UD5: Mantenimiento del estado. Autenticación y roles

Sitio: Aula Virtual do IES de Teis
Curso: Desarrollo web en entorno servidor 2023-24 (DAW-DUAL-A)
Libro: UD5: Mantenimiento del estado. Autenticación y roles

Impreso por: Abel Mahón Cortés
Data: Mércores, 31 de Xaneiro de 2024, 06:05

Táboa de contidos

1. Autenticación

2. Autorización RBAC

3. HTTP: Protocolo sin estado

4. Cookies en PHP

4.1. Cómo ver las cookies en el navegador

4.2. Buffer de salida: Output Buffering

5. Uso de sesiones en PHP

5.1. Ejemplo básico con sesiones

5.2. Ejemplo de uso de sesiones con la cesta de la compra

1. Autenticación

La **autenticación** es el proceso por el cual se identifica a los usuarios y se garantiza que los mismos son quienes dicen ser.

Para autenticarse, un usuario puede acreditar su identidad por un elemento (o varios) que:

- **conoce**: usuario + contraseña
- **posee**: Por ejemplo, una tarjeta magnética
- **es**: Un elemento biométrico (lectura de iris, huella digital, etc.)

Para autenticarse, un usuario proporciona, en general, al menos 2 elementos: usuario y contraseña

- El nivel de seguridad depende de la complejidad de la contraseña y de la medidas de seguridad del almacenamiento de la misma.

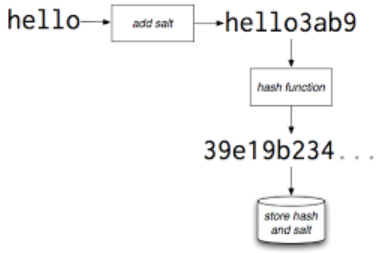
Las contraseñas normalmente no se almacenan en texto plano, sino que se les suele aplicar una función matemática llamada *hash.*

- Para un conjunto de datos dado siempre se ha de obtener el mismo valor del código hash
- El algoritmo ha de ser computacionalmente eficiente. Esto significa que la función ha de obtener el valor del código hash rápidamente.
- No ha de existir una forma directa de obtener el conjunto de datos original a partir del código hash
- Pequeños cambios en la entrada cambian completamente el código hash de salida.
- Pocas colisiones: La probabilidad de que dos conjuntos de datos diferentes generen el mismo código hash ha de ser altamente improbable.

Entre las amenazas de este tipo de autenticación se encuentran:

- **"diccionarios" de contraseñas**. En estos archivos, que circulan libremente en Internet, pueden encontrarse numerosas contraseñas que bien son muy populares o ya fueron interceptadas en el pasado y se prueban por fuerza bruta.
- **Tablas rainbow o tablas arcoiris:** Son ficheros que relacionan una entrada con el resultado de múltiples ejecuciones de funciones hash y de reducciones repetidas. Ofrecen un buen compromiso entre espacio (uso de memoria) y uso de CPU**.** Para saber más: https://www.ionos.es/digitalguide/servidores/seguridad/rainbow-tables/

Para dificultar el descifrado con tablas arco iris se utiliza el denominado valor **salt** ("sal" o "semilla"), una secuencia aleatoria que crea el sistema para la clave que define un usuario al crear una cuenta. Este valor se acopla a la clave y ambos se someten a la función hash, creando así un valor diferente que el que generaría la contraseña sin salt.
https://blogs.quickheal.com/password-security-a-dash-of-salt-and-little-of-hash-to-go-please/



Fuente: https://blogs.quickheal.com/password-security-a-dash-of-salt-and-little-of-hash-to-go-please/

Algunos algoritmos y funciones Hash:

- MD5: Message-Digest *Algorithm* 5 de 125 bits. Criptográficamente vulnerable.
- SHA-X: Tiene diferentes variantes SHA-1 (con vulnerabilidades), SHA-2 con diferentes longitudes (SHA-256, SHA-512, etc) y SHA-3
- Blowfish: Algoritmo de cifrado simétrico de propósito general.
- BCrypt: Está basado en Blowfish específicamente para hash de contraseñas. Permite usar un salt.

KDF (Key Derivation Function): la longitud de la salida con una función de derivación KDF es diferente. Entre estos, se encuentran:

- **Argon2d:** depende de los datos (d). Se supone que el primero es resistente al crackeo de la GPU.
- **Argon2i**: independiente de los datos (i). Es resistente a los ataques de canal lateral.
- **Argon2id** es una versión híbrida entre los 2 anteriores.

En PHP existen las funciones:

- password_hash(): Utiliza 4 posibles algoritmos: Blowfish, Bcrypt (es una versión basada en Blowfish mejorada para contraseñas), Argon2i y Argon2id. Con la opción `PASSWORD_BCRYPT`, utiliza un salt generado automáticamente que se incluye en el propio resultado de la función. Un ejemplo del resultado se puede ver en la siguiente imagen:

- password_verify(): Verifica que una contraseña de entrada coincide con el resultado del hash. Esto es posible porque el propio resultado del hash almacena el algoritmo utilizado para generar el hash, las opciones del mismo y el propio *salt*. Una misma contraseña de entrada, con el mismo algoritmo y mismas opciones, ¿producirá siempre el mismo resultado?
- https://www.php.net/manual/en/faq.passwords.php#faq.password.storing-salts

Para saber más:
https://www.redeszone.net/tutoriales/seguridad/criptografia-algoritmos-hash/
https://synkre.com/how-secure-is-bcrypt/

2. Autorización RBAC

La **autorización** es el proceso que **define a qué recursos del sistema un usuario autenticado podrá acceder**. Un usuario con permisos de lectura no podrá modificar ningún recurso. Un usuario editor podrá modificar sus propios recursos, pero no los ajenos. Un usuario administrador podrá modificar sus propios recursos y los ajenos.

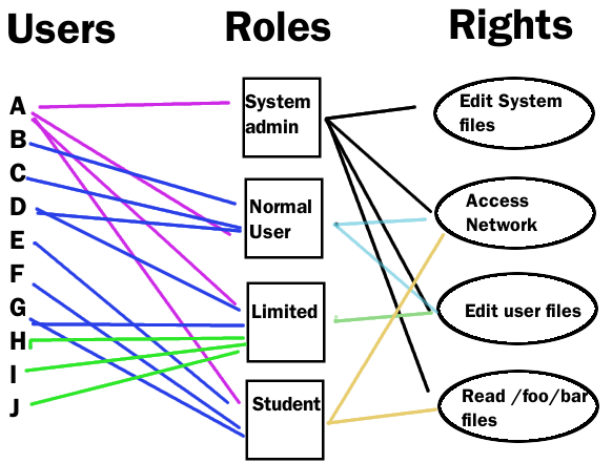
Es habitual que, para acceder a los recursos, se defina una **política de seguridad de acceso** basada en roles **Role-Based Access Control (RBAC)**. En ella suelen definirse:

- **Roles**: nombres abstractos para el permiso para acceder a un conjunto particular de recursos: Administrador global, usuario, administrador de ventas, comercial, etc.

- **Usuarios**: un individuo (o programa) que **ha sido autenticado**. Un usuario puede tener un conjunto de roles asociados con esa identidad, lo que le da derecho a acceder a todos los recursos protegidos por esos roles. Cada usuario debe tener al menos un rol. En algunas organizaciones también se pemite la creación de grupos de usuarios basados en el puesto de trabajo o perfil del cliente (vip, regular, casual, etc.) y una relación jerárquica entre los mismos.

- **Permisos**: Normalmente relacionados con las acciones permitidas: creación, edición, borrado o lectura. Estos permisos pueden especificarse para un determinado recurso (crear usuarios, crear ventas, etc.) o definirse globalmente. Un rol está normalmente asociado a una serie de permisos.

