MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO SERVIDOR
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Desarrollo de aplicaciones web utilizando código

05

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Mantenimiento del estado de aplicaciones 4

/ 3. Control de sesiones 4

3.1. Iniciar sesión 5

3.2. Cerrar sesión 6

3.3. Tiempo de acceso 7

/ 4. Cookies 8

4.1. Creación de ***cookies*** 9

/ 5. Seguridad. Perfiles, roles y usuarios 10

/ 6. Diseño del formulario y conexión 11

/ 7. Autenticación con base de datos 12

/ 8. OpenID 13

/ 9. Caso práctico 1: "Primer formulario de autenticación" 15

/ 10. Caso práctico 2: "Contador de sesiones" 16

/ 11. OAUTH 17

/ 12. Resumen y resolución del caso práctico de la unidad 18

/ 13. Bibliografía 18

OBJETIVOS

Identificar los mecanismos disponibles para el mantenimiento de la información que concierne a un cliente web concreto.

Señalar las ventajas del mantenimiento de sesiones.

Utilizar sesiones para mantener el estado de las aplicaciones web.

Utilizar cookies para almacenar información en el cliente y para recuperar su contenido.

Identificar y caracterizar los mecanismos disponibles para la autenticación de los usuarios.

Escribir aplicaciones que integran mecanizamos de autenticación de usuarios.

/ 1. Introducción y contextualización práctica

El desarrollo de aplicaciones y servicios web incorpora diferentes mecanismos que permiten mantener el estado de una sesión o muestran una u otra interfaz en función del usuario y del tipo de acceso al sitio.

A través de la autenticación de los usuarios, estos podrán mantener el estado de la sesión de una conexión a otra. Por ejemplo, si se accede a un sitio web creado para una tienda online, se deseará mantener los datos de envío, la dirección de facturación o los productos añadidos a una cesta pase el tiempo que pase, esto es, que se mantengan ciertos datos de sesión a sesión. Las *cookies* y el control de sesiones nos permitirán realizar estas acciones.

Fig. 1. Control de sesiones

Tanto las *cookies* como el control de sesiones almacenan información para garantizar el estado de la sesión. La diferencia radica en que, mientras que en el primer caso esta información queda almacenada en el cliente, en el segundo caso se lleva a cabo en el servidor.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema. Encontrarás su resolución en el apartado «Resumen y resolución del caso práctico de la unidad».

Audio Intro. "KAutorización y Autenticación".
https://bit.ly/3aj1EXe

/ 2. Mantenimiento del estado de aplicaciones

Los mecanismos que permiten controlar el estado de aplicaciones resultan claves para preservar los datos de un usuario entre sesiones. Sin duda, esta es una de las principales ventajas que ofrecen este tipo de mecanismos.

Un usuario podrá saltar de una página web a otra, cerrar la sesión, volver a abrirla a las dos horas o a las dos semanas, y siempre se mantendrán sus datos de sesión, si se han implementado correctamente los mecanismos de mantenimiento del estado de sesiones, encargados de guardar datos de los usuarios.

Los mecanismos más presentes en la actualidad son el uso de *cookies* y los elementos de control implementados a través de las sesiones:

- **Cookies:** Se modela el envío de información a través de un conjunto de parámetros (nombre *cookie*, tiempo de inactividad...) y son almacenados en el cliente. Cuando este vuelve a solicitar el mismo recurso al servidor para volver a acceder a él, podrá utilizar los datos almacenados gracias a la *cookie*. Estas permanecen en el equipo del cliente.

- **Sesiones:** A diferencia del caso anterior, ahora será en el servidor donde se almacena toda la información relativa a un usuario, la cual queda asociada a su *login* de acceso. Las sesiones se eliminan cuando el usuario cierra el navegador.

La autenticación de usuarios es clave para el desarrollo de aplicaciones web dinámicas que deben mantener una configuración específica para cada cliente. La autenticación permite confirmar que el usuario que está accedido a un sitio web es quien dice ser.

Para confirmar que es posible el uso de sesiones en nuestro entorno de desarrollo, en PHP la configuración de los mecanismos de control se realiza a través del fichero **php.ini**, en concreto, en el contenido de la variable **session.save_path.**

Fig. 2. Proporción áurea y relaciones numéricas.

