fondo"
       value="#ff0000">
```

```php
<?php
$color = $_POST['fondo'];

// Validar formato HEX (7 chars)
if (preg_match('/^#[a-f0-9]{6}$/i', $color)) {
    echo "Color válido: " . $color;
}
?>
```

Comportamiento: El navegador siempre envía el color en formato hexadecimal de 7 caracteres (ej: #RRGGBB). No soporta transparencia (alpha).

Search Bar

UX

```html
<!-- Semántico + Botón 'X' -->
<input type="search"
       name="busqueda"
       placeholder="Buscar ... ">
```

```php
<?php
// Limpiar espacios extra
$query = trim($_POST['busqueda'] ?? '');

if (!empty($query)) {
    // Proceder a buscar en BD...
    buscar_productos($query);
}
?>
```

Ventaja UX: En móviles, muestra la tecla "Ir" o "Buscar" en el teclado. En escritorio, suele añadir una "X" para limpiar el campo rápidamente.

Botones y Envío

type="submit"

Comportamiento por defecto. Envía el formulario al servidor.

```
<button type="submit">Enviar</button>
```

type="reset"

Restablece todos los campos a sus valores iniciales. No envía nada.

```
<input type="reset" value="Limpiar">
```

type="button"

Botón inerte. No hace nada por sí mismo. Se usa con JavaScript.

```
<button type="button" onclick="...">JS</button>
```

type="image"

Actúa como submit pero envía coordenadas (x, y) del clic.

```
<input type="image" src="go.png">
```

Múltiples botones de envío

Si das un name al botón submit, PHP recibirá ese valor solo si ese botón fue pulsado. Útil para formularios con acciones "Guardar" y "Borrar".

```
if (isset($_POST['btn_borrar'])) { ... }
```

Validación Cliente (I): Atributos HTML5

`required` BOOLEAN

Impide enviar el formulario si el campo está vacío. El navegador muestra un mensaje nativo.

`pattern=" ... "` REGEX

Valida el valor contra una Expresión Regular. Si falla, muestra el texto del atributo `title`.

`minlength / maxlength` INTEGER

Restringe la cantidad mínima y máxima de caracteres permitidos.

```
<!-- Ejemplo: Registro de Usuario --> <form> <!-- Campo obligatorio --> <label>Usuario:</label> <input type="text" name="username" required> <!-- Longitud restringida --> <label>Contraseña: </label> <input type="password" name="pass" minlength="8" maxlength="20"> <!-- Regex para DNI (8 dígitos + letra) --> <label>DNI:</label> <input type="text" name="dni" pattern="[0-9]{8}[A-Za-z]" title="8 números y una letra"> </form>
```

Nota de Seguridad: La validación HTML5 mejora la experiencia de usuario (UX), pero es fácilmente evitable. **Siempre debes validar los datos nuevamente en PHP.**

Validación Cliente (II): Feedback Visual

CSS Selectors

```css
/* Estado Inválido (Error) */
input:invalid {
  border-color: #ef4444;
  background-image: url('error.svg');
}

/* Estado Válido (Éxito) */
input:valid {
  border-color: #10b981;
  background-image: url('check.svg');
}
```

Resultado en Navegador

Email (Inválido)

usuario@

:invalid

Incluye un signo "@" en la dirección...

Email (Válido)

usuario@dominio.com

:valid

UX Tip: El feedback visual inmediato reduce la frustración del usuario, pero **nunca** confíes en él para la seguridad. Un usuario puede eliminar estas reglas con el Inspector Web.

Flujo de Procesamiento PHP

ESTRUCTURA RECOMENDADA

```php
<?php
$errores = [];
$nombre = "";
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $nombre = trim($_POST['nombre'] ?? '');
    if (empty($nombre)) { $errores[] = "El nombre es obligatorio"; }
    if (empty($errores)) {
        guardar_en_bd($nombre);
        header("Location: exito.php");
        exit;
    }
}
?>
<form method="POST">...</form>
```

Estrategia de Validación en Servidor

1. **`isset() / ??`**
   Verifica si el dato fue enviado. Evita el error "Undefined array key".

2. **`trim()`**
   Elimina espacios en blanco al inicio y final. Convierte " " en "".

3. **`empty() / === ''`**
   Verifica si el valor resultante tiene contenido real.

⚠ **¡Cuidado con empty()!** Considera `"0"` (string cero) como vacío. Si validas un campo numérico (ej: edad 0, cantidad 0), usa `$val === ''` o `is_numeric()` en su lugar.