Conviene que una política RBAC sea **flexible** permitiendo personalizaciones dependiendo incluso de la combinación de roles o de permisos. Esto conllevará una **mayor complejidad** en el diseño de la política.



Fuente: https://sites.google.com/site/cacsolin/role-based-access-control

3. HTTP: Protocolo sin estado

Se puede observar el flujo típico del protocolo HTTP en este recurso URL:
https://developer.mozilla.org/es/docs/Web/HTTP/Overview#flujo_de_http

1. El cliente abre una conexión TCP: la conexión TCP se usará para hacer una petición, o varias, y recibir la respuesta.

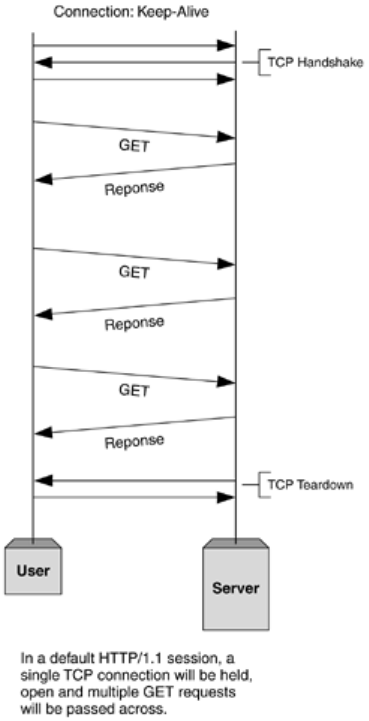
2. El cliente envía una petición HTTP: Los mensajes HTTP (previos a HTTP/2) son legibles en texto plano.

```
GET / HTTP/1.1 Host: developer.mozilla.org Accept-Language: fr
```

3. El cliente lee la respuesta enviada por el servidor

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

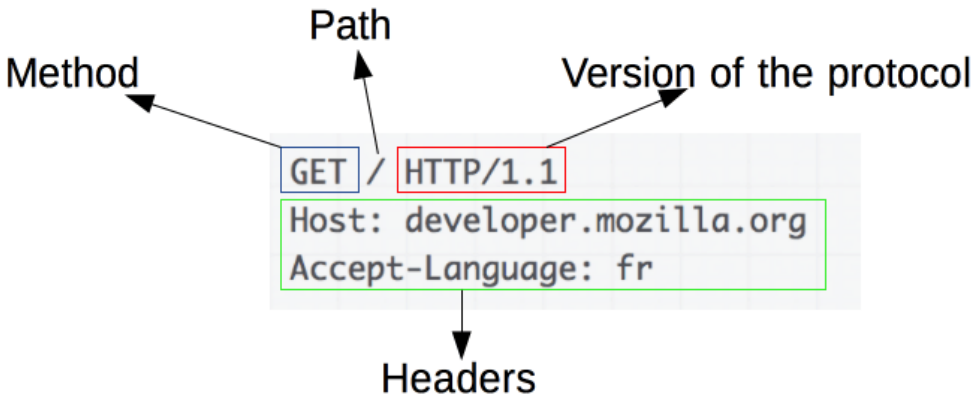
4. Se produce el cierre o reúso de la conexión para futuras peticiones.



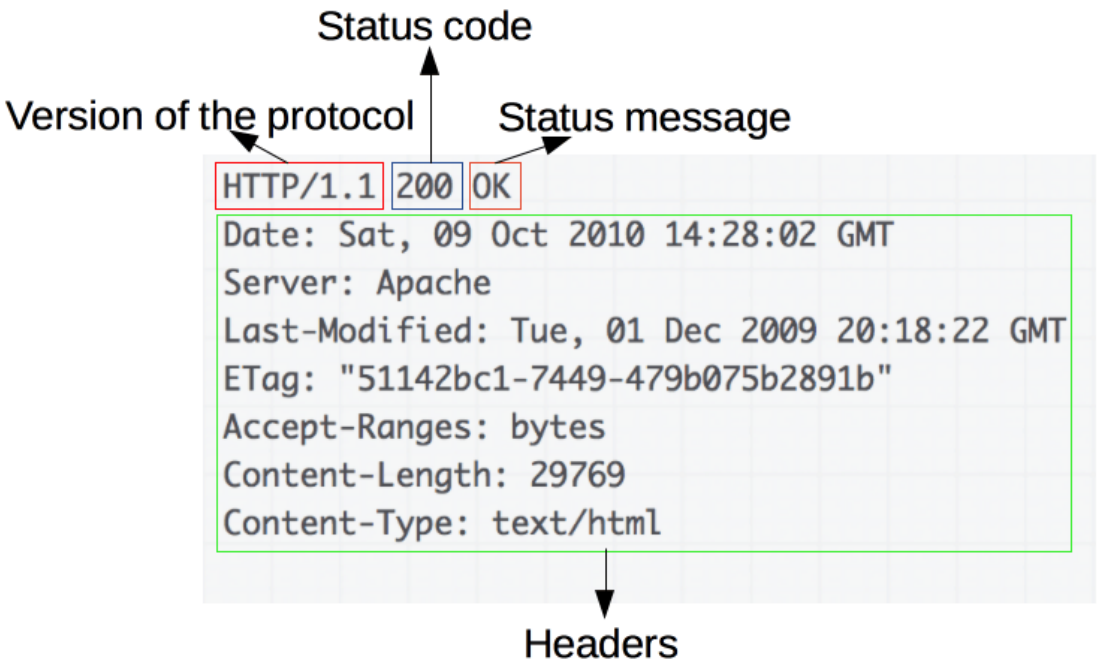
Fuente: https://medium.com/@he4am3id66/http-1-1-introduction-76d90b4834bf

En el mismo recurso se pueden observar los mensajes típicos de HTTP para peticiones y respuestas:
https://developer.mozilla.org/es/docs/Web/HTTP/Overview#mensajes_http

Peticiones HTTP



Respuestas HTTP



HTTP es un protocolo sin estado: no guarda ningún dato entre dos peticiones HTTP en la **misma sesión**. Una **sesión** es un intercambio de mensajes entre el cliente y el servidor durante un determinado período de tiempo

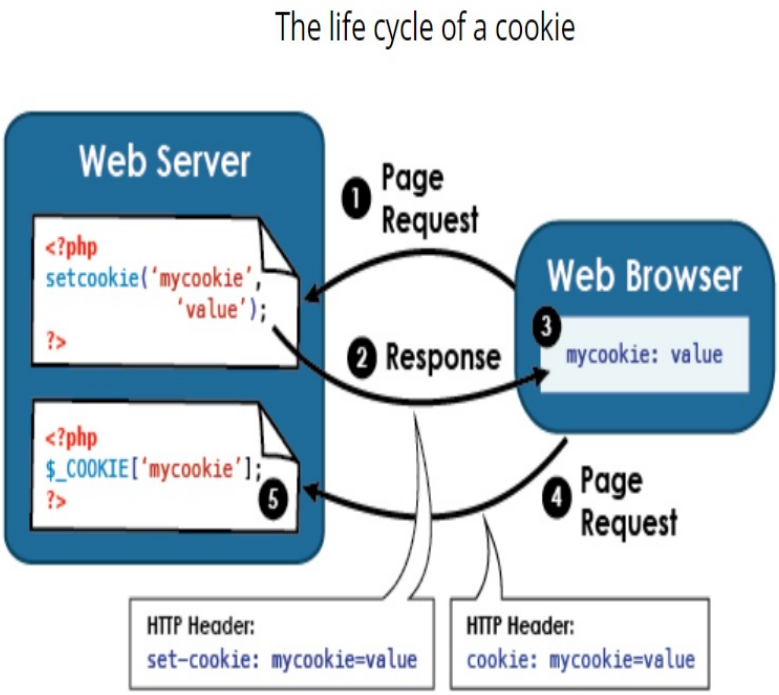
En caso de que los usuarios requieran interactuar con determinadas páginas Web de forma ordenada y coherente, por ejemplo, para el uso de "cestas de la compra" en el comercio electrónico, se requiere de mecanismos que permitan al servidor identificar que las peticiones pertenecen al mismo cliente (conservar los productos añadidos a la cesta, proceder al pago de la misma, conexión con plataformas de pago, etc.)

