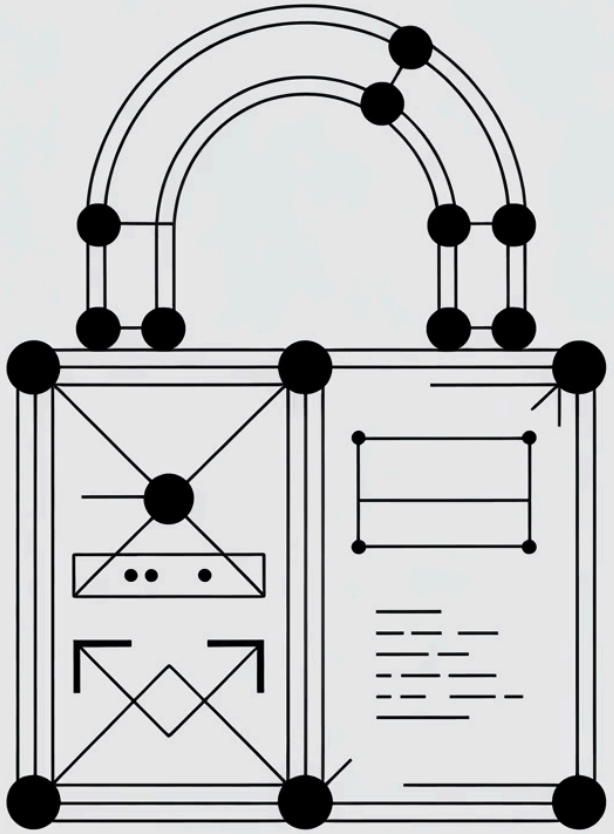
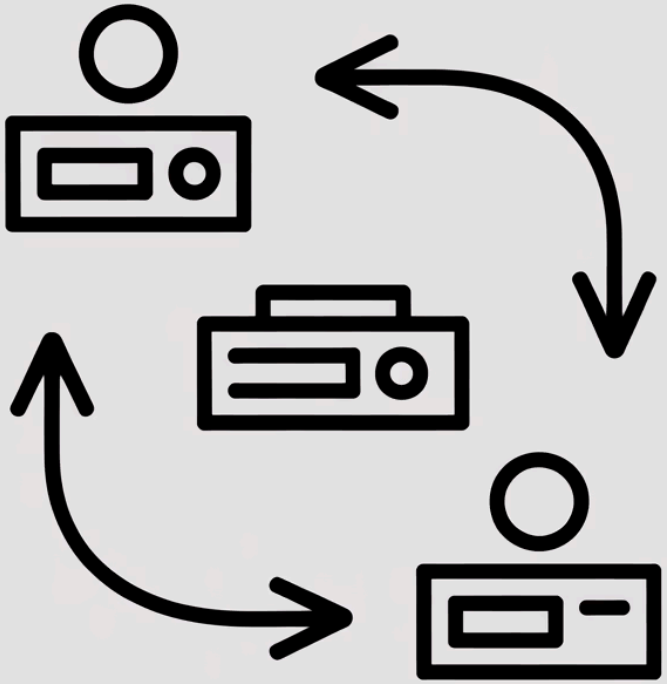
Sesiones, Cookies y Autenticación

MÓDULO: DESARROLLO WEB

Mantenimiento del estado y seguridad en aplicaciones web PHP. Aprenderemos cómo gestionar la información de usuarios entre peticiones HTTP y proteger nuestras aplicaciones mediante sistemas de autenticación robustos.





¿Por qué necesitamos "mantener el estado"?

El Protocolo HTTP

El protocolo **HTTP es "stateless"** (sin estado). Esto significa que, por diseño, el servidor web trata cada petición de manera completamente independiente, sin recordar nada sobre peticiones anteriores del mismo usuario.

La Solución

Sin estado: Cada clic es como si fueras un usuario completamente nuevo. El servidor no te reconoce.

Con estado: Podemos mantener información persistente como un carrito de compra activo, preferencias del usuario, o sesiones de autenticación.

