

Programación Web

**Práctica 2:**  
**Adaptación de la web**  
**de la práctica Evaluable I**



7 de mayo de 2025  
**Prof. Serafín Moral García**

Autor:  
**Ana Aragón Jerónimo**

**UNIVERSIDAD DE GRANADA**  
E.T.S. de Ingenierías Informática y de Telecomunicación

## ÍNDICE

<b>INTRODUCCIÓN.....</b>	<b>2</b>
<b>ESTRUCTURACIÓN DE LA PRÁCTICA.....</b>	<b>2</b>
<b>CONFIGURACIÓN PREVIA.....</b>	<b>3</b>
<b>DESARROLLO DE LOS DISTINTOS PUNTOS DE LA PRÁCTICA.....</b>	<b>4</b>
<b>ELEMENTOS INNOVATIVOS.....</b>	<b>13</b>
<b>REFERENCIAS BIBLIOGRÁFICAS.....</b>	<b>14</b>

## INTRODUCCIÓN

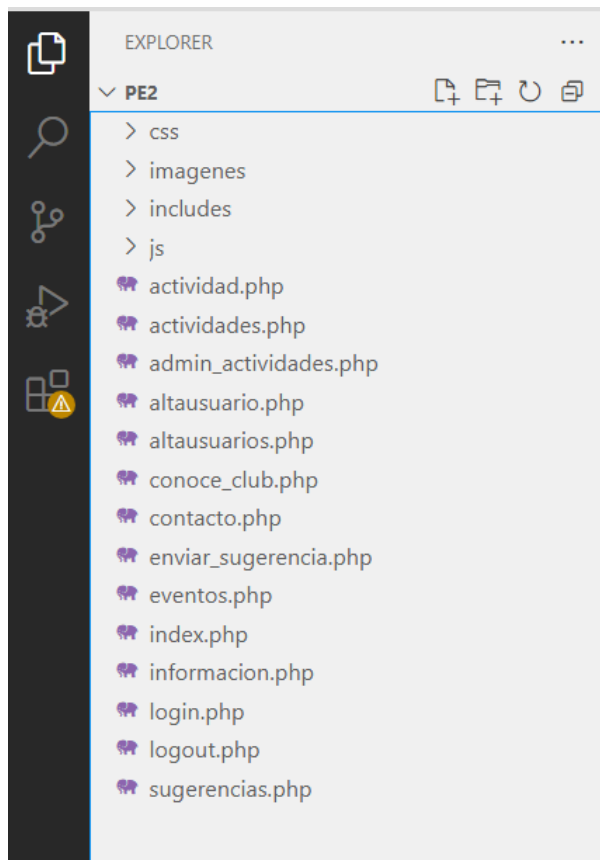
En esta segunda memoria se describe el proceso de desarrollo y adaptación de la página web estática sobre un club deportivo medieval, de la práctica anterior, a una versión dinámica e interactiva mediante el lenguaje PHP y MySQL como base de datos, desde la perspectiva del servidor; y mediante el lenguaje JavaScript, desde la perspectiva del cliente (utilizado en este caso principalmente para validaciones de los formularios).

También se han mantenido los archivos CSS de la anterior práctica para el aspecto visual de la página web, agregando nuevas cosas a las que adaptar estéticamente (como pueden ser el nuevo formulario del administrador para la gestión de las actividades, [admin\\_actividades.php](#), o la visualización del carrusel en la página de inicio, [index.php](#)).

Además de cumplir con los puntos a realizar para esta práctica, se han añadido elementos innovadores para enriquecer la seguridad del sitio (como por ejemplo, hasheando la contraseña al almacenarla en la base de datos cuando se registra un nuevo usuario), y su funcionalidad.

## ESTRUCTURACIÓN DE LA PRÁCTICA

La segunda práctica, pe2, sigue la siguiente estructura:



En primer lugar se encuentra una carpeta llamada **css**, conteniendo todas las hojas de estilo usadas para la estética de la página web del club deportivo (formato **.css**).

