MongoDB

GESTIÓN DE LA INFORMACIÓN EN LA WEB

PRÁCTICA 4

Grupo 04

Grupo 04

Claudia Gil Navarro

Ángel Luis Ortiz Folgado

Óscar Eduardo Pérez la Madrid

Esteban Vargas Rastrollo

MongoDB

Autenticación

A la hora de llevar a cabo la autenticación, seguimos una serie de pasos para transformar las contraseñas de los usuarios, de forma que su almacenamiento resultara más seguro.

Para cifrar la contraseña utilizamos la función HASH **SHA-256**, ya que pertenece a la familia SHA-2 y a día de hoy no se han encontrado colisiones (MD5, SHA-0 y SHA-1 fueron descartadas).

Para hacer que cada contraseña sea diferente e impedir un ataque por diccionario inverso, generamos una sal que se añade a la clave antes de calcular el HASH, utilizando la función de PHP *mcrypt_create_iv($size, $source).* Esta función crea un vector de inicialización (*iv*) desde una fuente aleatoria. El parámetro *$size* indica el tamaño del vector, y *$source* indica la fuente: MCRYPT_RAND (generador de números aleatorios del sistema), MCRYPT_DEV_RANDOM (lee datos de **/dev/random**) y MCRYPT_DEV_URANDOM (lee datos de **/dev/urandom**). En nuestro caso elegimos MCRYPT_DEV_RANDOM.

Al generar la sal se creaban una serie de símbolos que la base de datos no dejaba guardar, de modo que utilizamos la función *bin2hex* para convertirlo a hexadecimal y así poder almacenarlo sin errores. Una vez convertida, se concatena junto con la contraseña del usuario.

```
68
69      if ($operacion=='editar'){
70          $nuevaPass = $_POST['nuevaPass'];
71          $passAct = $_POST['PassAct'];
72          $datos = $teatro->datos_usr($usuario,$passAct);
73          if($datos){
74          $sal=mcrypt_create_iv($length, MCRYPT_DEV_RANDOM);
75          $sal=bin2hex($sal);
76          $pass=$nuevaPass;
77          $pass=$pass.$sal;
78          $pass=hash('sha256',$pass,false);
79
80         if($teatro->edit_user($usuario, $nombre, $apellidos,$dni, $correo, $pass,$sal, $rol)){
81           header('Location:indexx.php');
82         }
```

Entradas del usuario

En el archivo *inputs.php* están todas las funciones que comprueban el formato de las entradas y desinfectan, incluidas en la clase estática **Inputs**.

Entradas de usuario:

- **Nombre de usuario** (*nick*): cualquier letra o número. El guión bajo también está permitido. Longitud entre 5 y 15 caracteres.
- **Apellidos**: caracteres [A-Za-z], más un espacio o guión (para apellidos compuestos).
- **Nombre de usuario**: caracteres [A-Za-z]. Se permite espacio en blanco para nombres compuestos.
- **DNI**: 8 dígitos y una letra mayúscula.
- **Contraseña**: mismo formato que el *nick* del usuario: cualquier letra, número y guión bajo. Entre 5 y 15 caracteres.
- **Email**: Caracteres alfanuméricos, seguidos de @, más caracteres [A-Za-z]. Luego '.' y el dominio limitado a 'com', 'es' y 'org'. Formato xxxx@xxxx.es|com|org
- **Rol**: sólo admite 'admin' o 'user'.
- **Búsqueda de obra**: caracteres [A-Za-z] separados por espacios.
- **Nombre de obra**: caracteres [A-Za-z] separados por espacios.
- **Nombre del teatro**: los caracteres se limitan a [A-Za-z]
- **Descripción de la obra**: mismo formato que nombre de la obra.
- **Hora de las sesiones del teatro**: solo se permiten dos dígitos, separados por ':' y seguidos por otros dos dígitos. Formato XX:XX.
- **Número de fila y asiento**: puede ser cualquier número, limitado a 3 cifras.
- **Comentario sobre obra**: Mismo formato que la descripción de la obra.
- **Fecha de la sesión**:El año está indicado con 4 dígitos, seguido de un guión, otros dos números indican el mes y otros dos el día. Aunque está incluido el calendario html, esta función permite comprobar el formato en caso de error. Formato XXXX-XX-XX.
- **numeroBoolValido:** Comprueba si la cadena introducida es un 0 o un 1, es una función interna que usamos nosotros para parámetros de ordenación o búsqueda

La función *desinfectar* quita los espacios del principio y del final de la cadena que comprueba, así como las etiquetas de *<php>* y *html*. Con **addslashes($string)** se devuelve una cadena con barras invertidas delante de los caracteres que necesitan ser escapados. Se utiliza para verificar todas las cadenas entrantes.

