ÍNDICE

CAPÍTULO 6. SISTEMAS DE AUTENTICACIÓN

6.1 Sistemas y procedimientos de autenticación.

Se conoce como *autenticación*, al proceso por el cual se determina si una persona o cosa es quien o lo que dice ser.

En ningún caso debe confundirse el término *autenticación* con el término *autentificación* ya que, aunque están relacionados, la *autenticación* es un proceso más completo y costoso.

6.1.1 Procedimiento de autenticación.

La autenticación se suele dividir a su vez en dos procesos:

- ***Identificación,*** la fase de identificación es aquella en la que el usuario o cliente aporta al sistema sus datos identificativos, o lo que es lo mismo, la fase en la que el cliente dice quien es.

- ***Autentificación***, el objetivo de esta segunda fase es verificar la identidad que aportó el cliente durante la fase de identificación. Para llevar a cabo la fase de autentificación se recurre a contrastar la identidad mediante datos que sólo pueda conocer el cliente (contraseñas), algo que posee el cliente (tarjetas de acceso) o características físicas únicas del cliente (sistemas biométricos).

6.1.2 Mecanismos de autentificación.

Tal y como se ha dejado entrever anteriormente, existen diversos mecanismo de autentificación de usuarios. Dichos mecanismos de autentificación pueden dividirse en:

- ***Basados en algo conocido,*** en este grupo se enmarca cualquier mecanismo de verificación de identidad que emplee

contraseñas que el usuario debe recordar. Sus principales inconvenientes son la posibilidad de que un usuario olvide la contraseña, y que estos escojan contraseñas excesivamente simples para no olvidarlas.

- ***Basados en pertenencias,*** este grupo lo componen aquellos sistemas de seguridad que recurren a tarjetas magnéticas o dispositivos electrónicos para verificar la identidad de usuario. Su principal inconveniente es la posibilidad de pérdida del dispositivo, especialmente si dicho dispositivo cumple tanto con la función de identificación como con la de autentificación. Además, la mayoría de estos dispositivos son replicables por lo que pueden realizarse duplicados sin que el usuario sea consciente de ello.

- ***Basados en características o habilidades únicas,*** conocidos como sistemas biométricos, pueden averiguar por si mismos la identidad del usuario. Podemos distinguir dos grupos distintos: basados en rasgos físicos y basados en patrones de comportamiento.

    - ***Basados en rasgos físicos,*** utilizan rasgos físicos que se consideran únicos para identificar al usuario. Entre estos rasgos, los más extendidos son: huellas dactilares, iris, retina, reconocimiento de la cara y geometría de la mano. Un proyecto futuro a este respecto es el análisis de *ADN*.

    - ***Basados en patrones de comportamiento,*** dentro de este grupo se engloban mecanismos como el análisis del habla o de firma, y se sustentan en las diferentes formas de entonación y pronunciación en el caso del habla, y los trazos y presión en el caso de la firma. Un caso atípico dentro de este grupo, es el reconocimiento del ritmo de uso del teclado.

6.1.3 Funcionamiento del mecanismo de autenticación.

A grandes rasgos, el proceso que comúnmente se sigue es el siguiente:

- El servidor solicita la información de logado al cliente.

- El cliente procede a rellenar la información, y ésta se envía al servidor mediante los mecanismos de comunicación establecidos.

- El servidor recibe la información, y la contrasta contra algún tipo de origen de datos.

- Llegados a este punto existen dos posibilidades:

- Si la información es válida, el servidor procede a marcar la sesión de algún modo indicando que se trata de un usuario válido y, tal vez, indicando los permisos y/o privilegios de los que goza.

- Si la información es inválida, el servidor notifica al cliente que los datos no son válidos, y comienza de nuevo el proceso de autenticación.

6.1.4 Debilidades y mejoras del proceso de autenticación.

Entre las debilidades más comunes que podemos encontrar, están las siguientes:

- La información se almacena en texto claro. Debido a este hecho, aquel que logre acceso al origen de datos podrá usurpar la identidad de cualquiera de los usuarios. Para evitar este problema, la solución consiste en encriptar las contraseñas en el origen de datos, teniendo en cuenta el nivel de seguridad adecuado.

