



# UD6 – Sesiones y seguridad

**2º CFGS**  
**Desarrollo de Aplicaciones Web**  
**2024-25**

# 1.- Introducción

Las aplicaciones web funcionan con el **protocolo HTTP**, este protocolo solo se encarga de encapsular la información para poder enviarla, por lo que **no se mantiene información de estado** tratando cada petición como una conexión independiente.

Además, **la ejecución de un script PHP se termina cuando se llega al final del archivo.**

**Por esta razón, a priori, el servidor web no reconoce que un mismo usuario está navegando por las diferentes páginas de la aplicación web.**

Este comportamiento es la gran diferencia entre las aplicaciones web y las aplicaciones de escritorio.

Una aplicación de escritorio se mantiene abierta hasta que el usuario decide cerrarla por lo que la información siempre está disponible.

# 1.- Introducción

Para solventar este inconveniente existen diferentes soluciones:

- **Cookies:** se almacenan en el navegador web (cliente).
- **Sesiones:** se almacenan en el servidor.

En las aplicaciones web se suelen utilizar conjuntamente las dos técnicas.

Gracias al uso de las cookies y de las sesiones **el cliente y el servidor pueden compartir información** que permite al **servidor saber que un cliente determinado está navegando por las diferentes páginas de la aplicación web.**

## 2.- Cookies

Las **cookies** son pequeños ficheros de texto que **las aplicaciones web** almacenan **en el cliente**.

Se **guardan en el navegador** y están **asociadas a una aplicación web**.

### **Funcionamiento de las cookies:**

- Cuando el cliente hace una petición de un recurso al servidor, en las cabeceras de la petición le envía al servidor las cookies asociadas a ese recurso (si las hay).
- Cuando el servidor envía un recurso al cliente le puede indicar al cliente con una cabecera que cree una cookie con una información concreta.

## 2.- Cookies

Las cookies tienen **dos funciones** generales:

- **Recordar el acceso** a la página web: navegación, datos introducidos...
- **Conocer hábitos de navegación.**

La primera función es indispensable para el buen funcionamiento de las aplicaciones web.

La segunda es un quebradero de cabeza para la privacidad de los usuarios.

## 3.- RGPD

### Reglamento General de Protección de Datos

Las aplicaciones web generalmente se desarrollan para el su uso público y se deben seguir las diferentes leyes y normativas con respecto a ello.

La principal ley en España es la **LSSI**: Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico.

La **LSSI no es la única norma en vigor**, además, las leyes se actualizan y salen normativas nuevas, en el siguiente enlace se pueden consultar: <http://www.lssi.gob.es/Paginas/index.aspx>.

De entre toda la normativa uno de los puntos más importantes es el **uso de cookies**.

Guía sobre el uso de Cookies de la Agencia Española de Protección de datos:

[https://www.interior.gob.es/opencms/export/sites/default/.content/Documentacion/Guia\\_Cookies.pdf](https://www.interior.gob.es/opencms/export/sites/default/.content/Documentacion/Guia_Cookies.pdf)

## 3.- RGPD

La actual ley europea de cookies distingue entre cookies necesarias y no necesarias:

- **Cookies técnicamente necesarias:**

El almacenamiento de datos necesario incluye las cookies que son clave para el funcionamiento de una web. Esto significa, por ejemplo, guardar los datos de inicio de sesión, la cesta de la compra o la selección del idioma mediante las llamadas cookies de sesión (que se borran cuando se cierra el navegador).

- **Cookies técnicamente no necesarias:**

Como integrantes de este grupo se consideran los archivos de texto que no solo no sirven para la funcionalidad de la página web, sino que recogen otros datos:

- Cookies de seguimiento, que recogen datos sobre los usuarios como, por ejemplo, su ubicación
- Cookies de segmentación, que adaptan los anuncios a los usuarios de Internet
- Cookies de análisis, que aportan información sobre el comportamiento de los usuarios en la web
- Cookies de redes sociales, que vinculan una web con plataformas como Facebook, Twitter, etc.

[fuelle](#)

## 3.- RGPD

En esta unidad se verá como implementar el uso de cookies en la aplicación web.

Así que las cookies que se crearán serán de las llamadas "técnicamente necesarias", en principio para el uso de estas cookies no es necesario el consentimiento del usuario.

Como en la actualidad la protección de datos personales es un asunto muy importante es responsabilidad del desarrollador el conocer la situación actual de la normativa tanto estatal como europea en relación a las cookies para saber si se debe informar al usuario o no.

## 4.- Creación de Cookies en PHP

Para que el **servidor** le indique al cliente (navegador) que **cree una cookie** se utiliza la función **setcookie**. ([documentación](#))

```
setcookie('name', 'value', time()+3600);
```

Los parámetros que admite la función son:

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

# 4.- Creación de Cookies en PHP

Los parámetros que admite la función son:

Parámetro	Función del parámetro
\$name	Nombre de la cookie.
\$value	Valor de la cookie.
\$expires_or_options	Cuándo caduca (se borra) la cookie. Es una marca de tiempo Unix (segundos desde 1-1-1970). Si es 0 se borrará al cerrar la pestaña en el navegador. Ejemplo 30 días: time()+60*60*24*30
\$path	Ruta dentro del dominio donde estará disponible la cookie. Si es / estará disponible en todo el dominio. Si es / <b>cuenta</b> estará disponible solo en el directorio "cuenta" dentro del dominio.
\$domain	Domino o subdominios donde estará disponible la cookie. Ejemplos: www.miweb.com → disponible en www.miweb.com y subdominios como w2.www.miweb.com miweb.com → disponible en miweb.com y subdominios como www.miweb.com o cuenta.miweb.com
\$secure	Indica si la cookie solo debe enviarse bajo conexiones seguras HTTPS desde el cliente
\$httponly	Indica que la cookie solo se envíe a través del protocolo HTTP. Si es true, la cookie no será accesible desde lenguajes de scripting como JavaScript. Esto ayuda a reducir el robo de identidad mediante ataques XSS (Cross-Site Scripting).

