Actividad práctica

Sesiones y Cookies

1. Introducción

Sesiones y **cookies** son dos **mecanismos utilizados para gestionar la persistencia de datos entre múltiples solicitudes HTTP en aplicaciones web**.

HTTP es un protocolo sin estado, lo que significa que no guarda información entre una solicitud y otra. **Esto plantea un problema para las aplicaciones que necesitan mantener datos entre distintas páginas** (como información del usuario al iniciar sesión, datos de compras). Para resolver esto, se utilizan sesiones y cookies en las aplicaciones web:

2. Cookies:

- **Son pequeños archivos de texto que se almacenan en el navegador del cliente (aunque ahora se pueden guardar en una base de datos local al navegador).** Estos archivos **contienen pares clave-valor** y permiten que **la aplicación web recuerde información de un usuario entre visitas (Por ejemplo: por donde te quedaste en el episodio de tú serie)**

  Las cookies pueden persistir por un tiempo determinado o hasta que el usuario las elimine manualmente. **Son gestionadas directamente en el lado del cliente. Pero su información se manda al servidor en cada solicitud**

Estructura Típica de una Cookie

- Nombre: Identificador único de la cookie.
- Valor: Contenido o información que se almacena en la cookie.
- Dominio: El dominio para el cual la cookie es válida. Puede ser un dominio específico o un subdominio.
- Ruta (Path): La ruta dentro del dominio donde la cookie es válida. Si no se especifica, se asume que es la ruta de la página que la creó.

- Fecha de expiración (Expires): Fecha y hora en que la cookie dejará de ser válida. Si no se establece, la cookie se considerará "de sesión" y se eliminará cuando se cierre el navegador.
- Max-Age: Tiempo en segundos hasta que la cookie expire. Este valor puede ser usado en lugar de Expires.
- Secure: Indica que la cookie solo debe ser enviada a través de conexiones HTTPS.
- HttpOnly: Indica que la cookie **no puede ser accedida mediante JavaScript (ayuda a prevenir ataques XSS).**

  La opción HttpOnly en las cookies se utiliza principalmente por razones de **seguridad**.

- SameSite: Define si la cookie se enviará en solicitudes de origen cruzado (cross-site), que puede ser configurado como Strict, Lax o None.

La directiva SameSite se utiliza para controlar el comportamiento de las cookies en relación con las **solicitudes de origen cruzado (cross-site).** Esto es importante para proteger la privacidad y la seguridad del usuario al limitar cuándo se envían las cookies en situaciones en las que **se accede a un recurso en un dominio diferente.**

Las cookies con la directiva **SameSite=Strict** solo se enviarán en solicitudes del mismo sitio (same-origin). **Esto significa que la cookie no se enviará en ninguna solicitud de origen cruzado**

Esta configuración proporciona la máxima seguridad, ya que protege las cookies de ser enviadas a través de enlaces o redirecciones desde otros sitios. Sin embargo, puede afectar la funcionalidad de algunos flujos de trabajo, como el inicio de sesión a través de enlaces en correos electrónicos

La configuración **SameSite=Lax** proporciona una forma de permitir el uso de cookies en situaciones comunes (como navegar a un enlace) mientras se protege contra algunos tipos de ataques CSRF al **no permitir que las cookies se envíen en solicitudes de modificación (como POST)** desde sitios externos. Esto ayuda a mantener un equilibrio entre funcionalidad y seguridad**.**

Resumen de las directrices de seguridad

Configuración	Comportamiento	Uso
Strict	No se envía en solicitudes de origen cruzado.	Máxima seguridad, menor usabilidad.
Lax	Se envía en solicitudes GET de origen cruzado.	Equilibrio entre seguridad y usabilidad.
None	Se envía en todas las solicitudes (debe ser segura - `Secure`).	Necesario para ciertos flujos de trabajo.

