# Práctica 2

*Autor: José Antonio Álvarez Ocete*

### 1. Configuración de la base de datos

Empezamos la práctica configurando la base de datos. Abrimos una conexión *ssh* a `betatun.ugr.es` y ejecutamos el siguiente comando para acceder a nuestra BBDD:

```
mysql -u pw12345678 -p
```

Una vez conectados a MariaDB con nuestro usuario utilizamos el siguiente comando para seleccionar nuestra base de datos:

```
use db77553417_pw2021;
```

Necesitaremos tres tablas. Una de usuario, una de series y otra de secciones. El diagrama de la base de datos se puede encontrar aquí.

Creamos la tabla de los usuarios:

```
create table if not exists user (
    username varchar(255) not null,
    name varchar(255) not null,
    password varchar(255) not null,
    admin bool default false,
    email varchar(255) not null,
    phone int(9),
    profile_picture_path varchar(255) not null,
    primary key(username)
);
```

Donde la clave primaria es `username`, que tendrá que ser único por usuario. Utilizaremos el usuario para el *login* pero después mostraremos su nombre (`name`) en la parte superior derecha de la pantalla. Cabe destacar también el `profile_picture_path`, el path a la imagen de perfil que suba el usuario, que almacenaremos en el servidor.

Procedemos a crear la tabla de secciones:

```
create table if not exists section (
    section_id int auto_increment,
    name varchar(255) not null,
    primary key(section_id)
);
```

Donde hacemos uso del `auto_increment` para automatizar la asignación de ids. Y finalmente creamos la tabla de series:

```
create table if not exists serie (
    serie_id int auto_increment,
    section_id int not null,
    title varchar(255) not null,
    n_temps int not null,
```

```
    platform varchar(255) not null,
    pegi_18 bool not null,
    premiere_date date not null,
    description varchar(1023),
    serie_picture_path varchar(255) not null,
    primary key(serie_id),
    foreign key(section_id)
        references section(section_id)
);
```

Donde seleccionamos como clave externa la `section_id` para saber a qué sección pertenece este elemento. Utilizaremos esta asociación para mostrar todas las series de una sección en nuestra web.

Finalmente, para verificar que las base de datos está bien configurada utilizamos el siguiente comando para ver lass tablas existentes en neustra base de datos:

```
show tables;

+-----------------------------+
| Tables_in_db77553417_pw2021 |
+-----------------------------+
| section                     |
| serie                       |
| user                        |
+-----------------------------+
```

Para verificar los campos y las restricciones de una tabla utilizamos:

```
show fields from user;

+----------------------+--------------+------+-----+---------+-------+
| Field                | Type         | Null | Key | Default | Extra |
+----------------------+--------------+------+-----+---------+-------+
| username             | varchar(255) | NO   | PRI | NULL    |       |
| name                 | varchar(255) | NO   |     | NULL    |       |
| password             | varchar(255) | NO   |     | NULL    |       |
| email                | varchar(255) | NO   |     | NULL    |       |
| phone                | int(9)       | YES  |     | NULL    |       |
| profile_picture_path | varchar(255) | NO   |     | NULL    |       |
+----------------------+--------------+------+-----+---------+-------+
```

```
show fields from serie;

+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| serie_id           | int(11)      | NO   | PRI | NULL    | auto_increment |
| section_id         | int(11)      | NO   | MUL | NULL    |                |
| title              | varchar(255) | NO   |     | NULL    |                |
| genre              | varchar(255) | NO   |     | NULL    |                |
| n_temps            | int(11)      | NO   |     | NULL    |                |
| platform           | varchar(255) | NO   |     | NULL    |                |
```

```
| pegi_18            | tinyint(1)   | NO   |     | NULL    |                |
| premiere_date      | date         | NO   |     | NULL    |                |
| serie_picture_path | varchar(255) | NO   |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
```

Con esto ya tenemos nuestra base de datos configurada.

### 2. Alta de usuarios

Antes de nada, para hacer pruebas añadimos un usuario normal y una administradora a la base de datos para posteriores pruebas:

```
insert into user (username, name, password, email, profile_picture_path) values
('luigi', 'Luigi', 'luigi123', 'luigi123@gmail.com', 'Luigi.png');

insert into user (username, name, password, email, profile_picture_path, admin) values
('peach', 'Princess Peach', '12345678', 'peach@gmail.com', 'Peach.png', '1');
```

Estos usuarios deberían seguir activos hoy día, sea libre el lector de usarlos para ver el funcionamiento de la página.