## 4.- Creación de Cookies en PHP

Como hoy en día **se prioriza la seguridad** y **todas las aplicaciones deberían transmitirse mediante conexiones seguras HTTPS** una buena creación de cookie podría ser:

```
setcookie('name', 'value', time()+60*60*24*30, secure: true, httponly: true);
```

Para los ejercicios de clase ya que el servidor no está configurado con certificados no se debe poner la opción **secure** a true:

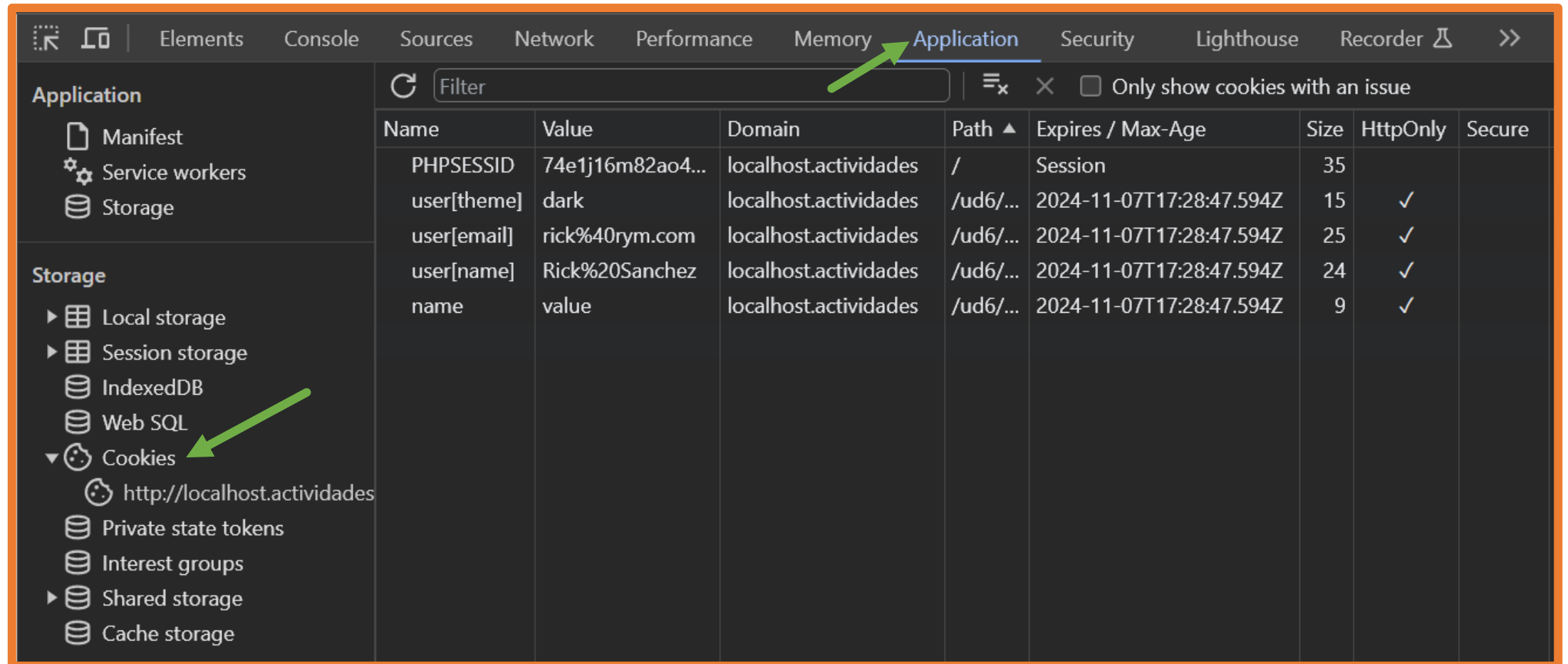
```
setcookie('name', 'value', time()+60*60*24*30, httponly: true);
```

Se puede crear un array dentro de una cookie:

```
setcookie('user[name]', 'Rick Sanchez', time()+60*60*24*30, httponly: true);  
setcookie('user[email]', 'rick@rym.com', time()+60*60*24*30, httponly: true);  
setcookie('user[theme]', 'dark', time()+60*60*24*30, httponly: true);
```

## 4.- Creación de Cookies en PHP

Desde el navegador, en el inspector, se puede ver la información de las cookies de la aplicación web:



The screenshot shows the Chrome DevTools Application tab. The left sidebar has a tree view with 'Storage' expanded and 'Cookies' selected, indicated by a green arrow. The main panel displays a table of cookies. A green arrow points to the 'Application' tab in the top navigation bar.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure
PHPSESSID	74e1j16m82ao4...	localhost.actividades	/	Session	35		
user[theme]	dark	localhost.actividades	/ud6/...	2024-11-07T17:28:47.594Z	15	✓	
user[email]	rick%40rym.com	localhost.actividades	/ud6/...	2024-11-07T17:28:47.594Z	25	✓	
user[name]	Rick%20Sanchez	localhost.actividades	/ud6/...	2024-11-07T17:28:47.594Z	24	✓	
name	value	localhost.actividades	/ud6/...	2024-11-07T17:28:47.594Z	9	✓	

## 4.- Creación de Cookies en PHP

### ¡IMPORTANTE!

- El servidor ejecuta el script PHP y genera el documento HTML.
- Cuando ya ha finalizado la ejecución del script PHP es cuando se tiene el documento HTML generado completo.
- En ese momento es cuando el servidor envía las cabeceras y el documento HTML al cliente.
- Cuando el cliente recibe las cabeceras y el documento HTML es cuando se debe crear la cookie y en ese momento el servidor ya finalizó la ejecución del script PHP.

Esta es la razón por la **que las cookies estarán disponibles en el servidor a partir de la siguiente petición o recarga de la página web.**

## 5.- Eliminación de Cookies en PHP

El parámetro **expires** indica el periodo de validez de una cookie.