Para resolver esta limitación fundamental de HTTP, utilizamos dos mecanismos complementarios: **Sesiones** y **Cookies**.

Comparativa: Cookies vs. Sesiones

Cookies (Cliente)

- Almacenadas en el **navegador** del usuario
- Inseguras por naturaleza - el usuario puede editarlas
- Ideales para datos no sensibles como preferencias o idioma
- Caducidad definida por fecha específica
- Limitadas a 4KB de tamaño

Sesiones (Servidor)

- Almacenadas en el **servidor** de manera segura
- Más seguras - el usuario solo tiene un identificador
- Ideales para autenticación y datos sensibles
- Se destruyen al cerrar el navegador (por defecto)
- Sin límite práctico de tamaño

La elección entre cookies y sesiones depende del tipo de información que necesites almacenar y del nivel de seguridad requerido. En la práctica, ambos mecanismos suelen trabajar juntos: las cookies almacenan el ID de sesión, mientras que los datos importantes permanecen seguros en el servidor.

Control de Sesiones en PHP

Una sesión crea un archivo temporal en el servidor identificado por un **ID único** llamado PHPSESSID. Este identificador se envía automáticamente al navegador como cookie, permitiendo al servidor reconocer al usuario en peticiones futuras.

01

session_start()

Inicia o reanuda la sesión existente. **Debe ejecutarse al principio del script**, antes de cualquier salida HTML, incluyendo espacios en blanco.

02

$_SESSION

Array superglobal para almacenar y recuperar datos de sesión. Los datos persisten entre diferentes páginas mientras la sesión esté activa.

03

session_destroy()

Elimina todos los datos de la sesión y destruye el archivo temporal del servidor. Útil para implementar la funcionalidad de logout.

Importante: El archivo de sesión se almacena por defecto en el directorio temporal del servidor (generalmente /tmp en Linux). La ubicación puede configurarse en php.ini mediante session.save_path.

Implementación de Sesiones: Ejemplos de Código

Veamos cómo trabajar con sesiones en PHP mediante ejemplos prácticos. Estos patrones son fundamentales para cualquier aplicación web que necesite mantener el estado del usuario.

Iniciar sesión y guardar datos

```
<?php
// 1. Iniciar la sesión (SIEMPRE primera línea)
session_start();

// 2. Guardar datos en el servidor
$_SESSION['usuario'] = 'JuanDev';
$_SESSION['rol'] = 'admin';
$_SESSION['ultima_actividad'] = time();

echo "Sesión iniciada para: " . $_SESSION['usuario'];
?>
```

Recuperar datos en otra página

```
<?php
// En cualquier otra página del sitio
session_start();

// Verificar si existe la variable de sesión
if (isset($_SESSION['usuario'])) {
 echo "Bienvenido de nuevo, " . $_SESSION['usuario'];
 echo "<br>Tu rol es: " . $_SESSION['rol'];
} else {
 echo "No has iniciado sesión";
 header("Location: login.php");
 exit();
}
?>
```

Gestión y Cierre de Sesiones

1

Eliminar una Variable Específica

```
unset($_SESSION['clave']);
```

Borra una variable específica de la memoria de sesión, pero mantiene la sesión activa y el resto de variables intactas. Útil para limpiar datos temporales.

2

Cerrar Sesión Completamente

```
session_destroy();
```

Elimina todos los datos de la sesión del servidor. Es el método apropiado para implementar el botón "Cerrar Sesión" o "Logout".