/ 3. Control de sesiones

El control del estado de las aplicaciones utilizando sesiones se basa en la creación o apertura de estas sesiones, así como en el almacenamiento de datos de usuario en el servidor, que podrán ser recuperados en posteriores accesos.

En PHP, para el control de las sesiones, existen varias funciones importantes:

Función	Descripción
session_start()	Permite crear una nueva sesión o abrir una ya creada previamente en una sesión anterior
session_destroy()	Borra la información de la que ya no se desee almacenar información
session_register()	Permite crear una nueva sesión o abrir una ya creada previamente en una sesión anterior. Además, permite el registro de más de una variable al mismo tiempo
session_unregister()	Borra todos los registros que ya no se desean almacenar

Tabla 1. Funciones control de sesiones.

Para implementar el control de sesiones, se utilizan las funciones descritas anteriormente, la primera, nada más abrir el fichero PHP contenido en el servidor, a continuación, será necesario acceder a cada una de las variables de sesión. Este acceso se realiza utilizando la siguiente sintaxis:

```
$_SESSION[“variable”];
```

Código 1. Sintaxis extracción información de sesión.

Este elemento es un *array* que almacena toda la información de un usuario durante su acceso al sitio web. Podríamos decir que se trata de un repositorio de información a la que el usuario podrá acceder siempre que se mantenga la sesión abierta.

Fig. 3. Procedimiento control de sesiones.

3.1. Iniciar sesión

En primer lugar, se debe iniciar la sesión o se debe crear (si aún no existe). Para llevar a cabo este primer paso, se utiliza la función **session_start()**. Una vez se ha establecido esta conexión entre el cliente y el servidor, ya serán accesibles todos los datos almacenados en el *array* $_SESSION. Veamos un ejemplo.

1. En primer lugar, se crea una primera sesión en la que se almacena el nombre de un nuevo usuario. Para ello, se utiliza una clave que se utilizará para volver a acceder a la posición exacta del *array* en la que se encuentra contenida la información buscada.

```
<?php
    session_start(); // Se crea la sesión
    echo “Sesión iniciada <br>”;
  // Se guardan datos de sesión
    $_SESSION[“id”] = “Usuario1”;
    echo “Se ha creado una sesión con el
  nombre usuario: “.$_SESSION[“id”].”<br>”;
?>
<!-- Se accede nuevo a la sesión -->
<a href=”accesso2.php”>Pincha aquí para acceder de nuevo a la sesión</a>
```

Código 2. Creación de sesión.

2. Ahora bien, en el segundo archivo volvemos a iniciar la sesión y comprobamos si el nombre de usuario ha quedado guardado en el *array* $_SESSION creado en el paso anterior.

```
<?php
session_start(); // Se inicia la sesión
// Se accede a los datos de sesión
    echo "Bienvenido de nuevo ".$_SESSION["id"];
?>
```

Código 3. Acceso a la sesión previamente creada en Código 2.

3. Como podemos comprobar, todo ha funcionado correctamente. Gracias a la apertura de una misma sesión, se almacenan, en el *array* de la sesión, datos que son accesibles (conocida la clave).

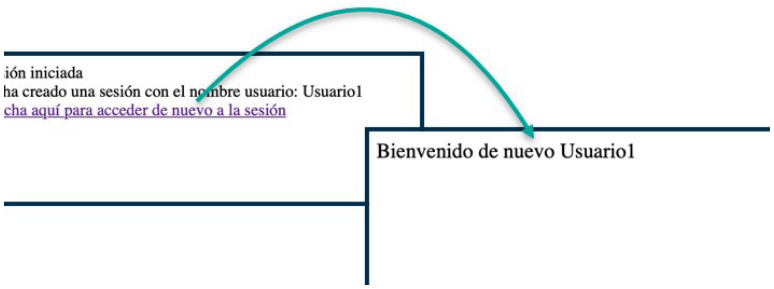

Fig. 4. Resultado acceso sesiones código 2 y 3.

3.2. Cerrar sesión

Para realizar el cierre de una sesión, se utiliza habitualmente la función **session_destroy()**, que se coloca al final de *script*.

```
<?php
session_start(); // Se inicia la sesión
// Se accede a los datos de sesión
    echo "Bienvenido de nuevo ".$_SESSION["id"];
session_destroy();
?>
```

Código 4. Cierre de sesión.