A continuación configuramos la alta de usuarios. Para ello alteramos el archivo `altausuario.html` para que al enviar el formulario se llame a `altausuario.php` usando el método *POST*. En este segundo archivo daremos de alta al nuevo usuario, si las condiciones son idóneas.

En primer lugar, revisamos que todos los campos estén correctamente rellenados. Aunque más adelante realizaremos esto mismo con Javascript, no quería obtener errores con la base de datos en este momento del desarrollo. Revisamos también que las dos contraseñas introducidas sean la misma.

A continución abrimos una conexión a la base de datos y realizamos una simple consulta para comprobar si el *username* que pretendemos utilizar ya está dado de alta en la abse de datos. Esta es la clave primaria de nuestra tabla así que no podemos incluir dos iguales.

```
$db = connect_to_db();
$query1 = "SELECT username FROM user";
$resultados = compute_query($db, $query1);

foreach ( $resultados as $fila ) {
    if ($fila["username"] == $_POST["username"]) {
        echo "Nombre de usuario ya existente!";
        echo exit();
    }
}
```

Como se puede ver en el código, he encapsulado ciertas funciones que conectan a la base de datos y repetiremos en multitud de ocasiones en un archivo llamado `utils.php` que incluimos con `require 'utils.php';` al principio del script. Esto facilita la lectura y nos permite hacer transparentes ciertas funcionalidades, como el tratamiento de errores que realiza la función `compute_query()`.

Tras realizar esta comprobación, añadimos al nuevo usuario a la base de datos:

```
$query2 = build_insert_query('user' , $campos, $_POST);
compute_query($db, $query2);
```

Finalmente, actualizamos la sesión actual para que quede reflejado el login del usuario. La sesión queda explicada en el siguiente apartado.

```
$_SESSION['user'] = $_POST["name"];
```

El último paso es siempre cerrar la conexión con la base de datos:

```
disconnect_from_db($db);
```

### 3. Sesión: login y logout

En esta práctica utilizaré las sesiones para comprobar si el usuario está loggeado o no, y si es administrador. Para ello, en cada documento en el que vayamso a utilizas las sesiones es necesario ejecutar `session_start();` para iniciar o reanudar la sesión existente. Adicionalmente, utilizaré dos variables en el diccionario `$_SESSION` para mantener estos indicativos: `$_SESSION['user']` contendrá el nombre del usuario activo y `$_SESSION['admin']` estará activa solamente si el usuario es administrador. Dado que estas variables se almacenan en el servidor, el usuario no podrá "autoproclamarse administrador", como ocurriría en el caso de utilizar cookies.

Con esto en mente, crear el login y logout será sencillo. En primer lugar, alteramos el formulario inicial de login para que ejecute el script `login.php` con el método post. En este script comprobamos si el usuario y la contraseña aparecen correctamente. Actualizamos `$_SESSION['user']` en ese caso, y actualizamos también `$_SESSION['admin']` si es administrador:

```
if ( $fila["password"] == $_POST["password"] ) {

    $_SESSION['user'] = $fila["name"];

    if ($fila["admin"]) {
        $_SESSION['admin'] = "true";
    }

    back_to_index();
}
```

Finalmente utilizamos la función `back_to_index()` para devolver al usuario a la página principal. Como hemos actualizado la sesión, podemos utilizar esta variable en `index.php` para mostrar un header distinto:

- Si el usuario no está logeado, mostramos un formulario de logeo y el botón para darse de alta en `altausuario.php`.
- Si el usuario está logeado, mostramos su nombre de usuario (almacenado en `$_SESSION['user']`) y un botón de cierre de sesión.
- Si además de estar logeado, el usuario es administrador, mostramos un botón para ir a la página de configuración.