En siguiente lugar hay una carpeta llamada **imagenes**, igual a la anterior práctica, con todas las imágenes mostradas en la página web (formato **.jpg/.JPG**).

Seguidamente, una carpeta llamada **includes**, en la que se encuentran las clases reutilizables, la configuración de conexión a la base de datos y archivos auxiliares (**cabecera.php**). (Formato **.php**).

Hay una última carpeta con el nombre de **js**, en la que se encuentran todos los archivos de extensión **.js** (lenguaje JavaScript).

Fuera de las carpetas están el resto de archivos, de extensión **.php**, las cuales son las distintas páginas que se van mostrando

en el club deportivo Ludus Tempus, y algunas auxiliares en las que se apoyan dichas páginas. Falta la memoria de la práctica, la cual estará fuera de las carpetas junto a estos archivos.

## CONFIGURACIÓN PREVIA

En primer lugar, se ha creado un archivo de configuración ([configuracion.inc.php](#)) para la conexión con la base de datos:

```
<?php
define("DB_DSN", "mysql:host=localhost;dbname=dbanarajer_pw2425");
define("DB_USUARIO", "pwanarajer");
define("DB_CONTRASENIA", "24anarajer25");
define("TAMANIO_PAGINA", 9);
define("TABLA_USUARIOS", "usuarios");
define("TABLA_ACTIVIDADES", "actividades");
define("TABLA_SUGERENCIAS", "sugerencias");
?>
```

Así, se especifican los parámetros necesarios para la conexión a la base de datos, además de establecer el tamaño de página para la paginación (9), y definir las tablas de la base de datos (usuarios, actividades y sugerencias).

A continuación se muestran las tablas en la base de datos:

```
MariaDB [dbanarajer_pw2425]> SHOW TABLES;
+-----+
| Tables_in_dbanarajer_pw2425 |
+-----+
| actividades                  |
| sugerencias                  |
| usuarios                     |
+-----+
3 rows in set (0,001 sec)
```

También se ha creado una clase abstracta que ofrece las operaciones más habituales para gestión de la base de datos ([datosObject.class.inc](#)). Esta clase tiene un constructor, una función para devolver los valores, y otras 2 funciones, una para conectarse a la base de datos y otra para desconectarse.

Y por último, se han implementado 2 clases principales ([Usuario](#) y [Actividad](#)) que encapsulan la lógica de acceso a datos mediante PDO. Ambas clases permiten realizar operaciones CRUD sobre sus respectivas tablas. La clase Actividad también incluye otras funciones adicionales como filtrado por categorías o por paginación, necesarias para algunos puntos a realizar en esta práctica. Por otro lado, la clase Usuario contempla el registro con control de duplicados.

Estas clases heredan de `DataObject` y centralizan la conexión y desconexión a la base de datos.

## DESARROLLO DE LOS DISTINTOS PUNTOS DE LA PRÁCTICA

### **1. Permitir que el público general se dé de alta en la aplicación de forma autónoma (ellos mismos), con el correspondiente formulario de registro.**

Para que un usuario se pueda dar de alta, se parte del formulario simulado de la práctica anterior, cambiada ahora a PHP, [alta\\_usuarios.php](#). Se han quitado todos los required, debido a que ahora se puede realizar esa misma acción mediante validaciones.

Enlazado a este archivo hay un script llamado [validacion\\_altausuarios.js](#), en el cual hay una función llamada `validarRegistro`, en la que se realizan diversas comprobaciones: todos los campos del apartado de información personal son obligatorios, por lo que se deben rellenar; el correo a introducir debe ser uno válido, con @ y que contenga un . (por ejemplo, para añadir .com, .es, etc.); longitudes mínimas en el usuario y contraseña, la contraseña y la confirmación de la contraseña deben ser iguales; y se debe seleccionar obligatoriamente un plan con el número de meses a apuntarse.

