DESARROLLO DE APLICACIONES WEB

17-11-2023

ROBERTO RODRÍGUEZ JIMÉNEZ

roberto.rodjim.1@educa.jcyl.es

Desarrollo Web en Entorno Servidor

Trabajar con bases de datos en PHP

Tarea 03

# 

Contenido

[Tarea online DWES03 2](#_Toc152265225)

[1.- Descripción de la tarea 2](#_Toc152265226)

[**Caso práctico** 2](file:///E:\cfgs\rodriguez_jimenez_roberto_DWES03_Tarea.docx#_Toc152265227)

[Enunciado 2](#_Toc152265228)

[2.- Información de interés 5](#_Toc152265229)

[Recursos necesarios 5](#_Toc152265230)

[Consejos y recomendaciones 5](#_Toc152265231)

[3.- Evaluación de la tarea 5](#_Toc152265232)

[Criterios de evaluación implicados 5](#_Toc152265233)

[¿Cómo valoramos y puntuamos tu tarea? 6](#_Toc152265234)

[RESOLUCIÓN 7](#_Toc152265235)

[Estructura del sitio 7](#_Toc152265236)

[Conexión con la base de datos 8](#_Toc152265237)

[Bootstrap 9](#_Toc152265238)

[listado.php 9](#_Toc152265239)

[Errores 11](#_Toc152265240)

[crear.php 13](#_Toc152265241)

[detalle.php 16](#_Toc152265242)

[update.php 17](#_Toc152265243)

[borrar.php 20](#_Toc152265244)

# Tarea online DWES03

**Título de la tarea**: Cesta de la compra.  
**Unidad 3**: Trabajar con bases de datos en PHP.  
**Ciclo formativo y módulo**: Desarrollo de Aplicaciones Web. Módulo DWES.  
**Curso académico**: …