3

Implementar Timeout de Seguridad

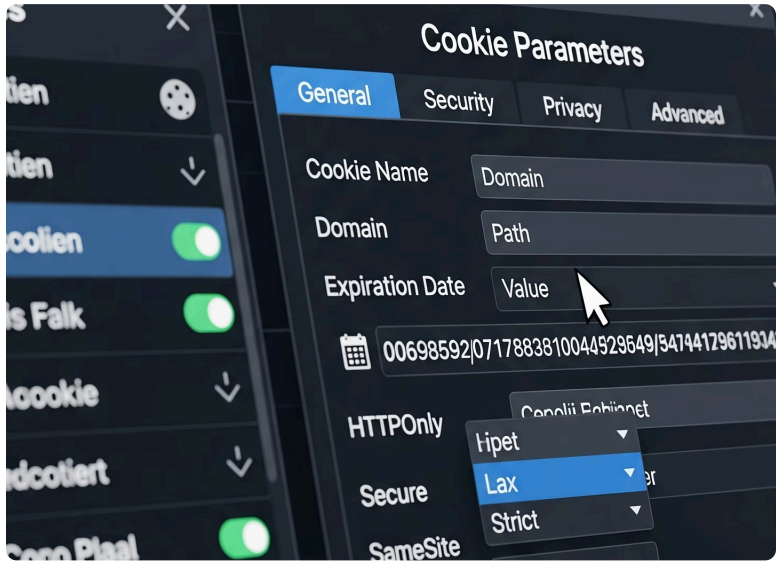
```
<?php
$timeout = 600; // 10 minutos
if (isset($_SESSION['ultima_actividad'])) {
    if (time() - $_SESSION['ultima_actividad'] > $timeout) {
        session_destroy();
        header("Location: login.php?timeout=1");
    }
}
$_SESSION['ultima_actividad'] = time();
?>
```

Por seguridad, es fundamental cerrar sesiones tras períodos de inactividad. Esto previene ataques de session hijacking en ordenadores compartidos.

Uso de Cookies en PHP

¿Qué son las Cookies?

Las cookies son pequeños archivos de texto plano que el servidor envía al navegador y este almacena localmente. En cada petición posterior, el navegador envía automáticamente estas cookies de vuelta al servidor.



Parámetros Fundamentales

- **Name:** Identificador único de la cookie
- **Value:** La información a guardar (máximo 4KB)
- **Expires:** Fecha de caducidad en formato timestamp UNIX
- **Path:** Ruta del servidor donde la cookie es válida
- **Domain:** Dominio para el cual es válida la cookie
- **Secure:** Solo se envía por HTTPS
- **HttpOnly:** Inaccesible desde JavaScript (protección XSS)

Precaución de Seguridad: Nunca almacenes contraseñas, números de tarjetas de crédito o cualquier dato sensible en cookies sin encriptación robusta. Las cookies son fácilmente accesibles y editables por el usuario.

Creación de Cookies en PHP: Código

La función `setcookie()` debe llamarse **antes de cualquier salida HTML**, incluyendo espacios en blanco, etiquetas HTML o echo statements. Esto se debe a que las cookies se envían en las cabeceras HTTP.

Crear una cookie básica

```
<?php
// Sintaxis completa: setcookie(nombre, valor, expiración, ruta, dominio, secure, httponly)
$nombre = 'usuario_idioma';
$valor = 'es_ES';
$expira = time() + 3600; // Caduca en 1 hora (3600 segundos)

setcookie($nombre, $valor, $expira, '/');
echo "Cookie guardada exitosamente.";
?>
```

Recuperar el valor de una cookie

```
<?php
// Las cookies están disponibles en el array superglobal $_COOKIE
if (isset($_COOKIE['usuario_idioma'])) {
    echo "Tu idioma es: " . $_COOKIE['usuario_idioma'];
    // Salida: Tu idioma es: es_ES
} else {
    echo "No se ha establecido preferencia de idioma";
}
?>
```

Eliminar una cookie

```
<?php
// Para borrar una cookie, establecemos fecha de expiración en el pasado
setcookie('usuario_idioma', '', time() - 3600, '/');
echo "Cookie eliminada";
?>
```

Seguridad: Perfiles y Roles de Usuario

Sistema de Roles

Definir niveles de acceso es vital para proteger funcionalidades críticas de la aplicación:

Administrador

Control total del sistema

Editor

Crear y modificar contenido

Usuario

Acceso básico de lectura

Conceptos Fundamentales

Autenticación: Verifica la identidad del usuario. Responde a la pregunta "¿Quién eres?" mediante credenciales como usuario y contraseña.

Autorización: Define qué puede hacer un usuario autenticado. Responde a "¿Qué permisos tienes?" mediante roles y privilegios.