Uno de los mecanismos que permiten identificar las peticiones provenientes del mismo cliente es el uso de **HTTP cookies**. Se trata de información que los servidores web establecen en la **cabecera de respuesta HTTP** y que posteriormente el navegador enviará al servidor web.

Las cookies almacenan datos en el cliente remoto para monitorizar o identificar a los usuarios que interactúan con el sito web. Los datos contenidos en las cookies pueden ser las preferencias de idioma del usuario, el tamaño de su pantalla o los productos de su carrito de la compra, su dirección IP, el navegador que utiliza o su comportamiento cuando navega por Internet.

4. Cookies en PHP

El ciclo de vida de una cookie



Fuente: https://www.beta-labs.in/2020/03/cookies-in-php.html

1- El navegador realiza una petición HTTP al servidor web

2- El servidor web, en función de una lógica determinada, crea una cookie con un nombre, valor, una duración determinada y otros posibles parámetros. La respuesta HTTP del servidor web incluye en la cabecera HTTP la cookie con su valor y otros posibles parámetros

3- El navegador guarda la cookie que le ha enviado el servidor

4-Cuando el navegador vuelve a interactuar con el mismo servidor web, envía en la petición http las cookies que ha guardado para ese servidor web que no hayan expirado ya

5- Las cookies que le ha enviado el navegador al servidor web, están disponibles en el array superglobal $_COOKIE **a partir de la próxima petición del navegador.**

Cookies en PHP

En PHP las cookies se manejan con:

setcookie(): **setcookie**(

```
    string $name,
    string $value = "",
    int $expires = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
): bool
```

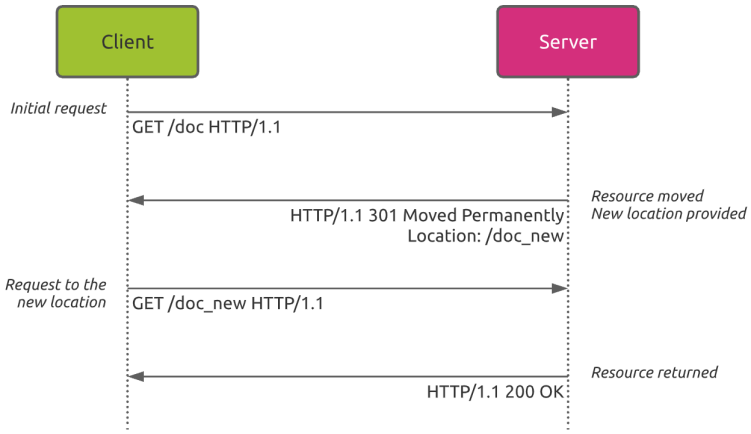
Los 3 primeros argumentos son los más importantes: `setcookie("TestCookie", $value, time()+3600);`

- El nombre de la cookie, con el que se recuperará del array superglobal **$_COOKIE["name"]** **en la siguiente petición del cliente**.
- El valor de la cookie.
- El número de segundos de validez de la cookie: se usa la función **time()**+nº de segundos de validez. Si se establece a 0, o es omitido, la cookie expirará al final de la sesión (cuando el navegador se cierre). Las **cookies de sesión** se eliminan del equipo cuando se cierra el navegador, mientras que las **cookies persistentes** permanecen almacenadas en el ordenador hasta que se eliminen, o hasta que llegue su fecha de caducidad.

Para **borrar una cookie**, ha de forzarse la fecha de expiración en el pasado: `setcookie("TestCookie", $value, time()-3600);`

A tener en cuenta:

- Las cookies de $_COOKIE no serán visibles en el servidor dentro del array superglobal $_COOKIE hasta la próxima petición del mismo cliente. Si hubiese lógica en el servidor dependiente de la existencia de cookies creadas en la petición actual, es posible generar una redirección con la función de PHP header que permite enviar una cabecera Location HTTP. `header('Location: doc_new');`
  Se trata de una cabecera http que forzará una **redirección**: envía el encabezado al navegador devolviendo el código HTTP de status (302) `REDIRECT` al navegador a no ser que el código de status `201` o `3xx` ya haya sido enviado.



Fuente: https://kadiska.com/how-http-redirections-impact-your-web-performance/
El cliente enviará una nueva petición al servidor al recurso indicado por la redirección.
En esta nueva petición, la cookie creada en el servidor en la primera petición ya será visible en $_COOKIE.

- Las cookies deben ser **borradas usando los mismos parámetros con que fueron creadas.** Si el argumento del valor es **un string vacío o** `false`, y todos los demás argumentos coinciden con una llamada anterior a setcookie, entonces la cookie con el nombre especificado será eliminada del cliente remoto. Internamente esto se logra estableciendo el valor a 'deleted' y el tiempo de expiración en el pasado.
- Ya que configurar el valor de una cookie como `false` intentará eliminar la cookie, no se deben utilizar valores booleanos. En su lugar, deberían usarse, *por ejemplo, 0* como `false` y *1* como `true`.
- Los nombres de las cookies pueden declararse como nombres de arrays y estarán disponibles en sus scripts PHP como arrays, pero en el sistema del usuario se guardarán como cookies separadas. Ver ejemplo 3 en https://www.php.net/manual/es/function.setcookie.php#example-4419

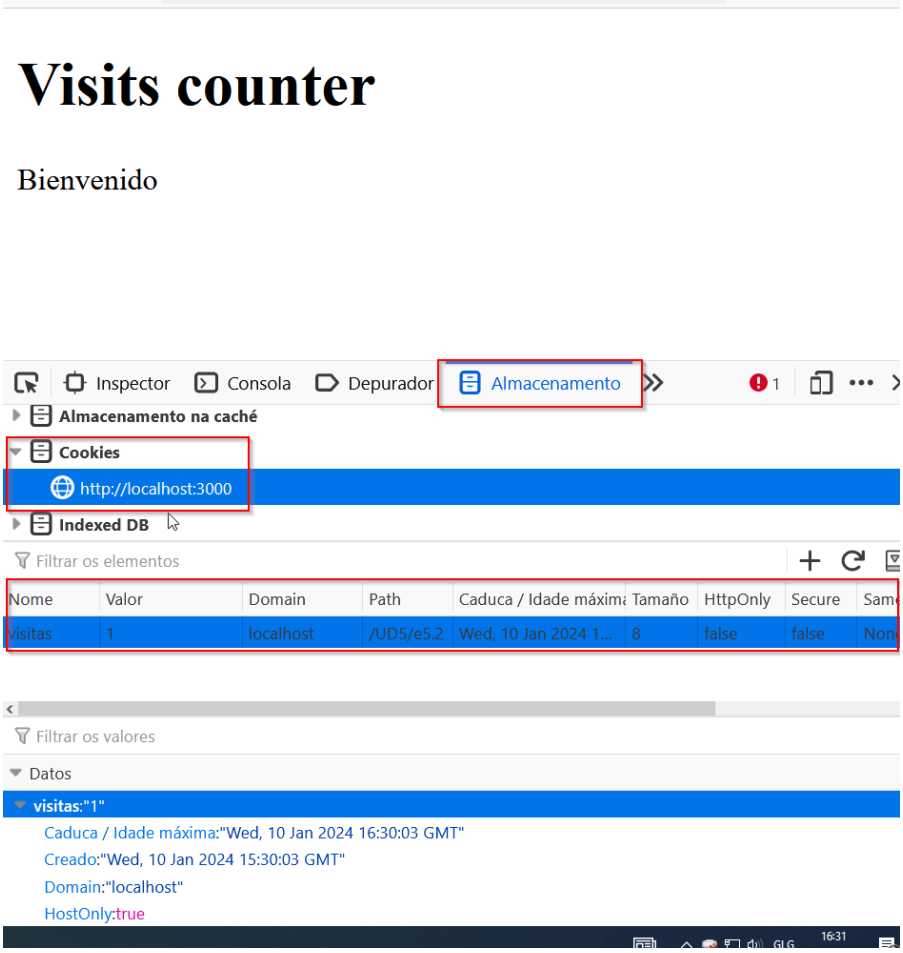
4.1. Cómo ver las cookies en el navegador

Es posible ver las cookies con las herramientas del navegador.

