<table>
<tr><td colspan="2">EXAMEN</td><td colspan="2">PRUEBA PARCIAL: PRÁCTICA</td></tr>
<tr><td colspan="4">Número de curso:IFC303LOE</td></tr>
<tr><td colspan="4">Denominación: Técnico Superior Desarrollo de Aplicaciones Web</td></tr>
<tr><td>Horas: 150</td><td>Modalidad: presencial</td><td>Fecha inicio:
17/09/2025</td><td>Fecha fin:
30/06/2026</td></tr>
<tr><td colspan="3">Centro: AFA FORMACION CONTINUA</td><td>Prueba Temas 3 y 4</td></tr>
<tr><td colspan="4">Código y denominación del módulo formativo:
0613: DESARROLLO WEB EN ENTORNO SERVIDOR</td></tr>
<tr><td colspan="3">Profesora: Miguel Méndez</td><td rowspan="2">CALIFICACIÓN</td></tr>
<tr><td colspan="3">Alumno/a:</td></tr>
<tr><td colspan="4">Tiempo de realización: 120 minutos</td></tr>
<tr><td colspan="4">Descripción de la prueba: Prueba relativa a la primera evaluación del módulo 0613, Desarrollo Web en Entorno Servidor, que abarca el tema 3 y 4 de la asignatura.</td></tr>
<tr><td colspan="4">Sistema de calificación:
• Cada problema vale 2,5 puntos.</td></tr>
<tr><td colspan="4">Instrucciones:
La prueba consiste en 4 ejercicios prácticos.
• 2 problemas de cokies
• 2 problemas de comentario de código de PDO
La prueba se superará con un 5.
Tiempo para la realización de la prueba: 120 minutos.
Cada problema se nombrará según: NOMBREALUMNO_ProblemaX_ExT3-4.php</td></tr>
<tr><td>Firma del Profesor:</td><td>Firma del Alumno/a:</td><td colspan="2">Fecha:
11/12/2025</td></tr>
</table>

1. Autenticación Persistente con Cookies (Login Básico) **(2,5 puntos)**

Implemente un sistema básico de login y logout que utilice cookies para mantener la sesión abierta. A diferencia de un login temporal, la cookie de autenticación debe ser **persistente** y recordar al usuario incluso si cierra el navegador (establezca una duración de, por ejemplo, 7 días).

Requisitos:

1. Crear un formulario POST para ingresar usuario y clave.

2. Si las credenciales son operador y seguro123, establecer una cookie persistente con el nombre de usuario.

3. La cookie debe caducar en una semana (time() + 7 * 24 * 3600).

4. Si el usuario está autenticado (existe la cookie), mostrar un mensaje de bienvenida y un enlace para cerrar sesión.

5. Implementar la lógica de cierre de sesión eliminando la cookie (estableciendo un tiempo de expiración pasado).

Requisitos Técnicos Clave: setcookie() con expires, $_POST, $_COOKIE, manejo de redirección con header() y exit().

2. Contador de Acciones Persistente **(2,5 puntos)**

Cree una página que cuente el número de veces que el usuario realiza una acción específica (simulada por una recarga de página). El contador debe ser **persistente** durante 24 horas y el valor se debe incrementar en cada recarga.

Requisitos:

Utilizar una cookie llamada contador_acciones.

Al cargar la página, comprobar si la cookie existe (isset($_COOKIE['contador_acciones'])).

Si existe, incrementar el valor en 1; si no, inicializarlo en 1.

Establecer o actualizar la cookie con el nuevo valor, garantizando que expire en 24 horas (time() + 24 * 3600).

Mostrar el valor actual del contador al usuario.

Requisitos Técnicos Clave: isset($_COOKIE), casteo de tipo ((int), (string)), setcookie() con expires.

3. Análisis de la Inserción Atómica y Gestión de Errores **(2,5 puntos)**

El siguiente código simula la inserción de un nuevo producto en dos tablas relacionadas (productos y stock_inicial), garantizando que ambas operaciones tengan éxito o que ninguna se aplique (transacción atómica).

Código a comentar:

```
<?php

// Snippet 2.3 - Insertar Producto y Stock Inicial

// Asumimos que $pdo está inicializado y conectado

$nombreProducto = $_POST['nombre'] ?? '';

$cantidadStock = (int) ($_POST['stock'] ?? 0);


try {

   // A. [Comentar esta línea]

   $pdo->beginTransaction();


   // B. [Comentar estas líneas]

   $stmtProd = $pdo->prepare('INSERT INTO productos (nombre) VALUES (:n)');

   $stmtProd->execute([':n' => $nombreProducto]);


   // C. [Comentar esta línea]

   $productoId = (int) $pdo->lastInsertId();


   // D. [Comentar estas líneas]

   $stmtStock = $pdo->prepare('INSERT INTO stock_inicial (producto_id, cantidad)
VALUES (?, ?)');

   $stmtStock->execute([$productoId, $cantidadStock]);


   // E. [Comentar esta línea]

   $pdo->commit();
```

```
} catch (PDOException $e) {
    // F. [Comentar estas líneas]
    if ($pdo->inTransaction()) {
        $pdo->rollBack();
    }
    // G. [Comentar esta línea]
    error_log($e->getMessage(), 3, __DIR__ . '/db_errors.log');
    echo 'Error en la operación. Intente de nuevo.';
}
?>
```