Ahora bien, existe la posibilidad de eliminar los datos de la sesión, pero no eliminarla; para ello, se utiliza la función *unset*:

- Para eliminar un solo dato, se utiliza la función *unset*, que recibe como parámetro el *array* de sesiones con referencia a la posición exacta que se va a eliminar; para ello, se utiliza su clave identificadora:

Dentro del contexto del marketing, en ocasiones, la comunicación posee un papel secundario. A menudo se trata únicamente como una función de relaciones públicas o solo entra en juego a nivel táctico. Sin embargo, para que las campañas de marketing sean exitosas, debe haber una estrategia de comunicación efectiva.

```
unset($_SESSION[“nombreClave”]);
```

Código 5. Eliminar un dato de sesión.

- Para eliminar todos los datos:

```
unset();
```

Código 6. Eliminar todos los datos de sesión.

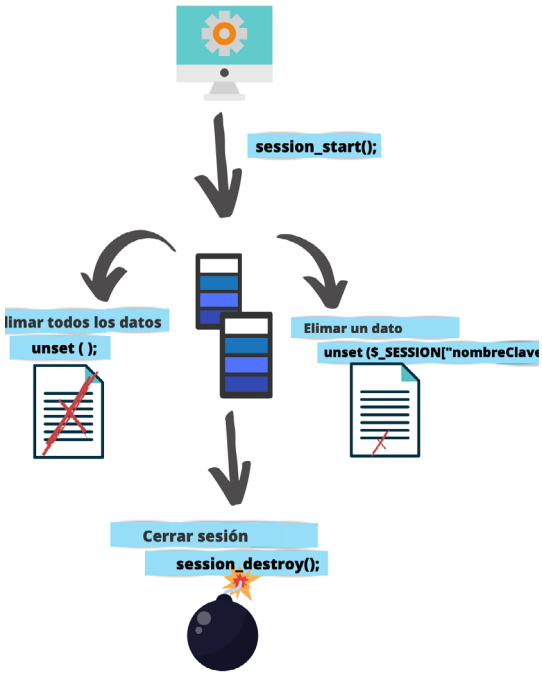

Fig. 5. Diagrama completo funcionamiento de sesión.

Video 1. “Creación y cierre de una sesión”
https://bit.ly/3gTFKwd

3.3. Tiempo de acceso

La configuración de las sesiones permite establecer una serie de parámetros que garantizan la seguridad de las mismas. Uno de los más frecuentes es el establecimiento de un tiempo máximo de conexión a la sesión, lo cual evita que, si el usuario ha olvidado salir de ella, esta pueda ser utilizada por personas ajenas.

Para configurar este tiempo de acceso, se deben tener en cuenta los siguientes datos:

- **$_SESSION[“timeout”]:** Si la conexión establecida devuelve un valor.
- **$sessionTTL:** Se realiza el cálculo necesario para conocer el tiempo exacto que la sesión lleva activa.
- **header():** Función que permite en PHP crear las cabeceras necesarias para HTTP. Recibe por parámetro en este caso, el PHP del servidor al que se reenvía el usuario si trata de acceder a la sesión, después del tiempo de inactividad establecido.

En el siguiente ejemplo, se establece un tiempo máximo de inactividad de 10 segundos. Si, pasado este tiempo, el usuario trata de volver a entrar, se le reenvía a otro PHP para indicarle que la sesión ha sido cerrada. Para volver a acceder, debería volver a iniciar la sesión. Esto se hace habitualmente desde el inicio que le autentifica en el sitio o aplicación.

```
<?php
    session_start();
  //Se indica el tiempo de actividad (seg)
  $tiempoInactivo = 10;
 // Si existe un valor para la clave timeout, la sesión ha sido establecida y se procede con el cálculo restante
 if(isset($_SESSION["timeout"])){
 //Se calcula el tiempo que ha transcurrido desde que se conectó
   $sessionTTL = time()-$_SESSION["timeout"];
  //Si el tiempo de inactividad supera al establecido se cierra la sesión y se lanza un fichero PHP con un aviso
      if($sessionTTL > $tiempoInactivo){
        session_destroy();
        header("Location: sesionCerrada.php");
      }
   }
 //Se almacena la hora exacta del inicio o creación de sesión
   $_SESSION["timeout"] = time();
?>
```

Código 7. Control de tiempo de sesión.

