Unidad04 Desarrollo de aplicaciones web usando código embebido

2º DAW – Desarrollo Web Entorno Servidor

Basados en los apuntes del Profesor JCM

CURSO 2020/2021

4 Control de Sesiones

4.1 Introducción

Funcionamiento de la web se basa en modelo estático o no conectado. El salto de una página web a otra implica la pérdida de datos y no preserva los estados de las mismas y la única forma de garantizar la permanencia de dichos estados es mediante el control o manejo de sesiones.

4.1.1 Control de Sesiones

El manejo de sesiones nos permite controlar que se ejecuten de manera correcta y segura los recursos o los servicios ofrecidos por la aplicación web.

El objetivo principal es guardar información específica de cada usuario mientras este navega por una aplicación web.

Cada usuario que entra en un sitio web abre una sesión que es diferente e independiente de la sesión de otros usuarios.

La sesión evita que un usuario pase de una página a otra y pierda la información o tenga el impedimento de continuar realizando alguna operación on-line en la que tiene privilegios.

Por otro lado, si no existieran las sesiones el servidor no tendría control de lo que ocurre en el cliente y podría estar asignando un espacio de trabajo a un usuario no autorizado o viceversa.

4.1.2 Establecer una Sesión

Procesos cuando se establece una sesión:

- URL (Variables de sesión)
- Uso de las cookies

4.1.2.1 URL o Variables de Sesión

Se pasan las variables o los datos de un punto a otro mediante los métodos GET o POST

Cuando un cliente se identifica al servidor éste envía al cliente un código único de identificación y gestión de sesión llamado **SID**:

- Número aleatorio generado por el servidor y que se usa para gestionar la sesión
- Será almacenado en el cliente hasta que se destruya la sesión (cookies)

En el lado del **servidor** además del SID se almacena un conjunto de datos relacionados con las variables de sesión declaradas.

4.1.2.2 Cookies

En caso de las cookies o galletas de información en el cliente se almacena un archivo que guarda SID y otros parámetros relacionados con la expiración.

Mediante este pequeño archivo el servidor puede identificar al cliente y puede controlar la sesión garantizando la funcionalidad operativa.

Sin embargo, el uso de las cookies puede estar condicionado por lo siguiente:

- Muchos navegadores web no lo soportan
- El cliente no permite su uso
- Está desactivado su uso en el navegador

Para evitar que este inconveniente condicione la funcionalidad del sistema algunos lenguajes embebidos como PHP fuerzan a utilizar ambos sistemas, de forma que en caso de no poder usar cookies el sistema siga operando mediante el control de sesión mediante URL.

4.2 Control de Sesiones en PHP

PHP cuenta con todo un catálogo de funciones relacionadas con el manejo y control de sesiones y de variables de sesión:

- http://php.net/manual/es/book.session.php

Una de las grandes utilidades de las sesiones es la de guardar información acerca de la visita de un usuario específico tales como nombre, usuario, correo electrónico, contraseña, etc. Información que será preservada mientras la sesión esté vigente.

4.2.1 Funciones de Inicio y Cierre de Sesión

- **session_start()**: crea una sesión o continúa con la actual. Hay que ponerlo en cada página donde queramos acceder a las variables de la sesión (es lo que se llama propagar la sesión). Al llamar a esta función se verifica si se ha creado el identificador de sesión **SID**, en tal caso devuelve dicho valor y en caso contrario crea uno nuevo.
- **session_destroy()**: destruye toda la información asociada con la sesión actual. No destruye ninguna de las variables globales asociadas con la sesión, ni destruye la cookie de sesión. Para volver a utilizar las variables de sesión se debe llamar a session_start().

  Para destruir la sesión completamente, como desconectar al usuario, el id de sesión también debe ser destruido. Si se usa una cookie para propagar el id de sesión (comportamiento por defecto), entonces la cookie de sesión se debe borrar. setcookie() se puede usar para eso. Lo veremos en apartados posteriores.