Para ello, veremos un ejemplo sencillo que cuenta las visitas de un mismo navegador: https://github.com/dudwcs/UD5_Ejemplo_Contador_Visitas_Cookies.git

Por ejemplo, en Firefox > F12 > Sección Almacenamiento > Cookies > http://localhost:3000

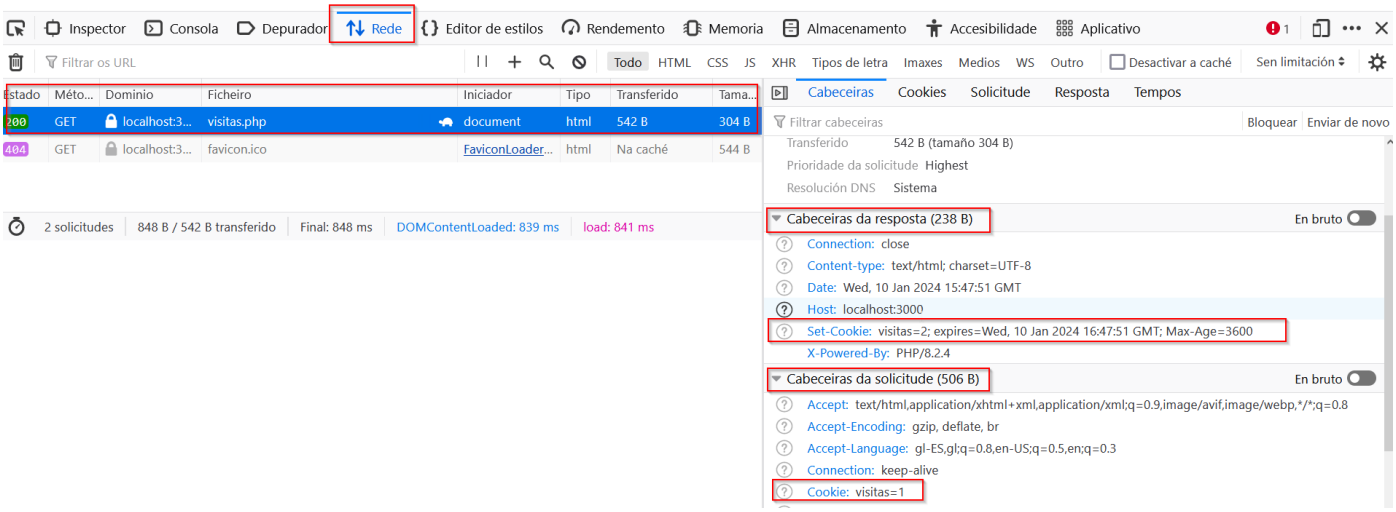
Vemos la cookie "visitas" y su valor 1, además de otros parámetros.



También es posible visualizarlos desde la sección Rede > seleccionamos la petición >el apartado cookies > vemos las cookies de la petición y de la respuesta

Visits counter

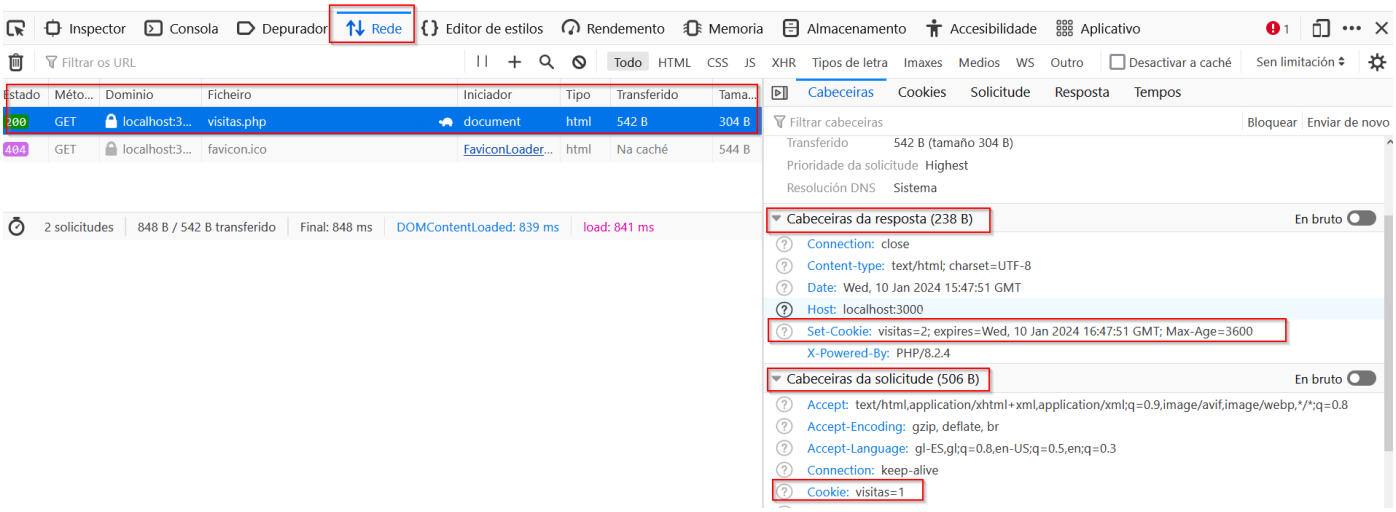
Nos ha visitado 2 veces



En la sección de Rede, también es posible ver las cabeceras de la petición HTTP y de la respuesta HTTP:

Visits counter

Nos ha visitado 2 veces



4.2. Buffer de salida: Output Buffering

Se denomina ***Output Buffering*** a la característica de PHP que nos permite no enviar al cliente la salida a medida que se va generando, sino que se almacena en un buffer controlado por el script.
Existen varias funciones PHP – *session_start, setcookie, header, etc.* – que no se pueden llamar si ya se ha producido alguna salida (estática o dinámica), pues modifican la cabecera HTTP de la respuesta del servidor y se supone que ésta ya se ha generado y ha viajado al cliente antes del primer carácter generado.