- La información de logado se envía en texto claro del cliente al servidor. El problema de esta debilidad es que algún intruso puede interceptar la información enviada, y capturar la información de logado pudiendo, por tanto, usurpar la identidad del usuario. La mejor solución a este problema es la encriptación de la información, para lo que se pueden emplear mecanismos como *SSL*.

- Usurpación de la sesión desde el terminal de origen. Esta debilidad es inherente al funcionamiento de los protocolos utilizados en la web, y permite que un intruso pueda acceder físicamente al mismo terminal en el que se hallaba un usuario legítimo utiizando su misma identidad. Aunque la solución más eficaz es que los usuarios cierren sesión siempre que se alejen del terminal y cuando no requieran su uso, esta solución no es viable debido a que los usuarios pueden no seguir las normas establecidas. Teniendo esto en cuenta, la manera de minimizar este problema es temporizar las sesiones para lo que se pueden emplear dos temporizadores distintos. En primer lugar, se establece un tiempo máximo de sesión que limita el tiempo que puede pasar desde que el usuario se registra en el sistema hasta que puede realizar la última acción. En segundo lugar, se puede establecer un tiempo máximo de respuesta de manera que si el tiempo entre dos tareas es superior al establecido la sesión se cerrará automáticamente.

- Usurpación de la sesión mediante man-in-the-middle. Mediante esta técnica un intruso situado en algún punto estratégico entre el cliente y el servidor puede utilizar la información transferida en las comunicaciones para interceptar la comunicación y usurpar la identidad del cliente. El método más sencillo para robar la sesión consiste en averiguar el código de sesión, y emplearlo para continuar la comunicación. Entre las soluciones que podemos encontrar tenemos la encriptación de las comunicaciones, y en segundo lugar configurar el servidor de manera que tenga en cuenta los datos de la máquina cliente para verificar que las comunicaciones no han sido alteradas.

6.2 Mecanismos de autenticación con WAMP/XAMP.

6.2.1 Autentificación de Apache.

El servidor web *Apache* dispone de un mecanismo estándar para proteger zonas del sitio web.

Existen dos formas de proteger una zona, bien mediante las secciones de tipo "<Directory>" en el fichero principal de configuración, o bien utilizando ficheros de configuración por directorios, denominados ".htaccess".

Para utilizar estos últimos es necesario especificar en el fichero principal de configuración que se permite su uso. Para ello se emplea la directiva "*AllowOverride*" de la siguiente manera:

```
AllowOverride AuthConfig
```

En primer lugar necesitaremos crear un fichero de contraseñas. Este fichero contendrá la información de logado de los usuarios. Para crear el fichero deberemos ejecutar la aplicación "*htpasswd*" incluida con el servidor *Apache*.

Para crear el fichero de contraseñas junto con un usuario denominado *rlopez* cuya contraseña fuera *mypassword*, utilizaríamos el siguiente comando, asegurándonos de utilizar la opción "*-c*" para crear el fichero.

```
# htpasswd -c /usr/local/apache/passwd/passwords rlopez
New password: mypassword
Re-type new password: mypassword
Adding password for user rlopez
```

Después de haber creado el fichero, para añadir un segundo usuario ejecutaríamos el mismo comando sin marcar la opción "*-c*", tal y como se muestra a continuación:

```
# htpasswd /usr/local/apache/passwd/passwords pmartinez
New password: yourpassword
Re-type new password: yourpassword
Adding password for user pmartinez
```

Una vez hayamos creado el fichero de contraseñas, y esté ubicado en la ruta adecuada, procederemos a añadir las directivas que configuración que vinculen nuestro fichero de contraseñas con la seguridad de la zona escogida.

Como ya se ha indicado, estas directivas pueden incluirse tanto en una sección "*<directory>*" del fichero de configuración general, como en un fichero de configuración de directorio.

Las directivas a incluir serían las siguientes:

```
AuthType Basic
AuthName "Restricted Files"
AuthUserFile /usr/local/apache/passwd/passwords
Require valid-user
```

La directiva *AuthType* selecciona el método de autentificación que se empleará. Aunque el método más común es *Basic*, al usar este método la información de logado se envía sin encriptar por lo que resulta más sencillo capturarla. Por este motivo, puede ser preferible utilizar el método *Digest*, que resulta más seguro pese a que sólo los navegadores más recientes lo soportan.