El logout es un script super simple que limpia la sesión y te devuelve a la página principal:

```
unset($_SESSION['user']);
unset($_SESSION['admin']);
back_to_index();
```

Finalmente, a lo largo de la página utilizaremos las variables de sesión para restringir el acceso a determinadas páginas. Este es el caso de las páginas de administración: dar de alta items y secciones. Para ello añadimos el siguiente bloque de código a `administracion.php`, `altaitem.php` y `altaseccion.php`:

```
if ( empty($_SESSION['admin']) ) {
    back_to_index();
}
```

Adicionalmente, si el usuario ya está logeado no le permitimos el acceso a `altausuario.php`:

```
if ( !empty($_SESSION['user']) ) {
    back_to_index();
}
```

### 4. Alta de secciones e items

Una vez hecha la creación de nuevos usuarios, la de nuevas series y secciones es totalmente similar. En estos casos no procede hacer una consulta previa ya que la llave primaria de la tabla tiene un `auto-incremet` para el id. Cabe destacar que solamente el administrador puede dar de alta nuevos items y secciones.

Una vez podemos añadir items de forma sencilla a nuestra base de datos, he añadido un total de 10 series a la sección de sitcoms para hacer pruebas (hacían falta tantas de cara a la paginación). A continuación he alterado el resto de la página para que muestre los datos que obtiene de la base de datos en vez de datos estáticos. Iremos explicando poco a poco los cambios realizados.

### 5. Refactoring de HTML en `utils_html.php`

Llegado este punto he decidio que por cohesión de todo el sitio web debería poner el mismo header, footer y barra de navegación en todas las páginas. Esto nos evita además distintas páginas duplicadas como era el caso de `index.html` y `sitioweb.html`. Para ello, he descrito estos tres elementos usando código php y los he añadido como simples funciones al archivo `utils_html.php`. Veamos un ejemplo de uso. La función para crear el footer en `utils_html.php` es:

```
function print_footer() {
    echo '
      <footer class="header_and_footer hide-mobile">
        <a class="clickable" href="/~pw77553417/pe2/src/contacto.php">Contacto</a> -
        <a class="clickable" href="/~pw77553417/pe2/como_se_hizo.pdf">Cómo se hizo</a>
      </footer>
    ';
}
```

Ahora en cualquier punto del sitio web basta con incluir el archivo y ejecutar:

```
<?php print_footer(); ?>
```

Para obtener un perfecto footer. Así, los cambios realizados se propagan correctamente por todo el sitio web. De cara a la barra de navegación, la he alterado para que muestre dinámicamente las secciones desde la base de datos. Para ello hace una rápida consulta y utiliza los nombres de las secciones para construir los botones.

```
function print_nav_bar() {
    $result = get_all_sections();

    echo '<nav class="nav_section_button">';

    foreach ($result as $section) {
      echo '<a class="blue-button" href="/~pw77553417/pe2/src/section.php?id='.
        $section['section_id'].'">'.$section['name'].'</a>';
    }
     echo '</nav>';
  }
```

El header está explicado previamente así que no volverá a comentarse. En esta última función podemos apreciar uno de los cambios más importantes realizados a `section.php` e `item.php`: la páginación utilizando la url.

Como última nota, las rutas que aparecen en este archivo han de ser desde la raiz de ficheros, ya que serán invocadas desde distintos puntos en el arbol de ficheros.

### 6. Selección de series y secciones con parámetros en la URL

De cara a acceder a los distintos items y secciones he utilizado la url, asociada en php al vector `$_GET`. Por ejemplo, accediendo a `item.php?id=1`, se llama a `item.php` con `$_GET = [ [id] => 1 ]`. De esta forma podemos acceder a distintos items colocando unicamente su id en la url.

Cuando accedamos a `item.php`, se utilizará el id para hacer una consulta a la base de datos y extraer el item requerido. Tras esto, se rellena la página con los datos del item:

```
<section class="item_container">
              <section class="info_container">
                <img src="../imgs/series/'.$serie['serie_picture_path'].'"
class="movie_box_image"
                  alt="'.$serie['title'].'"/>

                <aside class="form_container">
                  <p class="label">Título: </p> '.$serie['title'].' <br>
                  <p class="label">Género: </p> '.$section_name.' <br>
                  <p class="label">Temporadas: </p> '.$serie['n_temps'].' <br>
                  <p class="label">Plataforma: </p> '.$serie['platform'].' <br>
                  <p class="label">Fecha de estreno: </p> '.$serie['premiere_date'].'
<br>
                  <p class="label">Para mayores de 18: </p>';
                    if ( $serie['pegi_18'] ) echo 'Si';
                    else echo 'No';
                echo '</aside>
              </section>
```

```
            <h4>Descripción</h4>
            <p> '.$serie['description'].' </p>
          </section>
```

También hemos de contemplar los casos en los que el id no está en nuestra base de datos o no pasamos un id como parámetro en la url. En estos casos, simplemente se muestra un mensaje de "Serie no encontrada".