4.2.2 Funciones para Identificar Sesión

- **session_id()**: se usa para obtener o establecer el SID de sesión para la sesión actual. Si se especifica **id**, reemplazará el id de sesión actual. session_id() necesita ser llamado antes de session_start() para este propósito. Dependiendo del gestor de sesión, no todos los caracteres están permitidos dentro del id de sesión
- **session_name()**: devuelve el nombre de la sesión actual. Si se da el nombre **name**, session_name() actualizará el nombre de la sesión y devolverá el nombre *antiguo* de la sesión, así, **session_name('nombre')** permite asignar un nuevo nombre a la sesión actual.

4.2.2.1 Variables de Sesión

La información relacionada con una sesión se puede almacenar además de con las **cookies** con **variables de sesión**. Las variables de sesión se diferencian del resto en que se almacenan en un espacio de memoria del servidor junto con el **SID** de la sesión correspondiente.

Declaración de una variable de sesión:

- $_SESSION[‘nombre variable’] = valor

Hay que tener en cuenta que para poder declarar una variable de sesión previamente hay que llamar a la función **session_start()**. Este método hay que llamarlo cada vez y en cada una de las páginas que estemos utilizando la variable de sesión.

$_SESSION es una array asociativo y el valor que se le asigne puede ser de cualquier tipo, incluso puedo contener un nuevo array de cualquiera de los tipos: asociativo, indexado, etc…

Destruir una variable Sesión

Hay que tener en cuenta que **session_destroy()** reinicializa la información de la sesión pero no libera la variable, si queremos liberarla tenemos que usar **session_unset().**

4.2.3 Ejemplos

4.2.3.1 Comprobación de Variable de Sesión

```
<?
      session_start();
      if (!isset($_SESSION['nombre'])) $_SESSION['nombre']="Miguel";
?>
```

Y en otra página podríamos poner

```
<?    session_start();
      if (isset($_SESSION['nombre'])) echo $_SESSION['nombre'];

?>
```

4.2.3.2 Variables de Sesión tipo Array

```
<?
      session_start();
      $_SESSION["personas"][0]["nombre"]="Pepe";
      $_SESSION["personas"][0]["Edad"]=17;
      $_SESSION["personas"][0]["Peso"]=80;
      $_SESSION["personas"][1]["nombre"]="Julián";
      $_SESSION["personas"][1]["Edad"]=25;
      $_SESSION["personas"][1]["Peso"]=120;
      // y para visualizar
      echo $_SESSION["personas"][0]["nombre"];
?>
```

Las sesiones se almacenan en el servidor, en un directorio temporal. En el fichero **php.ini** en la directiva **session.save_path**

4.2.3.3 Destruir una Sesión

```
<?php
      # Si está usando session_name("miSesion")
      session_start();

      # Destruir todas las variables de sesión.
      $_SESSION = array();

      # Destruir la información de la sesión.
      session_destroy();
?>
```

4.2.3.4 Contador de Visitas

```
<?php
      # Iniciamos la sesión o recuperamos la anterior sesión existente
      session_start();

      # Comprobamos si la variable ya existe
      if (isset($_SESSION['visitas']))
            $_SESSION['visitas']++;
      else
            $_SESSION['visitas'] = 1;
?>
```

Si en lugar del número de visitas, quisieras almacenar el instante en que se produce cada una,

la variable de sesión "visitas" deberá ser un array y por tanto tendrás que cambiar el código anterior por:

```
<?php
      # Iniciamos la sesión o recuperamos la anterior sesión existente
      session_start();

      # En cada visita añadimos un valor al array "visitas"

      $_SESSION['visitas'][] = mktime();
?>
```

4.3 Cookies

Ya sabemos que una de las principales utilidades de las cookies (galletas) es la de solventar el problema de la falta de estado en la navegación a través de las páginas web.

Con las cookies, pequeñas porciones de información se quedan registradas en el navegador permitiendo identificar a este a través de diferentes páginas de un mismo sitio e incluso durante visitas entre distintos días.

Realmente las cookies no son más que cadenas de texto que son enviadas desde el servidor al cliente (navegador) y almacenadas en este, luego el navegador envía estas cookies al servidor permitiendo así la identificación del cliente en el servidor.