## 1.- Descripción de la tarea

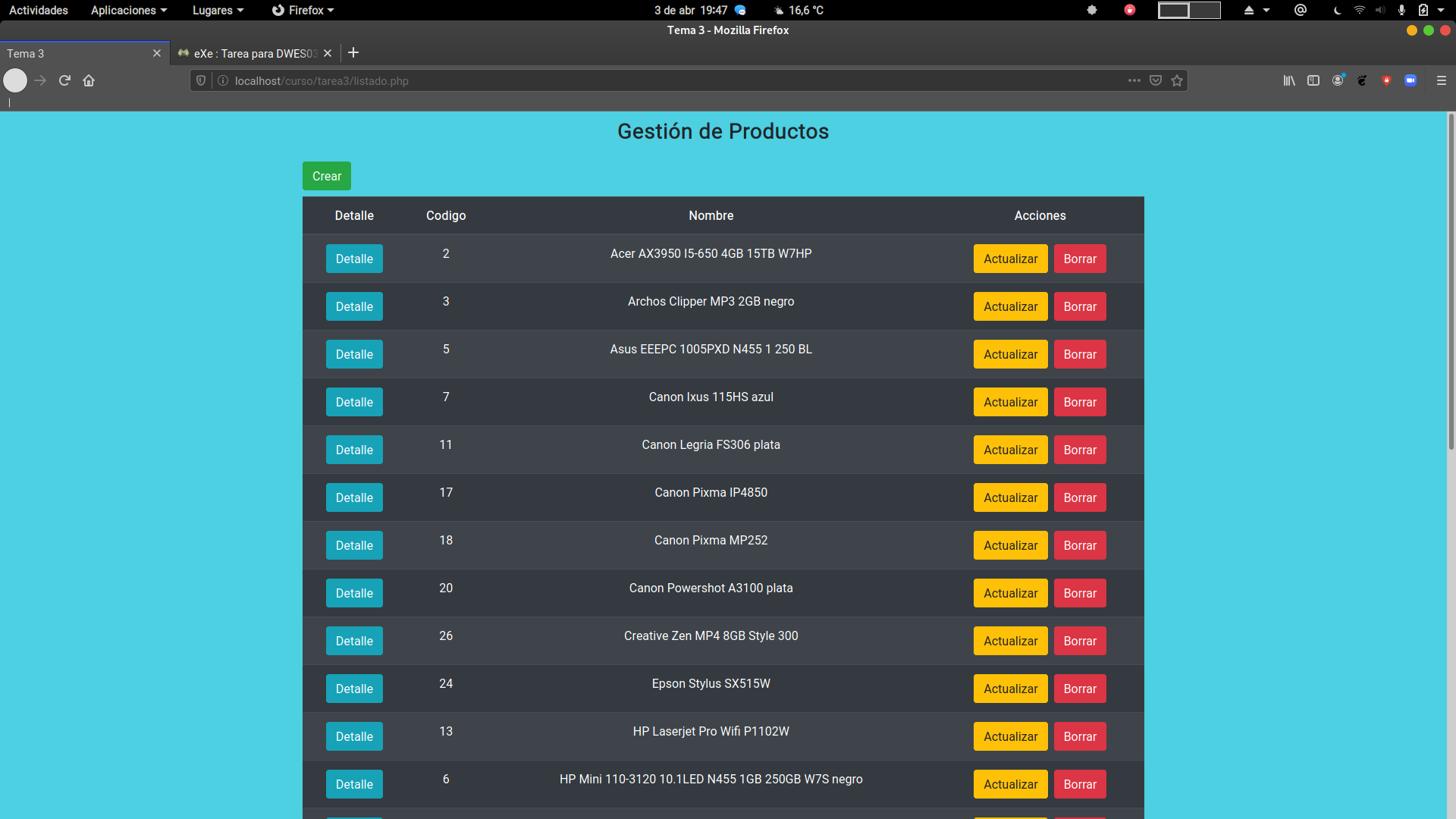
### **Caso práctico**

Carlos tiene un amigo con una tienda de informática, que ha decidido vender sus productos por Internet. Carlos piensa que con sus conocimientos de acceder a una base de datos desde la Web podría echarle una mano, y aunque es consciente de que aún le faltan herramientas para que la aplicación sea completamente funcional, se pone a hacer una primera implementación de esta.