De cara a la sección, utilizaremos el `section_id` y su relación en la base de datos para obtener las series asociadas a esta sección. A continuación, rellenamos la sección con los datos (imágenes, títulos y descripciones) de los items obtenidos.

En realidad, no son necesarias todas las series asociadas a la sección en cada página, basta con las 9 que vamos a mostrar. De cara a implementar la paginación en cada sección utilizaremos la opción ***LIMIT*** de MariaDB. La consulta realizada es:

```
SELECT * FROM serie WHERE section_id = "' .$section_id. '" LIMIT ' .$offset. ', 9 ;
```

Donde `$offset` nos indica a partir de qué fila hemos de coger, tomando un máximo de 9. Para saber el valor de `$offset`, pasamos un parámetro adicional en la url. De esta forma `section.php?id=1&page=2` nos mostrará la tercera oágina (empezamos a contar en 0) de la sección con id 1. Esto es, los elementos de la fila 18 a la 26.

Finalmente, hemos de actualizar los botones inferiores para que funcionen correctamente. A dónde redirecciona cada botón es sencillo: basta con mantener el id de la sección y sumar o restar uno al valor de la página. La dificultad reside en saber si debemos habilitar dichos botones o no.

Para el botón de página previa, mostraremos el botón si y sólo si la página acutal es mayor que 0. Para el de página siguiente, hemos de hacer una rápida consulta a la base de datos para averiguar si hay elementos en la página siguiente, ya que no absta con comprobar los de la página actual. Englobamos esta funcionalidad en la siguiente función:

```
function should_display_next_button($id, $page) {
    $db = connect_to_db();
    $offset = ($page+1)*9;
    $query = 'SELECT serie_id FROM serie WHERE section_id ="'.$id.
        '" LIMIT '.$offset.',9;';
    $result = compute_query($db, $query)->fetchAll();
    disconnect_from_db($db);
    return count($result) > 0;
  }
```

He intentado utilizar `COUNT(*)` per no funciona correctamente con `LIMIT`, asi que he optado por simplemente tomar los ids, que no es demasiada información que transmitir y como mucho son 9, y contar cuántos.

## 7. Repercusión en la página inicial

Finalmente, he actualizado el `index.php` para que lleve a los items correctamente. El que aparece más grande como destacado (Juego de tronos) lo he actualizado a mano, mientras que para los demás podemos hacer una rápida consulta a la base de datos para tomar las 5 primeras series y mostrarlas en portada.

Con esto termina la propagación de hacer la página dinámica.

### 8. Modificación y eliminación de series y secciones

Para implementar esta nueva funcionalidad he modificado los archivos `item.php` y `section.php` para que cuando los visita un administrador, en vez de simplemente mostrar los datos obtenidos de la base de datos, muestre un formulario ya relleno con los mismos. El administrador tiene entonces la posibilidad de modificar estos datos y actualizar la correspondiente serie o sección utilizando un botón asociado.

Naturalmente, este botón nos lleva con el método *POST* a un script en PHP que se encarga de actualizar debidamente la base de datos. De la misma forma, se han añadido botones correspondientes para eliminar series y secciones de la base de datos.

La implementación de estos scripts es análoga a la de dar de alta nuevos elementos: recogemos los campos necesarios del array `$_POST`, construimos la query utilizando una función aparte (en `utils.php`) y la ejecutamos.