Un simple carácter de espacio en blanco antes de un bloque <?php ...ya es una salida, o incluso aunque en el editor de texto no lo veamos, guardar el script en formato UTF-8 (en vez de UTF8-sin BOM – Byte Order Mark – indica el orden de los bytes, big-endian o little endian), también produce una salida y veremos el siguiente mensaje:

Warning: Cannot modify header information - headers already sent by

Para evitar este error podemos programar adecuadamente el script almacenando internamente el contenido generado, comprobando las condiciones necesarias y llamando a las funciones en el orden correcto antes de generar dicho contenido. O también podemos utilizar los mecanismos existentes en PHP para el control del buffer:

Directivas en los archivos de configuración **php.ini**
output_buffering=Off|On|NumBytes

- **Off**: deshabilita el buffer de salida
- **On**: lo habilita de modo ilimitado (usar con cuidado)
- **NumBytes**: Especifica el nº de bytes reservado para el buffer

Funciones PHP de control del buffer

Podemos programar utilizando las funciones de control del buffer de salida (en cada página), para no depender del contenido del **php.ini** al que, probablemente, no tengamos acceso en producción:

- **ob_start()** Activa el almacenamiento en el buffer. Cualquier salida del script será almacenada en dicho buffer (excepto las cabeceras, que irán saliendo en su orden).
- **ob_flush():** Envía el contenido del buffer a la salida y el contenido se descarta
- **ob_end_flush()**: Lo mismo que ob_flush() y, además, deshabilita el output buffering
- **ob_clean()**: desecha el contenido del búfer de salida
- **ob_end_clean()** Limpian el contenido del buffer y lo deshabilita

El uso de las funciones que gestionan el buffer de salida ob_*(), funcionan independientemente de la configuración existente en la directiva **output_buffering**

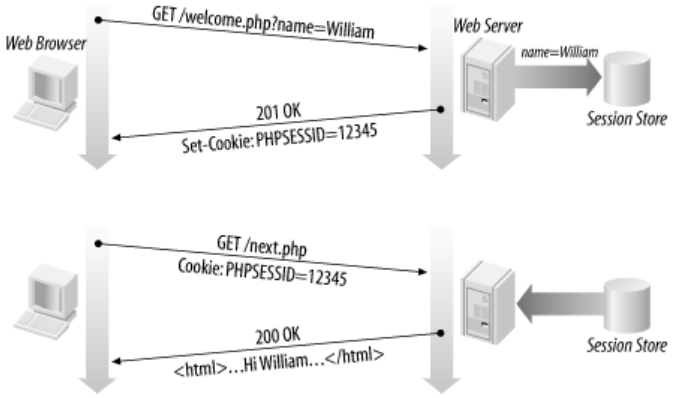
5. Uso de sesiones en PHP

Al ser HTTP un protocolo sin estado, se necesita un mecanismo para relacionar las diferentes peticiones de un mismo cliente a un determinado servidor web.

Las cookies pueden ayudarnos, pero tienen algunas limitaciones y vulnerabilidades. El uso de sesiones implica un almacenamiento del lado servidor.

Conceptualmente se puede pensar en una sesión como un contexto de comunicación cliente-servidor **limitado en el tiempo**.

- Normalmente se usa un identificador de sesión SID que se transmite en forma de cookie al navegador. También es posible pasar el identificador en la URL, pero está desaconsejado por motivos de seguridad, por ejemplo:
  `http://www.mitienda.com/tienda/listado.php&PHPSESSID=34534fg4ffg34ty`
- El servidor almacena el SID de forma persistente, por defecto, en un fichero (aunque se permiten otros mecanismos, por ejemplo, en Bases de datos). El servidor es capaz de recuperar la sesión a través de ese SID único.
- En la sesión se pueden almacenar variables a través del array superglobal $_SESSION: `$_SESSION['clave_variable']=variable;` Se pueden retirar variables de la sesión con unset: `unset ($_SESSION['clave_variable']);`



Fuente: https://litux.nl/mirror/webdatabaseapplicationsphpmysql/webdbapps_snode69.html

En la documentación de PHP podemos ver el flujo básico de utilización de una sesión.

1- Con la llamada a session_start():

- Si existe información con el SID (bien por cookie o por parámetro de la URL), PHP recuperará la sesión existente usando el SID.
- Si no se pasa el SID, se creará una sesión nueva (si la directiva session.auto_start=1). Si no, no se creará una nueva sesión y no estará disponible el array superglobal $_SESSION.

2- PHP rellenará la variable superglobal *$_SESSION* con los datos de la sesión iniciada previamente (si los hubiese) y permitirá que se vayan añadiendo a $_SESSION.

3- Cuando PHP se cierra, automáticamente toma el contenido de la variable superglobal *$_SESSION*, la serializa, y la envía para almacenarla usando el gestor de almacenamiento de sesiones.

Por defecto, PHP usa el gestor interno de almacenamiento `files`, el cual se establece mediante la directiva session.save_handler. Éste guarda los datos de sesión en el servidor en la ubicación especificada por la directiva de configuración session.save_path.

session.save_path="\xampp\tmp" (por defecto en XAMPP)

Funciones relacionadas con el uso de la sesión en PHP:

- session_start(): **crea** una sesión o **reanuda** la actual basada en un identificador de sesión pasado mediante una petición GET o POST, o pasado mediante una cookie. Siempre que se quiera acceder a una sesión **es importante llamar a session_start(),** no solo para crearla.
- session_name(): Obtiene y/o establece el nombre de la sesión actual. Por defecto, PHPSESSID
- session_id(): Obtiene y/o establece el id de sesión actual
- session_save_path(): Obtiene y/o establece la ruta de almacenamiento de la sesión actual
- session_destroy(): Teóricamente, destruye toda la información registrada de una sesión, **pero no destruye ninguna de las variables globales asociadas con la sesión, ni destruye la cookie de sesión.** Para destruir la sesión completamente 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 esto.
- No confundamos session_destroy() con session_unset(), que deshabilitará el registro de variables de sesión a través del array superglobal *$_SESSION*.

Más funciones de sesión: PHP: Funciones de sesión - Manual

Algunas directivas relevantes para trabajar con sesiones:

`session.use_cookies`	Indica si se deben usar cookies (1) o propagación en la URL (Uniform Resource Locator) (0) para almacenar el SID (Session Identifier).
`session.use_only_cookies`	Especifica si el módulo **solo** usará cookies para almacenar el id de sesión en la parte del cliente. Habilitar este ajuste ayuda a prevenir ataques que implican pasar el id de sesión en la URL. Por defecto es `1` (habilitado).
`session.save_handler`	Se utiliza para indicar a PHP (Hypertext Preprocessor) cómo debe almacenar los datos de la sesión del usuario. Existen cuatro opciones: en ficheros (`files`), en memoria (`mm`), en una base de datos SQLite (`sqlite`) o utilizando para ello funciones que debe definir el programador (`user`). El valor por defecto (`files`) funcionará sin problemas en la mayoría de los casos.
`session.name`	Determina el nombre de la cookie que se utilizará para guardar el SID (Session Identifier). Su valor por defecto es `PHPSESSID`.
`session.auto_start`	Su valor por defecto es 0, y en este caso se deberá usar la función `session_start` para gestionar el inicio de las sesiones.
`session.cookie_lifetime`	Si se usa la URL (Uniform Resource Locator) para propagar el SID (Session Identifier), éste se perderá cuando se cierre el navegador. Sin embargo, si se utilizan cookies, el SID (Session Identifier) se mantendrá mientras no se destruya la cookie. En su valor por defecto (0), las cookies se destruyen cuando se cierra el navegador. Si se desea que se mantenga el SID (Session Identifier) durante más tiempo, se debe indicar en esta directiva ese tiempo en segundos.

Más directivas de configuración asociadas a la sesión se pueden consultar aquí

5.1. Ejemplo básico con sesiones

Para ver un ejemplo sencillo de uso de sesiones, tenemos el código disponible en la URL:
https://github.com/dudwcs/UD5_Ejemplo_Basico_Sesiones.git

login.php contiene un formulario de login se realiza una autenticación contra un array en memoria con usuario y contraseña. Si las credenciales aportadas por el usuario son correctas, se considera al usuario autenticado y se almacenan en la sesión:

- El email del usuario: $_SESSION["user"] = $user;
- El último acceso del usuario: $_SESSION["ultimoAcceso"]=time(); /* La función time() devuelve el momento actual medido como el número de segundos desde la Época Unix (1 de Enero de 1970 00:00:00 GMT). */

Con la directiva session.auto_start=0, el array superglobal $_SESSION no se crea en memoria de forma automática hasta que se llama a session_start(). Por lo tanto, antes de almacenar datos en $_SESSION, habrá que llamar a session_start() previamente.