/ 4. Cookies

Las *cookies* consisten en un archivo de texto que se almacena en el equipo y que contiene algunos datos relevantes, como el número SID (Identificador de Seguridad, en inglés, *Security IDentifier*) y otros de carácter opcional. El servidor, gracias al SID, es capaz de identificar a un cliente, mostrando, así, todos los datos que tiene almacenados.

La transmisión de las *cookies* se realiza, también, utilizando el protocolo HTTP. A través del navegador, se envían, indicando un conjunto de datos:

- **Set-Cookie:** Indica la creación de una nueva cookie.
- **Name:** Se establece el nombre de la nueva cookie, es muy importante para que queden identificadas.
- **expires:** Se indica la fecha y la hora máxima de expiración, así se controla el tiempo de acceso.
- **domain:** Se indica el dominio dónde la cookie es válida.
- **path:** Ruta exacta dentro del domino a la que se envía la cookie.

```
Set-Cookie: Name=content data; expires=Sun, 25-Apr-2020 23:59:59
GMT; path=/; domain=.destino.com
```

Código 8. Ejemplo datos de cookie.

El uso de *cookies* presenta múltiples ventajas, como la rapidez de acceso o la sencillez de implementación. Además, resultan extremadamente útiles para el desarrollo de sitios web que incorporan opciones de compras online.

Pero también implica ciertas desventajas, como el abuso de estas para para la creación de publicidad.

Fig. 6. Ventajas y desventajas uso cookies.

4.1. Creación de *cookies*

En primer lugar, para la creación de cookies, se desarrolla una función propia en PHP, la cual presenta la siguiente sintaxis. Los parámetros que se expresan entre corchetes son opcionales.

```
setcookie ( string $name [, string $value [, int $expire = 0 [, string
$path [, string $domain [, bool $secure = false [, bool $httponly = false ]]]]]]
) : bool
```

Código 9. Sintaxis completa función setcookie().

En función de los elementos clave que presenta una *cookie*, que acabamos de describir, para la creación de *cookies,* se utiliza la función **setcookie()**, que recibe por parámetro varios elementos (no todos son obligatorios), se deben indicar en el siguiente orden:

- **nombre:** Parámetro obligatorio que indica el nuevo nombre de la *cookie*.
- **value:** Indica el valor de la *cookie*.
- **time:** Indica el tiempo de vida máximo de la *cookie*
- **domain:** Indica el dominio de la *cookie*.
- **path:** Ruta exacta dentro del domino.
- **envioSinConexionSegura:** Parámetro de tipo booleano que permite que una cookie sea enviada (*true*) o no (*false*), si no ha establecido una conexión segura con HTTPs.
- **envioProtocolo:** Este parámetro, también de tipo booleano, permite la disponibilidad de la *cookie* solo a través de HTTP (*true*) o no (*false*).

Al igual que en el caso de las sesiones, se crea un *array* llamado **$_COOKIE** donde quedan almacenados todos los valores asignados a una cookie, que serán recuperados atendiendo al nombre de la clave con la que han sido guardados en su creación.

Como se puede ver en el siguiente ejemplo, para la creación de una *cookie* se utiliza la función **setCookie()** y, entre paréntesis, se reciben el resto de parámetros de la función.

```
<?php
setcookie('cookie','valorCookie',time()+3600*24*31);
?>
```

Código 10. Creación de cookie.

No existe una función específica para el borrado de una *cookie*. Si se desea, se puede crear una nueva con el nombre de otra ya existente que puede ser borrada, utilizando **setcookie ("mismoNombre")** y se sobrescribiría la primera.

Video 2. "Creación de una cookie".
https://bit.ly/2PSgScd

/ 5. Seguridad. Perfiles, roles y usuarios

Para modelar el acceso a un sitio o aplicación web, es necesario establecer mecanismos de seguridad que no permitan el acceso a «cualquiera» ni que tampoco permita a cualquier usuario «acceder a todo».

Habitualmente, el usuario accede a la aplicación tras completar unos datos de acceso. Previamente, se debe haber registrado, o el administrador del sitio le debe haber creado una cuenta de acceso. Se comprueba si estos datos son correctos; si lo son, se le dará acceso.