Hay un par de detalles particularmente interesantes en este proceso. Por un lado, de cara a pasar a los scripts el correspondiente `$section_id` y `$serie_id` desde el formulario he utilizado campos ocultos, añadiendo lo siguiente a la declaración del formulario (hecha en PHP):

```
<input type="hidden" id="serie_id" name="serie_id" value="' . $serie['serie_id'] . '">
```

Tras ello, podemos del array `$_POST`, y después eliminarlo del mismo para que no interfiere con la creación de la consulta. Un ejemplo, en `modificación_item.php`:

```

$campos = array('title',
                'section_id',
                'n_temps',
                'platform',
                'premiere_date',
                'description',
                'serie_picture_path');

$serie_id = $_POST['serie_id'];
unset($_POST['serie_id']);

$db = connect_to_db();
$query = build_update_query('serie' , $campos, $_POST, 'serie_id', $serie_id);
compute_query($db, $query);

--------------------

function build_update_query ($table, $campos, $dict_with_values, $id_name, $id_value)
{
    $changes = array();
    foreach ( $campos as $c ) {
      array_push($changes, $c.'="'.$dict_with_values[$c].'"');
    }

    $query = "update $table set ".implode(", ", $changes).' where '.$id_name.' =
```

```
"'.$id_value.'";';
    return $query;
}
```

El otro detalle interesante es que, naturalmente, no se pueden simplemente eliminar secciones ya enlazadas a series. Para ellos hemos de eliminar todas las series asociadas primero. Una consulta sencilla nos devuelve los ids de las series asociadas a la sección, basta con eliminarlas una a una de la base de datos. Este se muestra en el siguiente trozo de código, proveniente de `eiliminar_seccion.php`:

```
$section_id = $_POST['section_id'];
$db = connect_to_db();

// Obtenemos todas las series de la sección
$query = 'SELECT serie_id FROM serie WHERE section_id = "' . $section_id . '";';
$result = compute_query($db, $query);

// Eliminamos las series una por una
foreach ($result as $serie_id) {
    $query = build_delete_query('serie', 'serie_id', $serie_id['serie_id']);
    compute_query($db, $query);
}

// Finalmente eliminamos la sección y desconectamos
$query = build_delete_query('section', 'section_id', $section_id);
compute_query($db, $query);
disconnect_from_db($db);
```

A todos los scripts de modificación y eliminación de series y secciones se les han añadido las siguiente líneas de código para que no puedan ser ejecutados (ni accediendo directamente a través de la URL) por no administradores:

```
// Si el usuario no es admin lo devolvemos al inicio
if ( empty($_SESSION['admin']) ) {
    back_to_index();
}
```

### 9. Alterar información de usuario

Llegado este punto me he dado cuenta de unicamente almacenamos en la sesión el nombre del usuario, ya que esto es lo único que hemos necesitado hasta ahora. Sin embargo, no podemos recuperar la información de la base de datos con unicamente el nombre de forma fiable y única sólo con el nombre ya que no es una clave. Es por ello que he alterado la sesión para añadir una nueva variable: `$_SESSION['username']`. Puesto que la sesión solamente se altera en tres puntos, ha sido sencillo propagar el cambio. Estos sitios son:

1. El login.
2. El logout.
3. Cuando damos de alta a un nuevo usuario y de manera automática lo loggeamos.

Una vez alterada la sesión en esos puntos, procedemos a configurar la modificación de datos por parte del usuario. Para ello añadimos un nuevo botón en el header, **Cambiar mi info**, que nos llevará a `modificar_usuario.php`. Aquí se presenta al usuario con un

formulario parcialmente relleno con sus datos para que los altere si lo desea con algunas particularidades.

En este caso, no rellenamos automáticamente la contraseña del usuario si no que la dejamos en blanco. Adicionalmente, cada vez que el usuario quiera modificar su contraseña tendrá que incluir su contraseña actual en un campo específico para verificar su identidad.

En el lado del servidor, es sencillo comprobar que la contraseña es correcta y el usuario puede modificar sus datos como hemos hecho en el pasado. En esta ocasión lo interesante ha sido darme cuenta que al dejar el campo contraseña en blanco, este automáticamente se pone en blanco en la base de datos! Aunque el usuario no quisiese cambiar la contraseña.