Para eliminar una cookie se debe poner una marca de tiempo de una fecha ya pasada:

```
setcookie('name', expires_or_options: time()-1, httponly: true);
```

## 6.- Acceso a las Cookies en PHP

Desde PHP las cookies creadas se pueden consultar mediante el **array asociativo** superglobal **\$\_COOKIE**.

```
if (isset($_COOKIE['name'])) {  
    echo 'El nombre almacenado es:'. $_COOKIE['name'];  
}
```

El siguiente código permite recorrer todas las cookies:

```
foreach ($_COOKIE as $cookie => $value) {  
    echo $cookie .': '. $value .'  
<br>';  
}
```

```
// Si alguna de las cookies almacena un array  
foreach ($_COOKIE as $cookie => $value) {  
    if (is_array($value)) {  
        foreach ($value as $arrayKey => $arrayValue) {  
            echo $arrayKey .': '. $arrayValue .'  
<br>';  
        }  
    } else {  
        echo $cookie .': '. $value .'  
<br>';  
    }  
}
```

# 7.- Ejemplo de uso de Cookies en PHP

index.php

```
ud6 > pruebas > index.php
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Cookies</title>
7  </head>
8  <body>
9      <h1>Hola <?=$_COOKIE['name']??'persona invitada'?></h1>
10
11      <?php
12          if (isset($_COOKIE['name'])) {
13              echo '<a href="logout.php">Salir del sistema</a>';
14          } else {
15              echo '<a href="login.php">Accede al sistema</a>';
16          }
17      ?>
18  </body>
19  </html>
```



# 7.- Ejemplo de uso de Cookies en PHP

## login.php

```
ud6 > pruebas > login.php
1  <?php
2      if (!empty($_POST['name'])) {
3          setcookie('name', $_POST['name'], time()+69*69*24*30, httponly: true);
4          header('location:index.php');
5          exit;
6      }
7  ?>
8  <!DOCTYPE html>
9  <html lang="es">
10 <head>
11     <meta charset="UTF-8">
12     <meta name="viewport" content="width=device-width, initial-scale=1.0">
13     <title>Título de la página</title>
14 </head>
15 <body>
16     <h1>Introduce tu nombre</h1>
17
18     <form action="#" method="post">
19         <label for="name">Nombre:</label>
20         <input type="text" name="name" id="name">
21         <input type="submit" value="Enviar">
22     </form>
23 </body>
24 </html>
```

← → ↻ 🏠 ⚠ No es seguro | localhost.actividades/ud6/pruebas/login.php

## Introduce tu nombre

Nombre:

## 7.- Ejemplo de uso de Cookies en PHP

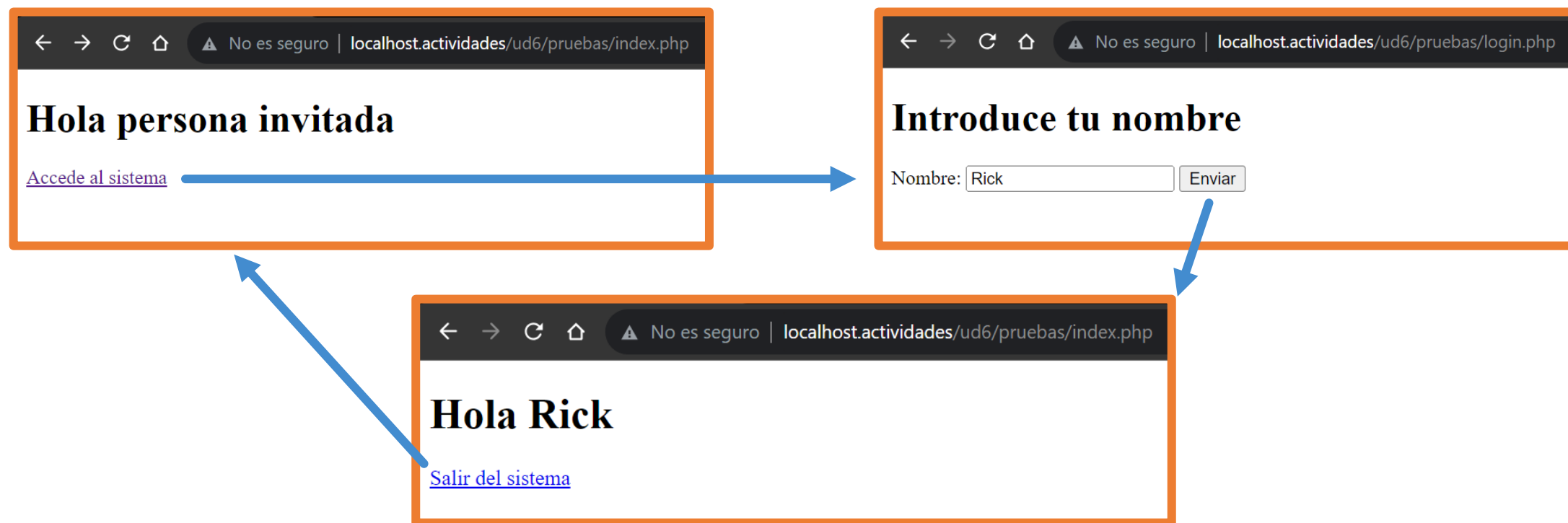
logout.php

```
ud6 > pruebas >  logout.php
You, 1 minute ago | 1 author (You)
1  <?php
2      setcookie('name', expires_or_options: time()-1, httponly: true);
3      header('location:index.php');
4
```

\*no es necesaria la instrucción **exit** debido a que no hay más instrucciones después de **header**.

## 7.- Ejemplo de uso de Cookies en PHP

Una vez que se rellene el formulario y se envíe, se creará la cookie y estará disponible en todo el dominio para ser utilizada.



# Práctica

## **Actividad 1:**

Triki: el monstruo de las Cookies.

## 8.- Sesiones en PHP

Se ha estudiado que mediante las **cookies** se puede almacenar información relativa al usuario y sus preferencias.