Después aparece en ese mismo script un evento DOM, el cual espera a que el archivo del formulario esté totalmente cargado, asignando seguidamente la función anterior para que se ejecute al intentar enviar un formulario de alta de sesión. Si todas las validaciones son correctas, se envía sin problema alguno, pero si alguna validación no se ha realizado correctamente, se muestra un mensaje de error y no se envía.

Por último, también se hace uso de un archivo llamado `altausuario.php`, encargado de procesar el formulario de registro de nuevos usuarios (datos de alta). Al enviarse el formulario desde `altausuarios.php` mediante el método POST, este script recoge los datos introducidos, realiza unas validaciones extra que son básicas, y encripta la contraseña utilizando `password_hash`.

```
//Encriptación de la contraseña en la BD
$hash = password_hash($contrasena, PASSWORD_DEFAULT);
```

Si los datos son válidos, se insertan en la base de datos a través de la clase Usuario ([Usuario.class.inc.php](#)), se inicia la sesión del nuevo usuario y se redirige a la página principal. En caso de error, se muestra un mensaje informativo.

A continuación se muestran varios usuarios que se han dado de alta al rellenar el formulario de registro y que han sido almacenados junto con toda la información rellenada en dicho formulario en la base de datos:

```

MariaDB [(none)]> USE dbanarajer_pw2425
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [dbanarajer_pw2425]> SELECT * FROM usuarios;

```

id	nombre_completo	email	nombre_usuario	password	desea_newsletter	plan	tipo	genero	intereses
1	Ana Aragon	aj@gmail.com	anarajer	\$2y\$10\$XjU3PRtiFd083DjrQ5v6TuxSmq1J76bW8C22LCzhXpFvNK6Ez9Lnq	1	2-meses	admin	femenino	Me gusta mucho
2	Pepe Sanz	pepe@gmail.com	pepesz	\$2y\$10\$ysUjTWP8yILucNUiYl2hm.NI4KiVTm7HGLPDPI3BmiBwCFAARLkZW	0	6-meses	usuario	masculino	Desde siempre
3	Antonia RamÃ-rez	rami@gmail.com	rami2003	\$2y\$10\$ac21nCy9qpFukS/QEmho708PRvr9bPtQLzgP5gho4mXbOP5VDieIW	0	6-meses	usuario	otro	Siempre me han
4	prueba forma	prueba@gmail.com	pru	\$2y\$10\$7NfkQ5YNRyo2wm/HXb79BucCa3XptVBOT7fmlLCQUV6jEBPLCLLWy	0	6-meses	usuario	masculino	djiedji3edirdi
5	prueba segunda	pru@gmail.com	prueba	\$2y\$10\$llqIOSScym6VNXH98C6MrOtcOxE/kisF60upwp0n6VHMJ2NlnWcdG	1	12-meses	usuario	masculino	ndeiodhiehfier
6	Marcos RodrÃ-guez	marcos@gmail.com	marcosRod23	\$2y\$10\$5jDE4XxHbkFg0Kgn70VYDuTrmZfcQn5xcip3PAQ8WYuXxLd00HBW	1	6-meses	usuario	masculino	Me gusta mucho
9	Pepita GarcÃ-a	pepi@gmail.com	pepi1973	\$2y\$10\$lycEVtRMzee0/ZgtwfiYsOGIDD0FCEfn9L6xxzwODTRJ368nISuR2	0	12-meses	usuario	femenino	Desde pequeÃ-ata
14	Calimero	calimero@gmail.com	calimero2	\$2y\$10\$em9wgJGgLYJ3kmEOEtq7exz1PW8/hocHbbLOYsrXQWcv.4E9YP0a	1	1-mes	usuario	masculino	pruebas con un
15	Marian	marian@gmail.com	marian1973	\$2y\$10\$Fe.4oSxxV1CujYQyFCfie01ohnnWa07EkUgmgo.7A9S5SF2Ep1Jcda	0	6-meses	usuario	femenino	Me gustan los

```

9 rows in set (0.001 sec)

```

## 2. Modificar el documento principal para permitir identificarse al usuario y abrir así una sesión, manteniéndola activa hasta que el usuario, de alguna forma, cierre la misma.