Para arreglar este comportamiento he añadido el siguiente código antes de construir la consulta:

```
$campos = array('username',
                  'name',
                  'password',
                  'email',
                  'phone',
                  'profile_picture_path');

[...]

// Eliminamos los campos vacíos para no ponerlos en blanco en la base de datos
foreach ($campos as $key => $value) {
  if ( !isset($_POST[$value]) or $_POST[$value] === "") {
    unset($campos[$key]);
  }
}

$query = build_update_query('user' , $campos, $_POST, 'username', $_POST['username']);
compute_query($db, $query);
```

Eliminamos así los posibles campos que estén en blanco, luego no se modificarán en la base de datos. He aplicado esta misma técnica a la modificación de series en caso de vaciarse manualmente algún campo.

### 10. Fotos de usuario

Puesto que el servidor no permite la subida por parte de scripts de imágenes al servidor, realizaremos una pequeña "triquiñuela" para gestionarlas. Por un lado, subimos todas las imágenes de perfil al servidor en `imgs/users`. Una vez subidas, cada vez que el servidor necesite una imagen de usuario, la buscará en esa carpeta. Por lo tanto, si al darnos de alta o alterar nuestros datos subimos una imagen, lo que guarda el servidor en realidad es solamente su nombre, y la buscará en dicha carpeta.

Con eso en mente tenemos que hacer poco más para implementar las imágenes de usuario. Una vez hemos configurado correctamente las imágenes de los usuarios, basta con mostrarla en el header. Para ello alteramos la función que crea el header en `utils_html.php`. Esta buscará en la base de datos el path a la imagen de usuario, en

caso de haber un usuario loggeado, utilizando la sesión. Una vez obtenido el path, basta con mostrarla.

Aunque no queda precisamente estético en el centro del header, Esta práctica no va de estilo y CSS persé, así que no me centraré particularmente en ello.

### 11. Iteración sobre todos los items

En la práctica anterior implementamos unos botones de *Anterior* y *Siguiente* en `item.html`. Con ellos podíamos visualizar distintos items saltando de uno a otro. Aunque no creo que fuese necesario, he implementado estos botones de forma dinámica para esta práctica. Para ello unicamente tenemos que preocuparnos de encontrar el anterior y el siguiente id a partir del id de la serie actual. Sin embargo, no basta con sumar o restar uno, ya que no todos los números tienen por que estar presentes. Ni basta con sumar o restar hasta llegar al próximo, porque puede no haberlo.

Para implementar esto de forma eficiente hemos de utilizar SQL con un poco más de cuidado. Busquemos primero el siguiente índice. Realizamos una consulta doble, la primera para obtener todos los el menor id de todos los mayores que el actual. Con la segunda consulta, obtenemos la fila entera con ese índica, aunque no nos es estrictamente necesaria:

```
'SELECT * FROM serie WHERE serie_id =
    (select MIN(serie_id) FROM serie WHERE serie_id > ' . $serie_id. ' );'
```

Con esta técnica podemos encontrar el id de la siguiente serie. Para el id anterior, la consulta es análoga:

```
'SELECT * FROM serie WHERE serie_id =
    (select MAX(serie_id) FROM serie WHERE serie_id < ' . $serie_id. ' );'
```

Finalmente, ocultamos el botón de *Anterior* o *Siguiente* si la respectiva consulta está vacía.

### 12. Pop up sobre elementos mostrando la sección.

Pasamos a la parte de Javascript con la funcionalidad más sencilla: mostrar un pequeño pop-up con la sección del elemento sobre el que pasamos el ratón.

Originalmente se pide mostrar tanto el título como la sección, pero ya que el título siempre se muestra, me remitiré a mostrar la sección. Adicionalmente, solo añadiremos esta funcionalidad en `index.php`. Para añadirla en cualquier otro lugar basta con repetir la misma mecánica.

En primer lugar, cuando creamos las cajitas correspondientes a cada serie, necesitamos poder saber el nombre de la sección a partir de su id. Para ello hacemos una rápida consulta al comenzar el archivo y preparamos un diccionario de la forma `[section_id] -> [name]` :

```
$query2 = 'SELECT * FROM section;';
$result = compute_query($db, $query2);

$sections = array();
foreach ($result as $s) {
```

```
    $sections[$s['section_id']] = $s['name'];
}
```

Una vez creado este diccionario, basta con añadir el siguiente código dentro de la caja que engloba a cada serie:

```
foreach ($series as $serie) {
    echo'
        [...]

        <span class="popuptext"> Sección: '.$sections[ $serie['section_id'] ].'</span>

        [...]
    ';
}
```