Las cookies son muy útiles pero conllevan una serie de inconvenientes:

- Cantidad de cookies que puede almacenar el navegador.
- Tamaño máximo de las cookies.
- Posible robo de identidad.
- Las cookies se almacenan en el cliente.
- Tráfico generado al enviar las cookies.

Para solventar estos inconvenientes habitualmente se usan las **sesiones** en el servidor.

En la programación en el lado del cliente, como solución a los problemas de las cookies, hace unos años se implementaron **Local Storage** y **Session Storage**.

## 8.- Sesiones en PHP

Al trabajar con sesiones **cada ventana de usuario del navegador que accede a la aplicación web tiene su propia sesión.**

Cada sesión tiene un identificador de sesión **SID**.

El **servidor almacena el SID** de una petición determinada y **lo asocia a todas las peticiones que se realicen desde la misma ventana del navegador.**

El SID se almacena de manera automática en el servidor generalmente en archivos aunque existen otros mecanismos como en bases datos.

## 8.- Sesiones en PHP

### ¿Y cómo sabe el cliente cuál es su sesión?

Como el SID identifica al cliente, es un dato que el cliente tiene que conocer para indicarle al servidor cuál es su sesión.

Se pueden utilizar dos métodos que están automatizados en PHP:

- **Propagar el SID en la URL**
- **Mediante cookies**

## 8.- Sesiones en PHP

### Propagar el SID en la URL

El SID se genera en el servidor pero se debe añadir a todas las URL que se usen en la aplicación web.

`http://localhost/index.php?PHPSESSID=4vjekic8fl7sqr0np45nfdrl6p`

Añadir el SID se puede hacer de manera manual o dejar que se encargue PHP activando la opción **`session.use_trans_sid`**.

Es una buena opción cuando el cliente no acepta el uso de cookies.

Inconvenientes:

- No se puede mantener el SID entre sesiones.

- Al compartir una URL se comparte el SID.

## 8.- Sesiones en PHP

### Mediante cookies

Al iniciar la sesión el servidor genera el SID y crea una cookie en el cliente para que se almacene.

Como en la cabecera de cada petición al servidor se envían todas las cookies, el servidor sabe cuál es la sesión del cliente.

La gestión del SID se realiza de manera automática por el servidor.

Es la opción por defecto de PHP.

Inconvenientes:

- Si el cliente no admite el uso de cookies este sistema no funciona.

- Si se accede al navegador se pueden robar las cookies.

## 8.- Sesiones en PHP

Tanto la propagación por URL como el uso de cookies para el control de sesiones tienen inconvenientes.

**El método más seguro es el uso de cookies.**

Por esa razón la configuración de PHP por defecto es con cookies.

En este curso se usará la configuración por defecto: cookies.

## 8.- Sesiones en PHP

Cuando se utilizan las sesiones, Apache se encarga de gestionarlas de manera automática.

Si no se indica lo contrario una sesión se borrará cuando se alcance el tiempo de vida de la sesión que por defecto son **180 minutos** que es el tiempo de vida por defecto de la cookie de sesión.

Esto significa que pasado ese tiempo se borrará la cookie con el SID y en la siguiente petición al no enviar la cookie, el servidor interpretará que es una sesión diferente.

# 9.- Configurar la sesión

PHP incorpora el soporte de sesiones y por defecto está activado para poder usarlas.

Mediante la función **phpinfo()** se puede consultar la configuración de las sesiones.

En el archivo **php.ini** se puede cambiar la configuración.

session		
Session Support	enabled	
Registered save handlers	files user	
Registered serializer handlers	php_serialize php php_binary	
Directive	Local Value	Master Value
session.auto_start	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	no value	no value
session.cookie_httponly	Off	Off
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_samesite	no value	no value
session.cookie_secure	Off	Off
session.gc_divisor	1000	1000
session.gc_maxlifetime	36000	36000
session.gc_probability	1	1
session.lazy_write	On	On
session.name	PHPSESSID	PHPSESSID
session.referer_check	no value	no value
session.save_handler	files	files
session.save_path	C:/laragon/tmp	C:/laragon/tmp
session.serialize_handler	php	php
session.sid_bits_per_character	5	5
session.sid_length	26	26
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On
session.upload_progress.freq	1%	1%
session.upload_progress.min_freq	1	1
session.upload_progress.name	PHP_SESSION_UPLOAD_PROGRESS	PHP_SESSION_UPLOAD_PROGRESS
session.upload_progress.prefix	upload_progress_	upload_progress_
session.use_cookies	On	On
session.use_only_cookies	On	On
session.use_strict_mode	Off	Off
session.use_trans_sid	Off	Off

## 9.- Configurar la sesión

Si no se tiene acceso al archivo **php.ini** se configura la sesión en tiempo de ejecución desde los scripts **PHP** mediante la función **ini\_set**.

Los cambios en la configuración se tienen que realizar antes de indicar al servidor que inicie la sesión.

En la imagen se muestra un ejemplo de uso de **ini\_set**:

```
ini_set('session.cookie_httponly', 1);  
ini_set('session.cookie_secure', 1);
```

En la documentación se pueden consultar todas las variables que se pueden configurar :  
<https://www.php.net/manual/es/session.configuration.php>

## 9.- Configurar la sesión

Para más seguridad se deben configurar las siguientes opciones con el valor 1:

**session.cookie\_httponly** → evita que JavaScript acceda a la cookie.

**session.cookie\_secure** → solo se envían las cookies a través de https.

**session.cookie\_lifetime** → tiempo de vida de la cookie de sesión.

**session.name** → nombre de la sesión (visible en la cookie).

```
ini_set('session.cookie_httponly', 1);  
ini_set('session.cookie_secure', 1);  
ini_set('session.cookie_lifetime', 300); // 300 segundos = 5 minutos  
ini_set('session.name', 'miSesion');
```

\*en clase al estar en un entorno seguro no es necesaria esta configuración (a no ser que se indique lo contrario), pero en un entorno de producción sí.