Para ello, se han creado 2 archivos, siendo uno de inicio de sesión: [login.php](#); y otro de cierre de sesión: [logout.php](#).

El primer archivo, [login.php](#), gestiona el proceso de autenticación. Cuando un usuario envía el formulario de inicio de sesión mediante el método POST, se recogen el nombre de usuario y la contraseña que se hayan introducido.

En caso de ser correctos (que el usuario exista en la base de datos y el usuario y contraseña coincidan con los almacenados. En cualquier otro caso se mostrará un mensaje de error), se inicia una sesión mediante [session\\_start\(\)](#) y se guardan en la variable de sesión los datos relevantes del usuario (su nombre de usuario y el tipo de usuario que es (usuario o admin). Tras esto, el usuario con la sesión iniciada es redirigido a la página de inicio: [index.php](#).

Por otro lado se encuentra el archivo [logout.php](#), el cual permite cerrar la sesión activa. Al acceder a este, se destruyen todas las variables de sesión activas mediante [session\\_unset\(\)](#) y [session\\_destroy\(\)](#), redirigiendo tras esto nuevamente a la página de inicio: [index.php](#).

A continuación se muestra el código implementado para el cierre de sesión:

```
<?php
session_start();
session_unset();
session_destroy();
header("Location: index.php");
exit();
?>
```

**3. Una vez identificado, el usuario activo se quedará en el index.php pero aparecerá su nombre de usuario y su tipo en la esquina superior derecha, así como un enlace o botón para cerrar sesión.**

Tras haber iniciado sesión un usuario y redirigirlo a la página de inicio, su nombre de usuario y su tipo de cuenta (usuario o admin) se muestran en la parte superior derecha de la página principal, dentro de la cabecera. Al situarse siempre en ese lugar dentro de la cabecera, se ha decidido separar dicha cabecera en un archivo por separado, llamado `cabecera.php`, debido a que todas las páginas del club deportivo tienen la misma cabecera. Con esto, se reutiliza el código de manera que no se repita todo el bloque de cabecera en cada archivo.

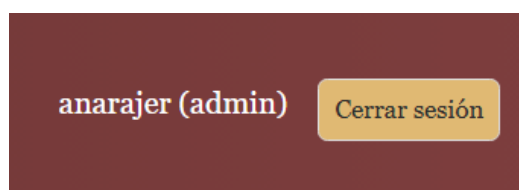
Se incluye de la siguiente manera en todos los archivos de cada página, dentro del body, antes del main:

```
<?php include("includes/cabecera.php"); ?>
```


Al cargar la cabecera, se comprueba si existe una sesión activa con un usuario identificado:

```
<?php if (isset($_SESSION["usuario"])): ?>
```

En caso de ser así, se muestra el nombre del usuario junto con su tipo (usuario o admin) al lado, junto con un botón de cierre de sesión (logout). Este botón envía una solicitud POST al archivo `logout.php`, el cual destruye la sesión y redirige al usuario (sin la sesión iniciada) a la página de inicio, `index.php`.



Si no hay ninguna sesión iniciada, se muestra en su lugar un formulario de acceso rápido con los campos para el nombre de usuario y la contraseña, así como un enlace al lado a la página con el formulario de alta de usuarios en caso de ser un nuevo usuario que no se haya registrado previamente.



El formulario de acceso rápido está ubicado en la parte superior de la página. Contiene dos campos de texto: 'Usuario' y 'Contraseña'. A la derecha de estos campos hay un botón de color naranja con el texto 'Entrar'. Al lado del botón, hay un enlace de texto 'Alta de usuarios'.

