

Programación Web
Grado en Ingeniería Informática
2024/2025



UNIVERSIDAD DE GRANADA

Práctica II Evaluable - Memoria

Pablo Rejón Camacho

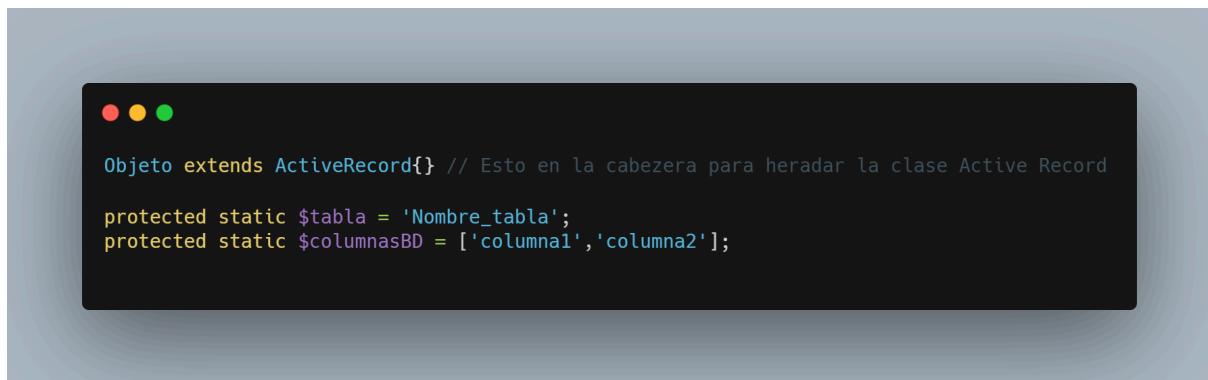


Nombre de la Web: **Pablito's CLUB**

PRE: Antes de nada quiero explicar la clase ubicada en /models/ActiveRecord.php. Esta clase la he creado para centralizar y automatizar las operaciones básicas del CRUD (Crear, Leer, Actualizar y Borrar) en los modelos de mi aplicación.

La idea es que cualquier clase que represente una tabla de la base de datos extienda de ActiveRecord. De este modo, hereda automáticamente todos los métodos necesarios para interactuar con la base de datos sin tener que escribir consultas SQL a mano.

Es una manera fácil y muy ordenada de tener todo.



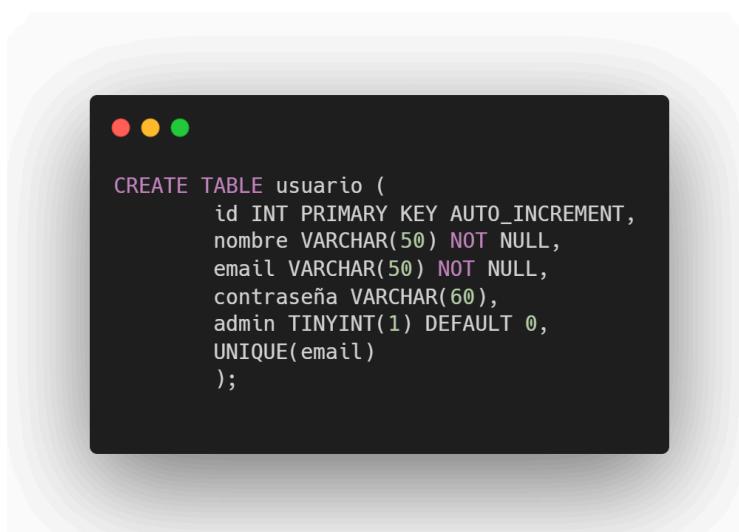
```
● ● ●

Objeto extends ActiveRecord{} // Esto en la cabecera para heredar la clase Active Record

protected static $tabla = 'Nombre_tabla';
protected static $columnasBD = ['columna1','columna2'];
```

Mediante esto la clase que se hereda puede saber el nombre de la tabla y los atributos correspondientes que posee, así es como podrá hacer la clase ActiveRecord todas las operaciones correspondientes.

Primeramente he puesto el css , y las imágenes que había utilizado en la práctica anterior , tras esto he pasado el index y lo he transformado en .php y he creado la tabla de usuarios que va a ser :



```
● ● ●

CREATE TABLE usuario (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL,
    contraseña VARCHAR(60),
    admin TINYINT(1) DEFAULT 0,
    UNIQUE(email)
);
```

Tras saber la estructura de mi tabla usuario , creó una clase usuario con sus mismos atributos y con un constructor que parte de una array para así poder inicializar un objeto usuario directamente de un POST por ejemplo.

Es una estructura básica pero que nos sirve para esta práctica. Una vez ya tengo la tabla realizo la conexión con la base de datos, incluyo una carpeta includes/ con un archivo database.php , donde voy a tener la conexión con la base de datos y varias funciones que me van a ayudar para debuguear el código cuando sea necesario.

