

VALORACIÓN DE PRODUCTOS



A Jeans

B Shirt

C Blouse

D Shoes

E Culottes

F Jacket

<César Miguel Ramírez Lucas>

ÍNDICE

Introducción

Este proyecto está enfocado a la creación de una página web dinámica usando “fetch” combinando JavaScript con PHP para evitar la recarga de páginas a la hora de actualizar información gestionada con un SGBD.

Cabe destacar que este proyecto tiene algo añadido a lo que se pedía para combinar todo lo trabajado en el curso (*incluyendo aspectos de otras asignaturas*) como es Tailwind CSS y el uso de control de versiones con Git y GitHub.

En concreto, el proyecto trata sobre una página web donde un usuario podrá valorar varios productos una única vez, siendo la votación enviada desde el cliente, tratada por el servidor y devuelta al cliente de forma dinámica.

La explicación de todo el proyecto que viene a continuación se centrará en explicar los aspectos más importantes de cada archivo de la aplicación (PHP y JS) y cómo inicializar el proyecto para poder trabajar con él.

Instalación y requisitos previos.

Descarga de recursos.

El proyecto al tener Tailwind CSS como framework de maquetación, este necesita de Node.js para funcionar, por ello es necesario descargarlo e instalarlo desde su página oficial. También es importante en tu editor de código (recomendable VSCode) configurar la ruta de tu “php.ini”. Y por último pero no menos importante, nuestro XAMPP que nos permitirá desplegar MySQL y Apache para poder trabajar con nuestra página web.

Recursos:

- [Node.js](#)
- [PHP](#)
- [XAMPP](#)
- [Visual Studio Code](#)

Una vez descargados e instalados, procederemos a la descarga del proyecto desde GitHub realizando un “clone”, para ello descargamos la herramienta Git ([clic aquí](#)) e instalamos y/o podemos descargar el proyecto desde la entrega del mismo.

Descarga del proyecto e instalación.

Para clonar el proyecto utilizaremos este comando en la carpeta localhost de nuestro XAMPP:

```
git clone https://github.com/elchimeneas/PHP-TailwindCSS-Valorations.git
```

Y se nos creará la carpeta con ese mismo nombre, la cual **NO DEBEMOS CAMBIAR**.

Una vez dentro de la carpeta abrimos una terminal en esa carpeta y/o abriremos el proyecto en VSCode y usamos la terminal del editor de código para instalar las dependencias de node o actualizarlas con el siguiente comando:

```
npm install
```

Importación de la base de datos.

Abrimos phpMyAdmin (*localhost/phpmyadmin*) y creamos una nueva base de datos llamada “**valorations**”. Importamos el archivo que se encuentra en la carpeta “**resources/tables.sql**” o copiamos su contenido y lo ejecutamos como una consulta SQL.

Despliegue de Tailwind CSS.

Asegurate de abrir una terminal en la carpeta “PHP-TailwindCSS-Valorations” y ejecuta este comando desde la terminal de VSCode:

```
npx tailwindcss -i ./css/input.css -o /css/output.css --watch
```

Esto generará el archivo output.css y permitirá que TailwindCSS sea reconocido por XAMPP para aplicar los estilos correspondientes, la salida de la consola debería ser algo como:

```
Rebuilding...  
Done in 277ms.
```

Una vez tengas encendidos los servicios de MySQL y Apache, podrás acceder al proyecto desde la URL:

<http://localhost/PHP-TailwindCSS-Valorations/app/index.php>

Estructura del proyecto y explicación de archivos.