```php
<?php
// Patrón Moderno (PHP 7.4+): Recibir y Validar
$nombre = trim($_POST['nombre'] ?? '');
$email  = trim($_POST['email']  ?? '');

if ($nombre === '') { $errores[] = "El nombre es obligatorio"; }
if ($email === '')  { $errores[] = "El email es obligatorio"; }
?>
```

Filtros de Validación PHP

filter_var()

```php
// Valida una variable existente $email = "test@example.com"; if (filter_var($email, FILTER_VALIDATE_EMAIL)) { // Es válido }
```

filter_input() Recomendado

```php
// Accede directamente a $_POST/$_GET // Evita warnings si la clave no existe if (filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL)) { // Email válido y seguro }
```

Filtros Comunes

FILTER_VALIDATE_EMAIL	Formato de correo estándar
FILTER_VALIDATE_URL	URLs absolutas (http/https)
FILTER_VALIDATE_INT	Números enteros
FILTER_VALIDATE_IP	Direcciones IP (v4/v6)

Valor de Retorno: Estas funciones devuelven el **dato validado** si es correcto, o `false` si falla.

(Cuidado: validar el entero 0 puede devolver 0, que PHP evalúa como false en un if simple).

Sanitización de Datos

Validación

Verifica si los datos cumplen con un formato o criterio específico. No modifica los datos.

`Retorna: true / false`

Sanitización

Limpia o modifica los datos para eliminar caracteres peligrosos o no deseados.

`Retorna: String modificado`

La Regla de Oro: Escapar al Salir (Output Escaping)

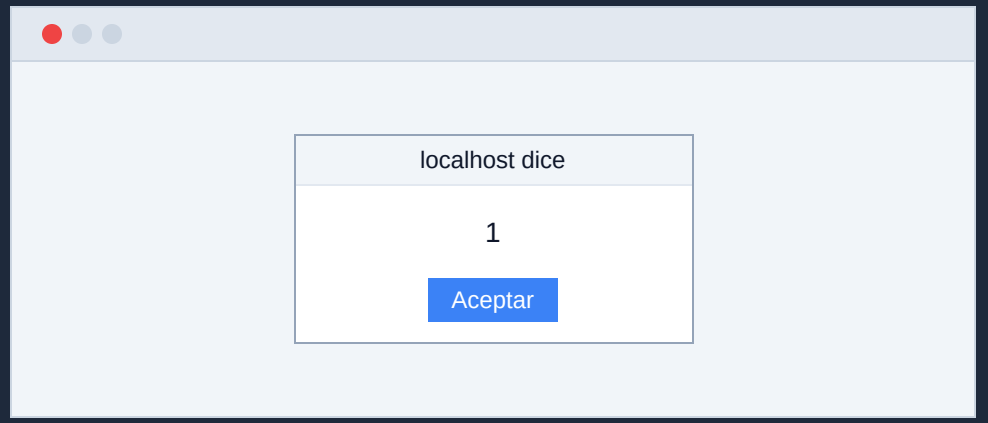
```php
// Al mostrar datos de usuario en HTML, SIEMPRE usa: echo htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
// Convierte caracteres especiales en entidades HTML seguras // < se convierte en &lt; // > se convierte en &gt;
```

Seguridad: Prevención XSS

Vulnerable

```php
<?php // Salida directa (PELIGRO) echo $_GET['msg']; ?>
```

Si el input es `<script>alert(1)</script>`, el navegador lo interpreta como código ejecutable.



Seguro

```php
<?php // Salida escapada (CORRECTO) echo htmlspecialchars($_GET['msg']); ?>
```

Convierte caracteres especiales en entidades HTML. El navegador muestra el texto, no lo ejecuta.

Seguridad: Prevención CSRF

CSRF (Cross-Site Request Forgery): Un atacante engaña al navegador de la víctima para que envíe una petición (ej: cambiar contraseña) en una sesión donde el usuario ya está autenticado.

1 Generar Token (Formulario)