Una vez ya he realizado y comprobado que la conexión me funciona voy a hacer algunos reajustes en alta usuario , cambiandolo a .php y gestionando el registro de los usuarios, no voy a hacer todavía lo de javaScript , eso lo dejo para más tarde.

Una vez me doy cuenta que funciona la línea:

```
usuario = new usuario($_POST);
```

Ya solo queda hacer las comprobaciones pertinentes , pero como esas se hacen con javaScript ya solo quedaría hashear la contraseña y introducir el nuevo usuario en la base de datos. Para hacer el hashing de la contraseña he hecho una función de la clase usuario que cambia la contraseña actual del objeto usuario por una hasheada, una vez hasheada podemos introducir nuestro nuevo usuario en la base de datos.

Para la gestión de errores voy a crear un array \$alertas = [] , el cuál se mostrará si en algún momento hay alguna alerta que decirle al usuario para ello voy a realizar:

He creado un file en /templates/alertas.php para que en caso de haber errores se gestione de esa forma. Y lo he añadido en altausuario en un lugar donde quiero que se muestre el error en caso de haberlo , esto es solo para gestionar errores al insertar en la base de datos.

Ahora compruebo que se haga el insert de manera correcta y tras esto paso al index.php para hacer y gestionar el login del usuario en la página web.

Tenía un error de ActiveRecord ya que tengo que asociar la base de datos una vez ya he solucionado este problema se inserta de manera correcta por lo que paso con el index.php.

Me he dado cuenta de que no tenía hecho la comprobación de si el usuario existe por lo que he tenido que volver a hacerlo después de esto he realizado, que si se registra de manera correcta paso con un método get sencillo que lo he registrado bien al index y lo muestro por pantalla.

```
● ● ●  
if( $resultado ){  
    header( 'Location: /index.php?login=exito' );  
}
```

Así redirecciono al login y lo gestino así para mostrar la alerta:

```
● ● ●  
if ( $_SERVER[ 'REQUEST_METHOD' ] === 'GET' )  
{  
    if( isset( $_GET[ 'login' ] ) && $_GET[ 'login' ] === 'exito' ){  
        Usuario::setAlerta('exito', 'El usuario ha sido registrado de manera correcta');  
        $alertas = Usuario::getAlertas();  
    }  
}
```

Ahora en el index cuando devuelvo un método POST lo que voy a hacer es comprobar que tanto el email como la contraseña son correctos , para ellos encuentro el usuario según el email en la base de datos y compruebo que la contraseña pasada por el método POST es igual a la que tengo en la base de datos , como he hasheado la contraseña necesito utilizar una función del estilo:

```
● ● ●  
// Verificaciones con la contraseña  
public function ComprobacionContraseña($valor) {  
    return password_verify($valor, $this->contraseña);
```

Una vez ya lo he comprobado muestro el nombre y el tipo en la parte superior derecha de la página, para ello primero inicio sesión una vez se han realizado las comprobaciones de la forma:

```
if( $encontrado && $contraseña_correcta ){
    session_start();
    $_SESSION['id'] = $encontrado->id;
    $_SESSION['nombre'] = $encontrado->nombre;
    $_SESSION['email'] = $encontrado->email;
    $_SESSION['admin'] = $encontrado->admin;

    if( $encontrado->admin == 1 ){
        header('Location : /');
    }
}
```

Todavía no sé si necesitaré más información en el session por ahora se queda así y como todavía no he realizado la página donde se crearán las actividades dejó la redirección cuando es admin así para luego cambiarla en el futuro.

Ahora necesito mostrar la información por pantalla para lo que voy a utilizar este código:

```
<?php
    if( isset($_SESSION['nombre']) && isset($_SESSION['admin'])):
?>

<p class="">Nombre: <?php echo $_SESSION['nombre'] ?></p>
<p>Tipo: <?php
    if( $_SESSION['admin'] == 0)
        echo 'Usuario';
    else
        echo 'Admin';
?></p>
<?php
    endif;
?>
```

Ahora tengo que gestionar las alertas si el correo no está registrado o la contraseña del usuario introducida es incorrecta. Para ello incluyo esto en el código y como el template ya está no hay que modificar como se muestra:

```
else{
    Usuario::setAlerta('error', 'El usuario no existe , o la contraseña es incorrecta');
    $alertas = Usuario::getAlertas();
}
```

Se me había olvidado introducir el botón de logout, pero se añade al lado de el tipo de usuario y el nombre , ahora voy realizar el logout.php que básicamente va a gestionar destruir la sesión y volver a redirigir al index pero ahora con un mensaje de que se ha finalizado la sesión de manera correcta, para ello en el logout.php introduzco el siguiente código:

```
● ● ●

if ($_SERVER['REQUEST_METHOD'] === 'POST' && !isset($_POST['logout']))
{
    $_SESSION = [];
    header('Location: /index.php?logout=exito');
}
else{
    header('Location: /index.php?logout=error');
}
```

Básicamente eliminó toda la información que tenía del usuario registrado y lo redirijo , también tengo en cuenta que no se pueda acceder si no es con el método post , para ello he hecho esto en el botón de cerrar sesión:

```
● ● ●

<form action="logout.php" method="POST">
    <input type="hidden" name="logout" id="logout" >
    <button class="boton" type="submit">Cerrar
    sesión</button>
```

He añadido algo de seguridad añadiendo una variable en el POST aunque no se si es estrictamente necesario.