Ejemplo de creación de Cookie que será enviada al cliente. Esta información será enviada por el cliente al servidor EN CADA SOLITUD. El servidor sabrá que se llama Antonio

```
// Crear una cookie llamada "usuario"
setcookie("usuario", "Antonio", [
    'expires' => time() + 86400, // Expira en 1 día
    'path' => '/', // Disponible en todo el dominio
    'domain' => 'ejemplo.com', // Dominio donde se envía la cookie
    'secure' => true, // Solo a través de HTTPS
    'httponly' => true, // No accesible a través de JavaScript
    'samesite' => 'Lax' // Envío en solicitudes de origen cruzado permitido
]);
```

a. Prevención de Ataques XSS

Cross-Site Scripting (XSS): Este es un **tipo de vulnerabilidad** en la que un atacante inyecta scripts maliciosos en páginas web que son vistas por otros usuarios. Si un sitio web es vulnerable a XSS, un atacante podría ejecutar scripts que roben cookies de sesión, incluyendo información sensible como tokens de autenticación.

Al marcar las cookies como HttpOnly, se evita que los scripts maliciosos accedan a estas cookies, dificultando la extracción de información crítica.

Reducir el Riesgo de Suplantación de Identidad

- Si un atacante puede robar una cookie de sesión, puede usurpar la identidad del usuario y acceder a su cuenta. Usar HttpOnly dificulta este proceso, protegiendo la sesión del usuario de accesos no autorizados.

3. Sesiones:

Una **sesión** es el **período de interacción entre un usuario y un sistema, en el que se establece una conexión que permite el intercambio continuo de información**. Durante una sesión, **se gestionan y mantienen los datos relacionados con las actividad del usuario dentro de ese periodo** para garantizar una experiencia coherente, como la autenticación, la navegación o la realización de tareas específicas en el sistema.

En términos más concretos:

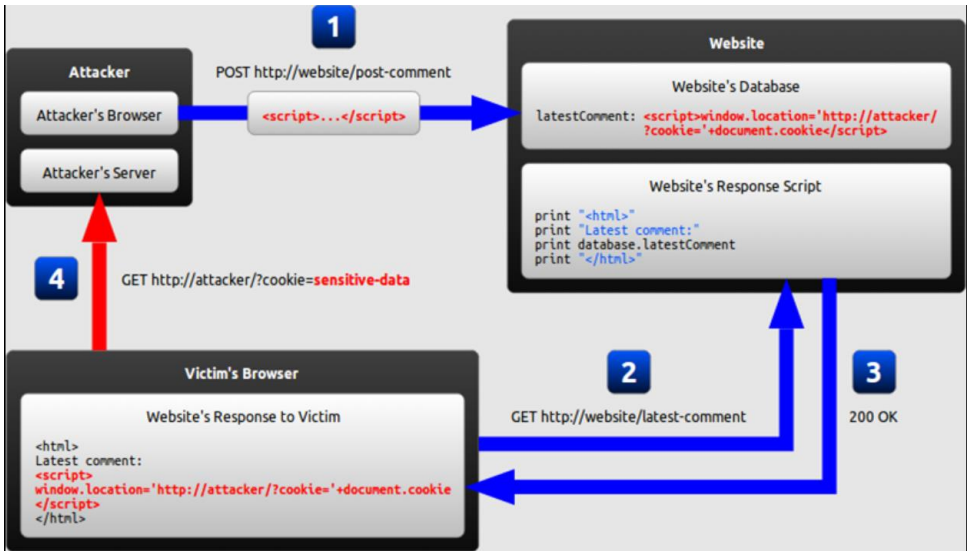
- Una sesión **comienza cuando el usuario inicia su interacción** con el sistema, por ejemplo, al iniciar sesión en una aplicación o sitio web.
- **Durante la sesión, se pueden almacenar datos temporales**, como las preferencias del usuario, información de autenticación o los pasos intermedios de un proceso.
- **La sesión finaliza cuando el usuario cierra explícitamente su sesión**, cierra el navegador, o la sesión expira debido a la inactividad.

A diferencia de las cookies, **las sesiones almacenan información en el servidor y solo un identificador de la sesión se guarda en el cliente**, generalmente a través de una cookie especial. Esto permite mantener datos seguros en el servidor sin exponerlos al cliente. Las sesiones se destruyen automáticamente cuando el usuario cierra el navegador o después de un período de inactividad.