4. Permitir que el administrador del club deportivo pueda gestionar las actividades, creando nuevas, rellenando sus datos, y modificarlas o eliminarlas. Las actividades, junto con su información, mostradas en la aplicación web son las dadas de alta por el administrador y almacenadas en la base de datos. Si el número de actividades a mostrar es mayor que nueve (tres filas de tres actividades) se gestionará la paginación de las mismas en bloques de nueve actividades.

Dado que el administrador puede gestionar las actividades, se ha creado un archivo específico, llamado `admin_actividades.php`, el cual está restringido exclusivamente a usuarios con el tipo admin, tras iniciar sesión con un usuario de tipo admin previamente. Si, en cambio, el que inicia sesión es un usuario normal, esta página permanece oculta.

En esta página, el administrador puede crear nuevas actividades rellenando los datos relativos a la nueva actividad a través de un formulario con los siguientes campos: título, descripción, categoría e imágenes de actividad y de detalle de actividad, siendo los 2 últimos campos de selección de entre todas las imágenes de la carpeta local imagenes, gracias al siguiente fragmento de código:

```
//array para seleccionar la imagen dada de la carpeta de imágenes
$imagenes = array_filter(scandir("imagenes/"), function ($archivo) {
    return preg_match("/\.(jpg|JPG|jpeg|png|gif)$/i", $archivo);
});
```

También se pueden editar y eliminar actividades ya registradas, estando en un formato de tabla que muestra todas las actividades existentes junto con los botones de acción para cada operación. Al seleccionar el botón de actualizar, se puede cambiar cualquiera de los campos de la actividad seleccionada en el formulario.

Para gestionar la información de las actividades, estas se almacenan en la base de datos y son manipuladas mediante métodos de la clase Actividad (`Actividad.class.inc.php`). Las operaciones de creación, actualización y eliminación se procesan por el mismo script a través de solicitudes POST.



Respecto a la visualización de las actividades nuevas o actualizadas, junto al resto de actividades (así como ya no aparecen aquellas actividades que se hayan eliminado siendo administrador), se muestra en la página [actividades.php](#), en la que se implementa una paginación para facilitar la navegación en la página.

Si el número de actividades total es superior a 9, se dividen el número de actividades en bloques de 9 actividades por página, y se hace de la siguiente manera:

```
//Paginación-->9 por pág.  
define("PAGINA_ACTUAL", isset($_GET['pagina']) ? (int)$_GET['pagina'] : 1);  
define("ELEMENTOS_POR_PAGINA", TAMANIO_PAGINA);  
$inicio = (PAGINA_ACTUAL - 1) * ELEMENTOS_POR_PAGINA;
```

Y el tamaño de página se especificó en el archivo de configuración ([configuracion.inc.php](#)), concretamente **TAMANIO\_PAGINA**:

```
<?php  
define("DB_DSN", "mysql:host=localhost;dbname=dbanarajer_pw2425");  
define("DB_USUARIO", "pwanarajer");  
define("DB_CONTRASENIA", "24anarajer25");  
define("TAMANIO_PAGINA", 9);  
define("TABLA_USUARIOS", "usuarios");  
define("TABLA_ACTIVIDADES", "actividades");  
define("TABLA_SUGERENCIAS", "sugerencias");  
?>
```

Para mostrar los enlaces de paginación, se hace de la siguiente manera:

```
<?php if($totalPaginas > 1): ?>  
    <div class="pagination">  
        <?php for ($i = 1; $i <= $totalPaginas; $i++): ?>  
            <a href="actividades.php?pagina=<?= $i ?><?= $categoriaFiltrada  
? '&categoria=' . urlencode($categoriaFiltrada) : '' ?>"  
                class="<?= $i == PAGINA_ACTUAL ? 'active' : '' ?>"><?= $i ?>  
            </a>  
        <?php endfor; ?>  
    </div>  
<?php endif; ?>
```

Y en la clase de Actividad ([Actividad.class.inc.php](#)), se han definido unas funciones para la paginación: una función para obtener todas las actividades, otra función para obtenerlas todas de forma paginada (9 por página), y una última función en la que se obtienen las actividades por categoría y paginadas (en caso de haber más de 9 actividades por página).