Así hemos configurado el texto de los pop-ups. Ahora combinaremos Javascript y CSS para hacer que aparezcan y desaparezcan. Como podemos ver en el código anterior, se le ha asociado al texto del pop-up que mostraremos la clase `popuptext`. Por otro lado, al contenedor que engloba a cada serie añadiremos la clase `popup` y un par de *EventListeners*: `onmouseover` y `onmouseout`:

```
foreach ($series as $serie) {
    echo'
        <a class="movie_box popup"
                onmouseover="show_pop_up(this)"
                onmouseout="hide_pop_up(this)"
                href="src/item.php?id='.$serie['serie_id'].'">

            <span class="popuptext"> Sección: '.$sections[ $serie['section_id']
].'</span>

            [...]
        </a>
    ';
}
```

Gestionamos los eventos con el siguiente código en Javascript añadido al `<header>`:

```
<script type="text/javascript">
    function show_pop_up(elem) {
        elem.classList.add('show');
    }

    function hide_pop_up(elem) {
        elem.classList.remove('show');
    }
</script>
```

De esta forma añadimos y quitamos una nueva clase dinámicamente utilizando el DOM al contenedor `<a>`. Utilizando CSS podemos verificar si un cierto objeto tiene una clase y su padre tiene otra. En particular:

```
.show .popuptext {
  visibility: visible;
  -webkit-animation: fadeIn 1s;
  animation: fadeIn 1s;
}
```

Estas reglas de estilo solo se aplican a elementos `popuptext` con padre `show`. Me he inspirado en este tutorial para hacer los pop-ups, aunque en el mismo acceden al elemento `<span>` (el popup en sí) utilizando `getElementById()` para cambiarle la clase a él. En vez de eso, yo simplemente se la cambio al padre. Esto nos permite omitir el uso de un identificador para el popup, que es algo más complicado al tener múltiples elementos con popups.

### 13. Verificación de formularios con Javascript

Empezaremos el desarrollo de esta compleja funcionalidad en el login y después los extrapolaremos a los demás formularios. Teniendo en cuenta que vamos a repetir este tipod e comprobaciones a lo largo de todo el sitio web, lo más sencillo será colocar todo lo referente a esta funcionalidad en un archivo aparte e independiente de cada formulario en si, para que sea facilmente escalable a los demás y sencillo de importar desde cualquier sitio. Este archivo será `validate.js`.

En primer lugar, los elementos que necesitaremos para comprobar un único campo del formulario son: el valor del campo, el tipo y su identificador. Aunque el identificador parece no ser necesario, lo utilizaremos para mostrar el campo incorrecto al usuario (normalmente se parsería un mensaje más bonito a partir del mismo, pero nos va a servir para la práctica). El siguiente paso es darse cuenta de que desde Javascript podemos encontrar el valor del input con el id utilizando `document.getElementById( id ).value;`. Asi que nuestra función de validación recibirá dos vectores: uno de los ids y otros de los tipos a comprobar para cada id.

Extraeremos los valores utilizando los ids mediante la siguiente función:

```
function extract_values_using_ids (ids) {
    var values = [];

    ids.forEach((id) => {
        values.push( document.getElementById(id).value );
    });

    return values;
}
```

Utilizamos la función `map` de Javascript para crear un único vector que en cada posición tiene el valor, el id y el tipo de un campo del formulario (parecido a lo que hace `zip` en Python). Por ejemplo, el formulario de login tienes los campos `username` y `password`. El vector mappeado tendrá los siguientes valores cuando Luigi se loggee:

```
zipped = [
    ['luigi', 'username', 'text'],
    ['luigi123', 'password', 'text']
]
```

Ahora recorremos este vector comprobando cada campo independientemente utilizando su tipo. La función de comprobación en su totalidad es la siguiente:

```
function validate (ids, types) {
    var msg = '';

    values = extract_values_using_ids(ids);

    var zipped = values.map(function(e, i) {
        return [e, ids[i], types[i]];
    });

    zipped.forEach((row) => {
        value = row[0];
        id = row[1];
        type = row[2];

        if (type == 'text') {
            if ( !validate_alfanumeric_text(value) ) {
                msg = msg.concat(`\n Texto inválido en ${id}.`);
            }
        }

    });

    if (msg !== '') {
        alert(msg);
    }
    return msg === '';
}
```