4. Problemas de seguridad ssociados a las Cookies

4.1. Robos de Sesión (Session Hijacking):

- Un atacante **puede robar una cookie de sesión (por ejemplo, a través de un ataque XSS)** y **usarla para suplantar la identidad del usuario**. Esto le permite acceder a la cuenta de la víctima sin necesidad de conocer sus credenciales.
- Ejemplo de XSS Directo (persistente)



Un ataque de **Cross-Site Scripting (XSS) es una vulnerabilidad de seguridad que permite a un atacante inyectar scripts maliciosos** en las páginas web vistas por otros usuarios. Esto ocurre cuando una aplicación web incluye datos proporcionados por el usuario (por ejemplo, a través de un formulario**) sin sanitizarlos adecuadamente.**

Ejemplo de código

Index.php

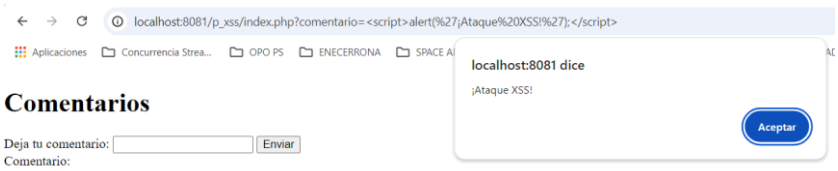
```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Página de Comentarios</title>
</head>
<body>
    <h1>Comentarios</h1>
    <form action="comentarios.php" method="GET">
        <label for="comentario">Deja tu
comentario:</label>
        <input type="text" name="comentario"
id="comentario">
        <button type="submit">Enviar</button>
    </form>
```

```
    <div>
        <?php
        if (isset($_GET['comentario'])) {
            echo "Comentario: " . $_GET['comentario'];
        }
        ?>
    </div>
</body>
</html>
```

Código malicioso inyectado en el formulario

```
Localhost:8180/index.php?comentario=<script>alert('¡Ataque XSS!');</script>
```

Cuando un usuario hace clic en el enlace, el script se inyecta en la página y se ejecuta en su navegador:



¿Qué pasa si en lugar de este mensaje inocente, se ejecuta un javascript que me roba la info de cookie y la envia al atacante?

4.1.1. Consecuencias de un Ataque XSS

- Robos de sesión: El atacante **puede robar cookies de sesión** y suplantar la identidad del usuario.
- Desviación de usuarios: **Puede redirigir a los usuarios a sitios maliciosos o de phishing.**
- Propagación de malware: **Puede inyectar código malicioso** que infecte a los visitantes del sitio.

4.1.2. Prevención de Ataques XSS

Para protegerse contra ataques XSS, se deben seguir algunas prácticas recomendadas:

1. **Sanitización y validación de entradas**: Asegúrate de que todos los datos ingresados por los usuarios sean sanitizados(limpiados) y validados adecuadamente.

```
$comentario = htmlspecialchars($_GET['comentario'], ENT_QUOTES, 'UTF-8');

//Esta función convierte caracteres especiales en sus entidades HTML, lo que evita que el navegador los interprete como código HTML.


$email = filter_var($_POST['email'],FILTER_SANITIZE_EMAIL);

//Esta función evita que se introduzca algo diferente a un email


$comentario = preg_replace('/[^A-Za-z0-9\s]/', '', $_GET['comentario']);

// Sólo admite caracteres convencionales en la entrada No se podría escribir <script> por ejemplo
```

Ejemplo de uso

```
<?php

// Supongamos que esta es una entrada del usuario

$comentario = "<script>alert('¡Ataque XSS!');</script>";


// Escapar la salida

$comentario_escapado = htmlspecialchars($comentario, ENT_QUOTES, 'UTF-8');


// Mostrar el comentario escapado en la página

echo "Comentario: " . $comentario_escapado;
```

```
?>
```

El navegador mostrara la salida como texto , no como código java script

```
Comentario: &lt;script&gt;alert('¡Ataque XSS!');&lt;/script&gt;
```

2. **Uso de HttpOnly y Secure en cookies**: Esto ayuda a proteger las cookies de acceso mediante JavaScript.
3. **Uso de Content Security Policy (CSP):** Implementa una política de seguridad de contenido que limite las fuentes de scripts y recursos.
4. **Escapar salidas**: Asegúrate de escapar los datos al mostrarlos en la página, para que no se interpreten como código HTML o JavaScript.