Los **perfiles** definen el tipo de acceso de usuarios. Suele ser habitual que existan diferentes tipos de perfiles. Por ejemplo, tenemos los de tipo administrador, que se encargan de la gestión y organización de una aplicación. Por otro lado, encontramos otros **perfiles de usuario**, los cuales se suelen otorgar a los diferentes clientes que va a acceder al sitio. Además, en función del tipo de usuario, también se podrán establecer unos permisos de acceso u otros.

Los roles permiten establecer un conjunto de permisos de acceso a la aplicación para cada usuario; de esta forma, tras superar con éxito el proceso de autenticación, se recuperan los roles asociados a cada uno de los usuarios del sitio.

Para completar el proceso de reconocimiento de roles, habitualmente se construye un formulario destinado para tal fin, que incluye varios campos de datos, entre ellos, el identificador de usuario. Estos datos se recogen, se envían y, posteriormente, son recuperados a través de las conocidas **$_GET o $_POST**.

A continuación, se comprueba si el usuario existe. En caso afirmativo, se crea o abre una sesión y se almacena en el array de datos de sesión **$_SESSION** el identificador de usuario. Finalmente, se redirecciona al usuario al fichero PHP que modela su tipo de acceso a la aplicación, en función del rol.

Fig. 7. Modelado de acceso en función de cliente.

/ 6. Diseño del formulario y conexión

En este proceso, se analiza cómo se ejecuta el mecanismo de autenticación de usuarios, comprobando en la base de datos de usuarios si estos existen.

Fig. 8. Diagrama mecanismos de autenticación de usuarios.

En primer lugar, se crea un **formulario** básico como el siguiente, en el que se recogen el nombre de usuario y la contraseña de acceso. Si estos datos son válidos, se realizará la conexión.

```
<FORM ACTION=”sesiones.php” METHOD=”POST”>
Usuario: <input name=”login” type=“text”/>
Pass: <input name=”pass” type=”password”/>
<input type=”submit” value=”Enviar”
class=”enviar”>
</FORM>
```

Código 11. Formulario envío de datos autenticación de conexión.

En este tipo de casos, se aconseja utilizar un método POST, puesto que se trata de datos sensibles que no deberían ser visibles. El funcionamiento completo utilizado bases de datos MySql se verá en los próximos temas, ya que es necesario aprender ciertos conceptos básicos sobre el manejo de consultas con MySQL.

```
$servidor = “localhost”;
$usuario = “root”; $contrasenha = “”;
$BD = “usuarios”;
$conexion = mysqli_connect($servidor, $usuario, $contrasenha);
```

Código 12. Parámetros de conexión a BBDD.

/ 7. Autenticación con base de datos

Una vez implementado el formulario, se modela el fichero **sesiones.php.** El primer paso será obtener el valor de los datos de usuario y contraseña enviados a través del formulario anterior:

```
$idUsuario = $_POST [“login”];
$pass = $_POST [“pass”];
```

Código 13. Obtención datos de usuario.

La conexión con base de datos, así como el acceso para la consulta, inserción o borrado de datos se verá en detalle en temas posteriores.

En este caso, simplemente se modela una consulta utilizando la función *mysql_query*, que recibe por parámetro el literal de la consulta que se va a ejecutar. De esta forma se comprueba si el *login* de usuario se encuentra almacenado y, por lo tanto, se puede autenticar.

```
    $usuario = mysql_query(
      "SELECT * FROM tablaUsuarios WHERE
    usuario = '$idUsuario' AND password =
    '$pass'"
);
```

Código 14. Modelado consultado base de datos con datos de usuario.

Ahora bien, en el bloque de código anterior, se ha extraído de la tabla de usuarios aquellos registros que coincidan con el *login* introducido en el formulario. Debería haber solo uno. Por esta razón, cuando se realiza un registro en una web, el login debe ser único.

A continuación, comprobamos si existe algún usuario con el *login* de acceso y, si es así, creamos una nueva sesión. Además, se debe incluir en el *array* de **$_SESSION** la información necesaria.

```
    //Si el usuario se encuentra en los registros recuperados de la BBDD
  if(mysql_num_rows($usuario)>0){
    session_start();//Se crea una nueva sesión
   //Se almacena en el array de SESSION, utilizando una clave común de
inserción
    $_SESSION['login']="$usuario";
   //Se redirecciona a la página de usuario autenticado
    header("Location: paginaUsuario.php");
    exit();
  }
```

Código 15. Búsqueda usuaria en datos recuperados.