La directiva *AuthName* establece el dominio a usar en la implementación. Sirve principalmente para diferenciar donde nos vamos a logar, y que contraseña debemos utilizar.

La directiva *AuthUserFile* sirve para seleccionar la ruta del archivo de contraseña que hemos creado mediante el comando *htpasswd*.

Finalmente, la directiva *Require* nos sirve para establecer que usuarios pueden acceder a la zona protegida. En nuestro caso podrá acceder cualquier usuario que disponga de información de logado en el fichero de contraseñas. Si sólo quisiéramos que pudiera logarse un usuario en concreto podríamos usar esta misma directiva de la siguiente manera:

```
AuthType Basic
AuthName "Restricted Files"
AuthUserFile /usr/local/apache/passwd/passwords
Require user rbowen
```

En último lugar, es importante recalcar que aunque este método es totalmente válido, el uso de ficheros planos no es recomendable en sistemas con un gran número de usuario, o un alto nivel de carga.

Se puede ampliar esta información en la web oficial de *Apache* [http://httpd.apache.org/docs/].

6.2.2 Autentificación *HTTP* con *PHP*.

En primer lugar, es necesario recalcar que el método de autenticación que se describirá en este apartado tan sólo es accesible si empleamos el servidor *Apache* ejecutando *PHP* como módulo.

La base de este mecanismo de autenticación es la fusión de *PHP* y el protocolo *HTTP*. Para ello es necesario utilizar el método *header* de *PHP*, que nos permite enviar cabeceras *HTTP*. Al enviar dichas cabeceras debemos tener en cuenta que, tal y como establece el protocolo *HTTP*, las cabeceras deben enviarse al principio de la comunicación por lo que no podremos enviar al cliente ni un sólo carácter antes de ellas.

Teniendo este hecho en cuenta, utilizaremos un script similar al siguiente para realizar la validación:

```
<?php
  if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header("WWW-Authenticate: Basic realm=\"My Realm\"");
    header("HTTP/1.0 401 Unauthorized");
    echo "Text to send if user hits Cancel button\n";
    exit;
  } else {
    echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
    echo "<p>You entered {$_SERVER['$PHP_AUTH_PW']} as your password.</p>";
  }
?>
```

El código anterior envía dos cabeceras *HTTP* que sirven para solicitar los datos de logado al cliente mediante un formulario de sistema. En caso de cancelar la operación, el script seguiría su curso pero si se introdujera los datos se volvería a acceder a la misma página rellenando los valores *$_SERVER['PHP_AUTH_USER']* y *$_SERVER['PHP_AUTH_PW']* con los valores aportados por el cliente.

Como puede observarse, una vez se hubieran introducido los datos, el script procedería a mostrar automáticamente los datos del usuario logado. Sin embargo, este no es el comportamiento esperado ya que por lo general querremos contrastar esta información contra algún origen de datos.

Es especialmente importante ser cuidadoso con el texto de las cabeceras ya que el protocolo *HTTP* y los servidores web pueden ser muy escrupulosos en cuanto a la sintaxis del mismo.

6.3 Planificación de un sistema de logado.

Un sistema de logado, o autenticación de usuarios, puede adquirir una gran complejidad, hasta el punto de tener que integrarse con sistemas

de vistas según el perfil del usuario, gestores de contenidos y demás módulos de una misma plataforma.

A continuación se muestran los conceptos fundamentales a tener en cuenta para implementar un sistema de autenticación de usuarios profesional.

6.3.1 Funcionalidades de comprobación.

Un sistema de logado deberá constar de dos funciones básicas: comprobación de credenciales, y comprobación de validez de la sesión.

La comprobación de credenciales deberá realizarse únicamente cuando el usuario introduce sus datos en el sistema para autenticarse en el mismo. Está función se encargaría de dadas las credenciales del usuario, por ejemplo login y contraseña, comprobar que dichos datos son válidos, y que el usuario está activo. En caso de que las comprobaciones fueran positivas se procedería a marcar la sesión como válida, aportando a la misma cualquier información que pudiera ser de interés como: nombre del usuario, roles del usuario o grupos a los que pertenece.