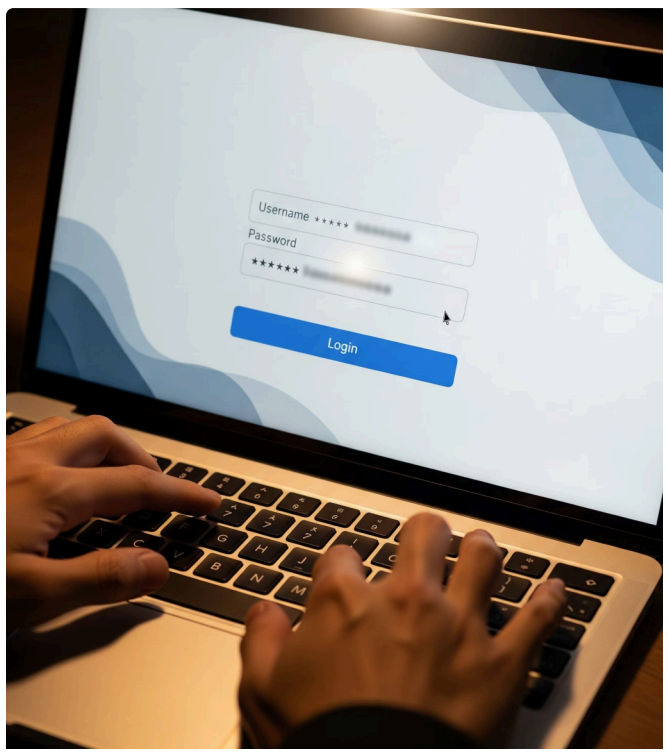
Ejemplo de implementación de roles

```
<?php
session_start();

// Verificar rol antes de mostrar contenido sensible
function verificarRol($rol_requerido) {
 if (!isset($_SESSION['rol']) || $_SESSION['rol'] !== $rol_requerido) {
 header("Location: acceso_denegado.php");
 exit();
 }
}

// Uso en una página de administración
verificarRol('admin');
echo "Bienvenido al panel de administración";
?>
```

El Proceso Completo de Login



Formulario HTML

Recoge las credenciales del usuario (username y password) usando el método POST por seguridad.

Recepción en PHP

El servidor recibe los datos mediante $_POST['username'] y $_POST['password'].

Consulta a Base de Datos

Se busca el usuario en la base de datos y se verifica la contraseña usando password_verify() con hash seguro.

Crear Sesión

Si las credenciales son correctas, se crean variables de sesión: $_SESSION['user_id'], $_SESSION['username'], $_SESSION['rol'].

Redirección

Finalmente, se redirige al usuario a su panel mediante header("Location: dashboard.php").

Ejemplo de formulario de login

```
<form method="POST" action="procesar_login.php">
 <input type="text" name="username" required>
 <input type="password" name="password" required>
 <button type="submit">Iniciar Sesión</button>
</form>
```

Validación con Base de Datos: Código Seguro

La seguridad en el proceso de autenticación es crítica. Veamos un ejemplo correcto usando **prepared statements** para prevenir inyección SQL y **password hashing** para proteger contraseñas.

❌ Ejemplo INSEGURO (nunca hacer esto)

```
<?php
// MAL: Vulnerable a SQL Injection
$user = $_POST['username'];
$pass = $_POST['password'];
$sql = "SELECT * FROM users WHERE username = '$user' AND password = '$pass'";
$result = mysqli_query($conn, $sql);
?>
```

✅ Ejemplo SEGURO (usar siempre)

```
<?php
session_start();
$conn = new mysqli("localhost", "usuario", "contraseña", "basedatos");

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Preparar consulta segura
    $stmt = $conn->prepare("SELECT id, username, password, rol FROM users WHERE username = ?");
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows === 1) {
        $user = $result->fetch_assoc();

        // Verificar password hasheado
        if (password_verify($password, $user['password'])) {
            // Login exitoso
            $_SESSION['user_id'] = $user['id'];
            $_SESSION['username'] = $user['username'];
            $_SESSION['rol'] = $user['rol'];
            $_SESSION['ultima_actividad'] = time();

            header("Location: dashboard.php");
            exit();
        }
    }

    // Si llegamos aquí, las credenciales son incorrectas
    $error = "Usuario o contraseña incorrectos";
}
?>
```

Crear contraseñas hasheadas

```
<?php
// Al registrar un nuevo usuario
$password_plano = "miContraseña123";
$password_hash = password_hash($password_plano, PASSWORD_DEFAULT);
// Guardar $password_hash en la base de datos
?>
```

Autenticación Moderna: OpenID

¿Qué es OpenID?