```php
<?php session_start();
if (empty($_SESSION['csrf_token'])) {
  $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
?>

<form method="POST">
  <input type="hidden"
         name="csrf_token"
         value="<?= $_SESSION['csrf_token'] ?>">
  ...
</form>
```

2 Verificar Token (Procesar)

```php
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  $token_post    = $_POST['csrf_token'] ?? '';
  $token_session = $_SESSION['csrf_token'] ?? '';
  if (!hash_equals($token_session, $token_post)) {
    die("Error de seguridad: Token inválido");
  }
  // Token válido: Procesar...
}
?>
```

Subida de Archivos (I): Configuración

Frontend (HTML)

```
<form action="upload.php" method="POST"
enctype="multipart/form-data"> <input
type="file" name="fichero">
<button>Subir</button> </form>
```

¡Crítico!
Sin `enctype="multipart/form-data"`, el navegador solo enviará el nombre del archivo, no el contenido binario.

Backend (php.ini)

```
; Habilitar subidas file_uploads = On ; Tamaño
máx. por archivo individual upload_max_filesize =
2M ; Tamaño máx. de toda la petición POST ; (Debe
ser mayor que upload_max_filesize) post_max_size =
8M ; Tiempo máx. de ejecución (segundos)
max_execution_time = 30
```

JERARQUÍA DE LÍMITES

`memory_limit` > `post_max_size` > `upload_max_filesize`

Subida de Archivos (II): Procesamiento

$_FILES['fichero']

```
[ 'name' ⇒ "foto.jpg", 'full_path'⇒
"foto.jpg", 'type' ⇒ "image/jpeg", //
¡Inseguro! 'tmp_name' ⇒ "/tmp/phpYzdqkD",
'error' ⇒ 0, // 0 = UPLOAD_ERR_OK 'size' ⇒
102400 // Bytes ]
```

tmp_name: El archivo se guarda temporalmente aquí. Si no lo mueves con `move_uploaded_file()` antes de que termine el script, **se eliminará automáticamente**.

</> Lógica Segura

```
<?php $file = $_FILES['fichero']; // 1. Verificar
errores de subida if ($file['error'] ===
UPLOAD_ERR_OK) { // 2. Validar tipo REAL (MIME) $mime
= mime_content_type($file['tmp_name']); if ($mime ===
'image/jpeg') { // 3. Mover a destino final $destino =
'uploads/' . basename($file['name']);
move_uploaded_file($file['tmp_name'], $destino); echo
"Subida exitosa"; } } ?>
```

Seguridad: Nunca confíes en `$_FILES['...']['type']`, ya que es enviado por el navegador y puede ser falsificado. Usa siempre `mime_content_type()` o `finfo_file()` para verificar el contenido real.

Usabilidad: Sticky Forms

Definición: Un formulario "Sticky" (pegajoso) mantiene los valores introducidos por el usuario después de enviarlo, especialmente si hubo errores de validación. Evita que el usuario tenga que reescribir todo.

A Inputs de Texto