4.1.3. Buenas Prácticas

- **Siempre escapa los datos de salida**: Escapa todos los datos que se muestran en la página, especialmente aquellos que provienen de entradas de usuario.
- **No confíes en la sanitización sola**: Sanitizar los datos es importante, pero siempre debes escapar las salidas al mostrarlas para asegurarte de que no se ejecute código malicioso.
- **Utiliza configuraciones de charset adecuadas**: Asegúrate de utilizar el mismo charset en tu aplicación y en las funciones de PHP para evitar problemas de codificación.

4.2. Cross-Site Scripting (XSS):

- Si un sitio web es vulnerable a ataques XSS, un atacante puede inyectar scripts maliciosos que pueden acceder a las cookies y enviarlas a un servidor controlado por el atacante.

4.3. Cross-Site Request Forgery (CSRF):

- Los ataques CSRF **permiten que un atacante envíe solicitudes no autorizadas a un sitio web en nombre de un usuario autenticado**, utilizando las cookies de sesión. Esto puede llevar a acciones no deseadas (como cambios de configuración o envíos de formularios) sin el conocimiento del usuario.

Cómo Funciona un Ataque CSRF

- **Autenticación del Usuario**: El usuario inicia sesión en un sitio web (por ejemplo, un banco o una red social) y **su sesión se mantiene activa mediante cookies.**
- **Visita a un Sitio Malicioso**: Mientras el usuario está autenticado en el sitio legítimo, **visita un sitio web malicioso** o hace clic en un enlace malicioso. NOTA ESTO ES CLAVE AL ACCEDER A UN SITIO MALICIOSO CON EL MISMO ASPECTO EL USUARIO NO SE DA CUENTA
- Cuando el usuario visita un sitio malicioso**, el navegador sigue enviando automáticamente todas las cookies asociadas con el dominio de BancoEjemplo.com**, incluyendo la cookie de sesión.Siempre que el destino de la solicitud (en este caso, BancoEjemplo.com) sea uno que el navegador ya conoce (es decir, al que el usuario ha accedido previamente y tiene cookies válidas).
- **Solicitud No Autorizada**: **El sitio malicioso envía una solicitud a la aplicación web legítima utilizando la sesión activa del usuario**. **Dado que la solicitud incluye las cookies del usuario (que se envían automáticamente por el navegador),** la aplicación web la acepta como válida.

Prevención de CSRF

Para prevenir ataques CSRF, se pueden implementar varias estrategias:

- **Tokens CSRF**: Incluir un token único en cada formulario que se verifica en el servidor. Este token debe ser único para la sesión del usuario y se verifica al procesar la solicitud.

```
// Generar un token al iniciar la sesión
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
<input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
```

- **Verificación de Referer**: Comprobar **el encabezado Referer de las solicitudes** para asegurarse de que **provienen de la misma aplicación**.
- **Métodos HTTP Seguros**: Usar métodos HTTP seguros (como PUT o DELETE) para las operaciones críticas que deberían requerir autenticación adicional.

4.4. Acceso Inadecuado a Cookies (Cookie Leakage):

1.1. Si las cookies no están configuradas correctamente, pueden ser accesibles desde orígenes no seguros (HTTP), lo que aumenta el riesgo de que un atacante las intercepte.

4.5. Pérdida de Privacidad:

- Las cookies pueden usarse para rastrear a los usuarios a través de diferentes sesiones y sitios web, lo que plantea preocupaciones sobre la privacidad.

4.6. Conclusión

Las cookies de sesión son un aspecto crucial para la autenticación en la web, y su manejo inadecuado puede abrir la puerta a ataques como CSRF. Comprender cómo se envían las cookies, incluso en dominios diferentes, es esencial para desarrollar aplicaciones web seguras. Implementar medidas de protección como tokens CSRF y atributos SameSite puede ayudar a mitigar estos riesgos.

5. Cookies en PHP (variable superglobal $_COOKIE)

En PHP, las cookies **se gestionan utilizando la variable superglobal $_COOKIE**. PHP permite crear, leer y eliminar cookies con las funciones **setcookie() y $_COOKIE**.

Las cookies son enviadas al servidor en cada solicitud, lo que permite recordar preferencias de usuario, autenticación básica, entre otras cosas.