Procedemos a explicar ya el proyecto en cuestión y sus archivos de aplicación, la estructura que nos vamos a encontrar en la parte de aplicación es la siguiente:

```
├─ app/                # Archivos PHP
│   ├── db.php         # Conexión a la base de datos
│   ├── index.php      # Archivo principal
│   ├── script.js      # Script JS para actualizar la valoración de un
producto
│   ├── vote.php       # Manejo de votos de los productos.
│   └── valorations.php # Panel de valoraciones de productos
```

Vamos a ir viendo cada archivo por el orden de estructura.

db.php

Esta es la parte más sencilla del proyecto, la conexión a la base de datos que incluimos posteriormente en otros archivos.

```
$db_host = '127.0.0.1';
$db_name = 'valorations';
$db_user = 'root';
$db_password = '';
```

En esta parte declaramos las variables que guardarán todos los datos necesario que nos pide el objeto para crear la conexión a la base de datos:

```
$link = "mysql:host=$db_host;dbname=$db_name;charset=utf8";
```

Creamos otra variables llamada “link” para definir el nombre de la base de datos y el host de la misma, además añadimos la característica del charset en UTF8 para evitar problemas con caracteres especiales.

```
try {  
    $db_PDO = new PDO($link, $db_user, $db_password, [  
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // Activa el modo  
de errores para PDO.  
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC, // Define el  
modo de obtención predeterminado.  
    ]);  
  
} catch (PDOException $e) {  
    die($e->getMessage());  
}
```

Realizamos un try para capturar los errores que nos pueda arrojar la conexión a la base de datos. Creamos el objeto PDO con los datos necesarios y anteriormente declarados.

También le asignamos el modo error y que la respuesta de obtención de datos predeterminada sea “FETCH_ASSOC” (array asociativo).

index.php

El index en nuestro proyecto es la página para iniciar la sesión, lo más importante de este archivo es el manejo de sesiones y del formulario de inicio de sesión, veamos las partes más importantes del código.

Iniciar la sesión y la asignación de valores en la misma.

```
session_start();

if (!isset($_SESSION['isLogged'])) {
    $_SESSION['isLogged'] = 'unlogged';
} elseif (isset($_SESSION['isLogged']) && $_SESSION['isLogged'] != 'logged') {
    $_SESSION['isLogged'] = 'unlogged';
}
```

Iniciamos la sesión y en caso de que no exista en la misma “isLogged” su valor será “unlogged”, es decir, no ha iniciado sesión el usuario, además, por seguridad si existe pero su valor es diferente a “logged” pues la asignamos a “unlogged”.

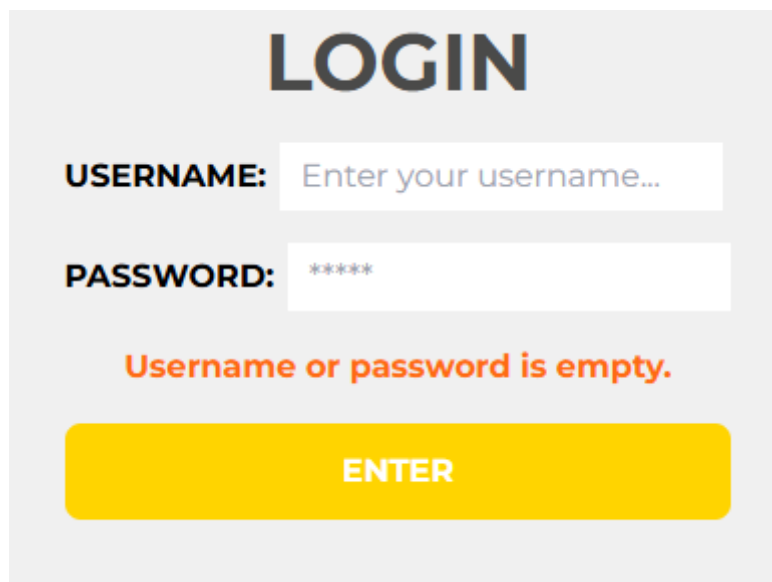
Posteriormente realizamos el registro de los datos del formulario por el método POST y dentro de esa petición comprobamos que los valores no estén vacíos, si lo están se añade un error a un array llamado “errors” los cuales si existe algún error se mostrará.