Al inicio de **login.php** y de **welcome.php,** se llama a la función isUserLoggedIn() que comprueba que $_SESSION["user"] existe. Por el mismo motivo, llama a session_start() con anterioridad.

Si se llama 2 veces seguidas a session_start() veremos un mensaje alertando de que la segunda llamada será ignorada:
Notice: session_start(): Ignoring session_start() because a session is already active in ...

Para evitar este mensaje, una posible solución es comprobar el estado de la sesión antes de llamar a session_start(), que es la aproximación utilizada en la función iniciarSession():

```
function iniciarSesion(): bool {
    $iniciada = true;
    if (session_status() !== PHP_SESSION_ACTIVE) {
        $iniciada = session_start();
    }
    return $iniciada;
}
```

Control de acceso

- Si un usuario está autenticado (si existe $_SESSION["user"]) y pretende entrar en **login.php,** se le redirige con la función header() **a welcome.php**

```
 if(isUserLoggedIn()){
        header("Location: welcome.php");
        exit;
    }
```

- Si un usuario no autenticado intenta acceder a **welcome.php,** es redirigido a **login.php**

```
if (!isUserLoggedIn()) {
    header("Location: login.php");
    exit;
}
```

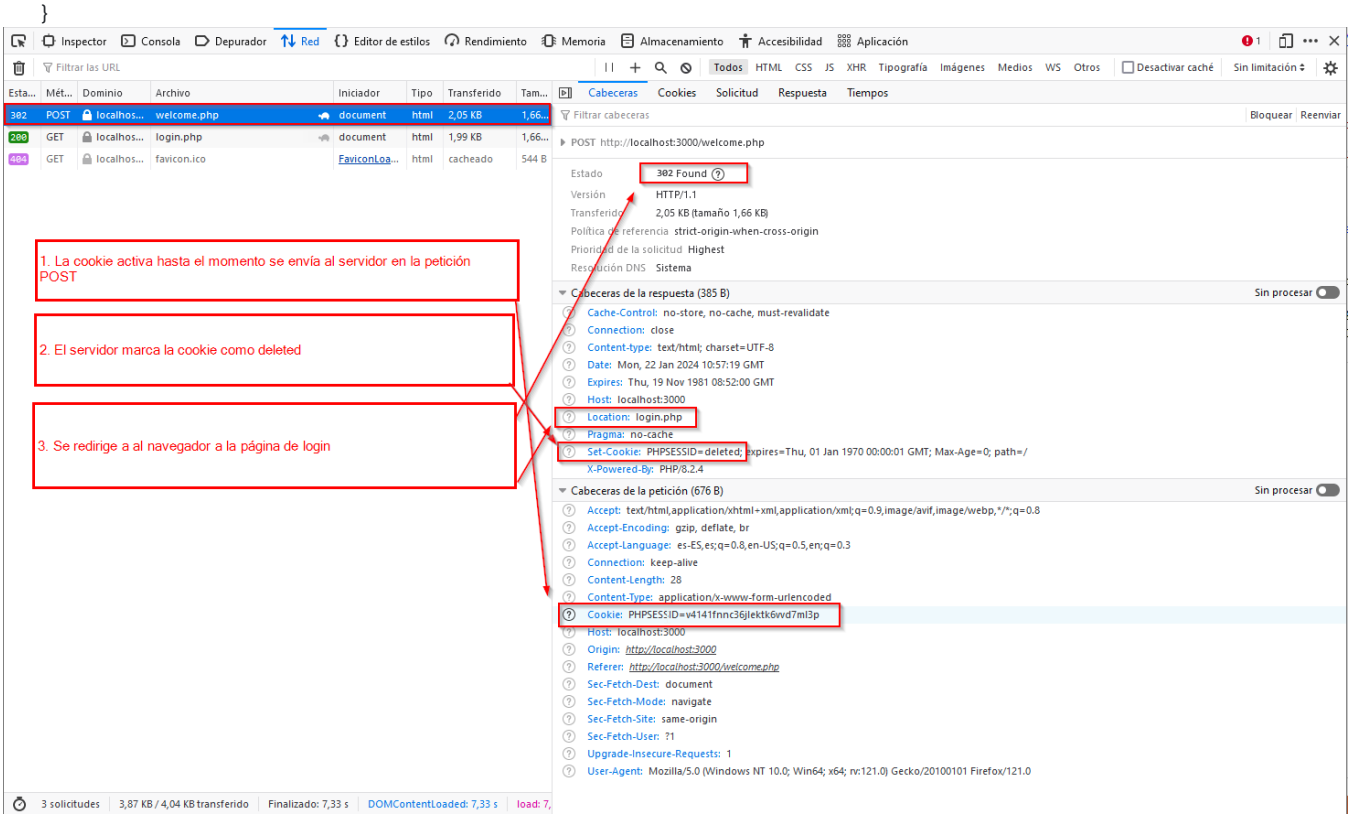
La función isUserLoggedIn() llama a iniciarSesion, que, si no está la sesión iniciada, llamará a session_start().

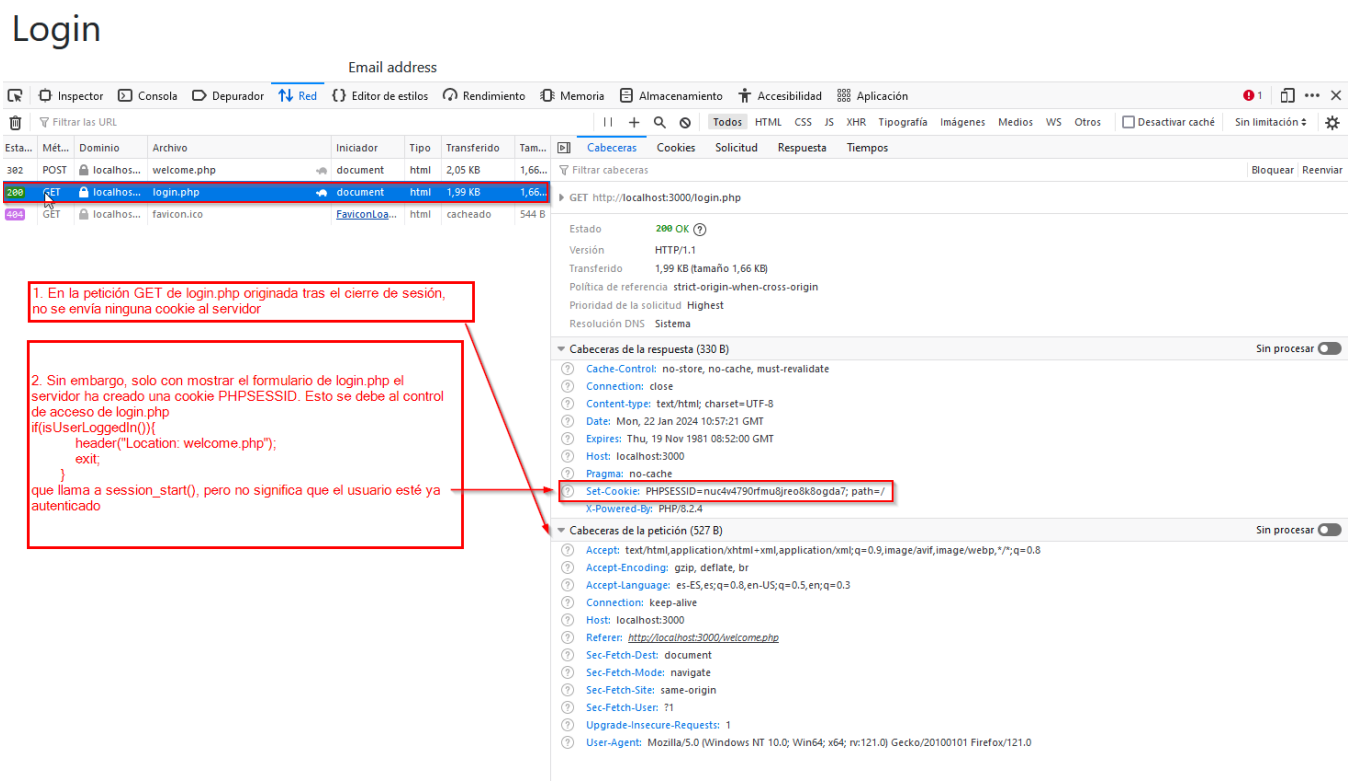
```
function isUserLoggedIn() {
    $autenticado = iniciarSesion() && (session_status() === PHP_SESSION_ACTIVE) && isset($_SESSION["user"]);
    return $autenticado;
}
```