Devuelver verdadero si no hay ningún problema y falso en otro caso, además de mostrar por pantalla los campos donde hay problemas. Como podemos ver, ahora mismo hay unicamente un tipo de comprobación, para los textos, que se realiza con la siguiente función:

```
function validate_alfanumeric_text (text) {
    const regex = new RegExp('^[a-zA-Z0-9]+$');
    return regex.test(text);
}
```

Para comprobar otros tipos basta añadir mas funciones de comprobación utilizando las expresiones regulares apropiadas. Una vez tenemos la función de validación preparada tenemos que asegurarnos de que se llama antes de enviar el formulario, y que este no se envíe si la función devuelve *falso*. Para ello añadimos a la cabecera del formulario un *EventListener* que llame a una nuestra función:

```
<form method="post" onSubmit="return submit_login_form();"
    action="/~pw77553417/pe2/src/login.php" id="loginForm" >

------------------------------------------------------------

// Dentro del header añadimos:
```

```
<script src="/~pw77553417/pe2/src/validate.js"></script>

<script type="text/javascript" type="module">
    function submit_login_form(){
        var ids = ["username", "password"];
        var types = ["text", "text"];

        return validate(ids, types);;
    }
</script>
```

De esta forma se llama a la función de comprobación antes de enviar el formulario y esta solo se llama si devuelte verdadero. A continuación he añadido estos cambios a todo el sitio web (pues el login aparece en todas las páginas) de forma similar a la que se hizo con `utils_html.php`: añadiendo una línea de código a la cabecera de cada página que añada estas funciones dinámicamente.

Finalmente, utilizamos esta implementación añadiendo nuevos tipos de comprobaciones en `validate.js` para el resto de formularios. En cada uno bastará con tomar los ids con sus tipos y llamar a `validate()`. Los tipos que se han tomado han sido los siguiente:

- Texto alfanumérico (corto): Revisamos que tengo longitud máxima 10 y la siguiente expresión regular: `^[a-zA-Z0-9]+$`.
- Email: Revisamos la siguiente expresión regular: `^\\S+@\\S+\\.\\S+$`.
- Teléfono: Revisamos la siguiente expresión regular: `^[0-9]{9}$`.
- Path de archivo: Revisamos la siguiente expresión regular: `^(.+)\\([^\\]+)$`.
- Fecha: Revisamos la siguiente expresión regular: `^[0-9]{4}(\-[0-9]{2}){2}$`.
- Número (positivo): Revisamos la siguiente expresión regular: `^^[0-9]+$`.
- Texto largo con espacios: Revisamos que tengo longitud máxima 1000 y la siguiente expresión regular: `^[ a-zA-Z0-9]+$`.

Hay un par de detalles interesantes a comentar. Por un lado, la expresión regular de path de archivo no se puede crear correctamente con el constructor normal de expresiones regulares: `new RegExp();`. No tengo claro por qué, pero funciona definiéndolo directamente como expresión regular con `/---/`: `const regex = /^(.+)\\([^\\]+)$/ ;`.

Por otro lado, he implementado manualmente una comprobación adicional. Como ya se aclaró al modificar el usuario, no siempre queremos cambiar la contraseña. En este caso, comprobamos si el campo de contraseña está vacío. Si lo está, no realizamos ninguna comprobación sobre la misma (y esta no se modificará al guardar los nuevos cambios). Si la nueva contraseña no está vacía, ejecutamos manualmente `validate_alfanumeric_text()` sobre ella para comprobar que tiene el formato adecuado.

# 14. Cosas innovadores que he hecho

No estoy seguro de que sean innovadoras:

- Usar la sesión no sé si es innovador.
- Utilizar los parametros en la URL para acceder a items y secciones
- La paginación en las secciones.

Higlihts:

- Comprobar si el usuario ya está registrado antes de añadirlo a la base de datos.
- Revisar la contraseña antes de cambiar los datos del usuario.
- Verificar con cuidado a qué cosas tiene acceso el user y a qué cosas el admin.
- Iteración sobre todos los items.
- La forma de verificar los formularios es super simple y limpia.