## 9.- Configurar la sesión

### `session.cookie_lifetime`

Si se configura un tiempo de vida para la cookie de sesión **demasiado grande** se aumenta la posibilidad de un robo de sesión si alguien accede al dispositivo.

Si se configura un tiempo de vida para la cookie de sesión **demasiado corto** la experiencia de usuario se verá afectada ya que podría interrumpir su actividad en la aplicación.

En caso de configurar un tiempo de vida corto se puede ir actualizando la cookie de sesión mientras el usuario esté navegando por la aplicación.

## 10.- Uso de sesiones en PHP

Con la configuración por defecto de PHP para iniciar una sesión hay que indicarlo con la siguiente instrucción **al principio de cada script** que se quiere que pertenezca a la sesión:

```
session_start();
```

Esta instrucción debe incluirse en todos los scripts PHP que se quiera que pertenezcan a la sesión.

## 10.- Uso de sesiones en PHP

Si se modifica la configuración por defecto en tiempo de ejecución:

```
ini_set('session.cookie_httponly', 1);  
ini_set('session.cookie_secure', 1);  
ini_set('session.cookie_lifetime', 300);  
ini_set('session.name', 'miSesion');  
session_start();
```

Estas instrucciones deben incluirse en todos los scripts PHP que se quiera que pertenezcan a la sesión.

## 10.- Uso de sesiones en PHP

Como el uso de sesiones requiere del uso de cookies, y las cookies se envían en las cabeceras, la instrucción **session\_start()** debe realizarse antes de que se genere cualquier carácter HTML (a continuación de phpDoc).

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cookies</title>
</head>
<body>
```

## 10.- Uso de sesiones en PHP

Mientras una sesión se encuentre abierta se puede usar la variable superglobal **\$\_SESSION** para guardar en ella información relativa a la sesión.

La variable **\$\_SESSION** es un **array asociativo**.

En todos los scripts donde se haya iniciado la sesión mediante la instrucción **session\_start()** estarán disponibles los elementos almacenados en el array asociativo **\$\_SESSION**.

# 10.- Uso de sesiones en PHP

## Ejemplo 1:

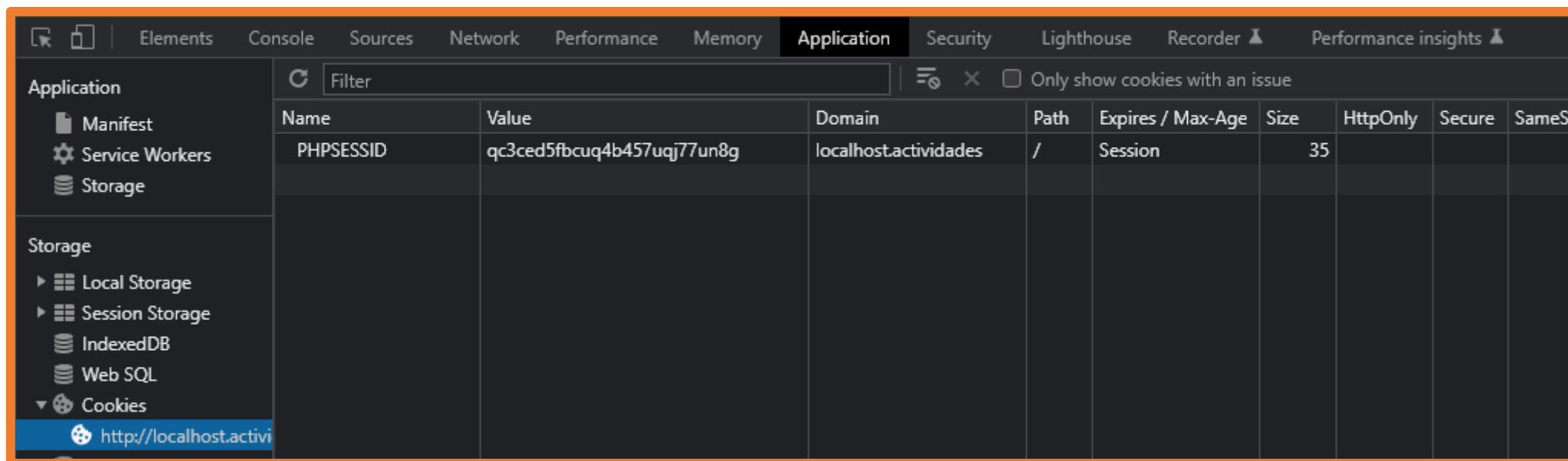
```
<?php
    // Se inicia la sesión o se recupera la anterior sesión existente
    session_start();
    // Se comprueba si la variable de sesión ya existe
    if (isset($_SESSION['visits']))
    |     $_SESSION['visits']++;
    else
    |     $_SESSION['visits'] = 1;
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cookies</title>
</head>
<body>
    |     Has visitado esta página <?=$_SESSION['visits']??0?> veces.
</body>
</html>
```

# 10.- Uso de sesiones en PHP

## Ejemplo 1:

En la configuración del navegador se puede observar que se ha creado la cookie con el SID de la sesión.



# 10.- Uso de sesiones en PHP

## Ejemplo 2:

```
<?php
// Se inicia la sesión o se recupera la anterior sesión existente
session_start();
// En cada visita se añade un valor al array 'visits'
$_SESSION['visits'][] = date('G:i:s D j M Y');