```
<!-- Usar el atributo value --> <input type="text" name="nombre" value="<?= htmlspecialchars($nombre) ?>"> <!-- Textarea (dentro de las etiquetas) --> <textarea name="bio"> <?= htmlspecialchars($bio) ?> </textarea>
```

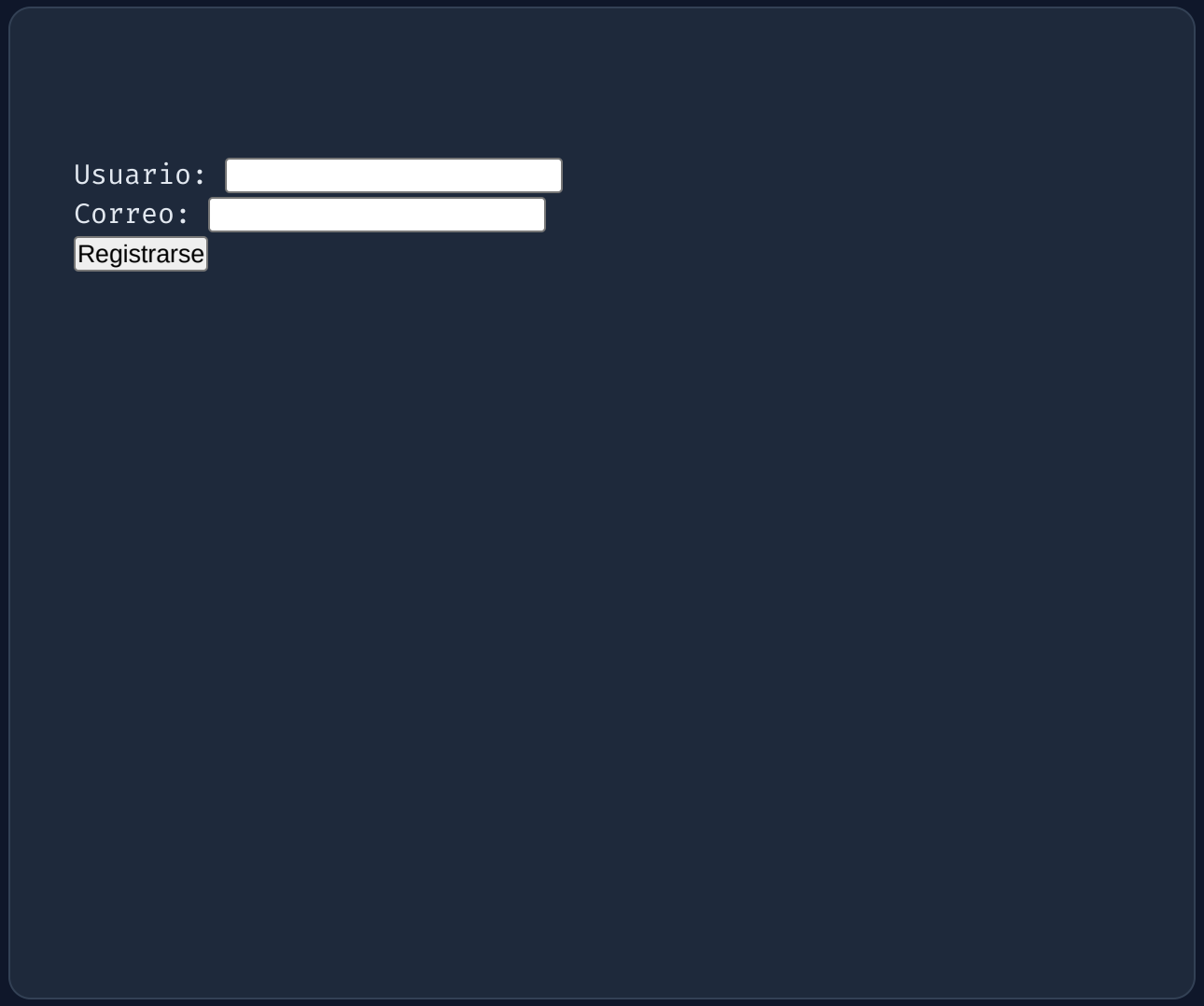
Nota: Siempre usa `htmlspecialchars()` para evitar que comillas dobles en el texto del usuario rompan el atributo HTML.

Selecciones (Checkbox/Select)

```
<!-- Checkbox: atributo checked --> <input type="checkbox" name="terms" <?= $terms_accepted ? 'checked' : '' ?>> <!-- Select: atributo selected --> <select name="pais"> <option value="es" <?= ($pais == 'es') ? 'selected' : '' ?>> España </option> </select>
```

Usa el operador Null Coalescing al inicio: `$nombre = $_POST['nombre'] ?? '';` para evitar errores en la primera carga.

Ejemplo Completo (I): HTML



1 Seguridad Base

Uso de `POST` para datos sensibles y un campo oculto (`hidden`) para el token CSRF.

2 Accesibilidad

La etiqueta `<label>` se vincula explícitamente al input mediante el atributo `for` y el `id`. Esto es vital para lectores de pantalla.

3 Semántica y UX

Usar `type="email"` activa el teclado correcto en móviles y validación básica del navegador.

Ejemplo Completo (II): Validación PHP

VALIDAR.PHP