4.3.1 Funcionamiento de las Cookies

Las cookies se transmiten entre el navegador y el servidor web utilizando los encabezados del protocolo HTTP. Por ello, las sentencias **setcookie** deben enviarse antes de que el navegador muestre información alguna en pantalla.

El proceso de recuperación de la información que almacena una cookie es muy simple. Cuando accedes a un sitio web, el navegador le envía de forma automática todo el contenido de las cookies que almacene relativas a ese sitio en concreto. Desde PHP puedes acceder a esta información por medio del array **$_COOKIE**.

Siempre que utilices cookies en una aplicación web, debes tener en cuenta que en última instancia su disponibilidad está controlada por el cliente. Por ejemplo, algunos usuarios deshabilitan las cookies en el navegador porque piensan que la información que almacenan puede suponer un potencial problema de seguridad. O la información que almacenan puede llegar a perderse porque el usuario decide formatear el equipo o simplemente eliminarlas de su sistema.

Si una vez almacenada una cookie en el navegador quieres eliminarla antes de que expire, puedes utilizar la misma función **setcookie** pero indicando una fecha de caducidad anterior a la actual.

4.3.2 Función setcookie()

El manejo de cookies en PHP se realiza mediante el uso de la función **setcookie**(), esta función está disponible a partir de la versión 3 de PHP.

Sintaxis de setcookie():

```
int setcookie (string Nombre [, string Valor [, int Expire [, string
Path [, string Dominio [, int Secure]]]]])
```

Setcookie() define una cookie que es enviada junto con el resto de la información de la cabecera(header). Las cookies deben ser enviadas antes de cualquier tag de html, por lo tanto deberemos realizar la llamada a estas funciones antes de cualquier <HTML> o <HEAD>. Esta es una restricción de las cookies no de PHP.

Todos los argumentos excepto el nombre son opcionales.

- **Nombre**. Nombre de la cookie. Si creamos una cookie solamente con el nombre, en el cliente se eliminará la cookie que exista con ese nombre. También podemos reemplazar cualquier argumento con una cadena vacía ("").
- **Value**. Valor que almacenará la cookie en el cliente.
- **Expire**. El argumento expire es un argumento entero que indica la hora en que se eliminara la cookie en el formato de hora que devuelven las funciones UNIX time() y mktime(). Normalmente se usa time() + N. segundos de duración, para especificar la duración de una cookie.
- **Path**. Subdirectorio en donde tiene valor la cookie.
- **Dominio**. Dominio en donde tiene valor la cookie. Si ponemos como dominio www.domain.com la cookie no se transmite para domain.com, mientras que si ponemos domain.com la cookie se transmite tanto para domain.com como para www.domain.com
- **Secure**. El argumento secure indica que la cookie solo se transmitirá a través de una conexión segura HTTPS.

4.3.3 Leer contenido Cookie

Para leer el contenido de una cookie sólo tenemos que usar el array asociativo:

- $_COOKIE[‘variable’]

4.3.4 Eliminar Cookie

La forma más sencilla de eliminar una cookie es volviéndola a declarar pero sin la asignación de ningún valor:

```
Setcookie('variable')
```

Pero este método a veces falla, por ejemplo algunos navegadores no borran la cookie sino coincide el path, por ello la forma más segura de borrar una cookie es colocar una nueva con el mismo nombre, sin valor, mismo path y nombre de dominio si se especificaron y con un tiempo de vida negativo, por ejemplo time() – 3600

```
setcookie("nombre", "", time()-3600)
```

4.3.5 Ejemplos de uso Cookies

4.3.5.1 Registro número de visitas

```
<?php
If (isset($_COOKIE['visitante']))
{
      $numero=$_COOKIE['visitante'];
      $numero+=1;
}
else
     $numero=1;
setcookie("visitante",$numero,time()+86400);
print "Es la $numero ª vez que visitas esta página";
?>
```

4.3.5.2 Varios valores a una misma cookie

Podemos asignar **varios valores a una misma cookie**, aunque realmente se crean tantas cookies como valores se introducen

```
setcookie("Libro[0]","El médico",time()+300);
setcookie("Libro[1]","Noah Gordon",time()+300);
setcookie("Libro[2]","1992",time()+300);
```