Aquí tienes las operaciones más comunes con cookies : crear, leerla y borrarla

```
// Crear una cookie
setcookie("usuario", "Juan", time() + 3600); // Expira en una hora
```

```
// Leer una cookie
if (isset($_COOKIE['usuario'])) {
    echo "Bienvenido" . $_COOKIE['usuario'];
}
```

```
// Eliminar una cookie
setcookie("usuario", "", time() - 3600); // Tiempo en el pasado para eliminarla
```

6. Sesiones en PHP (variable superglobal $_SESSION)

PHP proporciona una forma sencilla de trabajar con sesiones a través de la superglobal **$_SESSION**.

Las sesiones permiten almacenar información como un array en el servidor, y el navegador solo maneja el identificador de la sesión. **Para iniciar y usar una sesión, es necesario usar session_start() al inicio de cada script que manipule sesiones.**

Iniciar una sesión y establecer un valor

```
// Iniciar sesión
```

```
session_start();


// Guardar información en la sesión
$_SESSION['usuario'] = "Juan";
```

Leer información de la sesión

```
// Leer información de la sesión
echo "Usuario: " . $_SESSION['usuario'];
```

Eliminar los datos de una variable de sesión y destuirla

```
// Eliminar una variable de la sesión
unset($_SESSION['usuario']);


// Destruir la sesión
session_destroy();
```

Configuración de la duración de la sesión

PHP, por defecto, destruye las sesiones una vez que el navegador se cierra o después de un período de inactividad. Para gestionar sesiones más largas, se puede ajustar la duración del tiempo que la sesión permanece activa en el servidor. Esto se hace configurando los siguientes parámetros en el archivo **php.ini** o mediante las funciones ini_set() en el código:

- **session.gc_maxlifetime**: Este **parámetro define el tiempo máximo en segundos que una sesión** puede permanecer activa antes de ser considerada "basura" y eliminada por el proceso de "garbage collection" (recolección de basura). Si necesitas sesiones largas, puedes aumentar este valor.
- **session.cookie_lifetime**: Este valor define la duración de la cookie que guarda el identificador de la sesión en el cliente. Si se configura a 0, la cookie expira cuando se cierra el navegador. Para sesiones largas, se puede establecer un valor mayor (en segundos).
-

Almacenamiento en el servidor

Las sesiones en PHP se almacenan en archivos por defecto, en una carpeta temporal del servidor (especificada por session.save_path). PHP guarda los datos

de la sesión en el servidor, mientras que el navegador **del cliente solo maneja una cookie con el ID de la sesión.**

- **El servidor revisa el archivo de sesión cada vez que recibe una solicitud del cliente** y lo actualiza con los nuevos datos o lo elimina si ha expirado.
- Para sesiones largas, es importante aumentar los tiempos de vida de las sesiones, pero también tener en cuenta la capacidad del servidor para manejar archivos de sesión grandes o numerosos.
-

Control de recolección de basura (Garbage Collection)

PHP tiene un mecanismo de recolección de basura que **elimina las sesiones inactivas para liberar espacio en el servidor.** Este proceso ocurre de manera periódica, según tres configuraciones importantes:

- **session.gc_probability** y **session.gc_divisor**: Controlan la probabilidad de que la recolección de basura se ejecute en cada solicitud. Se calcula como gc_probability/gc_divisor. Para sesiones largas, puedes reducir la frecuencia con la que se ejecuta la recolección de basura si no deseas que las sesiones se eliminen prematuramente.
- **session.gc_maxlifetime:** Define cuánto tiempo después de la última actividad se eliminará la sesión. Ajustar este valor, como se mencionó antes, es clave para sesiones largas.

Sesiones y cookies persistentes

Otra estrategia para gestionar sesiones largas es combinar sesiones con cookies persistentes. Si una sesión expira, pero **se mantiene una cookie con un token de usuario (por ejemplo, un identificador o un hash)**, se puede restaurar la sesión o crear una nueva automáticamente.

7. Diferencias Fundamentales entre Sesiones y Cookies

1. **Almacenamiento**:
    - **Cookies**: Se almacenan en el navegador del usuario como archivos de texto (o en una bd local). Cada vez que el usuario realiza una solicitud a un servidor, **se envían todas las cookies asociadas a ese dominio.**
    - **Sesiones**: Los datos de la sesión se almacenan en el servidor. El cliente recibe solo un identificador de sesión (normalmente

almacenado en una cookie o en la URL) que se utiliza para acceder a la información almacenada en el servidor.