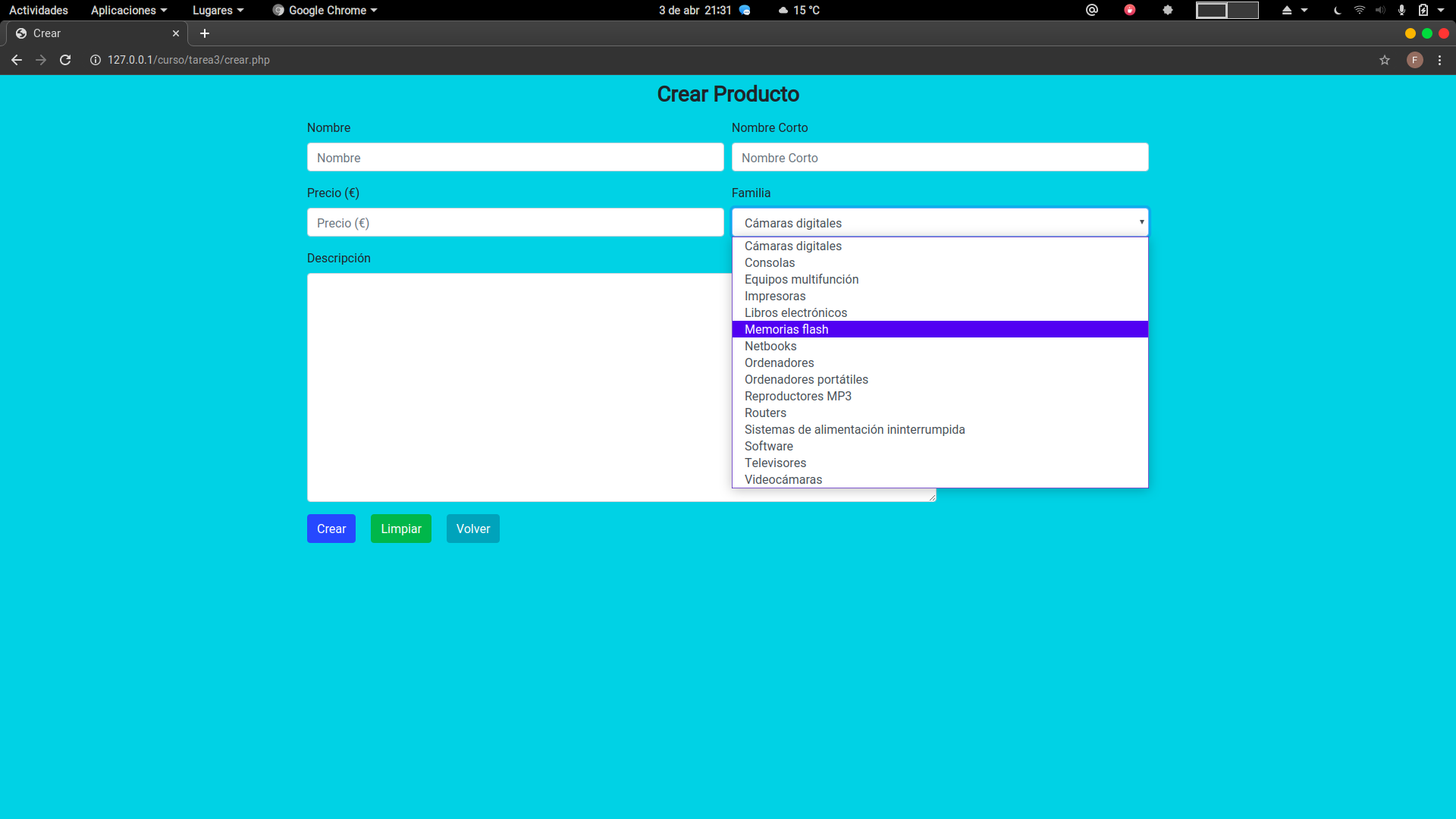
### Enunciado

Partiendo de la base de datos 'proyecto' usada en los ejemplos y ejercicios de la unidad , se trata de programar un CRUD (create, read, update, delete) que permita gestionar los registros de la tabla 'productos'. La aplicación se dividirá en 5 páginas:

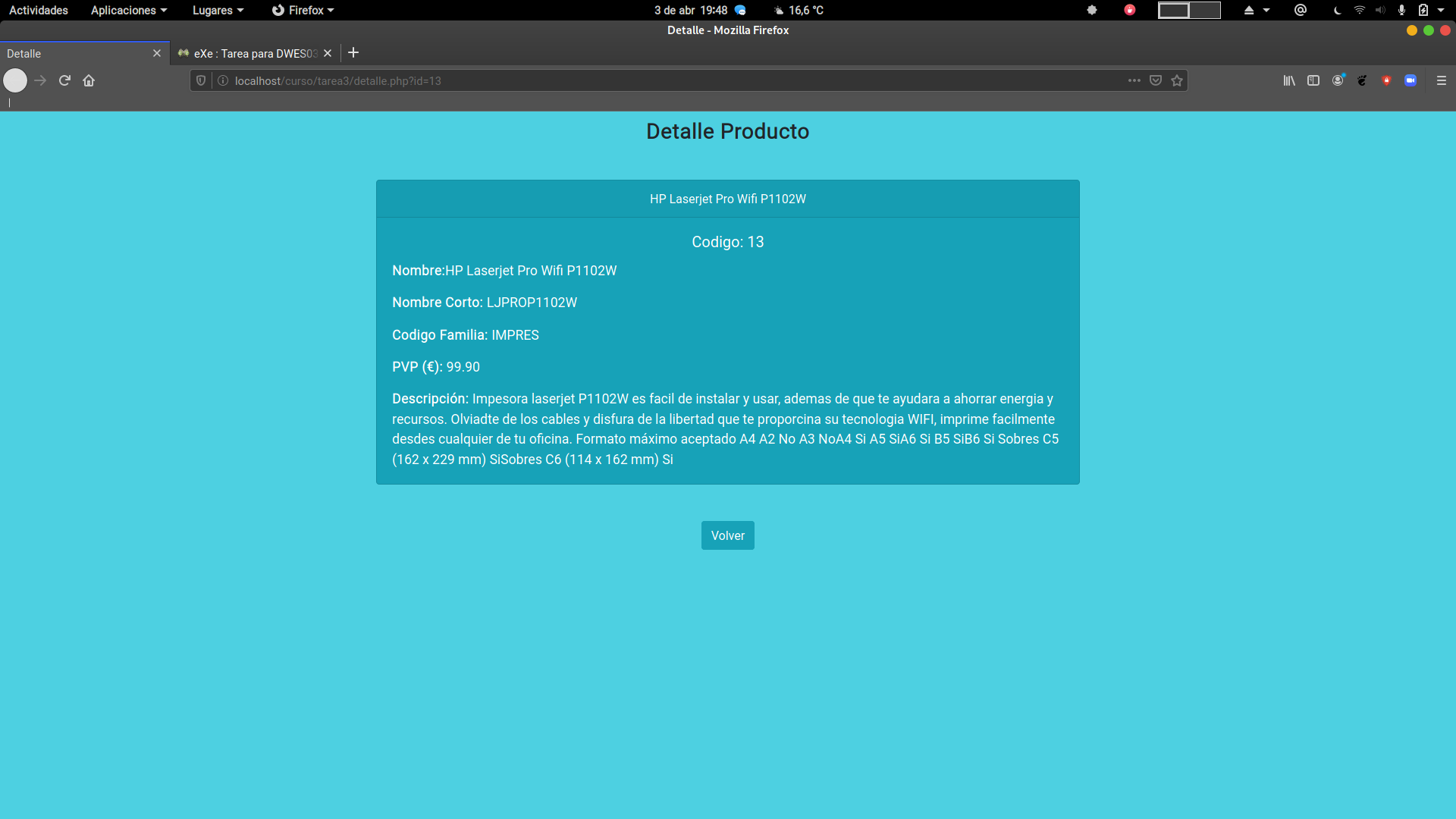
* listado.php. Mostrará en una tabla los datos código y nombre y los botones para crear un nuevo registro, actualizar uno existente, borrarlo o ver todos sus detalles.