La segunda función se utilizaría en la totalidad de la zona privada de la aplicación, y tendría como objetivo comprobar que la sesión se hubiera validado previamente mediante la función anterior, y que en la actualidad siga siendo válida. En un sistema más completo, se comprobaría que la sesión tuviera permisos específicos para acceder a la página web que el usuario ha solicitado. En el caso de que no tuviera suficientes permisos para acceder se le desviaría al usuario a una página web predeterminada.

6.3.2 Clasificación de páginas según el acceso.

En toda aplicación de acceso restringido podemos clasificar sus páginas y funcionalidades en tres grupos:

- Zona pública o de acceso libre: se caracteriza porque cualquier usuario, esté o no validado, puede acceder a ella.

- Zona privada o restringida: se caracteriza porque solamente los usuarios validados pueden acceder a la información y funcionalidades contenidas en esta zona.

- Punto/s de entrada a la zona privada: se trata de aquellas páginas que, perteneciendo generalmente a la zona pública, sirven para autenticar las credenciales de un usuario, y permitirle la entrada a la zona restringida.

En sistemas de validación orientados a roles y/o perfiles de usuarios podemos disponer de distintas zonas privadas dependiendo del usuario y sus privilegios. De este modo se puede limitar el acceso a ciertas funcionalidades de la aplicación dependiendo del rol que desempeñe el usuario validado, permitiendo únicamente el acceso a aquellas funcionalidades que realmente requiera para desempeñar sus tareas.

6.3.3 Control de tiempos de la sesión: inactividad y caducidad.

Al igual que ocurre con los salvapantallas que al reanudar la actividad solicitan una contraseña, casi la totalidad de los sistemas y aplicaciones que se consideran seguros implementan un subsistema de control de inactividad.

El objetivo de estos controles de tiempo no es otro que evitar la usurpación de una sesión validada por parte de un falso usuario, mientras el autentico usuario se ha ausentado de su equipo informático.

Aunque los mecanismos de control de tiempos más extendidos son aquellos que actúan sobre el tiempo de inactividad, podemos encontrar dos controles de tiempo diferenciados:

- Control de inactividad de la sesión.
- Control de caducidad de la sesión.

Estos controles de tiempo deben de realizarse periódicamente, tal y como se explica en los siguientes puntos.

6.3.3.1 Control de inactividad.

Como se ha introducido anteriormente, se llama tiempo de inactividad a aquel que transcurre entre dos acciones de un mismo usuario. La forma más práctica de verlo es considerar el tiempo que ha transcurrido desde la última acción del usuario hasta el instante actual.

El principal objetivo de controlar el tiempo de inactividad consiste en evitar que un usuario malintencionado interactúe con nuestra aplicación en nombre de un usuario legítimo que se ha ausentado temporalmente. O lo que es lo mismo, evitar que si el usuario autenticado se aleja de su equipo informático otra persona aproveche para acceder a la aplicación en su nombre.

Para aplicar esta técnica podríamos plantearnos la posibilidad de implementar algún tipo de disparador que transcurrido un tiempo comprobara si se ha superado el máximo tiempo de inactividad permitido, tal y como hacen los salvapantallas.

Sin embargo, la técnica más práctica consiste en realizar una comprobación rutinaria cada vez que el usuario realiza una acción. Dicha comprobación consistiría en ver cuanto tiempo ha transcurrido desde la acción anterior, y en el caso de que se hubiera superado el tiempo máximo permitido cancelaríamos la sesión del usuario, y éste se vería obligado a volver a validarse en el sistema.

De este modo resulta relativamente sencillo convertir dicha técnica en un mecanismo rutinario permitiéndonos, en muchos sistemas, convertirlo en una herramienta automatizada.

6.3.3.2 Control de caducidad.

Se denomina tiempo de caducidad al intervalo máximo de tiempo que puede permanecer un usuario validado en el sistema. O expresado de otra manera, es el tiempo tras el que expulsaremos del sistema a un usuario, a contar desde el instante en que se validó en el sistema.