Etiqueta	Comentario Requerido
A.	Explicar la función beginTransaction() y su propósito en operaciones múltiples.
B.	Explicar el uso de prepare() con marcadores nombrados (:n). ¿Por qué esta es una buena práctica de seguridad?
C.	Explicar la función lastInsertId() y cuándo se utiliza.
D.	Describir el uso de marcadores anónimos (?) y cómo execute() maneja la vinculación de $productoId y $cantidadStock.
E.	¿Qué garantiza commit()?
F.	Describir el papel de rollBack() y el condicional if ($pdo->inTransaction()) dentro del bloque catch.
G.	Describir la buena práctica de manejo de errores aplicada aquí (vs. mostrar el error crudo al usuario).

A- Inicia una transacción que permite agrupar varias operaciones en una sola para que por ejemplo una operación falla se pueden cancelar todas las anteriores, esto hace que se hagan todas las operaciones o ninguna.

B- Prepara la consulta usando el marcador :n como si fuera una variable. Lo bueno es que separa la consulta de los datos reales, entonces si alguien intenta meter código SQL malicioso por el formulario, se trata solo como texto normal y no se ejecuta nada raro.

C- Devuelve el id de la ultima fila insertada.

D- Los ? son placeholders posicionales que se rellenan según el orden de valores del array. En este caso remplaza el primer ? con $productoId y el segundo con $cantidadStock

E- Confirma permanentemente todas las operaciones de la transacción de la base de datos.

F- RollBack() revierte las operaciones si ocurre un error, condicional if ($pdo->inTransaction()) confirma que la transacción este activa antes de hacer el rollback() por si crea errores intentando revertir algo cerrado.

G- Se hace para evitar mostrar información delicada de la base de datos como credenciales nombre etc, en vez de eso se muestra un numero.

4. Análisis de la Inserción Atómica y Gestión de Errores **(2,5 puntos)**

El siguiente fragmento de código simula el proceso de autenticación de un usuario, recuperando un registro de la base de datos y verificando la contraseña ingresada con el hash almacenado, utilizando las buenas prácticas de seguridad de PDO y PHP (Argon2ID).

Código a comentar:

```
<?php

// Snippet 2.4 - Simulación de Login Seguro

// Asumimos que $pdo está inicializado y conectado

$email_input = $_POST['email'] ?? '';

$password_input = $_POST['password'] ?? '';


// A. [Comentar esta línea]

$stmt = $pdo->prepare('SELECT id, passhash FROM usuarios WHERE email = :email');


// B. [Comentar esta línea]

$stmt->execute([':email' => $email_input]);


// C. [Comentar estas líneas]

$user = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$user) {

   throw new RuntimeException('Credenciales inválidas.');

}

$hash_almacenado = $user['passhash'] ?? '';
```

```
// D. [Comentar esta línea]
if (!password_verify($password_input, $hash_almacenado)) {
   throw new RuntimeException('Credenciales inválidas.');
}

// E. [Comentar esta línea]
if (password_needs_rehash($hash_almacenado, PASSWORD_ARGON2ID)) {
   // Código para actualizar el hash...
}

// Login exitoso...
?>
```

Etiqueta	Comentario Requerido
A.	Explicar el propósito de prepare() con marcadores nombrados (:email) en el contexto de la seguridad.
B.	Describir el rol de execute() y cómo garantiza que el valor de $email_input se trate de forma segura.
C.	Explicar qué hace fetch(PDO::FETCH_ASSOC) y la importancia de no continuar si $user es nulo o falso.
D.	Describir la función password_verify() y por qué este método es obligatorio para la gestión de contraseñas.
E.	Explicar el propósito de password_needs_rehash() como buena práctica de seguridad.

A- Prepara la consulta usando el marcador :email, separa la consulta de datos reales asi si alguien intenta inyectar código malicioso se tarta como texto plano.

B- Execute() rellena el valor de :email con el valor de $email_input de forma segura.

C- Fetch(PDO::FETCH_ASSOC) obtiene el registro como array. El if (¡$user) verifica que el usuario exista si no lanza una excepción.

D- Password_verify() compara la contraseña ingresada con el hash almacenado de forma segura. Es obligatorio porque nunca comparas contraseñas en texto plano, tienen que estar siempre hasheadas.

E- Password_needs_rehash() verifica si el hash necesita actualizarse al algoritmo mas seguro. Es una buena practica para tener actualizado los hashes