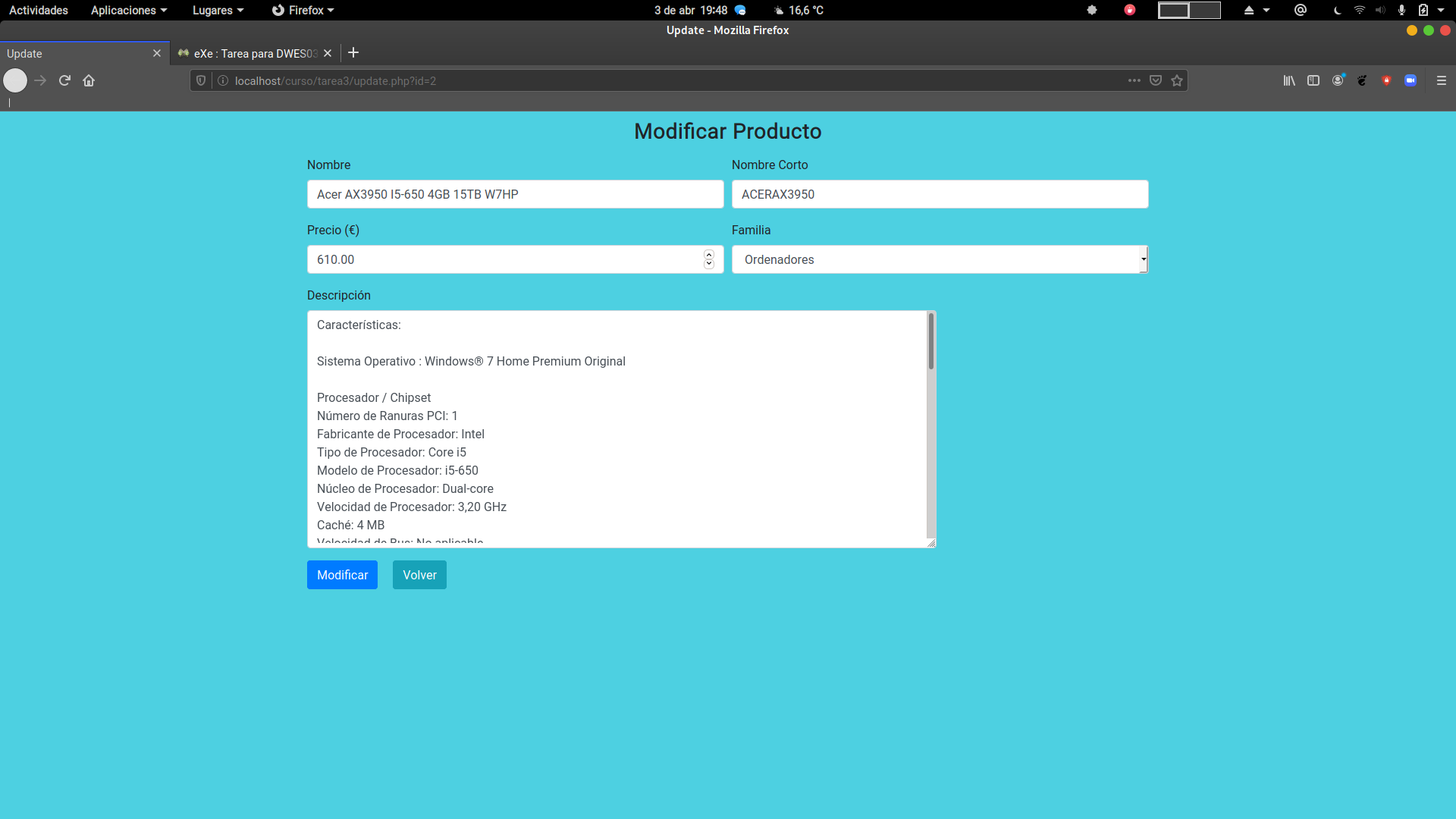
* crear.php. Será un formulario para rellenar todos los campos de productos (a excepción del id). Para la familia nos aparecerá un "select" con los nombres de las familias de los productos para elegir uno (lógicamente, aunque mostremos los nombres por formulario enviaremos el código).