Aunque, de cara al usuario final, podríamos argumentar que el objetivo de este control de tiempo es el mismo que en el caso anterior y sería cierto, tiene otro objetivo que puede resultar de mayor importancia: la liberación de recursos.

Todo acceso al sistema conllevará un determinado consumo de recursos, que en la mayoría de los casos será relativamente pequeño. Sin embargo, el tiempo potencial de dicho consumo será infinito ya que el usuario puede mantener la sesión abierta indefinidamente. Dado que una plataforma puede tener un número de usuarios considerablemente grande, podría llegar a repercutir en el rendimiento del sistema.

En pro de limitar el consumo total de recursos por parte de los usuarios podemos optar por fijar un límite al tiempo de permanencia de los usuarios en la aplicación.

Podríamos imaginar un puesto de información de un museo en el que se nos muestra de forma interactiva información sobre todas las obras del museo. Para fomentar la rotación de los usuarios, ya que el recurso consumido sería el propio puesto de información, podríamos optar por cerrar la sesión del usuario transcurridos 5 minutos desde el inicio de forma que estaríamos “sugiriendo” al usuario a que liberara el equipo. En nuestro caso, se trataría de un mecanismo similar aunque menos agresivo, ya que emplearíamos periodos bastantes más amplios.

Al igual que en caso anterior, podríamos establecer algún tipo de disparador que se accionara transcurrido un tiempo desde el inicio de la sesión. Sin embargo, el mecanismo más sencillo consistiría en dos

acciones sencillas. La primera establecería una marca temporal en el momento de la autenticación del usuario en el sistema. La segunda acción, de forma análoga al tiempo de inactividad, evaluaría antes de cada acción si se ha superado el límite de tiempo de la sesión. En el momento que se hubiera superado dicho límite se cancelaría la acción del usuario, y se desviaría la sesión al punto de entrada al sistema con un mensaje de advertencia para que el usuario pudiera volver a entrar de forma sencilla.

6.4 PEAR:Auth.

Con el objetivo de desarrollar el sistema de autenticación, el proyecto *PEAR* nos proporciona el paquete *Auth*. El paquete *Auth* es básicamente un objeto, que encapsula un conjunto de métodos, que permiten la creación de un sistema de autenticación mediante *PHP*.

El paquete *Auth* es capaz de interoperar con las capas de base de datos de *PEAR*, *MDB* y *MDB2*, así como con otros mecanismos como son: ficheros de texto plano, *LDAP*, *POP3*, *IMAP*, *vpopmail*, *RADIUS*, *SAMBA* o incluso *SOAP*.

Para acceder a cada uno de estos formatos de almacenamiento, *Auth* emplea los *storage drivers*, o drivers de almacenamiento. Los *storage drivers* implementan las funciones de lectura y escritura de información sobre el medio en concreto para el que han sido creadas. Además, estos drivers pueden ser desarrollados libremente para implementar el acceso a medios que hasta el momento no hubieran sido desarrollados dentro del marco del paquete *Auth*.

6.4.1 Formatos de almacenamiento.

El paquete *Auth* es capaz de emplear diversos formatos de almacenamiento a los que denomina "*storage containers*" o contenedores de almacenamiento.

A continuación se listan dichos contenedores:

TIPO	DESCRIPCIÓN
Array	Utiliza un vector con nombres de usuarios y contraseñas para autentificar.
DB	Utiliza el paquete PEAR:DB para validar contra una base de datos.
DBLite	Es una versión simplificada de DB.
File	Utiliza un fichero mediante PEAR:File_Passwd.
IMAP	Conecta a un servidor IMAP utilizando los datos suministrados.

KADM5	Autentifica contra un servidor Kerberos 5 utilizando la extensión kadm5 de PECL.
LDAP	Autentifica contra un servidor LDAP, incluso contra un Active Directory.
MDB	Utiliza el paquete PEAR:MDB para autentificar contra una base de datos.
MDB2	Utiliza el paquete PEAR:MDB2 para autentificar contra una base de datos.
PEAR	Autentifica directamente contra la web pear.php.net
POP3	Conecta a un servidor POP3 utilizando los datos suministrados.
RADIUS	Autentifica contra un servidor RADIUS utilizando el paquete AUTH_Radius y el PECL radius.
SAP	Autentifica contra un servidor SAP utilizando la extensión SAPRFC de la dirección http://saprfc.sourceforge.net.
SMBPasswd	Autentifica contra ficheros smbpasswd de SAMBA utilizando el paquete File_SMBPasswd.
SOAP	Autentifica contra un servicio SOAP utilizando el cliente SOAP de PEAR.
SOAP5	Autentifica contra un servicio SOAP utilizando la extensión de PHP para SOAP.
vpopmail	Autentifica contra un servicio vpopmail.
Custom	Sirve para construir sistemas de autentificación personalizados.