```php
<?php
// 1. Inicializar array de errores
$errores = [];
// 2. Recoger y limpiar datos
$user = trim($_POST['username'] ?? '');
$email = trim($_POST['email'] ?? '');
$pass = $_POST['password'] ?? '';
// 3. Validar Usuario (Obligatorio)
if (empty($user)) {
    $errores[] = "El nombre de usuario es obligatorio.";
}
// 4. Validar Email (Formato)
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $errores[] = "El formato del email no es válido.";
}
// 5. Validar Password (Longitud)
if (strlen($pass) < 8) {
    $errores[] = "La contraseña debe tener al menos 8 caracteres.";
}
?>
```

Estructura de $errores

```
$errores = [
  0 => "El formato del email...",
  1 => "La contraseña debe..."
];
```

Estrategia "Acumular Errores"

En lugar de detener el script al primer error (`die()`), acumulamos **todos** los problemas en un array. Esto permite mostrar al usuario todo lo que debe corregir de una sola vez, mejorando la UX.

Resultado Final

Si `empty($errores)` es **true**, los datos son válidos y podemos proceder a guardar.

Ejemplo Completo (III): Procesamiento

Lógica de Decisión

```php
<?php
// Si NO hay errores...
if (empty($errores)) {
    // 1. Guardar en BD (Simulado)
    save_user($user, $email, $pass);

    // 2. Redirigir (Patrón PRG)
    header("Location: index.php?status=ok");
    exit;
} else {
    // 3. Si hay errores, el script continúa
    // y muestra el formulario de nuevo
}
?>
```

Patrón PRG (Post-Redirect-Get): Siempre redirige después de un POST exitoso. Esto evita que el usuario duplique la acción (ej: doble pago) si pulsa "Actualizar" en el navegador.

Feedback al Usuario

```php
<!-- Mostrar Errores -->
<?php if (!empty($errores)): ?>
    <div class="alert alert-danger">
        <ul>
            <?php foreach ($errores as $error): ?>
                <li><?= $error ?></li>
            <?php endforeach; ?>
        </ul>
    </div>
<?php endif; ?>

<!-- Mostrar Éxito (tras redirect) -->
<?php if (($_GET['status'] ?? '') === 'ok'): ?>
    <div class="alert alert-success">
        ¡Registro completado con éxito!
    </div>
<?php endif; ?>
```

El nombre es obligatorio.

Ejercicio 1: Formulario de Contacto

Requisitos

- Crear un script PHP único que muestre y procese el formulario.
- **Campos:** Nombre, Email, Asunto (texto), Mensaje (textarea).
- **Validación:** Todos los campos son obligatorios. El email debe tener un formato válido.
- **Seguridad:** Implementar protección contra XSS (al mostrar datos) y CSRF (con tokens).
- **UX:** El formulario debe ser "pegajoso" (mantener valores si hay errores).
- **Feedback:** Mostrar una lista con todos los errores o un mensaje de éxito tras redirigir.

Solución: Puntos Clave

1. Validación

```php
<?php $errores = []; $datos = []; if ($_SERVER['...'] === 'POST') { // Validar CSRF... $campos = ['nombre', 'email', 'asunto', 'mensaje']; foreach($campos as $c) { $datos[$c] = trim($_POST[$c] ?? ''); if(empty($datos[$c])) { $errores[$c] = "El campo es obligatorio"; } } if (!filter_var($datos['email'], FILTER_VALIDATE_EMAIL)) { $errores['email'] = "Email inválido"; } } ?>
```

2. Renderizado

```php
<!-- CSRF Token --> <input type="hidden" name="csrf" value="<?= $_SESSION['token'] ?>"> <!-- Sticky Form + XSS Protection --> <label for="nombre">Nombre</label> <input type="text" id="nombre" value="<?= htmlspecialchars($datos['nombre'] ?? '') ?>"> <!-- Mostrar error específico --> <?php if(isset($errores['nombre'])): ?> <span class="error"> <?= $errores['nombre'] ?> </span> <?php endif; ?>
```

3. Flujo y Feedback