2. **Capacidad de Almacenamiento**:
    - **Cookies**: Tienen un tamaño limitado (generalmente alrededor de 4 KB) y pueden almacenar solo pares clave-valor.
    - **Sesiones**: **Pueden almacenar grandes cantidades de datos ya que se almacenan en el servidor**, lo que permite manejar información más compleja sobre el usuario.
3. **Persistencia**:
    - **Cookies**: Pueden tener una fecha de caducidad establecida, lo que permite que persistan incluso después de que el usuario cierra el navegador.
    - **Sesiones**: Típicamente, **se eliminan al finalizar la sesión** (por ejemplo, cuando se cierra el navegador o después de un período de inactividad).
4. **Seguridad**:
    - **Cookies**: **Son más vulnerables a** ataques como el secuestro de cookies (session hijacking) y Cross-Site Scripting (XSS) si no se configuran adecuadamente (por ejemplo, no usando HttpOnly y Secure).
    - **Sesiones**: Son generalmente más seguras porque los datos sensibles se mantienen en el servidor y solo se maneja un identificador en el cliente.

4. Desarrollo práctico . Uso de Cookies

Vamos a desarrollar una pequeña práctica donde pongamos en práctica lo aprendido.

Implementa los siguientes componentes de la aplicación web

index.html

Este componente tendrá un formulario de envío del nombre de usuario y un enlace para hacer solicitudes al servidor sin enviar nada fuera del formulario

```
<form action=”procesar_cookie.php”  method="post">
 Introduce tu nombre: <input type="text" name="nombre">
 <input type="submit" value="Enviar">
 </form>
<a href=”./procesar_cookie.php”> Acceso al servidor sin enviar nombre. </a>
<h1> Cómo le estoy enviando mi nombre en cada solicitud el lo puede recuperar</h1>
```

procesar_cookie.php.

En el servidor se atenderá la solicitud post del formulario anterior por este módulo el siguiente módulo**. Fíjate que el código es REENTRANTE!**

```
<?php
// Comprobar si el formulario ha sido enviado
if (isset($_POST['nombre'])) {
    // Crear una cookie con el nombre del usuario, que expira en 1 hora
    setcookie("nombre_usuario", $_POST['nombre'], time() + 3600);
} else{
if (isset($_COOKIE['nombre_usuario'])) {
    // Mostrar el nombre si la cookie existe
```

```
    echo "Bienvenido de nuevo, " . $_COOKIE['nombre_usuario'] .
"!";
}
}
```
?>

Cuestiones:

- Mira en tú navegador si efectivamente se guarda una variable denominada nombre_usuario en la información de cookies

5. Desarrollo práctico uso de sesión

Este es un ejemplo para implementar una especie de carrito de compras de **libros de informática**:

Los pasos de la práctica son los siguientes:

1. Crear un listado de libros de informática que el usuario puede agregar al carrito.
2. Al agregar un libro, se guardará en la sesión.
3. Mostrar el contenido del carrito una vez agregado , mediante un enlace especial

Los componentes de tú aplicación web son los siguientes

`index.html`

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tienda de Libros</title>
</head>
<body>
    <h1>Tienda de Libros de Informática</h1>
    <ul>
        <li>
            Libro 1: "Introducción a PHP"
            <a href="comprar.php?libro=Introducción+a+PHP">Comprar</a>
        </li>
        <li>
            Libro 2: "Programación en Java"
```

```
            <a
href="comprar.php?libro=Programación+en+Java">Comprar</a>
        </li>
        <li>
            Libro 3: "Algoritmos y Estructuras de Datos"
            <a
href="comprar.php?libro=Algoritmos+y+Estructuras+de+Datos">Compr
ar</a>
        </li>
    </ul>

    <p><a href="ver_compra.php">Ver Carrito</a></p>
</body>
</html>
```

Módulo comprar.php

```
<?php
session_start(); // Iniciar la sesión

// Verificar si se ha seleccionado un libro
if (isset($_GET['libro'])) {
    $libro = ¿…?// Sanitizar la entrada