6.4.2 Constantes.

El paquete *Auth* dispone de un conjunto de constantes predefinidas.

NOMBRE	VALOR	DESCRIPCIÓN
AUTH_IDLED	-1	Se ha excedido el tiempo de espera.
AUTH_EXPIRED	-2	La sesión ha expirado/caducado.
AUTH_WRONG_LOGIN	-3	No se han podido validar el conjunto usuario/password.
AUTH_METHOD_NOT_SUPPORTED	-4	La función solicitada no está disponible para el mecanismo de autentificación seleccionado.
AUTH_SECURITY_BREACH	-5	Se ha detectado una brecha de seguridad.

6.4.3 Funcionalidad del paquete *Auth*.

Las principales funciones del paquete *Auth* son:

- ***Auth::Auth***, se trata del método constructor de *Auth*. Define el *storage driver* que se empleará para acceder al medio que almacena la información de autenticación, de parametrizar el origen de datos sobre el que se realizarán las operaciones de

verificación, establecer la función de autenticación que comprobará la identidad de los usuarios, así como indicar si el logado del usuario será opcional u obligatorio.

- ***Auth::getAuth***, es una de las funciones de comprobación. Devuelve un valor lógico indicando si el usuario actual ha sido autenticado.

- ***Auth::getAuthData***, devuelve los datos almacenados en un campo determinado.

- ***Auth::getStatus***, de forma similar a *getAuth*, esta función devuelve el estado de la autenticación del usuario actual. Los valores devueltos pueden ser: *auth_idled*, *auth_expired* o *auth_wrong_login*; Indicando respectivamente que la sesión ha permanecido inactiva demasiado tiempo, que la autenticación ha caducado, o que el proceso de autenticación no se ha podido completar debido a que la información suministrada es errónea o inválida.

- ***Auth::getUsername***, como su nombre indica, devuelve el nombre de usuario de la sesión activa.

- ***Auth::sessionValidThru***, devuelve la cantidad de tiempo durante la que la sesión seguirá siendo válida. En el caso de no haberse definido un límite de tiempo, devolvería 0.

- ***Auth::setAdvancedSecurity***, activa funcionalidades de seguridad avanzadas dificultando la ejecución de diversos ataques como *man-in-the-middle* o *session hijacking*. Requiere la activación de cookies y javascript en el cliente.

- ***Auth::setAllowLogin***, establece si se permite el logado de los usuarios. Por defecto está activado.

- ***Auth::setAuth***, establece que el usuario indicado está logado en la sesión, independientemente de su veracidad.

- ***Auth::setAuthData***, Permite almacenar información extra junto con la información de logado del usuario.

- ***Auth::setExpire***, permite establece o incrementar el tiempo de validez de la autenticación.

- ***Auth::setIdle***, permite establecer o incrementar el tiempo máximo de espera del sistema, es decir, el tiempo máximo que puede pasar entre dos acciones del mismo usuario.

- ***Auth::setSessionName***, establece el nombre de la sesión. Esta opción resulta indispensable en servidores que lanzan varias aplicaciones y en servidores compartidos, ya que en caso de mantener el nombre de sesión por defecto podrían producirse colisiones de información entre las distintas aplicaciones.

- ***Auth::setShowLogin***, establece si debe mostrarse el formulario de logado de usuarios. Esta opción está incluida de manera opcional en la función inicial *Auth*. Por defecto está activo.

- ***Auth::start***, inicia el proceso de autenticación de usuarios.

- ***Auth::check_auth***, comprueba si existe alguna sesión con información de acceso autenticada.

- ***Auth::logout***, produce la desconexión de cualquier usuario logado.