OpenID es un protocolo de autenticación descentralizado que permite a los usuarios utilizar una **identidad única** para acceder a múltiples sitios web sin necesidad de crear nuevas cuentas en cada uno.

Características Principales

- **Single Sign-On (SSO):** Una sola cuenta para múltiples servicios
- **Descentralizado:** No depende de un único proveedor
- **Seguro:** Las credenciales nunca se comparten con terceros
- **Estándar abierto:** Implementación libre y gratuita

OpenID responde a la pregunta fundamental: **"¿Es este usuario quien dice ser?"**

Flujo de Autenticación

01

Usuario solicita login

En tu sitio web

02

Redirección al IdP

Proveedor de Identidad

03

Usuario se autentica

Con sus credenciales

04

IdP confirma identidad

Devuelve token

05

Acceso concedido

Usuario autenticado

Proveedores populares: Google, Facebook, Microsoft, Apple y GitHub son algunos de los proveedores de identidad OpenID más utilizados en la actualidad.

OAuth: Autorización Delegada

¿Qué es OAuth?

OAuth es un estándar abierto de autorización que permite a las aplicaciones acceder a recursos protegidos de un usuario **sin compartir las contraseñas**. En lugar de credenciales, utiliza tokens de acceso.

Diferencia Clave

OpenID: Se enfoca en *autenticación* (identidad)

OAuth: Se enfoca en *autorización* (permisos)

Roles en OAuth 2.0

Resource Owner

El usuario dueño de los datos que se quieren compartir

Client

Tu aplicación que solicita acceso a los datos

Authorization Server

Servidor que valida y emite tokens (ej: Google, Facebook)

Resource Server

Servidor que contiene los datos protegidos

Ejemplo práctico: "Iniciar sesión con Google"

```
<?php
// 1. Usuario hace clic en "Login con Google"
// 2. Se redirige a Google para autorización
$google_auth_url = "https://accounts.google.com/o/oauth2/auth?" . http_build_query([
 'client_id' => 'TU_CLIENT_ID',
 'redirect_uri' => 'https://tuweb.com/callback.php',
 'scope' => 'email profile',
 'response_type' => 'code'
]);

// 3. Google redirige de vuelta con un código
// 4. Intercambiar código por access token
// 5. Usar token para obtener datos del usuario
?>
```

Caso Práctico: Contador de Visitas

Implementaremos un sistema simple pero ilustrativo que cuenta cuántas veces un usuario ha visitado una página utilizando sesiones. Este patrón es fundamental para entender la persistencia de datos entre peticiones.

Código completo del contador

```
<?php
session_start();

// Verificar si es la primera visita
if (!isset($_SESSION['contador'])) {
    // Primera visita: inicializar contador
    $_SESSION['contador'] = 1;
    $_SESSION['primera_visita'] = date('Y-m-d H:i:s');
    $mensaje = "¡Bienvenido! Esta es tu primera visita.";
} else {
    // Visitas posteriores: incrementar
    $_SESSION['contador']++;
    $tiempo_transcurrido = time() - strtotime($_SESSION['primera_visita']);
    $mensaje = "Has visitado esta página " . $_SESSION['contador'] . " veces.";
}
?>

<!DOCTYPE html>
<html>
<head>
    <title>Contador de Visitas</title>
</head>
<body>
    <h1><?php echo $mensaje; ?></h1>
    <p>Primera visita: <?php echo $_SESSION['primera_visita']; ?></p>

    <a href="?reset=1">Resetear contador</a>

    <?php
    // Opción para resetear
    if (isset($_GET['reset'])) {
        unset($_SESSION['contador']);
        unset($_SESSION['primera_visita']);
        header("Location: " . $_SERVER['PHP_SELF']);
        exit();
    }
    ?>
</body>
</html>
```