Me he dado cuenta que mejor paso a través de un form al logout para que simplemente poniendo la ruta en el navegador no se pueda desloguear de la cuenta , y también he añadido una alerta de éxito cuando el usuario se inicia de manera correcta en la página y una de error cuando intentas acceder con la ruta de esta forma:

```
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    if( isset($_GET['login']) && $_GET['login'] === 'exito'){
        Usuario::setAlerta('exito', 'El usuario ha sido registrado de manera correcta');
        $alertas = Usuario::getAlertas();
    }

    if( isset($_GET['logout']) && $_GET['logout'] === 'exito'){
        Usuario::setAlerta('exito', 'Se ha cerrado la sesión');
        $alertas = Usuario::getAlertas();
    }

    if( isset($_GET['logout']) && $_GET['logout'] === 'error'){
        Usuario::setAlerta('error', 'Se ha accedido a un lugar sin autorización');
        $alertas = Usuario::getAlertas();
    }
}
```

Una vez ya controlo todo para el registro , el inicio de sesión y el logout , voy a pasar a empezar a gestionar un administrador que voy a tener que introducir en la base de datos inicialmente con un insert y poniendo que el boolean de admin sea uno para ello hago este insert en el sql:

```
INSERT INTO usuario ( nombre, email, edad, contraseña, admin )
VALUES (' Pablo', 'pablorejoncamacho@gmail.com', '18',
'$2y$10$vn94UFSQsdYGSCGmcvFUFuy8HAe0CL2rjT97RcjE94rW/oD/ycJYu', '1');
```

Ahora voy a empezar viendo a donde redirijo el admin cuando se inicia sesión, al parecer se queda también en el index.php así que no hace falta que haga ninguna redirección extra. Ahora voy a hacer una carpeta admin/ en la cuál estará todo lo que el admin puede realizar y que un usuario no admin no va a poder entrar.

Una vez he creado admin/actividades.php (lugar donde el admin creará y modificará y eliminará las actividades de la base de datos) voy a hacer un botón de redirección desde el index que solo podrá ver si eres admin:

```
<?php
    if( isset($_SESSION['admin']) && $_SESSION['admin'] == 1 ):
    ?>
        <a class="boton" href="/admin/actividades.php"> Admin </a>
<?php endif; ?>
```

Antes de nada voy a gestionar que los usuarios no puedan entrar en dicho archivo, para esto he creado un template que va a gestionar esta acción para así tener más limpio el código dicho template es /template/autorizado.php y en el :

```
<?php
    if( !isset($_SESSION['admin']) || $_SESSION['admin'] == 0 ){
        header('Location: /index.php?logout=error');
    }
    ?>
```

Básicamente comprueba que si no existe la variable o es un 0 nos redirige al inicio y te da el warning de que no tienes autorización para acceder a esta ubicación.

Para empezar a gestionar las actividades primero voy a hacer la tabla en la base de datos y le voy a añadir varias actividades para así poder ver que funciona , también voy a tener que gestionar como se ven dichas actividades, primero haré lo de sql de la forma:

```
CREATE TABLE actividades (
    id INT PRIMARY KEY AUTO_INCREMENT,
    tipo VARCHAR(50) NOT NULL,
    modalidad VARCHAR(50) NOT NULL,
    pistas VARCHAR(60)
);
INSERT INTO actividades (tipo, modalidad, pistas) VALUES
    ('Futbol', 'En Equipo', '1'),
    ('Natacion', 'En Equipo', '5'),
    ('Tenis', 'Individual', '3'),
    ('Futbol', 'En Equipo', '12');
```

Una vez he hecho esto tengo que crear la clase que voy a utilizar para gestionar las actividades que va a ser /models/Actividades.php , una vez ya hemos hecho esto podemos pasar a como vamos a traernos todas las actividades de la base de datos:

```
$actividades = Actividades::all();
```

Así tengo todas las actividades de la base de datos en un array que es \$actividades una vez tengo esto lo muestro por pantalla al administrador de la forma:

```
<?php
    foreach($actividades as $actividad):
?>
    <div class="actividad">
        <?php
            if( $actividad->tipo == 'Futbol'):
?>
            
<?php
            endif;
?>

        <?php
            if( $actividad->tipo == 'Natacion'):
?>
            
<?php
            endif;
?>

        <?php
            if( $actividad->tipo == 'Tenis'):
?>
            
<?php
            endif;
?>

            <h3><?php echo $actividad->tipo ?></h3>
            <p><strong>Modalidad:</strong> <?php echo $actividad->modalidad ?></p>
            <p>Pistas: <?php echo $actividad->pistas ?></p>
        </div>

    <?php
        endforeach;
?>
```