?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo visitas</title>
</head>
<body>
    Has visitado esta página <?=count($_SESSION['visits'])?> veces:
    <ul>
        <?php
        foreach ($_SESSION['visits'] as $visit) {
            echo '<li>'. $visit .'</li>';
        }
        ?>
    </ul>
</body>
</html>
```

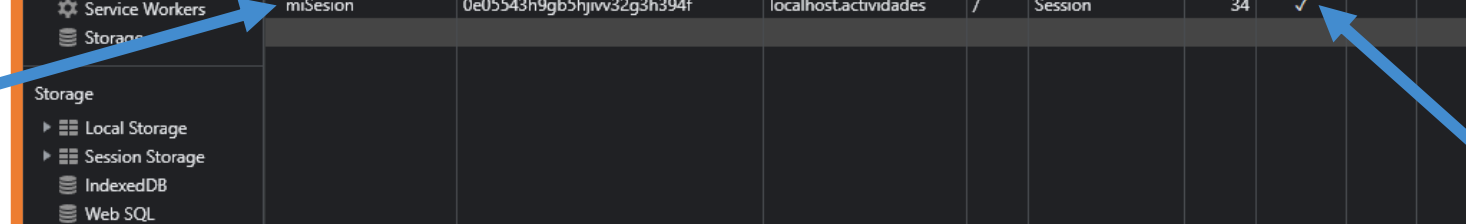
Has visitado esta página 5 veces:

- 12:46:29 Thu 29 Sep 2022
- 15:08:16 Sat 1 Oct 2022
- 09:21:37 Sun 2 Oct 2022
- 22:13:54 Sun 9 Oct 2022
- 18:37:07 Tue 11 Oct 2022

# 10.- Uso de sesiones en PHP

## Ejemplo 3:

```
<?php
// Se cambia La configuración por defecto (php.ini) de las sesiones
ini_set('session.name', 'miSesion');
ini_set('session.cookie_httponly', 1);
// Se inicia la sesión o se recupera la anterior sesión existente
session_start();
```



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameS
miSesion	0e05543h9gb5hjvw32g3h394f	localhost.actividades	/	Session	34	✓		

## 10.- Uso de sesiones en PHP

Si se necesita se puede borrar una sesión en un momento determinado, por ejemplo si se utilizan las sesiones para almacenar la información de **autenticación** (login) y el usuario decide **desloguearse**.

Para eliminar todas las variables creadas en la sesión pero mantener el identificador de sesión se utiliza la siguiente función:

```
session_unset();
```

Para eliminar completamente la sesión se utiliza la función:

```
session_destroy();
```

Para poder ejecutar estas instrucciones **primero** se debe iniciar la sesión con **session\_start()**.

## 10.- Uso de sesiones en PHP

Otra medida de seguridad consiste en regenerar el id de la sesión.

```
session_regenerate_id();
```

Regenerar el id de la sesión puede evitar el robo de la sesión mediante su id desde la cookie.

Existen más funciones para trabajar con sesiones que se pueden consultar en la [documentación](#).

# Práctica

## **Actividad 2:** Friki Shop.

# 11.- Autenticación de usuarios y control de acceso

Como ya se ha estudiado, la mayoría de las aplicaciones web actuales son **dinámicas**.

Además, casi todas las aplicaciones web permiten el **registro de usuarios** para así poder mostrar información personalizada.

Otra de las ventajas del **acceso autenticado** a la aplicación web es que se pueden **diferenciar los contenidos según los roles de los usuarios**.

Un comportamiento típico es que existan los roles **administrador** y **usuario**, siendo el administrador el que añade información a la aplicación web.

La sección solo accesible para el administrador se le denomina **back-office** y a las secciones públicas para los usuarios se le denomina **front-end**.

# 11.- Autenticación de usuarios y control de acceso

Se pueden usar técnicas como certificados electrónicos de los usuarios para la autenticación de los usuarios. por ejemplo el DNI electrónico. Este método de autenticación es conocido como **algo que tiene** el usuario.

Para implementar estos sistemas hay que tener un alto conocimiento de cómo funcionan los certificados y sus contenedores y además los usuarios en ocasiones deben disponer del hardware adecuado (lector de tarjetas identificadoras).

Por esto es más habitual usar un método conocido como **algo que sabe** el usuario, es el conjunto **identificador de usuario y contraseña**.

Para que un sistema de acceso sea efectivo se tendrían que realizar las conexiones http mediante protocolo seguro: **https** (estos contenidos se estudian en el módulo DAW).

## 12.- Autenticación por HTTP

Existe un método de control de acceso muy sencillo de implementar que utiliza un archivo donde se almacenan las credenciales de los usuarios.

En Apache se usa el archivo **htpasswd**.

El navegador mostrará una ventana (tipo Alert de JavaScript) para introducir las credenciales, si se introducen una credenciales erróneas el servidor responde con el código **401**: "Acceso no autorizado".

Este método no se va a estudiar en este módulo ya que bloquea el navegador dejando la pantalla en blanco lo cuál es una pésima experiencia para el usuario.

## 13.- Autenticación por PHP y BBDD

Existe una solución mejor y más portable que consiste en almacenar las credenciales en una **base de datos**.

Se pueden aislar las credenciales en una base de datos separada a la base de datos que con el contenido de la aplicación o bien dentro de la misma base de datos de la aplicación web en una tabla.

En caso de que las credenciales se encuentren en la misma base de datos de la aplicación web se suele optar por crear un usuario específico en la base de datos que tenga permisos sobre la tabla de credenciales.

## 14.- Encriptación de contraseñas en PHP

Las credenciales de acceso a la aplicación web se deben almacenar **encriptadas** en la base de datos.

Lo más habitual es **encriptar solo la contraseña** debido a que el resto de valores que pueden entrar en la autenticación (nombre usuario, mail...) se suelen utilizar en la aplicación.

Por ejemplo, si el nombre de usuario se almacenara encriptado no se podría mostrar en la aplicación web.

## 14.- Encriptación de contraseñas en PHP

Para encriptar contraseñas se usan funciones **hash**.

Estas funciones utilizan un algoritmo matemático para **convertir** un conjunto de datos **en un código alfanumérico de longitud fija**.

Por ejemplo, si una función hash determinada genera como salida un código de longitud 20, independientemente de la longitud de los datos que se encripten el resultado siempre será una cadena de longitud 20.

Una misma función hash, para dos datos de entrada diferentes nunca devuelven el mismo código de salida.

## 14.- Encriptación de contraseñas en PHP

Los algoritmos de encriptación evolucionan constantemente para mejorar la seguridad.

PHP recomienda la función **password\_hash** para encriptar contraseñas.