Si todo el proceso concluye con éxito, se redirecciona al usuario autenticado al recurso apuntado desde la función ***header()***.

/ 8. OpenID

Cada vez está más presente la posibilidad de una autenticación universal, es decir, la creación de cuentas e identificadores de usuarios que pueden ser utilizadas como acceso a múltiples servicios y aplicaciones web. Algunas de las ventajas de este tipo de mecanismos son:

VENTAJAS
Creación de una única cuenta para acceder y manejar diversos servicios
Reducción de *logins* y *password*, que pueden dar lugar a errores y pérdidas
Se reduce la cantidad de información que se proporciona a todos los sitios web en los que hay que crear una cuenta de acceso

Tabla 2. Ventajas mecanismos autenticación de usuarios.

Bajo esta filosofía, se encuentra el estándar **OpenID**, que permite que cualquier usuario de un sitio web pueda identificarse en un sitio web utilizando una URL. A continuación, podrá ser verificado desde cualquier servidor que soporte este protocolo. Por lo tanto, se trata de un estándar descentralizado de verificación.

Una de las ventajas de este estándar es que evita a todos los usuarios que acceden a sitios que soportan OpenID la necesidad de crear una nueva cuenta de acceso. Esto se consigue gracias al identificador de identidad, *Identity Provider (IdP)*.

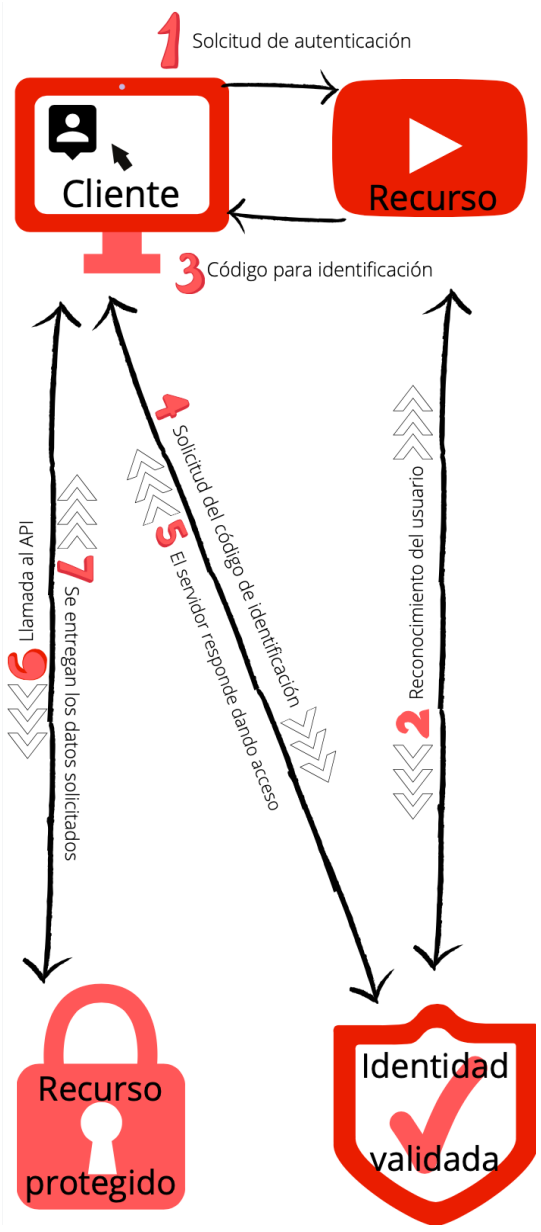

Fig. 9. Diagrama OpenID.

Audio 1. "Mecanismos de autenticación: OpenID, OAuth y SAML".
https://bit.ly/3iCCc23

/ 9. Caso práctico 1: “Primer formulario de autenticación”

Planteamiento: Se van a diseñar los ficheros PHP y el formulario HTML que permiten iniciar una sesión. En este caso, se comprueba si el usuario y la contraseña están previamente registrados utilizando las estructuras de control oportunas.

Por lo tanto, se va a diseñar lo siguiente:

- Formulario HTML que reciba usuario y contraseña.
- Fichero PHP que compruebe si el usuario introducido es «userCorrecto» y la contraseña «passCorrecto»; en el caso ser así, devolverá un mensaje de bienvenida donde aparece el nombre de usuario pasado por parámetro y validado; en este caso, «userCorrecto».