Ahora tengo que añadir los botones en cada una de ellas para poder modificarlas , o eliminarlas:

```
<a class="boton_admin" href="modificar.php?id=<?php echo $actividad->id; ?>">Modificar</a>
<a class="boton_admin" href="eliminar.php?id=<?php echo $actividad->id; ?>">Eliminar</a>
```

Como podemos observar he pasado a través del método get el id de la actividad, para saber a qué actividad nos estamos refiriendo, ahora voy a hacer los archivos php /admin/modificar.php y /admin/eliminar.php, empezamos por el de eliminar que es más simple:

```
● ● ●

require_once( __DIR__ . '/../templates/autorizado.php');

if ( $_SERVER[ 'REQUEST_METHOD' ] === 'GET' )
{
    $actividad = Actividades::where("id", $_GET[ 'id' ]);

    $resultado = $actividad->eliminar();

    if( $resultado ){
        header( 'Location: /admin/actividades.php?eliminar=exito' );
    }
    else{
        header( 'Location: /admin/actividades.php?eliminar=error' );
    }
}
else{
    header( 'Location: /admin/actividades.php?eliminar=error' );
}
```

Aquí gestión que si el id se ha pasado de manera correcta se elimine , y sigo añadiendo el template ya que nos encontramos en la carpeta admin. gestiono las alertas como lo he hecho en el índice , por lo tanto ahora en las actividades.php tengo que gestionarlo así:

```
if ($_SERVER['REQUEST_METHOD'] === 'GET')
{
    if( isset($_GET['eliminar']) && $_GET['eliminar'] === 'exito'){
        Actividades::setAlerta('exito', 'Se ha eliminado de manera correcta');
        $alertas = Actividades::getAlertas();
    }

    if( isset($_GET['eliminar']) && $_GET['eliminar'] === 'error'){
        Actividades::setAlerta('error', 'No se ha podido eliminar');
        $alertas = Actividades::getAlertas();
    }
}
```

Una vez ya he terminado y gestionado como eliminar las actividades , voy a pasar a cómo modificarlas, es decir , voy a pasar a hacer el código de admin/modificar.php:

```
if ($_SERVER['REQUEST_METHOD'] === 'GET')
{
    $actividad = Actividades::where("id", $_GET['id']);
}
```

Recojo qué actividad es la que quiero modificar , y la guardó en un objeto de la clase Actividades, una vez hecho esto voy a hacer el formulario para modificar el objeto:

```
<input type="text"
       id="tipo"
       placeholder="tipo..."
       name="tipo"
       value="php echo $actividad-&gt;tipo ?"
       required
/>
```

Es importante añadir el valor `value=...` , para que el que está modificando vea el valor actual que tiene dicha actividad, una vez ya lo sabe lo que tiene que hacer es borrar introducir un valor nuevo y darle al botón , que va a redireccionar a la misma página , y vamos a gestionarlo ahora con:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST')
{
    $encontrado = new Actividades($_POST);

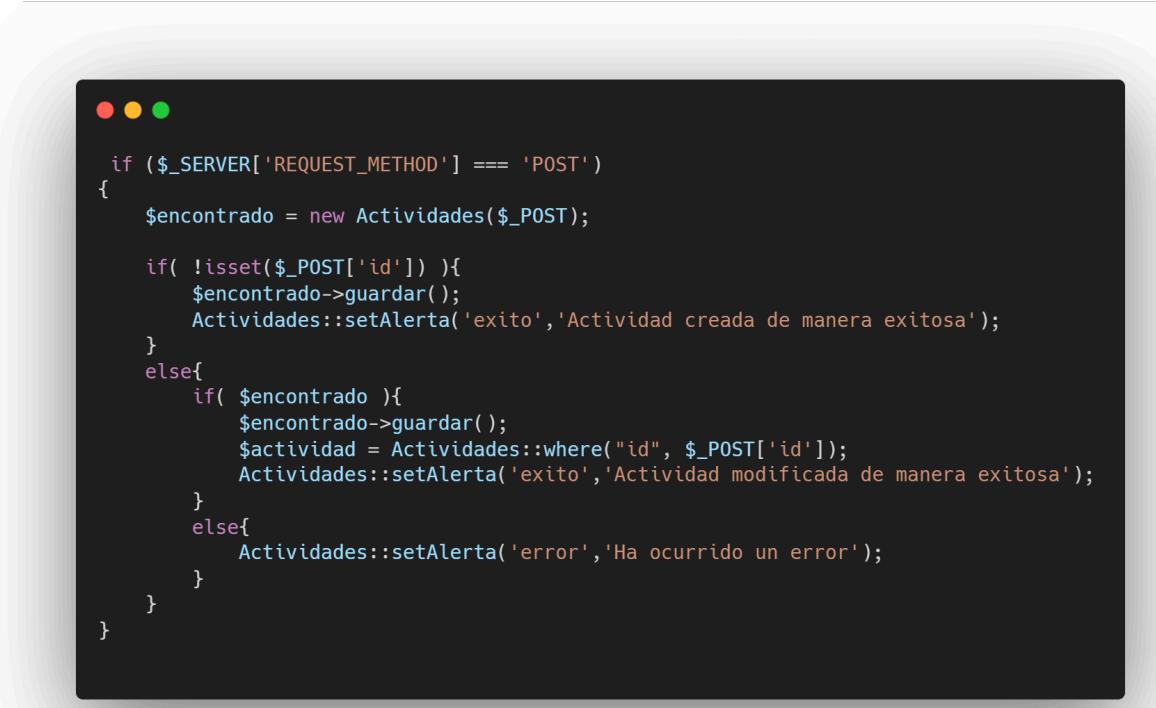
    if( $encontrado ){
        $encontrado->guardar();
        Actividades::setAlerta('exito','Actividad modificada de manera exitosa');