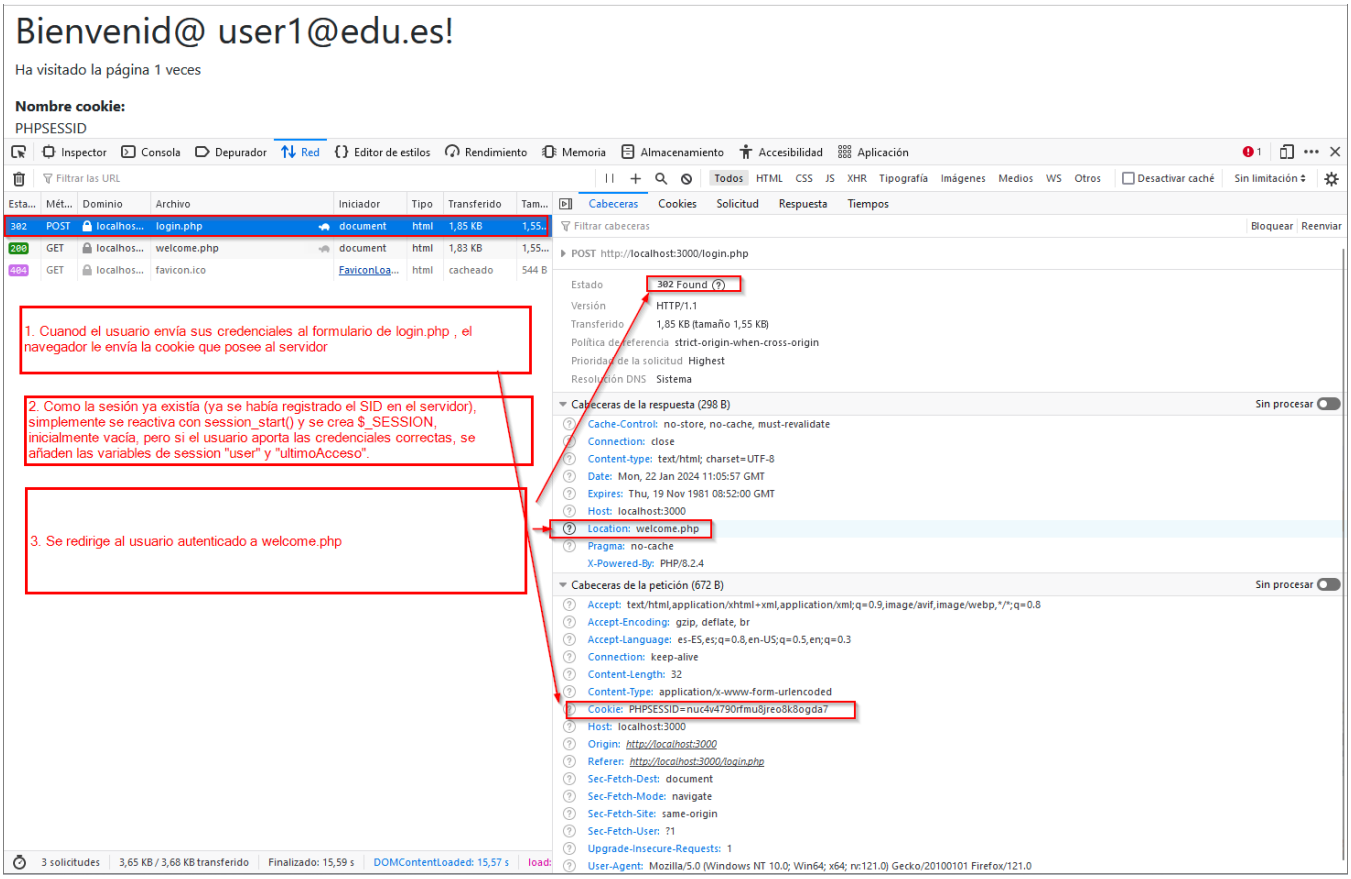
La llamada a session_start(), con la configuración por defecto del php.ini(session.use_only_cookies=1), si el navegador no le envía una cookie PHPSESSID, crea de forma transparente al desarrollador una cookie PHPSESSID.
El hecho de que exista la cookie no hace automáticamente que el servidor considere que el usuario esté autenticado, sino que ha interactuado previamente con el servidor. Para considerarlo autenticado, en este ejemplo, debe existir una variable en la sesión, en este ejemplo: $_SESSION["user"]

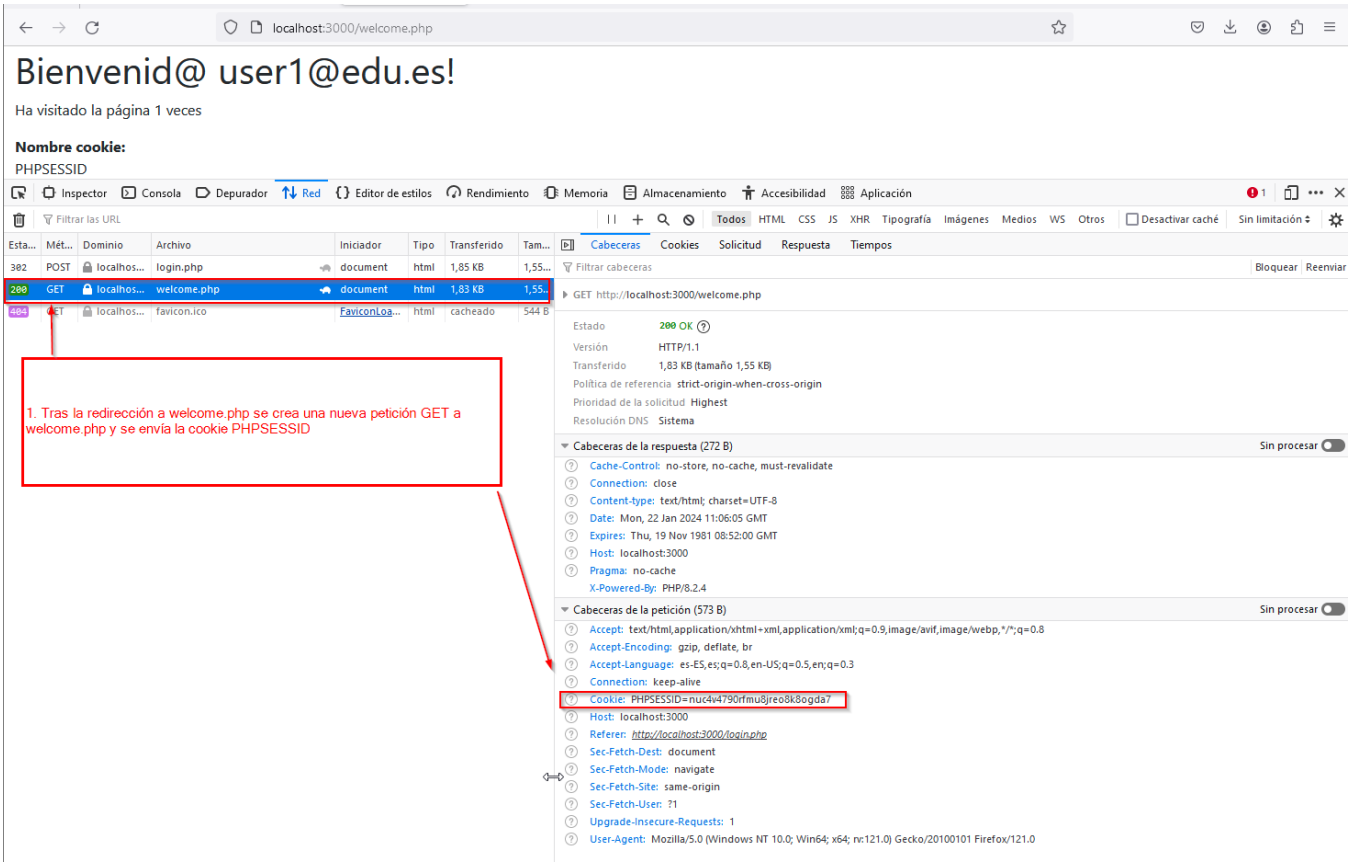
Esta interacción se ve claramente cuando se cierra sesión en **welcome.php**

```
if (isset($_POST["btnCerrar"])) {
        cerrarSesion();
        header("Location: login.php");
        exit;
    }
```