6.4.3.1 Métodos para el control de validaciones.

Disponemos de tres funciones que nos permiten asociar acciones adicionales a los mecanismos de logado del paquete *Auth*.

- ***Auth::setLoginCallback***, prepara la invocación a una función auxiliar en caso de que se realice correctamente el logado del usuario.

- ***Auth::setFailedLoginCallback***, prepara la invocación a una función auxiliar en caso de que falle el logado del usuario.

- ***Auth::setLogoutCallback***, prepara la invocación a una función auxiliar en caso de que el usuario desconecte.

6.4.3.2 Métodos de gestión de usuarios.

Existen diversos métodos para el control de usuarios:

- ***Auth::listUsers***. Informa de los usuarios disponibles en el contenedor actual.

- ***Auth::addUser***. Añade un nuevo usuario al contenedor actual.

- ***Auth::removeUser***. Elimina un usuario del contenedor.

- ***Auth::changePassword***. Modifica el password de un usuario.

6.4.4 Funcionamiento del paquete *Auth*.

El objetivo de este punto es suplir algunas carencias de la documentación oficial en cuanto al funcionamiento general del paquete, aportando una visión general de su funcionamiento.

Aunque el propio paquete *Auth* ofrece la posibilidad de utilizar un formulario predefinido para recopilar la información de logado del usuario, las actuales necesidades de la web en cuanto a presencia y estética, hacen que esta opción sea desaconsejable en muchos casos. En este punto, la documentación oficial no facilita, en modo alguno, la creación de un formulario de entrada.

El funcionamiento es sencillo, al invocar al método *start* se inicia una serie de procesos que, entre otras tareas, proceden a analizar diversas variables de sistema como son: $GLOBALS, $_POST, $_SESSION y $_COOKIES; durante este análisis, si se encuentran los índices "*username*" y "*password*" se utilizan para realizar la acción de logado del usuario.

Por este motivo, los únicos requisitos a cumplir en la creación del formulario de logado son enviar el nombre del usuario a través de un campo denominado "*username*" y la contraseña mediante el campo "*password*". Una vez comprobado los datos, y completado el proceso de entrada, el paquete *Auth* realizará las gestiones necesarias para garantizar que dicha información no se extravíe.

Una vez logado el usuario en el sistema, podemos utilizar las funciones *getAuth* y *getStatus* para comprobar la validez de las credenciales del usuario logado, y el correcto funcionamiento del sistema.

Mientras que la función *getAuth* tan solo indica mediante un booleano si el usuario se ha logado correctamente, la función *getStatus* sirve para consultar si ha ocurrido algún error, como la caducidad de la sesión, obteniendo un valor entero que informa del origen del error, tras lo que el sistema procede liberar dicho código de error devolviendo el sistema a la normalidad.

En último lugar, es imprescindible saber como almacena el paquete *Auth* la información de seguridad de la sesión. *Auth*, como es lógico, almacena la información de seguridad en la propia sesión. En su caso, *Auth* almacena un vector en la sesión cuyo nombre será el valor del *sessionName*, que suministremos al paquete.

6.4.5 Ampliaciones de Auth.

Existen principalmente dos proyectos que emplean el paquete *Auth* como base de autenticación:

- ***Auth_HTTP***, es un paquete para crear sistemas de autenticación *HTTP* similares a la autenticación de *Apache* basada en reinos mediante ficheros *.htaccess*. Este proyecto que ha alcanzado una versión estable dispone de las ventajas y potencia del proyecto original *Auth*.

- ***LiveUser***, el objetivo de este paquete, al contrario de *Auth*, es establecer un entorno completo de autenticación de usuarios y asignación de permisos. Se trata de un *framework* con grandes posibilidades cuyo principal problema es que se halla en una versión beta, pese a que sus autores consideran que su desarrollo actual permite su lanzamiento a proyectos reales. Una vez concluido este proyecto, podría llegar a integrarse en la base de sistemas de software concretos como mecanismo de autenticación, integrable con la gestión de vistas personalizadas.

6.4.6 Información adicional.

Para disponer de la información más detallada y más actualizada se recomienda consultar la información oficial del paquete en la siguiente dirección:
http://pear.php.net/manual/en/package.authentication.auth.php.