Programación Web: P2

Alejandro Alcalde, *Universidad de Granada*

4 de junio de 2016

Aunque solo se pide validación mediante JS, se ha hecho también con PHP, para que exista más seguridad. El motivo es que un atacante puede fácilmente saltarse la validación JS. Esto se ha realizado para todos los formularios de la página. La validación en JS se explicará luego, para PHP se comprueba mediante expresiones regulares que la entrada es correcta. Además, se "Sanatiza" la entrada antes de procesarla con la siguiente función:

```php
<?php
function test_input($data)
{
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);

    return $data;
}
?>
```

Como bien es conocido por todos, las contraseñas de usuarios no se pueden guardar en texto plano, para ello se ha hecho uso de la librería *password_compat*. En concreto se han usado dos funciones de esta librería:

- `password_hash($pwd, PASSWORD_BCRYPT, array("cost" => 10))`. Con esta función se realiza un *hash* sobre la contraseña introducida por el usuario, se usa el algoritmo *BCRYPT*, el más robusto hasta la fecha. El parámetro *cost* especifica el coste de la CPU al ejecutar el algoritmo, a más coste, más trabajo requiere de la CPU, esto se hace para protegerse contra ataques de fuerza bruta.

- `password_verify($pwd, $realHash)`. A esta función se le pasa la contraseña introducida por el usuario cuando quiere hacer log in, y se compara con el *hash* almacenado en la base de datos, si coinciden, la contraseña es correcta.

Se han credo algunas funciones *PHP* para reutilizar código, por ejemplo, el formulario de login, que aparece en todas las páginas, se refactorizó en una función `login_form`:

```php
<?php
function header_login()
{
  if (session_status() == PHP_SESSION_NONE) {
      session_start();
```

```
  }
  if (isset($_SESSION['logged_user'])) {
      ?>
      <div class="header-container">
          <header class="wrapper">
              <h1 class="title"><a href="/~x76625397/tiendamusicaII/"> La
↪   tienda de música
                      de <?php echo $_SESSION['logged_user']; ?></a></h1>
              <section id="login-container">
                  <form action="/~x76625397/tiendamusicaII/login.php"
↪   method="post">
                      <button name="logout">Log out</button>
                      <input type="hidden" name="logout" value="logout">
                  </form>
              </section>
              <?php if ($_SESSION['is_admin'])
                  echo '<a href="/~x76625397/tiendamusicaII/admin.php">Añadir
↪   Discos</a>';
              ?>
          </header>
      </div>
      <?php

  } else {
      ?>
      <div class="header-container">
          <header class="wrapper">
              <h1 class="title"><a href="/~x76625397/tiendamusicaII/"> Mi
↪   Tienda de Música </a></h1>
              <section id="login-container">
                  <form action="/~x76625397/tiendamusicaII/login.php"
↪   method="post">
                      <input type="text" name="username" value=""
↪   placeholder="Username" required>
                      <input type="password" name="password"
↪   placeholder="Password" required>
                      <button name="singlebutton"> Login</button>
                  </form>
                  <?php if (isset($_SESSION['incorrect_password'])) echo
↪   $_SESSION['incorrect_password']; ?>
              </section>
          </header>
      </div>
      <?php
  }
  ?>
  <?php
}
?>
```

Se aprovecha de paso para explicar por encima cómo se hace el loggin:

La primera comprobación que se hace es si existe una sesión abierta, de no existir se crea. A continuación se comprueba si el usuario está logeado, si lo está, se mostrará un botón para cerrar sesión. Si es administrador además aparecerá

un enlace al panel de administración para añadir nuevos discos. De no estar logeado, se mostrará el formulario de login.