        $actividad = Actividades::where("id", $_POST['id']);
    }
    else{
        Actividades::setAlerta('error','Ha ocurrido un error');
    }
}
```

Aquí gestiono la actividad que ha sido modificada con el POST , tras esto la guardo en la base de datos , ajusto la alerta de que me ha funcionado de manera correcta y por último actualizo \$actividad para que en el propio formulario se muestre ya modificada la actividad(es lo que he comentado antes de los value).

Una vez ya he terminado de crear el el modificar.php voy a hacer el crear.php , aunque dándome cuenta creo que no es necesario hacer ningún crear, ya que en el modificar.php puedo ajustarlo para que funcione para ambas cosas según le pases por referencia las actividades o no.

Para hacer todo esto he tenido que hacer una serie de cambios en el modificar.php para no tener que hacer dos archivos innecesariamente, ya que el formulario es el mismo. Lo primero que he realizado es cambiar cuando se recibe un método post:



```
if ( $_SERVER[ 'REQUEST_METHOD' ] === 'POST' )
{
    $encontrado = new Actividades($_POST);

    if( !isset($_POST['id']) ){
        $encontrado->guardar();
        Actividades::setAlerta('exito','Actividad creada de manera exitosa');
    }
    else{
        if( $encontrado ){
            $encontrado->guardar();
            $actividad = Actividades::where("id", $_POST['id']);
            Actividades::setAlerta('exito','Actividad modificada de manera exitosa');
        }
        else{
            Actividades::setAlerta('error','Ha ocurrido un error');
        }
    }
}
```

El cambio básicamente es que si no paso una actividad con el método get en el form no mando un id de la forma:

```
● ● ●  
<?php  
    if( $actividad->id):  
?>  
    <input type="hidden" name="id" value="<?php echo $actividad->id ?>">  
<?php  
    endif;  
?>
```

En la vista también he cambiado algunas cosas para que se vea que estás modificando o creando según:

```
● ● ●  
<?php  
    if( $actividad->id):  
?>  
    <input type="submit" class="boton submit" value="Modificar">  
<?php  
    else:  
?>  
    <input type="submit" class="boton submit" value="Crear">  
<?php  
    endif;  
?>
```

Además de:

```
● ● ●

if( is_null($actividad->id)):
?>      <h2>Crear una nueva actividad</h2>
<?php
    else:
?>      <h2>Estamos modificando la pista id= <?php echo $actividad->id ?></h2>
<?php
    endif;
?>
```

Ya teniendo hecho todo esto , voy a ver como mostrar más de nueve actividades con la paginación. Para realizar la paginación primero he realizado unos cálculos básicos que son:

```
● ● ●

$numero_actividades = count($actividades);
$num_pag = ceil($numero_actividades/9);
$inicio = ($pag - 1) * 9;
$fin = min($inicio + 9, $numero_actividades); // No pasarse del total
```

Estos me van a ayudar a saber el rango que tengo que manejar , teniendo en cuenta que \$pag inicia su valor en 1.Para hacer la paginación hago que se muestre del array de \$actividades que muestra desde \$inicio hasta \$fin según en que \$pag nos encontremos. Para poner que la flechas he utilizado el siguiente código:

```
● ● ●

<?php
    if ($pag > 1):
?>
    a class="boton" href="?pagina=<?php echo $pag-1 ?>">- Anterior</a>
<?php
    endif;
?>
<?php
    if ($pag < $num_pag):
?>
    <a class="boton" href="?pagina=<?php echo $pag + 1; ?>">Siguiente -</a>
<?php
    endif;
?>
```

Esto muestra que en la primera página no mostramos para dar a anterior, ya que no hay. Y para mostrar si hay más verificamos si la página en la que estamos es menor que el número total de páginas , ya que si es igual nos encontramos en la última página.

Claro otra cosa es que utilizamos como se puede ver el método get para saber en qué página nos encontramos actualmente, por lo que ahora tengo que actualizar el método eliminar , ya que al eliminar te mueve otra vez a la página inicial y no es la idea. Por eso le paso por el método get al eliminar la página en la que nos encontramos:

```
<a class="boton_admin" href="eliminar.php?id=<?php echo $actividad->id;
?>&pagina=<?php echo $pag; ?>">Eliminar</a>
```

Ahora tenemos que volver a modificar el eliminar de la forma:

```
if ($_SERVER['REQUEST_METHOD'] === 'GET')  
{  
    $actividad = Actividades::where("id", $_GET['id']);  
    $pag = $_GET['pagina'] ?? 1;  
  
    $resultado = $actividad->eliminar();  
  
    if ($resultado) {  
        header("Location: /admin/actividades.php?eliminar=exito&pagina=" . $pag);  
    } else {  
        header("Location: /admin/actividades.php?eliminar=error&pagina=" . $pag);  
    }  
} else {  
    header("Location: /admin/actividades.php?eliminar=error&pagina=1");  
}
```

Una vez hemos hecho esto , cuando elimino una actividad te retorna en la página en la que estabas previamente. Además he añadido una verificación para que si quieres acceder a una página y no existe te devuelva a la primera de todas:

```
if( $pag > $num_pag)  
header("Location: /admin/actividades.php?pagina=1");
```

Una vez ya he terminado la paginación puedo dar por concluida la parte del administrador ,es decir, la creación, eliminación y modificación de las actividades paso al punto 5, que es el menú de actividades de manera dinámica. Para esto , que es básicamente mostrar las actividades para todos los usuarios creo otra carpeta /actividades con un archivo dentro /actividades/actividades.php , este archivo se va a parecer mucho al que gestiona el administrador ya que es prácticamente igual , la única diferencia es que el usuario no puede modificar nada de las actividades.

Haciendo una copia pega rápida quitando los botones del administrador podemos tener ya hecho todo lo que necesitamos , ahora queda hacer el menú según las categorías. Ahora para gestionar el menú lateral voy a hacer primeramente esto:

```
<aside class="menu-lateral">
    <p>Categorías</p>
    <ul>
        <p class="categoria">Principal:</p> class="categoria">
            <li><a href="actividades.php">Todo</a></li>
            <p class="categoria">Por deporte:</p> class="categoria">
                <li><a href="actividades.php?deporte=Futbol">Fútbol</a></li>
                <li><a href="actividades.php?deporte=Tenis">Tenis</a></li>
                <li><a href="actividades.php?deporte=Natacion">Natación</a></li>

            <p class="categoria">Por modalidad:</p> class="categoria">
                <li><a href="actividades.php?categoria=Individual">Individual</a></li>
                <li><a href="actividades.php?categoria=En Equipo">Equipo</a></li>
        </ul>
    </aside>
```

Así redirecciono con el get para así poder luego filtrar según la modalidad o el tipo. Ahora teniendo esto tengo que ver que hago cuando lo encuentro a través del GET, para ello utilizo:

```
if( isset($_GET['categoria'])){
    $categoria = $_GET['categoria'];
    $actividades = Actividades::encontrarCategoria($categoria);
}

if( isset($_GET['deporte'])){
    $deporte = $_GET['deporte'];
    $actividades = Actividades::encontrarDeporte($deporte);
}
```

He tenido que crear varias funciones que se traen todas las actividades según el filtro, dichas funciones son:

```
public static function encontrarCategoria($categoria){
    $query = "SELECT * FROM ".static::$tabla ." WHERE modalidad =
'{$categoria}'"; self::consultarSQL($query);
    return $resultado;
}

public static function encontrarDeporte($deporte){
    $query = "SELECT * FROM ".static::$tabla ." WHERE tipo = '{$deporte}'";
    $resultado = self::consultarSQL($query);
    return $resultado;
}
```

Una vez ya tengo la forma de que las actividades que se muestran se filtren según el tipo o la modalidad, ahora sigue habiendo un problema y es que en la paginación si antes había aplicado una categoría se pierde por lo que necesito que esa información se mantenga, para ello cuando sé que lo he recibido en el get hago respectivamente:

```
$extra .= '&categoria=' . urlencode($_GET['categoria']);
$extra .= '&deporte=' . urlencode($_GET['deporte']);
```

Cabe añadir que la función urlencode() es necesario ya que una de las modalidades es "En Equipo" al tener un espacio es necesario hacer esto para que funciones de manera correcta. Con esto ya tengo forma de añadir la categoría o el deporte a la hora de hacer la paginación con la forma:

```
● ● ●  
<?php if ($pag > 1): ?>  
    <a class="boton" href="?pagina=<?php echo $pag - 1 . $extra ?>">- Anterior</a>  
<?php endif; ?>  
  
<?php if ($pag < $num_pag): ?>  
    <a class="boton" href="?pagina=<?php echo $pag + 1 . $extra ?>">Siguiente -</a>  
<?php endif; ?>
```

Así también tengo en cuenta la categoría en la paginación. Ya he terminado la página actividades/actividades.php y ahora voy a hacer el que enseña cada imagen sola individualmente. Para ello he creado el archivo /actividades/actividad.php donde mostraré las imágenes, para hacerlo pasará por el método get el id de la actividad:

```
● ● ●  
<a href="actividad.php?id=<?php echo $actividad->id ?>">
```

Añadiendo eso en cada actividad ahora tengo que gestionarlo en actividad.php de la forma:

```
● ● ●

if ($_SERVER['REQUEST_METHOD'] === 'GET')
{
    if( isset($_GET['id'])){
        $id = $_GET['id'];
    }
}

$actividad = Actividades::where('id',$id);
```

Así ya tengo la actividad que quiero tener ahora una vez ya he hecho tengo que ver como lo muestro, y es de la forma:

```
<?php
    if( $actividad->tipo == 'Futbol'):
?>
    
<?php
    endif;
?>

<?php
    if( $actividad->tipo == 'Natacion'):
?>
    
<?php
    endif;
?>

<?php
    if( $actividad->tipo == "Tenis"):
?>
    
<?php
    endif;
?>

<section class="contenido-actividad">
    <div class="info">
        <h2><?php echo $actividad->tipo ?></h2>
        <p><strong>Modalidad:</strong> <?php echo $actividad->modalidad ?></p>
        <p><strong>Pistas:</strong><?php echo $actividad->pistas ?></p>
    </div>
</section>
```

Una vez ya tengo como mostrar , ya puedo dar por finalizada toda esta parte, alomejor más tarde añado más información en la base de datos para que aparezca mucho más completo.

Ahora que ya he hecho esto he terminado con toda la parte de php , el próximo paso será hacer JS , haciendo el carrusel y la corrección de todos los formularios.

Para empezar con la validación de formularios con JS(JavaScript) voy a comenzar con el formulario de registro de usuarios. Primeramente voy a quitar los campos required del html y cualquier comprobación de tipo original de html. Ahora voy a añadir al final la etiqueta <script></script>.

Después de hacer esto introduzco un :

```
document.addEventListener('DOMContentLoaded', function() {  
  
    const formulario =  
    document.querySelector('.formulario');  
    formulario.addEventListener('submit', function(e) {  
  
    });  
});
```

Primero encuentro mi formulario a partir de la clase que le he asignado, y después tengo que conseguir poder hacer referencia a cada cambio de la forma:

```
document.getElementById('email');  
document.getElementById('nombre');  
document.getElementById('edad');  
document.getElementById('contraseña');
```

Antes de hacer nada también me he dado cuenta que voy a hacer un pequeño <p></p> con id para escribir en ellos cuando la validación sea errónea , y con esto me refiero a:

```
<div class="campo">
    <label for="email">Email</label>
    <input type="text"
        id="email"
        placeholder="Email..."
        name="email"
        />
</div>
<p class="alerta_php_error" id="alerta_email"></p>
```

Con esto puedo escribir en estos , además le he añadido una clase para darle estilo. Por lo tanto he añadido a java:

```
const alertaEmail = document.getElementById('alerta_email');
const alertaNombre = document.getElementById('alerta_nombre');
const alertaEdad = document.getElementById('alerta_edad');
const alertaContraseña = document.getElementById('alerta_contraseña');
```

Una vez ya he encontrado todos los campos del formulario tengo que ver qué validaciones le voy a aplicar a cada uno de los campos para ello voy a hacer una función de validación individual para cada uno:

-Validación email:

```
function validarEmail() {
    const email = document.getElementById('email').value.trim();
    const re = /^[^@\s]+@[^\s]+\.[^\s]+$/;

    if (email === '') {
        alertaEmail.textContent = 'El email es obligatorio';
        return false;
    } else if (!re.test(email)) {
        alertaEmail.textContent = 'Por favor ingresa un email válido';
        return false;
    } else {
        return true;
    }
}
```

He hecho una función que si es correcta devolverá true, y sino pone en el mensaje de debajo del input el mensaje correspondiente con una validación del email y si está vacío ahora hago lo mismo con el nombre.

-Validación nombre

```
function validarNombre() {
    const nombre = document.getElementById('nombre').value.trim();

    if (nombre === '') {
        alertaNombre.textContent = 'El nombre es obligatorio';
        return false;
    } else if (/[^a-zA-ZáéíóúÁÉÍÓÚññ\s]+$/i.test(nombre)) {
        alertaNombre.textContent = 'Solo se permiten letras y espacios';
        return false;
    } else {
        return true;
    }
}
```

-Validación Edad

```
function validarEdad() {
    const edad = document.getElementById('edad').value.trim();
    const numEdad = parseInt(edad);

    if (edad === '') {
        alertaEdad.textContent = 'La edad es obligatoria';
        return false;
    } else if (isNaN(numEdad)) {
        alertaEdad.textContent = 'Debe ser un número válido';
        return false;
    } else if (numEdad >= 12) {
        alertaEdad.textContent = 'Debe ser mayor de 12 para registrarse';
        return false;
    } else {
        return true;
    }
}
```

-Validacion Contraseña