Nudo: El fichero HTML donde se implementa el formulario para recoger los datos presenta el aspecto habitual, que se muestra a continuación, se selecciona GET o POST en función del resto de especificaciones del sistema.

```
   <html>
<body>
<FORM ACTION=”inicioSesion.php” METHOD=”GET”>
 Introduce usuario y pass y pulsar Aceptar
 </br>
Usuario: <input type=”text” name=”usuario”>
</br>
Password: <input type=”text” name=”pass”>
</br>
<input type=”submit” value=”Aceptar” />
</FORM>
</body>
</html>
```

Código 16. Formulario de acceso HTML.

Desenlace: Para el modelado de este fichero PHP, se utilizan estructuras de control if-else, donde evalúa si el usuario es el que se espera para poder acceder, y si lo es, se le da acceso; si no, no se le da.

```
<?php
$Usuario1=$_GET[“usuario”];
if ($Usuario1==”userCorrecto”){
session_start(); // Se crea la sesión
$_SESSION[“id”] = $Usuario1;

echo “Se ha creado una sesión con el nombre usuario: “.$_SESSION[“id”].”<br>”;
}else{
echo “No se puede iniciar una sesión con este usuario”;
}
?>
```

Código 17. Código PHP para comprobar usuario.

/ 10. Caso práctico 2: “Contador de sesiones”

Planteamiento:

El control del estado de las sesiones ofrece multitud de aplicaciones, como llevar un conteo sobre el número de veces que se ha visitado una misma página. En este segundo caso práctico, se debe implementar un programa en PHP que muestre por pantalla un contador que aumenta cada vez que el usuario accede de nuevo al sitio.

Nudo:

A continuación, se va a crear un fichero PHP que inicia una sesión y comprueba si es la primera vez; en este caso, comienza la cuenta y, en el caso de no ser el primer acceso, incrementa el valor del contador.

```
<html>
<head>
<title>Contador de visitas por usuario</title>
</head>
<body>
<?php
    session_start();
    if (!isset($_SESSION[“contador”])){
            $_SESSION[“contador”] = 1;
    }else{
            $_SESSION[“contador”]++;
    }
echo “Esta página se ha visto “ . $_SESSION[“contador”] . “ veces”;
?>
<br>
<br>
</body>
</html>
```

Código 18. Código PHP y HTML para contador de visitas.

Desenlace:

Si todo ha ido bien, cada vez que actualizamos la página o, incluso, si realizamos algún cambio en el código anterior, el conteo tiene que seguir aumentando, puesto que no se han eliminado los datos de la sesión actual, ni tampoco se ha reseteado el valor del contador.

/ 11. OAUTH

Se trata de un estándar abierto que **permite la autorización de APIS, lo que posibilita compartir información entre varios sitios sin compartir la identidad**. Actualmente, se utiliza bastante.

En el mecanismo habitual de conexión entre cliente y servidor, si se quieren mantener ciertos estándares de seguridad, **se requiere de un proceso de autenticación** en el que el usuario debe enviar sus credenciales.

Este estándar está cada vez más presente. Por ejemplo, en la actualidad, con nuestra cuenta de Google, se puede acceder a multitud de aplicaciones y servicios web sin necesidad de crear una nueva cuenta.

En OAuth podemos encontrar 4 roles: propietario del recurso (es el encargado de autorizar el acceso a una aplicación), clientes, servidores de recursos (aquí se sitúan las cuentas de usuario validadas) y servidores de autorización.

El proceso completo de funcionamiento se detalla a continuación:

1. El cliente realiza una petición de acceso.
2. El propietario del recurso autoriza, o no, la solicitud.
3. Se solicita al servidor de autorización el acceso; para ello, debe presentar su autenticación y la autorización otorgada por el propietario del recurso.
4. Si el acceso es validado, se envía un token de acceso al cliente.
5. Se realiza el envío de token de acceso al servidor de recursos.
6. Si el *token* es válido, el recurso es devuelto al cliente como respuesta a su petición.

Este estándar **resulta bastante útil en aquellos casos en los que se requiere la integración con aplicaciones de terceros**; por ejemplo, si desea publicar los resultados obtenidos en una aplicación deportiva en una red social, sin necesidad de proporcionarle a la primera los datos de usuario de la segunda, sí que se le permite el acceso para la publicación de los datos obtenidos.