```php
<?php // Al final de la validación if (empty($errores)) { // Lógica de envío de email... mail('...', '...', $datos['mensaje']); // Redirigir (PRG) header('Location: form.php?status=ok'); exit; } ?> <!-- En el HTML, para el feedback --> <?php if (($_GET['status'] ?? '') === 'ok'): ?> <div class="success"> Mensaje enviado con éxito. </div> <?php endif; ?>
```

Ejercicio 2: Perfil con Avatar

Especificaciones

Objetivo

Crear formulario para actualizar nombre de usuario y subir avatar.

Validación de Archivo

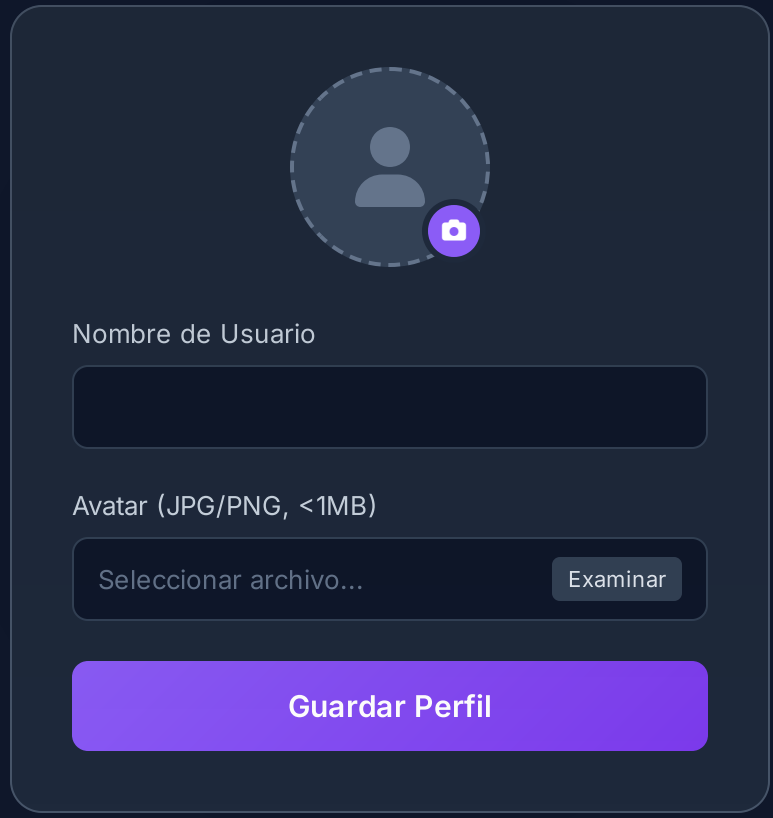
- Estado: `UPLOAD_ERR_OK`
- Tamaño: Máx 1MB
- Tipo: Solo `image/jpeg` o `image/png`

Seguridad

Ignorar nombre original. Generar nombre único: `uniqid() . ".ext"`.

Procesamiento

Mover con `move_uploaded_file()` a `/uploads`.



Solución: Puntos Clave

1. Verificación

```php
<?php $file = $_FILES['avatar'] ?? null; // Check de subida y error if (!$file || $file['error'] !== UPLOAD_ERR_OK) { $errores[] = "Error al subir el avatar."; } // Check de tamaño (1MB) $max_size = 1 * 1024 * 1024; if ($file['size'] > $max_size) { $errores[] = "El archivo supera 1MB."; } ?>
```

2. Seguridad

```php
<?php // Whitelist de tipos MIME $allowed_types = ['image/jpeg', 'image/png']; $mime_type = mime_content_type($file['tmp_name']); if (!in_array($mime_type, $allowed_types)) { $errores[] = "Solo se admiten JPG o PNG."; } // Generar nombre único y seguro $extension = pathinfo( $file['name'], PATHINFO_EXTENSION ); $new_name = uniqid('avatar_') . '.' . $extension; ?>
```

3. Procesamiento

```php
<?php // Si no hay errores, mover if (empty($errores)) { $destination = __DIR__ . '/uploads/' . $new_name; if (move_uploaded_file($file['tmp_name'], $destination)) { // Guardar $new_name en la BD... header('Location: perfil.php?success=1'); exit; } else { $errores[] = "Error al guardar el archivo."; } } ?>
```