Para simplificar la creación de las páginas que contienen discos, se consulta la base de datos para obtener discos, posteriormente se itera sobre todos ellos pra rellenar la página:

```
<?php
foreach ($data as $item) {
    if ($data[0] === $item) continue;
    $plurals = $item['numComments'] == 1 ? " Comentario " : " Comentarios ";
    ?>
    <article class="other-discs">
        <figure>
            <img src="<?php echo BASE_URL . $item['cover']; ?>" alt="Portada
 ↪  decimus" height="128px"
                 width="128px"/>
        </figure>
        <header>
            <h3><?php echo $item['titulo']; ?></h3>
        </header>
        <p><?php echo $item['numComments'];
            echo " " . $plurals; ?></p>
        <p><a href="<?php echo BASE_URL . "epic/details.php?id=" .
 ↪  $item['id']; ?>"
              title="Ver Decimus"> Ver</a></p>
    </article>
    <?php
}
?>
```

La validación de formularios mediante *JavaScript* se ha realizado con una combinación de etiquetas HTML5 y accediendo a la API de HTML5, en concreto se usó el método `setCustomValidity(message)`. Expliquemos un poco lo realizado:

A los elementos obligatorios se les ha añadido el atributo **`required`**, por ejemplo:

```
<input maxlength="20" size="25" required>
```

Además para cada uno se ha establecido un tamaño máximo y una longitud para el texto.

Para la validación se ha hecho uso de la atributo **`pattern`**, el cual no dejará que se envíe el formulario si no se cumple con el patrón especificado, por ejemplo, para el nombre:

```
<input id="Nombre" name="name" type="text" placeholder="Nombre"
       required maxlength="15" size="20" pattern="^[a-zA-Z ]*$">
```

Por último, para personalizar el mensaje por defecto cuando no se cumple con el patrón o no se rellena un campo obligatorio, se ha creado una función en *JavaScript* personalizada que cambia el mensaje de error. Para ello a cada elemento se le ha añadido el atributo `oninput` y/o `oninvalid`, por ejemplo:

```
<input id="Nombre" name="name" type="text" placeholder="Nombre" required
       maxlength="15" size="20" pattern="^[a-zA-Z ]*$"
       oninput="check(this)" oninvalid="check(this)">
```

Cuando se disparan cualquiera de estos eventos, se llama a la función `check` que recibe el campo:

```
function check(input) {

    if (input.validity.valueMissing) {
        input.setCustomValidity(input.id + ' es obligatorio');
    } else {
        input.setCustomValidity('');
    }

    if (input.name == "password") {
        if (input.validity.patternMismatch) {
            input.setCustomValidity('Mejora esa contraseña (Más de 8
↪   caracteres, mayúsculas, minúsculas, números y símbolos son
↪   obligatorios)');
        }
    }
    else if (input.name == "name" || input.name == "lastname") {
        if (input.validity.patternMismatch) {
            input.setCustomValidity(input.id + ' debe contener únicamente
↪   caracteres y/o espacios');
        }
    } // ...
    // ...
}
```

Para controlar el acceso al ingreso de nuevos discos por parte del administrador, la tabla de usuarios tiene un campo *rol*, que representa los privilegios de cada usuario, los valores son *subscriptor* o *admin*, solo el *admin* puede dar de alta nuevos discos. La comprobación se hace a la hora de hacer el login, como se vió antes, esto habilita el botón *Añadir discos*, el cual lleva a la página `admin.php`. Sin embargo, en `admin.php` también se deben hacer comprobaciones, ya que de

otro modo cualquier usuario podría dirigirse a dicha página y añadir discos. En `admin.php` al principio del fichero se añade:

```php
<?php

if (session_status() == PHP_SESSION_NONE) {
    session_start();
}

if (!isset($_SESSION['is_admin']) || !$_SESSION['is_admin']) {
    die('<h1>No tienes permisos para entrar en esta página</h1>');
}
?>
```

Recordemos que en el login se creaba la variable `is_admin` así:

```php
<?php
$_SESSION['is_admin'] = $loginResult["rol"] == "admin" ? true : false;
?>
```

Para el *tooltip* que muestra las canciones al pasar el ratón por encima se ha usado solo *CSS*. Cuando se pasa el ratón por encima se activa con el selector `:hover`. Este tooltip tiene varias propiedades. Si el número de canciones es demasiado y no entra en el recuadro, se muestra un *scroll* para que el usuario pueda desplazarse y ver el resto de las canciones. En la pantalla principal también se muestra el tooltip:

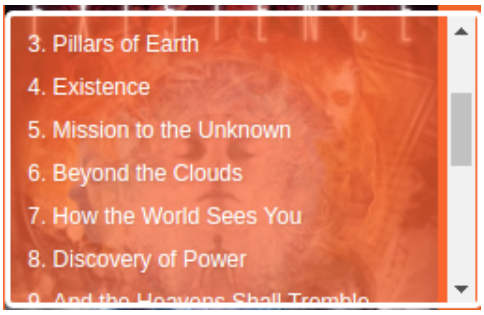

Figura 1: *Tooltip en la pantalla principal*

En el caso de que no existan canciones en la base de datos para un disco concreto, el *tooltip* no se mostrará.

Las canciones se almacenan en la base de datos.

1. Aspectos innovadores no vistos en clase

1.1. Pretty print

Mientras se ha desarrollado la práctica, fue de gran utilidad crear una función que mostrara la información de las variables que se le pasaran por parámetros, con propósitos de depurar. Esta función se declaró como sigue:

```php
<?php
function pretty_print($name = "Debug", $var)
{
    highlight_string("<?php\n" . $name . " =\n" . var_export($var, true) .
 ↪  ";\n?>");
}
?>
```

Y su salida es algo así:

```php
<?php
Discs =
array (
  0 =>
  array (
    'id' => '5',
    'titulo' => 'Existence',
    'genero' => 'EPIC',
    'precio' => '10',
    'productora' => 'Audiomachine',
    'valoracion' => '2',
    'cover' => 'img/existence.jpg',
    'numComments' => '4',
  ),
```

```php
<?php
Discs =
array (
  0 =>
  array (
    'id' => '5',
    'titulo' => 'Existence',
    'genero' => 'EPIC',
    'precio' => '10',
    'productora' => 'Audiomachine',
    'valoracion' => '2',
    'cover' => 'img/existence.jpg',
    'numComments' => '4',
  ),
```

Figura 2: *Salida en pantalla de pretty_print*

1.2. Plurales para comentarios

Al mostrar el número de comentarios de un disco, se ha comprobado si hay *1 Comentario*, varios o ninguno. De este modo se evita que aparezca el texto *1 Comentarios*. Se logró con el siguiente código:

```php
<?php
$plurals = $item['numComments'] == 1 ? " Comentario " : " Comentarios ";
?>
```

1.3. Eliminar parámetros GET de la url para hacerla más SEO-friendly

Para evitar mostrar en la url de *details.php* parámetros GET, una vez capturado el parámetro se limpia la url para que quede de la forma */details.php* en lugar de */details.php?id=XX*. Se logró con el siguiente código:

```php
<?php
if (!empty($_GET)) {
    $_SESSION['get'] = $_GET;
    header('Location: details.php');
}

if (isset($_SESSION['get'])) {

    $id = $_SESSION['get']['id'];

    $d = new Disc();

    if (!($disc = $d->getDisc($id))) {
        die('<h1>El disco no existe.</h1>');
    }

    $comments = $d->getComments($id);
    $tracks = $d->getTracks($id);

    $title = $disc['titulo'];
    $gender = $disc['genero'];
    $price = $disc['precio'];
    $discografy = $disc['productora'];
    $rating = $disc['valoracion'];
    $cover = $disc['cover'];

} else {
    die('<h1>El disco no existe.</h1>');
}
?>
```

1.4. Comentarios de varias líneas

Al principio los comentarios, aunque el usuario los introdujera con saltos de línea, luego salían en un único párrafo. Para conseguir que se mostrara bien se utilizó lo siguiente:

```php
<?php
$multiline_comment = preg_split("/\n/", $comment['comentario']);
foreach ($multiline_comment as $line)
    echo "<p><em>" . $line . "</em></p>";
?>
```