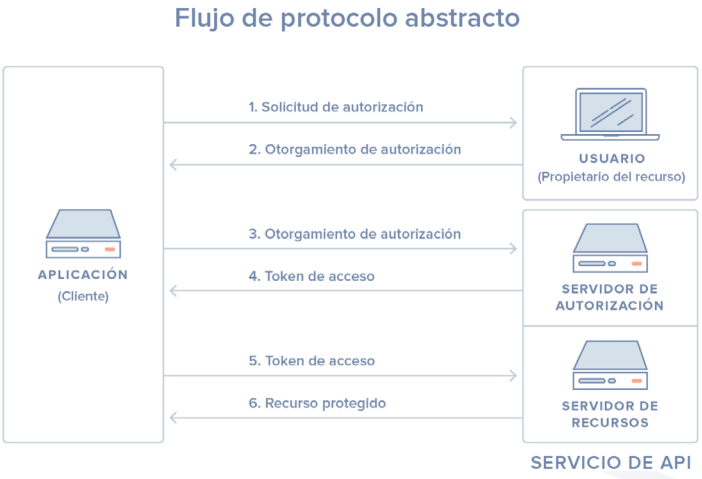

Fig. 10. Diagrama de funcionamiento OAuth.
Fuente imagen: https://www.digitalocean.com/

/ 12. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos analizado de forma práctica los principales mecanismos para el control y mantenimiento del **estado de sesiones** de usuarios en los sitios web.

Tanto el diseño basado en ***cookies*** como en **sesiones** presentan unos fines semejantes, la principal diferencia reside en el lugar en el que queda almacenado el estado de la sesión.

A través del lenguaje de programación PHP y las librerías de funciones desarrolladas para estos fines, es posible modelar e implementar los algoritmos de **acceso y autenticación** necesarios para garantizar a los usuarios una experiencia de navegación óptima.

Finalmente, también se han identificado algunos métodos de **autenticación universal**, cada vez más presentes en a la actualidad, que implementan ciertos mecanismos que permiten la autenticación en sitios web utilizando una única cuenta de acceso. Algunos de estos mecanismos son **OpenId** o **OAutch**, y son utilizados en ejemplos tan conocidos como el acceso con las cuentas de Google o de Facebook, que permiten la identificación y acceso a cada vez más sitios web.

Resolución del caso práctico de la unidad

Como se ha visto a lo largo del tema, el proceso de autorización que se definía en el audio inicial como el mecanismo que delimita los recursos disponibles o no, para según qué usuarios, **no es lo mismo que la autenticación.**

La autenticación se define como el proceso mediante el cual un usuario se identifica en una aplicación. Por lo tanto, la diferencia principal entre autenticación y autorización es que mientras que la primera determina si se tiene o no acceso a una aplicación o sitio web, la autorización establece a qué recursos de dicho sitio se permite o se deniega el acceso a un usuario autenticado.

En este tema, hemos analizado cómo es el proceso de autenticación. Esto es muy importante en lo que respecta al diseño web en entorno servidor, puesto que permite adaptar la aplicación al usuario que accede a él. Ya sabemos que esto sucede gracias a dos importantes mecanismos: las cookies y el control de mantenimiento mediante sesiones.

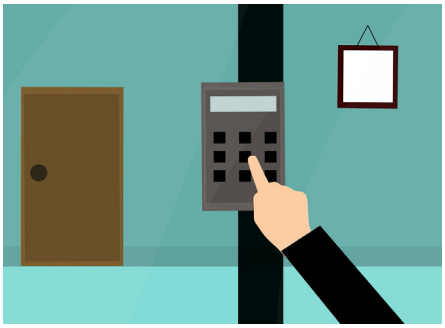

Fig. 11. Diagrama "autenticación" por código.

/ 13. Bibliografía

Ganzabal, X. (2019). Desarrollo web en entorno servidor. (1.a. ed.). Madrid: Síntesis.

Vara, J. M. (2012). Desarrollo en entorno Servidor. (1.a. ed.). Madrid: Rama.

Webgrafía

PHP. Recuperado de: https://www.php.net/manual/es/intro-whatis.php

Programación Lado Servidor. Recuperado de: https://developer.mozilla.org/es/docs/Learn/Server-side