Por otro lado, también se ha añadido un script de validación de actividades, llamado [validacion\\_actividades.js](#), en el que se comprueba que se escriban obligatoriamente los campos de título, descripción y categoría, con una extensión mínima por cada uno de estos. En caso de error, se manda una alerta y no se crea/actualiza la actividad dada.

- 5. El menú de actividades se generará de forma dinámica en función del contenido de actividades en la base de datos. Cada actividad tendrá asociada una categoría. Este menú estará organizado por dichas categorías. Las imágenes que se asignen a las actividades deberán estar ya en un directorio del servidor pues, por motivos de seguridad, no se permite subir ficheros al mismo desde ordenadores locales.**

El menú de actividades se ha generado de forma dinámica en función de los datos almacenados en la base de datos. Cada actividad registrada tiene asociada una categoría, lo que permite organizar el menú lateral (aside) agrupando las actividades según sus respectivas categorías.

Para la construcción de este menú lateral a través de las categorías, se usa la función [obtenerCategorias](#) de la clase Actividad ([Actividad.class.inc.php](#)), el cual recupera las distintas categorías que haya, listándose como enlaces en este menú, de forma que al pulsar sobre alguna de ellas, la página se filtre y se muestren solo aquellas actividades pertenecientes a la categoría seleccionada.

Al no poder subirse las imágenes asociadas a cada actividad manualmente desde ordenadores locales (por motivos de seguridad, como ya menciona el enunciado), deben estar almacenados en un directorio del servidor, siendo en este caso la carpeta `imagenes`, evitando así riesgos asociados a la subida de ficheros externos.

En la página de [actividades.php](#), cada tarjeta de actividad muestra su imagen correspondiente accediendo a la carpeta de `imagenes`, utilizando el nombre del archivo guardado en la base de datos.

Para garantizar que solo se usan imágenes válidas y evitar riesgos de seguridad como inyección de código o acceso no autorizado a archivos del sistema, se implementan algunas medidas de seguridad.

En [admin\\_actividades.php](#), las imágenes disponibles para seleccionar en el formulario se obtienen leyendo el contenido de la carpeta `imagenes` con el código visto anteriormente, asegurando que únicamente se muestren los archivos con extensiones típicas de imágenes (`jpg`, `jpeg`, `png`):

```
//array para seleccionar la imagen dada de la carpeta de imágenes
$imagenes = array_filter(scandir("imagenes/"), function ($archivo) {
    return preg_match("/\.(jpg|JPG|jpeg|png|gif)$/i", $archivo);
});
```

Por otro lado, en `actividades.php`, las imágenes se insertan de la carpeta `imagenes` de la siguiente forma:

```
devolverValor('titulo'))
?>" />
```

Se hace uso de `htmlspecialchars`, una función que convierte caracteres especiales en entidades HTML, evitando así inyecciones de código malicioso que pudiese estar almacenado en la base de datos.

También se concatena la ruta (`/imagenes`) de forma fija, haciendo imposible acceder a archivos fuera de esta carpeta. Además, solo se almacenan en la base de datos los nombres de las imágenes que ya estén en la carpeta `imagenes`, ya que el administrador solo puede elegir entre estos archivos a la hora de crear o actualizar una actividad.

6. **Validar la corrección de todos los formularios con JavaScript (sólo con JavaScript, quitando previamente los atributos `required` del html y convirtiendo todos los campos de texto a tipo `<input type="text">` o `<textarea>` y quitando cualquier comprobación del tipo original en html, por ejemplo, la longitud máxima y mínima). En caso de que se dé algún error, se deberá informar al usuario del mismo y la forma para arreglarlo.**

Para realizar la validación de los formularios, primero se eliminaron los atributos `required`, y se convirtieron los campos de los formularios a `<input type="text">` o `<textarea>`, permitiendo así las validaciones a través de JavaScript.