```
function validarContraseña() {
    const contraseña =
document.getElementById('contraseña').value.trim();
    if (contraseña === '') {
        alertaContraseña.textContent = 'La contraseña es obligatoria';
        return false;
    } else if (contraseña.length < 6) {
        alertaContraseña.textContent = 'Mínimo 6 caracteres';
        return false;
    } else {
        return true;
    }
}
```

Una vez ya tengo estas 4 funciones definidas , ahora vamos a hacer la función principal del formulario que es básicamente:

```
formulario.addEventListener('submit', function(e) {
    e.preventDefault(); // Prevenir envío para validar primero

    // Validar todos los campos
    const emailValido = validarEmail();
    const nombreValido = validarNombre();
    const edadValido = validarEdad();
    const contraseñaValido = validarContraseña();

    // Si todo es válido, enviar formulario
    if (emailValido && nombreValido && edadValido && contraseñaValido) {
        this.submit();
    }
});
```

Primero evitó el envío del formulario , y tras esto si las 4 validaciones son correctas , hago el submit el formulario.

Me he dado cuenta de varios errores , primeramente tengo mejor que añadir las clases y el texto de alerta desde java , y otra cuestión es que tengo que eliminarlo cuando me introduce bien el dato, de la forma:

```
function validarContraseña() {
    const contraseña =
document.getElementById('contraseña').value.trim();
    if (contraseña === '') {
        alertaContraseña.textContent = 'La contraseña es obligatoria';
        alertaContraseña.classList = "alerta_php error";
        return false;
    } else if (contraseña.length < 6) {
        alertaContraseña.textContent = 'Mínimo 6 caracteres';
        alertaContraseña.classList = "alerta_php error";
        return false;
    } else {
        alertaContraseña.textContent = '';
        alertaContraseña.classList = ''; // Limpia todas las clases
        return true;
    }
}
```

Así ya tenemos todas las verificaciones hecha. Ahora voy a pasar todo esto a un archivo externo para poder utilizarlo en el formulario del inicio de sesión, y el de las sugerencias. Lo importo de esta forma:

```
<script src="javascript/formulario.js"></script>
```

Ahora voy a ir al formulario de inicio de sesión que es exactamente igual , solo que no tiene la edad ni el nombre.

El siguiente formulario que voy a verificar es el de las sugerencias. Es básicamente igual así que he hecho otro archivo js y lo he modificado para sugerencias.

Ahora voy al que hay en Admin , al crear una actividad.

Ya he terminado de hacer todas las verificaciones de todos los formularios como se pide.

Ahora voy a pasar a hacer el carrusel de actividades:

Primeramente nos traemos las actividades de la basae de datos:

```
$actividades = Actividades::all();
```

Una vez ya tenemos las actividades en un array , tenemos que pasaselas a js, para ello vamos a utilizar:

```
const actividades = <?= json_encode($actividades) ?>;
```

Una vez ya tenemos las actividades , solo tenemos que mostrar las actividades moviéndonos por el array para ello he hecho una función:

```
● ● ●

function mostrarActividad(index) {
    if (!actividades.length) {
        carruselSlides.innerHTML = "<p>No hay actividades disponibles</p>";
        return;
    }

    const actividad = actividades[index];
    let imagenSrc = "";
    let altTexto = actividad.tipo;

    if (actividad.tipo === 'Futbol') {
        imagenSrc = 'imagen/futbol/futbol1.webp';
    } else if (actividad.tipo === 'Natacion') {
        imagenSrc = 'imagen/natacion/natacion1.webp';
    } else if (actividad.tipo === 'Tenis') {
        imagenSrc = 'imagen/tenis/tenis1.webp';
    }

    carruselSlides.innerHTML = `
        <div class="actividad">
            
            <h3>${actividad.tipo}</h3>
            <p><strong>Modalidad:</strong> ${actividad.modalidad}</p>
            <p><strong>Pistas:</strong> ${actividad.pistas.trim()}</p>
        </div>
    `;
}
```

Así muestro la actividad según un índice, ahora lo que tengo que hacer es ver como aumento el índice o lo disminuyo:

```
● ● ●

// Navegación circular
btnAnterior.addEventListener("click", () => {
    indiceActual = (indiceActual - 1 + actividades.length) % actividades.length;
    mostrarActividad(indiceActual);
});

btnSiguiente.addEventListener("click", () => {
    indiceActual = (indiceActual + 1) % actividades.length;
    mostrarActividad(indiceActual);
});
```

Así conseguimos una navegación circular. Para finalmente hacer que se redirija a la actividad tengo que añadir:

```
● ● ●

carruselSlides.innerHTML =
<a href="actividades/actividad.php?id=${actividad.id}">
    <div class="actividad">
        
        <h3>${actividad.tipo}</h3>
        <p><strong>Modalidad:</strong> ${actividad.modalidad}</p>
        <p>Pistas: ${actividad.pistas}</p>
    </div>
</a>
`;
```

Una vez hecho esto ya he finalizado la práctica II de PW de manera completa y cumpliendo paso a paso lo que se solicita en el pdf.