4.3.5.3 Cookies con valores tipo array

Es posible crear cookies en las que la variable contiene un array de tipo escalar o de tipo

asociativo. Es necesario incluir un setcookie por cada uno de los valores del array aunque a la hora de devolver esa variable el navegador del cliente envía el array completo:

```
<?
      $valores=Array("Verde","Verano","Rolls-Royce","Millonario");
      setcookie("cookie3[color]",$valores[0],time()+3600);
      setcookie("cookie3[estacion]",$valores[1],time()+3600);
      setcookie("cookie3[coche]",$valores[2],time()+3600);
      setcookie("cookie3[finanzas]",$valores[3],time()+3600);

      if (isset($_COOKIE['cookie3']) ) {
            foreach($_COOKIE['cookie3'] as $indice=> $valor) ) {

                  echo "$indice == $valor\n";

            }

      }
?>
```

4.3.5.4 *Comprobar si las cookies están activadas*

```
<?php
      setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
      if(count($_COOKIE) > 0) {
            echo "Cookies are enabled.";
      } else {
            echo "Cookies are disabled.";
}
?>
```

4.4 Autenticación y autorización.

En este punto trataremos distintos tipos de autenticación que podemos implementar con PHP y veremos la diferencia entre autenticación y autorización:

La autentificación es el procedimiento para comprobar la identidad de un usuario o proceso mientras que con la autorización permitimos o no a los distintos usuarios poder acceder los

distintos recursos.

Lo lógico en una aplicación web es que haya recursos accesibles por todos los usuarios, por lo tanto, estarán en un lugar público accesible por todo el mundo, y no será necesario estar autentificado para acceder a dichos recursos.

Por otro lado, habrá recursos accesibles solo por ciertos usuarios, para poder acceder a dichos recursos si es necesario estar autentificados.

4.4.1 Tipos de autenticación.

A continuación, veremos distintos tipos de autenticación que podremos implementar en nuestra aplicación PHP, nosotros en este curso nos vamos a centrar en el primer método, aunque también se va a proponer una práctica que trata sobre el segundo método.

4.4.1.1 Autenticación HTTP

Para este tipo de autenticación nos basamos en la cabecera http, dentro de esta forma básica de autenticación podemos usar dos métodos Basic y Digest. En PHP.net encontramos ejemplos con ambos métodos. La principal diferencia es que el método Digest utiliza un hash para la clave.

Este tipo de autenticación por sí sola, aunque enviemos los datos cifrados, pueden ser interceptados mediante un ataque Man in the middle, por lo que debería ser usada en combinación con una conexión TLS/HTTPS, y nunca solo con HTTP, en este curso lo trataremos con HTTP de manera ocasional al no ser objeto de estudio HTTPS.

A continuación, se indican algunos ejemplos extraídos de php.net:

Con la función header() se puede enviar un mensaje de "Autenticación requerida" al navegador del cliente para mostrar una ventana emergente donde introducir un usuario y una contraseña. Una vez introducidos estos datos, el URL que contiene el script de PHP será invocado de nuevo con las variables predefinidas PHP_AUTH_USER, PHP_AUTH_PW y AUTH_TYPE establecidas al nombre de usuario, contraseña y tipo de autenticación, respectivamente. Estas variables se encuentran en el array $_SERVER. Se admiten ambos métodos de autenticación, 'Basic' y 'Digest' (desde PHP 5.1.0). Véase la función header() para más información.