En caso de que existan datos enviados ejecutamos una consulta SQL para recuperar los datos de la base de datos con el mismo nombre y contraseña que los enviados, si se reciben datos iguales cambiamos el estado de la sesión “isLogged” a “logged” y además creamos otro valor en la sesión llamado “username” con el nombre de usuario que ha iniciado la sesión.

El código en cuestión es el siguiente:

```
$sql_user = "SELECT * FROM users WHERE username = '$username' AND  
password = '$password'";  
  
$gsent_user = $db_PDO->prepare($sql_user);  
// Ejecutar consulta  
$gsent_user->execute();  
// Obtenemos el resultado  
$result = $gsent_user->fetch(PDO::FETCH_ASSOC);  
if ($result) {  
    $_SESSION['isLoggedIn'] = 'logged';  
    $_SESSION['username'] = $username;  
    header('location: ./valorations.php');  
    die;  
} else {  
    $errors[] = "Username or password is incorrect.";  
}
```

Si no se encuentra ninguna coincidencia significa que el usuario o la contraseña es incorrecta.



LOGIN

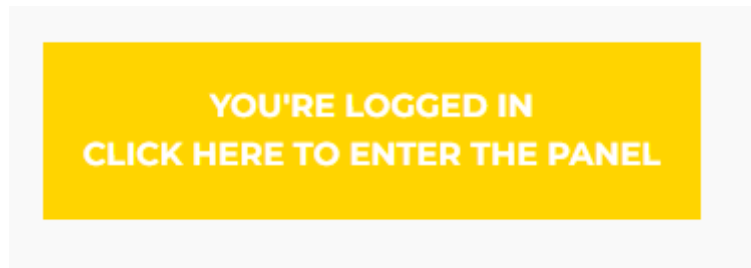
USERNAME:

PASSWORD:

Username or password is empty.

ENTER

Además, si el usuario ya ha iniciado sesión y vuelve a ingresar a index.php, le aparecerá un texto con un enlace para acceder al apartado de valoraciones.



script.js

En este script tratamos los datos que vienen desde vote.php (manejo de votos que se explicará más adelante), utilizando una función asíncrona para poder actualizar los datos en la parte del frontend sin tener que recargar la página, esto es posible gracias a **FETCH**.

Antes de nada, para evitar el envío del formulario sin que la función asíncrona haya recibido datos, debemos usar la función **“preventDefault();”** asignada al evento del click del botón del formulario en cuestión:

```
button.addEventListener('click', async function(event) {  
    event.preventDefault(); // Evita el envío tradicional del  
formulario
```

En este caso nosotros registramos los datos del formulario en variables de la siguiente manera:

```
const form = this.closest('form');  
const formData = new FormData(form);  
const voteInput = form.querySelector('input[name="vote"]');  
const productId = formData.get("productId");  
const averageRateElement =  
document.getElementById(`averageRate-${productId}`);
```

El objeto “FormData” nos permite registrar todos los datos de un formulario, en este caso el que hemos recogido en la variable “form”

Dentro del try que veremos a continuación, recogeremos la respuesta del fetch que viene de vote.php, transformando esta respuesta en formato json (ya que desde vote.php se envía de esta manera) y dependiendo de la respuesta pues actualizamos los contenedores de la información en el frontend o no, veamos:

```
try {
const response = await fetch('vote.php', {
    method: 'POST',
    body: formData
});
const result = await response.json(); // Obtiene la respuesta como JSON

if (result.success) {
responseDiv.innerHTML = `

${result.message}</p>`;
}


```

Si en la respuesta, el valor success está en true, pues en el “div” de respuesta (que se encuentra en valorations.php), añadiremos el mensaje que también viene dado en el json de la respuesta.

Por ejemplo, también actualizaremos el dato de la media de los votos del producto:

```
if (averageRateElement) {
console.log("Elemento encontrado:", averageRateElement); // Verifica
que se encuentre el elemento
averageRateElement.innerText = `Average: ${result.averageRate}`;
} else {
console.log("The average element was not found.");
}
```

El console.log lo utilizamos como desarrolladores para comprobar las respuestas.