Código que comprueba dichas entradas:

```php
    private static function desinfectar($cad){
        $cadena=trim($cad); //quita espacios del principio y del final del string
        $cadena=strip_tags($cadena); //quita etiquetas <php> y html
        return addslashes($cadena); //escapa caracteres
    }
    /*Funcion que comprueba si el nombre de usuario es válido*/
    static function usuarioValido($cad){
        $cadena=Inputs::desinfectar($cad);
        $patron = "/^[\w]{5,15}$/"; // cualquier letra o numero o guion bajo de longitud entre 5 y 15
        if(preg_match($patron, $cadena)){
            return $cadena;
        }
        else return "error";
        }
  /*Funcion que comprueba si los apellidos de un usuario es válido*/
    static function apellidosValido($cad){
        $cadena=Inputs::desinfectar($cad);
        $patron = "/^(([[:alpha:]])+([[:blank:]]|-)?)*([[:alpha:]])+$/"; //se permite un guion para apellidos compuestos
        if(preg_match($patron, $cadena)){
            return $cadena;
        }
        else return "error";
    }
  /*Funcion que comprueba si el nombre de un usuario es válido*/
    static function nombreUsuarioValido($cad){
        $cadena=Inputs::desinfectar($cad);
        $patron = "/^[[:alpha:]]+[[:blank:]]?[[:alpha:]]+$/"; //cualquier nombre simple o compuesto
        if(preg_match($patron, $cadena)){
            return $cadena;
        }
        else return "error";
    }
```

```php
  /*Funcion que verifica el formato de un DNI*/
    static function dniValido($cad){
      $cadena=Inputs::desinfectar($cad);
      $patron = "/^[[:digit:]]{8}[[:upper:]]$/"; // 8 digitos y una letra mayuscula
      if(preg_match($patron, $cadena)){
          return $cadena;
        }
        else return "error";
    }
  /*Funcion que comprueba si la contraseña introducida por el usuario es válida*/
    static function passwordValido($cad){
        return Inputs::usuarioValido($cad); //La contraseña sera igual que el nombre de usuario, a no ser que se quiera dar otro formatos
    }
  /*Funcion que comprueba si un correo electronico es valido*/
    static function correoValido($cad){
      $cadena=Inputs::desinfectar($cad);
      $patron = "/^[[:alnum:]]+@[[:alpha:]]+\.(com|es|org)$/"; // xxx@xxx.es|com|org se podria hacer que lo que viene despues del punto sea un dominio cualuqiera,
      if(preg_match($patron, $cadena)){
          return $cadena;
        }
        else return "error";
    }
    /*Funcion que comprueba si lo que se introduce en la busqueda de obras es correcto*/
    static function buscarObraValido($cad){
      return Inputs::nombreObraValido($cad); //en esencia es lo mismo que el nombre de la obra
    }
  /*Funcion que comprueba si el nombre de una obra es correcto*/
  static function nombreObraValido($cad){
    return Inputs::descripcionValido($cad); //en esencia es lo mismo que una descripcion
    }
  /*Funcion que comprueba si el nombre de un teatro es valido*/
  static function nombreTeatroValido($cad){
      return Inputs::descripcionValido($cad); //en esencia es lo mismo que una descripcion
    }
```

```php
  static function validaRol($cad){
    $cadena=Inputs::desinfectar($cad);
    $patron = "/^(admin|user)$/";
    if(preg_match($patron, $cadena)){
            return $cadena;
        }
        else return "error";

  }
```

```php
/*Funcion que comprueba si el formato de la hora de las sesiones es correcta*/
static function sesionValido($cad){
  $cadena=Inputs::desinfectar($cad);
  $patron = "/^[[:digit:]]{2}:[[:digit:]]{2}$/"; // 12:34
  if(preg_match($patron, $cadena)){
        return $cadena;
    }
    else return "error";
}
/*Funcion que comprueba si el numero de filas/asientos está bien puesto*/
static function numeroValido($cad){
  $cadena=Inputs::desinfectar($cad);
  $patron = "/^[[:digit:]]{1,3}$/"; // numero de filas/asientos puede ser cualquier numero, está limitado a 3 cifras
  if(preg_match($patron, $cadena)){
        return $cadena;
    }
    else return "error";
}
/*Funcion que comprueba si la descripcion de una obra es valida*/
static function descripcionValido($cad){
  $cadena=Inputs::desinfectar($cad);
  $patron = "/^([[:alpha:]]+[[:blank:]]?)*[[:alpha:]]+$/"; // Conjunto de letras separadas por espacios
  if(preg_match($patron, $cadena)){
        return $cadena;
    }
    else return "error";
}
/*Funcion que comprueba si el comentario de una obra al valorar es correcto*/
static function comentarioValido($cad){
  return Inputs::descripcionValido($cad); //en esencia el comentario es igual que la descripcion un conjunto de palabras separadas por espacios
}
/*Aunque ya tengamos el calendario html, por si acaso falla y se pone una fecha a mano o algo*/
static function fechaValido($cad){
  $cadena=Inputs::desinfectar($cad);
  $patron = "/^[[:digit:]]{4}-[[:digit:]]{2}-[[:digit:]]{2}$/"; // Fecha en formato legible XXXX-XX-XX
  if(preg_match($patron, $cadena)){
        return $cadena;
    }
    else return "error";
}
```