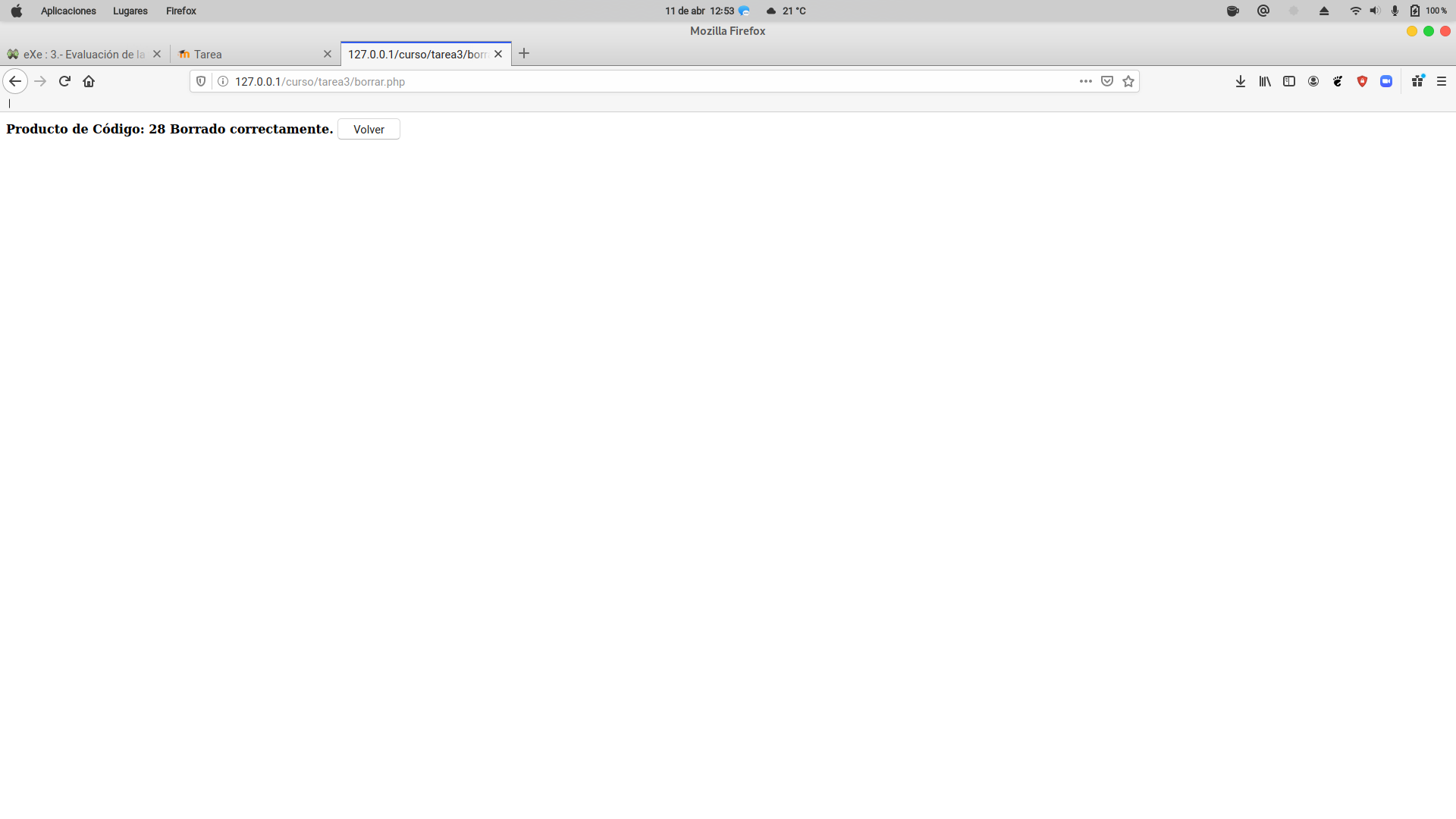
* detalle.php. Mostrará todos los detalles del producto seleccionado.