Bienvenid@ user1@edu.es!
Ha visitado la página 1 veces
Nombre cookie:
PHPSESSID
1. Cuanod el usuario envía sus credenciales al formulario de login.php , el navegador le envía la cookie que posee al servidor
2. Como la sesión ya existía (ya se había registrado el SID en el servidor), simplemente se reactiva con session_start() y se crea $_SESSION, inicialmente vacía, pero si el usuario aporta las credenciales correctas, se añaden las variables de session "user" y "ultimoAcceso".
3. Se redirige al usuario autenticado a welcome.php

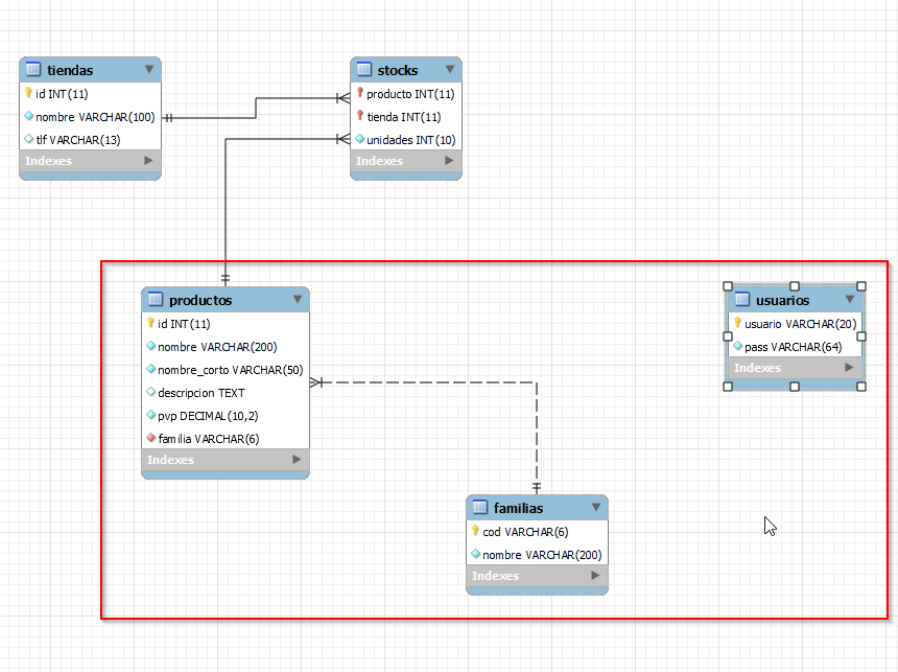

localhost:3000/welcome.php
Bienvenid@ user1@edu.es!
Ha visitado la página 1 veces
Nombre cookie:
PHPSESSID
1. Tras la redirección a welcome.php se crea una nueva petición GET a welcome.php y se envía la cookie PHPSESSID

5.2. Ejemplo de uso de sesiones con la cesta de la compra

Vamos a ver un ejemplo de uso de variables de sesión en https://github.com/dudwcs/UD5_Ejemplo_cesta_compra.git

Vamos a ejecutar los scripts que encontraréis en el propio repositorio en la carpeta scripts.

El script nº1 crea una nueva base de datos llamada **proyecto** con las siguientes tablas:



Se crea además un usuario para acceder a la base de datos llamado **gestor** con contraseña **secreto**.

En el ejemplo se trabaja solo con las tablas **productos y usuarios.**

El script nº 3, crea 2 usuarios en la **tabla usuarios**:

- **admin** con contraseña **secreto**
- **gestor** con contraseña **pass**

conexion.php

- Crea una conexión PDO y utiliza los datos de conexión a la base de datos proyecto con gestor *hardcoded* en el código (hemos visto que es una mala práctica y que deberían extraerse del código, por ejemplo, en un fichero .ini)
- Declara una función consultarProducto($id) que recupera todos los datos de la tabla productos filtrados por un determinado id
- Tiene un par de funciones para cerrar la conexión y el PDOStatement por referencia (para que afecte a las variables fuera de la propia función).

login.php

Permite comprobar si usuario y contraseña están en la tabla usuarios. Utiliza la función hash() para aplicar el algoritmo SHA256 en lugar de password_hash() y password_verify(). Si el proceso de autenticación es correcto se guarda en la sesión
$_SESSION['nombre'] = $nombre; donde $nombre es la columna usuario de usuarios y se redirige a listado.php
Si hay algún error, se guarda en la sesión un error. Este se borra, una vez mostrado.

listado.php

Realiza un control de acceso. Si el usuario no está autenticado se fuerza una redirección a login.php

Se obtienen todos los productos de la base de datos y se recuperan como objetos. Por cada objeto producto, se crea una fila en una tabla HTML con un formulario que tiene un input oculto con el id del producto y un <input> que permite añadir el producto a la cesta. El formulario será procesado por el propio fichero /listado.php que se puede obtener del array superglobal $_SERVER con la clave PHP_SELF: $_SERVER['PHP_SELF']
En la misma fila se muestran el nombre y el precio.
Cuando se añade un producto a la cesta se guarda en una variable de sesión 'cesta' un array que tiene tanto clave como valor el id del producto seleccionado.

En la parte superior se muestran:
- los elementos de la cesta
- el nombre del usuario autenticado
- un enlace para salir de la sesión

Sobre la tabla HTML se muestran
- un enlace a la cesta.php
- un formulario con un botón para vacíar la cesta

cesta.php

Realiza un control de acceso. Si el usuario no está autenticado se fuerza una redirección a login.php
Si la variable de sesión cesta no está vacía, consulta a la base de datos datos de cada producto por su id y crea un array asociativo con clave el id del producto y valor, un array con nombre y precio)
Muestra una lista desordenada con el nombre y precio del producto y el total de la cesta

En la parte superior se muestran:
- los elementos de la cesta
- el nombre del usuario autenticado
- un enlace para salir de la sesión

Se muestran un enlace a **pagar.php y a listado.php**

pagar.php

Realiza un control de acceso. Si el usuario no está autenticado se fuerza una redirección a login.php

Muestra un texto indicando que se ha pagado y un enlace a **listado.php**

cerrar.php

Se eliminan las variables cesta y user de la sesión y se fuerza la redirección a login.php