    // Agregar el libro al carrito (array de sesión)
    if (!isset($_SESSION['carrito'])) {
        $_SESSION['carrito'] = ¿…?// Inicializar carrito si no
existe
    }

    // Añadir el libro al carrito
    $_SESSION['carrito'][] = ¿…?;
```

```
    // Mostrar mensaje de éxito
    echo "<h1>Libro Comprado!</h1>";
    echo "<p>Has comprado: <strong>$libro</strong></p>";
    echo '<p><a href="index.html">Seguir comprando</a></p>';
} else {
    echo "<h1>Error</h1>";
    echo "<p>No se ha seleccionado ningún libro.</p>";
    echo '<p><a href="index.html">Volver a la tienda</a></p>';
}
?>
```

Módulo ver_compra.php

```
<?php
session_start(); // Iniciar la sesión

// Verificar si el carrito existe
if (isset($_SESSION['carrito']) && !empty($_SESSION['carrito']))
{
    echo "<h1>Contenido del Carrito</h1>";
    echo "<ul>";

    // Mostrar cada libro en el carrito
    Foreach(¿…?) {
        echo "<li>" . htmlspecialchars($libro) . "</li>";
    }

    echo "</ul>";
    echo '<p><a href="index.html">Volver a la tienda</a></p>';
} else {
```

```
    // Mensaje si el carrito está vacío
    echo "<h1>El carrito está vacío</h1>";
    echo '<p><a href="index.html">Volver a la tienda</a></p>';
}
?>
```

Cuestiones:

- Prueba la aplicación y compra varios libros. Después accede a la opción de ver la compra.¿ Qué productos se te muestran?. ¿ Cómo recuerda el servidor lo que has comprado?
- ¿**Qué información se almacena en la sesión** y cómo se utiliza para personalizar la experiencia del usuario?
- ¿Cómo se asegura que los libros comprados por un usuario permanezcan en el carrito **incluso si navega a diferentes páginas de la aplicación**? ¿Qué sucedería si no se utilizaran sesiones?

6. Anexo

Aquí se muestra las cookies que guardaría una web típica de comercio electrónico

Cookies de Sesión

- **Nombre**: PHPSESSID (o un nombre similar)
- **Propósito**: Almacenar el identificador de sesión del usuario para mantener la sesión activa mientras navega por el sitio. Estas cookies se eliminan cuando se cierra el navegador.

2. Cookies de Autenticación

- **Nombre**: auth_token o session_token
- **Propósito**: Permitir a los usuarios **permanecer autenticados entre sesiones**. Se utiliza para recordar la sesión del usuario después de que cierra y vuelve a abrir el navegador.

3. Cookies de Carrito de Compras

- **Nombre**: cart o cart_id
- **Propósito**: Almacenar el contenido del carrito de compras del usuario, lo que permite recordar los productos que ha añadido mientras navega por el sitio.

4. Cookies de Preferencias del Usuario

- **Nombre**: language, currency
- **Propósito**: Recordar las preferencias del usuario, como el idioma o la moneda seleccionada, para personalizar la experiencia de compra.

5. Cookies de Seguimiento y Análisis

- **Nombre**: ga (Google Analytics), fbp (Facebook Pixel)
- **Propósito**: Recopilar datos sobre el comportamiento del usuario en el sitio web para análisis de tráfico y marketing. Estas cookies ayudan a realizar un seguimiento de las conversiones y a optimizar campañas publicitarias.

6. Cookies de Marketing y Publicidad

- **Nombre**: advertising_id, utm_source
- **Propósito**: Almacenar información relacionada con las campañas de marketing y publicidad para medir la

efectividad de los anuncios y personalizar las ofertas según el comportamiento del usuario.

7. **Cookies de Seguridad**
   - **Nombre**: csrf_token
   - **Propósito**: Almacenar tokens de protección contra ataques CSRF (Cross-Site Request Forgery) para asegurar que las solicitudes que se realizan desde el navegador son legítimas.
8. **Cookies de Opt-in/Opt-out**
   - **Nombre**: cookie_consent
   - **Propósito**: Almacenar la decisión del usuario sobre el consentimiento de cookies, permitiendo que el sitio web sepa si puede utilizar cookies adicionales según las preferencias del usuario.