Variación: Contador con cookies (persiste al cerrar navegador)

```
<?php
if (!isset($_COOKIE['visitas'])) {
 $visitas = 1;
} else {
 $visitas = (int)$_COOKIE['visitas'] + 1;
}

setcookie('visitas', $visitas, time() + (86400 * 30)); // 30 días
echo "Visitas totales: " . $visitas;
?>
```

Ejercicios de Repaso

1

Diferencia Clave

Explica con tus propias palabras por qué es más seguro guardar el ID de un usuario en una **Sesión** en lugar de en una **Cookie**. Menciona al menos dos razones técnicas.

- ¿Qué podría hacer un usuario malintencionado con una cookie?
- ¿Dónde se almacenan físicamente los datos en cada caso?

2

Ciclo de Vida

Responde estas preguntas sobre la persistencia de datos:

- ¿Qué sucede con $_SESSION['usuario'] cuando cierras completamente el navegador?
- ¿Y si usas unset($_SESSION['usuario']) pero mantienes el navegador abierto?
- ¿Cómo difiere esto del comportamiento de las cookies con fecha de expiración?

3

Seguridad

Investiga y explica por qué debemos implementar:

- `session_regenerate_id()` después del login
- Timeouts de inactividad en sesiones sensibles
- El flag HttpOnly en cookies de sesión

4

Autenticación Moderna

Con tus propias palabras, diferencia **OpenID** de **OAuth**:

- ¿Cuál usarías solo para verificar la identidad de un usuario?
- ¿Cuál necesitas para acceder a la lista de contactos de Google?
- ¿Pueden trabajar juntos? Proporciona un ejemplo.

Reto 1: Oferta de Bienvenida

Objetivo del Reto

Crear un sistema que muestre una oferta especial solo la primera vez que un usuario visita tu sitio. Las visitas posteriores deben reconocer que el usuario ya vio la oferta.

Requisitos Técnicos

1. Iniciar una sesión al principio del script
2. Verificar si existe la cookie 'oferta_visto'
3. Si NO existe: crear cookie con validez de 24 horas y mostrar mensaje de bienvenida con la oferta
4. Si SÍ existe: mostrar mensaje indicando que ya conoce las ofertas
5. Bonus: Añadir un botón para "ver oferta de nuevo" que elimine la cookie

Pistas de Implementación

```
<?php
session_start();

// Verificar cookie
if (!isset($_COOKIE['oferta_visto'])) {
   // Primera visita
   $expiracion = time() + (24 * 60 * 60);
   setcookie('oferta_visto', 'si',
          $expiracion, '/');

   $mensaje = "¡Bienvenido! ...";
} else {
   // Ya visitó antes
   $mensaje = "Ya conoces...";
}

echo $mensaje;
?>
```

Extra: ¿Cómo modificarías el código para mostrar cuántas horas faltan hasta que expire la oferta?

Reto 2: Preferencias de Tema

Desarrolla una aplicación web que recuerde la preferencia de tema visual (claro/oscuro) del usuario utilizando cookies. El tema debe persistir incluso después de cerrar el navegador.

Estructura HTML del Formulario

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .light { background: #ffffff; color: #000000; }
        .dark { background: #1a1a1a; color: #ffffff; }
        body { transition: all 0.3s ease; padding: 20px; }
    </style>
</head>
<body class="<?php echo $tema; ?>">
    <h1>Selector de Tema</h1>
    <form method="POST">
        <select name="tema">
            <option value="light">Tema Claro</option>
            <option value="dark">Tema Oscuro</option>
        </select>
        <button type="submit">Guardar Preferencia</button>
    </form>
</body>
</html>
```

Código PHP para Gestionar el Tema