```
password_hash(string $password, string|int|null $algo, array $options = []): string
```

- **\$password**: string con la contraseña a encriptar.
- **\$algo**: algoritmo que se usará para encriptar.
- **\$options**: array con las opciones para el algoritmo de encriptación.

## 14.- Encriptación de contraseñas en PHP

Como algoritmo de encriptación se recomienda usar la constante **PASSWORD\_DEFAULT**, de esta manera será el propio PHP el que seleccione el mejor algoritmo de encriptación e entre los disponibles.

Actualmente el algoritmo de encriptación por defecto de PHP es **bcrypt**.

Con el algoritmo bcrypt el hash generado tiene una longitud de **60 caracteres** así que en la **base de datos** ese campo al menos debe tener esa longitud.

Se recomienda que en la base de datos la longitud del campo para almacenar el hash sea de **255 caracteres** para evitar problemas futuros si se mejoran los algoritmos.

# 14.- Encriptación de contraseñas en PHP

Ejemplo de uso de `password_hash`:

```
$password = 'Mi_Contraseña2022';  
$encryptedPassword = password_hash($password, PASSWORD_DEFAULT);
```

El valor de `$encryptedPassword` será el que se guarde en la base de datos.

El algoritmo **bcrypt** admite la opción **coste**, que es el tiempo de procesamiento necesario para calcular el hash, a mayor coste más difícil se vuelve descifrar dicho hash, por defecto tiene el valor 10.

Con el siguiente código se puede calcular el coste óptimo para un tiempo objetivo:

```
$targetTime = 0.05; // 50 milisegundos  
$cost = 8;  
do {  
    $cost++;  
    $startTime = microtime(true);  
    password_hash("test", PASSWORD_DEFAULT, ["cost" => $cost]);  
    $endTime = microtime(true);  
} while (($endTime - $startTime) < $targetTime);  
echo 'Valor apropiado para el coste: ' . $cost;
```

Valor apropiado para el coste: 10

## 14.- Encriptación de contraseñas en PHP

Si se dispone de hardware potente se puede aumentar el valor por defecto del coste.

Se puede calcular el coste para cada contraseña que se encripte.

Aunque es un proceso que consume recursos no es una tarea que se realice demasiadas veces.

También se puede calcular el coste durante el desarrollo de la aplicación y usar el valor resultante siempre.

Si se quiere calcular el coste para cada contraseña a encriptar, una buena técnica es crear una función.

```
function encryptPassword(String $password): String {  
    $targetTime = 0.05; // 50 milisegundos  
    $cost = 8;  
    do {  
        $cost++;  
        $startTime = microtime(true);  
        $encryptedPassword = password_hash($password, PASSWORD_DEFAULT, ["cost" => $cost]);  
        $endTime = microtime(true);  
    } while (($endTime - $startTime) < $targetTime);  
    return $encryptedPassword;  
}
```

## 14.- Encriptación de contraseñas en PHP

Para comprobar si una cadena de caracteres corresponde a un hash almacenado se usa la función **password\_verify**.

La función `password_verify` devuelve `true` si la cadena coincide con el hash y `false` en caso contrario.

```
if (password_verify($_POST['password'], $passwordInDB)) {  
    echo 'Login correcto';  
} else {  
    echo 'Login erroneo';  
}
```

## 15.- Diseño del sistema de registro y autenticación

En este momento se dispone de todos los elementos para la creación de un sistema de registro y login:

- **Formularios:** para el registro y el login.
- **Cookies:** para mantener la sesión mientras se está logueado.
- **Sesiones:** para mantener datos del usuario entre sesiones.

# 15.- Diseño del sistema de registro y autenticación

## Sesiones

En todas las páginas web de la aplicación se deberá **iniciar sesión** al principio del script.

Si no existe la variable de sesión con los datos del usuario es que no se ha hecho login.

Si existe la variable de sesión con los datos del usuario significa que se ha hecho login y el usuario está conectado.

Si el usuario está logueado se le podrá mostrar información personalizada.

## 15.- Diseño del sistema de registro y autenticación

### Registro

Se creará un **formulario** donde se piden los datos al usuario.

Si los datos cumplen las características requeridas se **almacenarán** en la base de datos, **encriptando la contraseña**.

# 15.- Diseño del sistema de registro y autenticación

## Login

Se creará un **formulario** donde se piden los datos de inicio de sesión.

Se verifica si el usuario y la contraseña coinciden con algún registro de la base de datos.

Si los datos son correctos se regenerará el id de la sesión y se creará una variable de sesión para almacenar los datos del usuario, principalmente el **nombre de usuario** y el **rol** del mismo en la aplicación.

```
session_regenerate_id();  
$_SESSION['name'] = $user;
```

# 15.- Diseño del sistema de registro y autenticación

## Logout

Se creará un enlace que dirija a un script que **destruya la sesión**.

De esta manera el usuario realizará el deslogueo.

```
session_start();  
session_destroy();  
header('location: /');  
exit;
```

# Práctica

## **Actividad 3:**

Friki Shop con registro y login.

# 15.- Diseño del sistema de registro y autenticación

## Autologin al entrar a la aplicación web

En ocasiones se ofrece la opción que permite que se realice el login de manera automática en la aplicación web:

- "Recordar durante 30 días"
- "Mantener conectado"
- ...

Para implementar esta funcionalidad **se suele utilizar un token** que se almacena en la base de datos.

Así, el sistema creado anteriormente tendrá unos pequeños cambios.

## 15.- Diseño del sistema de registro y autenticación

### Login

Si los datos de acceso son correctos, además de las acciones que ya se realizaban, deberá:

- **Crear un token** para el usuario.
- **Almacenar el token en una cookie.**
- **Almacenar el token en la bases de datos.**

# 15.- Diseño del sistema de registro y autenticación

## Creación del token

El token para la autenticación automática debe ser una cadena de caracteres aleatoria y única.

Mediante las siguientes funciones se consigue un token apropiado:

- **random\_bytes**: obtiene la cantidad de bytes aleatorios indicada como parámetro.
- **bin2hex**: convierte los bytes recibidos a representación hexadecimal.

La cadena obtenida tendrá una longitud del doble de caracteres que la cantidad indicada en la función random\_bytes.