Autenticación delegada

Para que en la aplicación convivan usuarios que se autentican con clave (los disponibles hasta ahora) y usuarios que se autentican a través de Google incluimos un mecanismo de autenticación delegada basada en el protocolo OpenID Connect, cuya autenticación se realiza a través de OAuth2.0.

Este protocolo presenta las mismas ventajas que OpenID (*single-sign-on* sin depender de ningún proveedor específico) pero aprovecha las implementaciones de OAuth 2.0.

Credenciales generadas para OAuth:

- **Id de Cliente**: 547638711794-hn5b8ikbbhvaqodjeh6v36hcm7i8uk94.apps.googleusercontent.com
- **Dirección de correo electrónico**: 547638711794-hn5b8ikbbhvaqodjeh6v36hcm7i8uk94@developer.gserviceaccount.com
- **Secreto de Cliente**: vho4P_EICcS98XdY7Lz4IGOf
- **Uris de redireccionamiento**: http://localhost/oauth2callback/index.php
- **Orígenes de Javascript**: http://localhost:8080

Pasos:

indexxx.php.

*Creamos la **anti-forgery state token** y la solicitud de autentificación a google*

index.php

Recibimos una respuesta en **/oauth2callback/index.php** (allí configuramos que devolvería la respuesta cuando nos registramos en console google developers), que almacenamos en la variable $code, y comprobamos que el estado de la aplicación sea el mismo que cuando enviamos la petición.

Luego una vez guardado el valor de $code, hacemos otra petición POST, para intercambiar el código recibido por el id_token y el token_access.

Esta petición la realizamos usando **curl,** y nos devuelve un Json Web Token el cual decodificamos usando al función json_decode(), para poder acceder a sus campos.

Guardamos el id_token y el access_token en las variables $id_token y $access_token respectivamente.

Ahora, obtendremos la información del usuario, usando el valor almacenado en **$id_token,** ahora hacemos otra petición

URL : "https://www.googleapis.com/oauth2/v1/tokeninfo?id_token=".$id_token"

y obtendremos otra respuesta en formato Json Web Token, que contendrá la información básica del usuario. La decodificamos con json_decode(), y almacenamos los datos que nos interesen.

Después de obtener la información del usuario desde su *id Token*, si el usuario existe en nuestra base de datos, iniciamos una sesión para dicho usuario.

Si el usuario no existe en nuestra base de datos, auto-registramos un usuario con la información proporcionada por Google.

Código para la autenticación delegada:

HttpPost.class.php

```php
<?php

class HttpPost {
    public $url;
    public $postString;
    public $httpResponse;
    public $ch;

    /**
     *
     *  Construimo sun objeto HttpPost e inicializamos CURL
     * @param url : URL al que vamos a acceder
     */
    public function __construct($url) {
        $this->url = $url;
        $this->ch = curl_init( $this->url );
        curl_setopt($this->ch, CURLOPT_URL, $this->url);
        curl_setopt($this->ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($this->ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($this->ch, CURLOPT_SSL_VERIFYHOST, false);
    }


    public function __destruct() {
        curl_close($this->ch);
    }


    public function setPostData($params) {

        $this->postString = $params;
        curl_setopt( $this->ch, CURLOPT_POST, true );
        curl_setopt ( $this->ch, CURLOPT_POSTFIELDS, $this->postString );
    }


    // Hacemos un post request al servidor
    public function send() {
        $this->httpResponse = curl_exec( $this->ch );
    }

    //Obtener la respuesta que nos ha dado el servidor
    public function getResponse() {
        return $this->httpResponse;
    }

}

?>
```