La validación se ha implementado con funciones específicas para cada formulario, que se asocian al evento `onsubmit`. Dentro de cada función, se obtienen los valores de los campos mediante `document.getElementById` o `document.querySelector`, y se usan condicionales para validar cada campo (como comprobaciones de longitud, campos obligatorios y que los campos coinciden, por ejemplo).

La estructura, por ejemplo, de `validacion_actividades.js`, empieza con la definición de la función `validarFormularioActividad`, la cual se ejecutará al enviar el administrador (admin) el formulario. Esta función accede a los valores de los campos mediante `document.querySelector` y verifica que cumplan con unas condiciones dadas.

Si alguno de los campos no cumple con la condición, se muestra un mensaje de alerta y se cancela el envío del formulario, devolviendo false.

La función es la siguiente:

```
function validarFormularioActividad() {
    const titulo =
document.querySelector('input[name="titulo"]').value.trim();
    const descripcion =
document.querySelector('textarea[name="descripcion"]').value.trim();
    const categoria =
document.querySelector('input[name="categoria"]').value.trim();

    if (titulo.length < 3) {
        alert("El título debe tener al menos 3 caracteres.");
        return false;
    }

    if (descripcion.length < 10) {
        alert("La descripción debe tener al menos 10 caracteres.");
        return false;
    }

    if (categoria.length < 3) {
        alert("La categoría debe tener al menos 3 caracteres.");
        return false;
    }

    return true;
}
```

Y para conectar esta función con el formulario, se usa un evento `onsubmit` que ejecuta la validación y bloquea el envío si algo falla:

```
document.addEventListener("DOMContentLoaded", function () {
    document.getElementById("formActividad").onsubmit =
validarFormularioActividad;
});
```

De esta manera se han hecho los demás scripts de validación con JavaScript: validando los campos necesarios con condiciones específicas y bloquear el envío en caso de no cumplir con dichas condiciones.

7. En el documento `index.php`, se deberá implementar un carrusel de actividades. Inicialmente aparecerá la información de una actividad (imagen, título y descripción) y debajo dos imágenes de flechas, una hacia la izquierda y otra hacia la derecha. Cada vez que el usuario haga clic en alguna flecha se pasará a la actividad siguiente o a la anterior, respectivamente. Se mostrará la primera si se ha llegado a la última actividad y se hace clic en la flecha hacia la derecha o, si se ha llegado al principio y se hace clic en la flecha de la izquierda, se mostrará la última. El usuario al hacer clic en una actividad será dirigido al documento descriptivo de esa actividad.

En la página de inicio, `index.php`, se implementa el carrusel mostrando las actividades del club Ludus Tempus. Las actividades se obtienen desde la base de datos mediante el método `obtenerTodas` de la clase `Actividad`, y se van presentando de 1 en 1, la imagen de la actividad correspondiente, con el título y descripción debajo.

A continuación se muestra el código del carrusel dentro de la página de inicio:

```
<?php foreach ($actividades as $index => $actividad): ?>
    <div class="actividad" style="display: <?= $index === 0
? 'block' : 'none' ?>;">
        <a href="actividad.php?id=<?=
$actividad->devolverValor('id') ?>">
            devolverValor('titulo')) ?>">
        </a>
        <h3><?=
htmlspecialchars($actividad->devolverValor('titulo')) ?></h3>
        <p><?=
nl2br(htmlspecialchars($actividad->devolverValor('descripcion')) ?></p>
    </div>
<?php endforeach; ?>
```

El usuario puede navegar entre las actividades del carrusel usando los 2 botones con las flechas:

```
<div class="flechas">
    <button id="anterior" class="boton-flecha"><</button>
    <button id="siguiente" class="boton-flecha">>>/button>
</div>
```

La lógica de navegación se gestiona desde `carrusel.js`, que escucha los clics en los botones y cambia la actividad mostrada, asegurando un bucle circular (cuando llega a la última actividad, regresa a la primera de nuevo).