```
<?php
// Procesar formulario
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $tema_seleccionado = $_POST['tema'];
    // Guardar por 30 días
    setcookie('tema_preferido', $tema_seleccionado,
              time() + (30 * 24 * 60 * 60), '/');
    $tema = $tema_seleccionado;
} else {
    // Cargar preferencia guardada o usar por defecto
    $tema = isset($_COOKIE['tema_preferido'])
            ? $_COOKIE['tema_preferido']
            : 'light';
}
?>
```

Mejora avanzada: Implementa detección automática del tema del sistema operativo usando JavaScript y guárdalo como preferencia inicial.

Reto 3: Límite de Intentos de Login

El Problema de Seguridad

Los ataques de fuerza bruta intentan adivinar contraseñas probando miles de combinaciones. Implementaremos un sistema que bloquee temporalmente la cuenta tras varios intentos fallidos.

Objetivos

- Limitar intentos de login a 3
- Bloquear acceso tras exceder límite
- Mostrar mensaje apropiado
- Permitir reset manual (simulado)

Implementación Completa

```
<?php
session_start();

// Inicializar contador si no existe
if (!isset($_SESSION['intentos'])) {
 $_SESSION['intentos'] = 0;
}

// Procesar intento de login
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
 $usuario = $_POST['usuario'] ?? '';
 $password = $_POST['password'] ?? '';

 // Simulación: credenciales correctas
 $usuario_correcto = 'admin';
 $pass_correcta = '1234';

 if ($usuario === $usuario_correcto
 && $password === $pass_correcta) {
 // Login exitoso
 $_SESSION['intentos'] = 0;
 $_SESSION['logueado'] = true;
 header("Location: dashboard.php");
 exit();
 } else {
 // Login fallido
 $_SESSION['intentos']++;
 }
}

// Verificar si está bloqueado
$bloqueado = $_SESSION['intentos'] >= 3;

// Procesar reset
if (isset($_GET['reset'])) {
 unset($_SESSION['intentos']);
 header("Location: " . $_SERVER['PHP_SELF']);
 exit();
}
?>

<!DOCTYPE html>
<html>
<body>
 <h1>Login Seguro</h1>

 <?php if ($bloqueado): ?>
 <div style="color: red;">
 <h2>
```

Reto 4: Mini Carrito de Compras

Las sesiones pueden almacenar estructuras de datos complejas como arrays. Construiremos un carrito de compras básico que mantiene una lista de productos seleccionados por el usuario.

Código completo del carrito

```
<?php
session_start();

// Inicializar carrito si no existe
if (!isset($_SESSION['carrito'])) {
 $_SESSION['carrito'] = [];
}

// Procesar acción de añadir producto
if (isset($_GET['producto'])) {
 $producto = htmlspecialchars($_GET['producto']);
 $_SESSION['carrito'][] = [
 'nombre' => $producto,
 'fecha_agregado' => date('H:i:s'),
 'id' => uniqid()
 ];
 header("Location: " . strtok($_SERVER['REQUEST_URI'], '?'));
 exit();
}

// Procesar vaciado de carrito
if (isset($_GET['vaciar'])) {
 unset($_SESSION['carrito']);
 header("Location: " . strtok($_SERVER['REQUEST_URI'], '?'));
 exit();
}

// Eliminar producto individual
if (isset($_GET['eliminar'])) {
 $id_eliminar = $_GET['eliminar'];
 $_SESSION['carrito'] = array_filter($_SESSION['carrito'],
 function($item) use ($id_eliminar) {
 return $item['id'] !== $id_eliminar;
 }
 );
 header("Location: " . strtok($_SERVER['REQUEST_URI'], '?'));
 exit();
}

$total_productos = count($_SESSION['carrito']);
?>

<!DOCTYPE html>
<html>
<head>
 <title>Mi Carrito de Compras</title>
 <style>
 .producto {
 padding: 10px;
 border: 1px solid #ddd;
 margin: 5px 0;
 display: flex;
 justify-content: space-between;
 }
 .botones a {
 padding: 10px 20px;
 margin: 5px;
 display: inline-block;
 background: #4CAF50;
 color: white;
 text-decoration: none;
 }
 </style>
</head>
<body>
 <h1>
```