index.php

```php
<?php
session_start();
error_reporting(E_ALL);
require('../config.php');
require('../HttpPost.class.php');

//echo $state;
if(isset($_GET['code'])) {
    $code = $_GET['code'];
    $stateRequested = " ";
    if(isset($_GET['state']))
        $stateRequested = $_GET['state'];



    /*
     *Nos aseguramos  que no hay falsificación de petición , y que el usuario
     *que envía  esta solicitud de conexión es el usuario que se suponía.
     */
    echo "Este es el estado autogenerado".$_SESSION['state'];
    echo "<br>";
    echo "Este es el estado segun google".$stateRequested;
    echo "<br>";

        if($_SESSION['state']!= $stateRequested)
            die('Invalid state parameter');

    $url = "https://accounts.google.com/o/oauth2/token";
    $post = array(
        "code" => $code,
        "client_id" => $oauth2_client_id,
        "client_secret" => $oauth2_secret,
        "redirect_uri" => $oauth2_redirect,
        "grant_type" => "authorization_code"
    );

```

```php

//convertimos $post en un post String que usaremos en nuestro HttpPost.
$postText = http_build_query($post);
//creamos un objeto HttpPost pasando por parametro la URL a la que vamos a acceder.
$request = new HttpPost($url);
$request->setPostData($postText);
$request->send();



//decodificamos el string con formato json
$data = json_decode($request->getResponse());
//Alamacenamos los tokens
$id_token = $data->id_token;
$access_token = $data->access_token;


//Url que nos da la informacion del usuario en formato JWT (json web token).
$url_id_token = "https://www.googleapis.com/oauth2/v1/tokeninfo?id_token=".$id_token;

```

```php
//hacemos la peticion para obtener los datos de la url anterior.
$ch = curl_init();
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($ch, CURLOPT_HEADER, false);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
    curl_setopt($ch, CURLOPT_URL, $url_id_token);
    curl_setopt($ch, CURLOPT_REFERER, $url_id_token);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
    $result = curl_exec($ch);
    curl_close($ch);

    echo $result;
echo "<br>";

    $obj = json_decode($result);
    $issuer =  $obj->issuer;
    $issued_to = $obj->issued_to;
    $audience = $obj->audience;
    $user_id = $obj->user_id;
    $expires_in = $obj->expires_in;
    $issued_at =  $obj->issued_at;
    $email = $obj->email;
    $email_verified =  $obj->email_verified;


//$accessToken seria el valor que necesitamos.
echo "Access Token -> ".$access_token;
echo "<br>";
echo "<br>";
echo "Id Token-> ".$id_token;
echo "<br>";

/*      Ahora nos aseguramos que los datos sean correctos       */

//asegurarnos que el token obtenido es para nuestra aplicacion.
if( $audience != $oauth2_client_id)
    die('El ID del cliente es invalido');
}

?>
```

config.php

```php
<?php
//Aqui guardo los datos obtenidos al registrarte en console.developers.google.com
$oauth2_client_id = '547638711794-hn5b8ikbbhvaqodjeh6v36hcm7i8uk94.apps.googleusercontent.com';
$oauth2_secret = 'vho4P_EICcS98XdY7Lz4IGOf';
$oauth2_redirect = 'http://localhost/oauth2callback/index.php';
?>
```

indexxx.php

```php
//para usar los datos obtenidos.
require('config.php');
session_start();

//creamos un state token para prevenir request forgery.
$stateToSave = md5(rand());

//Lo almacenamos en la session !
$_SESSION['state'] = $stateToSave;

$url = "https://accounts.google.com/o/oauth2/auth";

// construimos la  HTTP GET query
$params = array(
    "response_type" => "code",
    "client_id" => $oauth2_client_id,
    "state" => $_SESSION['state'],
    "redirect_uri" => $oauth2_redirect,
    "scope" => "openid profile email"
    );

 //"scope" => "https://www.googleapis.com/auth/plus.me"
$request_to = $url . '?' . http_build_query($params);

header("Location: " . $request_to);
?>
```

Validación de peticiones

Creamos un token aleatorio al principio en **indexx.php**, y lo guardamos en la sesión.

```
$token = md5(uniqid(rand(), TRUE));

$_SESSION['token'] = $token;
```

En cada formulario con method=’POST’ creamos un campo oculto:

```
<input type='hidden' name='token' value='$token' />
```

Luego cada página que reciba esta información por POST, comprobará que el token recibido coincide con el de la sesión, en caso que no coincida le redigiremos a una página de error.

```
if (!isset($_SESSION['token']) OR ($_POST['token'] != $_SESSION['token'])){
    header('Location:error.php');
}
else
{
        $token = $_SESSION['token'];
}
```

Hemos usado esta guía, para realizar este apartado:

http://shiflett.org/articles/cross-site-request-forgeries