Ejemplos:

Ejemplo #1 Ejemplo de autenticación HTTP 'Basic'

```php
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="Mi dominio"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Texto a enviar si el usuario pulsa el botón Cancelar';
    exit;
} else {
    echo "<p>Hola {$_SERVER['PHP_AUTH_USER']}.</p>";
```

```
    echo "<p>Introdujo {$_SERVER['PHP_AUTH_PW']} como su contraseña.</
p>";
}
?>
```

Ejemplo #2 Ejemplo de autenticación HTTP 'Digest'

Este ejemplo muestra cómo implementar un sencillo script de autenticación HTTP 'Digest'. Para más información lea el » RFC 2617.

```
<?php
$dominio = 'Area restringida';

// usuario => contraseña
$usuarios = array('admin' => 'micontraseña', 'invitado' => 'invitado')
;


if (empty($_SERVER['PHP_AUTH_DIGEST'])) {
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Digest realm="'.$dominio.
           '",qop="auth",nonce="'.uniqid().'",opaque="'.md5($dominio).
'"');

    die('Texto a enviar si el usuario pulsa el botón Cancelar');
}


// Analizar la variable PHP_AUTH_DIGEST
if (!($datos = analizar_http_digest($_SERVER['PHP_AUTH_DIGEST'])) ||
    !isset($usuarios[$datos['username']]))
    die('Credenciales incorrectas');


// Generar una respuesta válida
$A1 = md5($datos['username'] . ':' . $dominio . ':' . $usuarios[$datos
['username']]);
$A2 = md5($_SERVER['REQUEST_METHOD'].':'.$datos['uri']);
$respuesta_válida = md5($A1.':'.$datos['nonce'].':'.$datos['nc'].':'.$
datos['cnonce'].':'.$datos['qop'].':'.$A2);

if ($datos['response'] != $respuesta_válida)
    die('Credenciales incorrectas');

// Todo bien, usuario y contraseña válidos
echo 'Se ha identificado como: ' . $datos['username'];


// Función para analizar la cabecera de autenticación HTTP
function analizar_http_digest($txt)
{
    // Protección contra datos ausentes
    $partes_necesarias = array('nonce'=>1, 'nc'=>1, 'cnonce'=>1, 'qop'
=>1, 'username'=>1, 'uri'=>1, 'response'=>1);
    $datos = array();
    $claves = implode('|', array_keys($partes_necesarias));
```

```
    preg_match_all('@(' . $claves . ')=(?:([\'"])([^\2]+?)\2|([^\s,]+)
)@', $txt, $coincidencias, PREG_SET_ORDER);

    foreach ($coincidencias as $c) {
        $datos[$c[1]] = $c[3] ? $c[3] : $c[4];
        unset($partes_necesarias[$c[1]]);
    }

    return $partes_necesarias ? false : $datos;
}
?>
```

Como podéis observar, en estos ejemplos las credenciales están almacenadas en el propio código, lo normal es que éstas, estén almacenadas en una base de datos por ejemplo.

Otra opción es almacenarlas en el propio servidor Apache, aunque esto hace que la aplicación sea menos portable, en el siguiente enlace se explica cómo realizar la autenticación y autorización con las credenciales almacenadas en el servidor:

https://httpd.apache.org/docs/trunk/es/howto/auth.html

4.4.1.2 Autenticación manual

Consiste en realizar por nuestra cuenta un formulario de login, comprobando las credenciales de forma manual, ya sea contra una base de datos u otro sistema, permitir al usuario entrar a los recursos autorizados.

Al igual que en el caso anterior, este tipo de autenticación por sí sola, aunque enviemos los datos cifrados, pueden ser interceptados mediante un ataque Man in the middle, por lo que debería ser usada en combinación con una conexión TLS/HTTPS, y nunca solo con HTTP, en este curso lo trataremos con HTTP de manera ocasional al no ser objeto de estudio HTTPS.

4.4.1.3 OAUTH2

Es un método de autenticación utilizado por empresas como Twitter, Facebook o Google entre otros, y lo que permiten es, a otros proveedores autenticarse con sus credenciales sin llegarles a facilitar esta información a dichos proveedores.

Cuando usamos dicho método nos sale una pantalla similar a la siguiente, donde aceptamos que se compartirá información con el proveedor para poder usar dicho método:

Iniciar sesión con Google

Selecciona una cuenta

para ir a AliExpress

[redacted]

[redacted]

[redacted]

Usar otra cuenta

Para continuar, Google compartirá tu nombre, tu dirección de correo electrónico, tu preferencia de idioma y tu foto de perfil con AliExpress. Antes de usar esta aplicación, puedes leer la política de privacidad y los términos del servicio de AliExpress.