Además, cada actividad está envuelta en un enlace así:

```
<a href="actividad.php?id=<?= $actividad->devolverValor('id') ?>">
```

Lo que hace es llevar a la página con la información detallada de una actividad específica, mediante el método para obtener el id de la página específica.

## ELEMENTOS INNOVATIVOS

- ❖ Cifrado seguro de contraseñas: el sistema de autenticación utiliza `password_hash()`, para almacenar contraseñas de forma segura para su validación al iniciar sesión, evitando el almacenamiento en texto plano.

```
//Encriptación de la contraseña en la BD  
$hash = password_hash($contrasena, PASSWORD_DEFAULT);
```

- ❖ Gestión de imágenes del servidor: se eligen las imágenes disponibles de la carpeta `imagenes`, y se recorren todas las imágenes con `scandir`:

```
//array para seleccionar la imagen dada de la carpeta de imágenes  
$imagenes = array_filter(scandir("imagenes/"), function ($archivo) {  
    return preg_match("/\.(jpg|JPG|jpeg|png|gif)$/i", $archivo);  
});
```

- ❖ Paginación optimizada con `SQL_CALC_FOUND_ROWS`: para mejorar el rendimiento en la carga de actividades, se utiliza esta función de MySQL, que permite conocer el número total de resultados sin tener que ejecutar una segunda consulta `COUNT(*)`, haciendo más eficiente la visualización por páginas de 9 actividades por cada una. Se usa en la página `Actividad.class.inc.php` en diferentes métodos, como `obtenerPaginadas` y `obtenerPorCategoriaPaginadas`.
- ❖ Se ha hecho uso de `formnovalidate` en alguno de los formularios para poder realizar logout sin tener que introducir los campos obligatorios de dicho formulario antes de cerrar la sesión.
- ❖ Además, se ha dejado la funcionalidad de iniciar sesión/cerrar sesión/hacer alta de usuarios en todas las páginas accesibles para el usuario en el club deportivo.
- ❖ Se ha creado también la tabla de sugerencias en la base de datos para almacenar las sugerencias registradas (manejandolas desde la base de datos como administrador, a diferencia de las actividades), y se ha realizado la validación del formulario de sugerencias.

## **REFERENCIAS BIBLIOGRÁFICAS**

[https://www.w3schools.com/js/js\\_validation.asp](https://www.w3schools.com/js/js_validation.asp)

Enlace tomado de referencia para las validaciones de los formularios en JavaScript.

PW-P3-PHP7-IV.pdf

PDF de la asignatura acerca de los archivos de conexión a la base de datos y de objetos y tablas de la BD.

<https://www.php.net/manual/es/function.password-hash.php>

Contraseña cifrada con PHP.

<https://www.php.net/manual/es/function.password-verify.php>

Validación de contraseñas cifradas.

<https://www.php.net/manual/es/function.scandir.php>

Escáner para cargar dinámicamente las imágenes desde un directorio (carpeta imagenes en este caso).

<https://developer.mozilla.org/es/docs/Web/API/Document/querySelectorAll>

Selección de elementos HTML con JavaScript.

<https://developer.mozilla.org/es/docs/Web/API/EventTarget/addEventListener>

Manejo de eventos en JavaScript.

[https://dev.mysql.com/doc/refman/8.0/en/information-functions.html#function\\_found-rows](https://dev.mysql.com/doc/refman/8.0/en/information-functions.html#function_found-rows)

Optimización de la paginación en MySQL.

<https://www.php.net/manual/es/function.htmlspecialchars.php>

Uso de htmlspecialchars para la mejora de seguridad.

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style>

Manipulación de la visibilidad en el DOM con JavaScript (para el carrusel).