```
$token = bin2hex(random_bytes(90));
```

## 15.- Diseño del sistema de registro y autenticación

### Inicio de sesión automático

En todas las páginas web de la aplicación:

**Si no existe la variable de sesión** del usuario y **existe la cookie con el token:**  
se comprobará si en la base de datos existe ese token, en caso de existir se obtendrán los datos del usuario y se creará la variable de sesión con ellos.

Si no existe la variable de sesión del usuario ni la cookie con el token es que es un usuario no logueado.

# 15.- Diseño del sistema de registro y autenticación

## Logout

Además de destruir la sesión también deberá:

Eliminar la cookie con el token.

Borrar el token del usuario en la base de datos.

# Práctica

## **Actividad voluntaria:**

Añadir autologin a Friki Shop.

## 16.- Recuperación de contraseñas

En todo sistema de registro y autenticación se debe dar la oportunidad al usuario de recuperar la contraseña por si la olvida.

El sistema más habitual consiste en:

- El usuario introduce su email y pulsa recuperar contraseña.
- Si el usuario existe se realizan las siguientes instrucciones:
  - El servidor genera un **token**.
  - Se almacena el token y el usuario/mail en una tabla en la base de datos que se puede llamar **passrecovery** donde el email y el token son únicos.
  - Se envía un email al usuario. El email contendrá un enlace tipo:  
[www.miweb.com/recuperar.php?token=TOKEN](http://www.miweb.com/recuperar.php?token=TOKEN)
- El usuario entra al enlace y si el token coincide con uno almacenado se le muestra el formulario para introducir una contraseña nueva.
- Se almacena la contraseña nueva en la base de datos encriptándola.

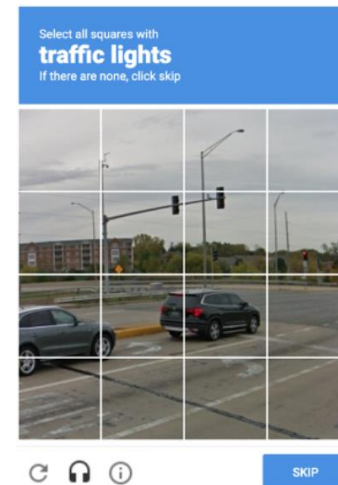
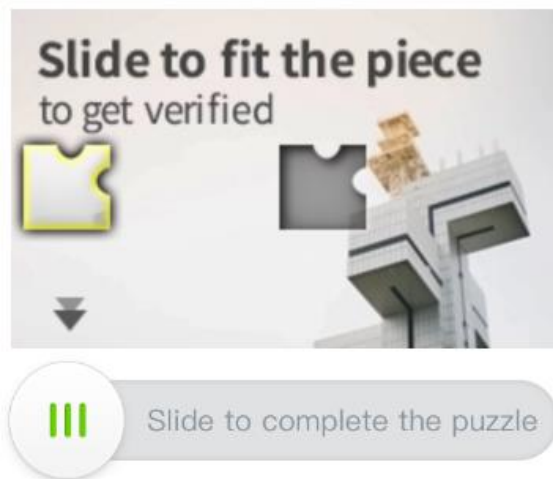
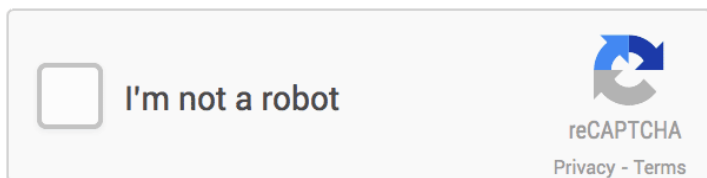
## 16.- Añadir seguridad al registro

**CAPTCHA** → Completely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part

Nacieron como la necesidad de evitar que los bots saturaran los servicios de internet.



Hoy en día los programas **OCR** (reconocimiento de texto) han mejorado gracias a las redes neuronales, los captcha han tenido que evolucionar también.



## 16.- Añadir seguridad al registro

Hoy en día si se necesita usar un sistema de CAPTCHA la mejor opción es utilizar alguno ya implementado.

Si se quiere desarrollar un CAPTCHA propio con PHP los pasos a seguir podrían ser los siguientes:

- Crear cadena aleatoria de caracteres.
- Almacenar la cadena en una variable de sesión.
- Crear lienzo con GD.
- Rellenar el lienzo con la cadena y otros elementos utilizando todas las técnicas de ofuscación posibles.
- Mostrar la imagen en el formulario junto a un campo input para introducir el texto.
- EN el script que recibe los datos del formulario comprobar que el input con el texto del CAPTCHA coincide con el valor almacenado en la variable de sesión.

## 16.- Añadir seguridad al registro

Para dificultar el reconocimiento de texto por parte de los bots se pueden usar diferentes técnicas como pueden ser:

- Escribir las letras sobre un lienzo con granulado.
- Crear líneas que atraviesen las letras.
- Cambiando el tamaño, ángulo, color, transparencia de las letras y líneas.
- Usando tipos de letra diferentes para cada letra.
- Usar imágenes de letras almacenadas que se encuentren distorsionadas con algún editor de imágenes.
- Usar la librería [Imagick](#) de PHP para aplicar distorsiones a las imágenes.
- Cualquier otro método que se te pueda ocurrir.

## 17.- Añadir seguridad al login

Para dificultar los ataques por fuerza bruta se puede limitar la cantidad de intentos de login erróneos.

Para implementar esto se puede crear una tabla en la base de datos donde se lleve la cuenta cada vez que se hace un login incorrecto.

Si se llega al límite no se debe permitir otro intento y se instará al usuario a recuperar la contraseña.

Cuando se hace login se debe reiniciar a cero la cuenta de intentos incorrectos.