* update.php. Nos aparecerá un formulario con los campos rellenos con los valores del producto seleccionado desde "listado.php" incluido el select donde seleccionamos la familia.



* borrar.php. Será una página php con el código necesario para borrar el producto seleccionado desde "listado.php" un mensaje de información y un botón volver para volver a "listado.php".



Para acceder a la base de datos se debe usar PDO. Controlaremos y mostraremos los posibles errores. Para los estilos se recomienda usar *Bootstrap*.

Pasaremos el código de producto por "get" tanto para "detalle.php" como para "update.php". Utilizando en el enlace "detalle.php?id=cod" .En ambas páginas comprobaremos que esta variable existe, si no redireccionaremos a "listado.php" para esto podemos usar "header('Location:listado.php')"

## 2.- Información de interés

### Recursos necesarios

Ordenador con un entorno AMP, instalado y configurado, un entorno de desarrollo como VSC con las extensiones adecuadas. Navegador con acceso a Internet, para poder consultar el manual online de PHP y poder trabajar online Bootstrap.

### Consejos y recomendaciones

* Además del manual online de PHP, se recomienda dar libre acceso a Internet para la búsqueda de información.
* Para no repetir el código de la conexión a la base de datos en cada archivo, se recomienda crear el archivo conexion.php y utilizar require o require\_once cada vez que lo necesitamos.
* Para borrar un producto, crea un formulario con el action apuntando a "borrar.php" y pasa por un campo oculto el código del producto a borrar.
* Recuerda que en el "option" del HTML "<option value="v">Nombre</option>", "v" es el valor que pasamos, "Nombre" es lo que mostramos en la lista desplegable.

## 3.- Evaluación de la tarea

### Criterios de evaluación implicados

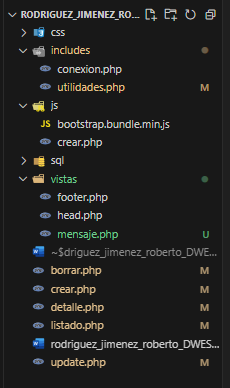
* Se han analizado las tecnologías que permiten el acceso mediante programación a la información disponible en almacenes de datos.
* Se han creado aplicaciones que establezcan conexiones con bases de datos.
* Se ha recuperado información almacenada en bases de datos.
* Se ha publicado en aplicaciones Web la información recuperada.
* Se han utilizado conjuntos de datos para almacenar la información.
* Se han creado aplicaciones Web que permitan la actualización y la eliminación de información disponible en una base de datos.
* Se han utilizado transacciones para mantener la consistencia de la información.
* Se han probado y documentado las aplicaciones.

### ¿Cómo valoramos y puntuamos tu tarea?

|  |  |
| --- | --- |
| **Rúbrica de la tarea** | |
| En "listado.php", generar el listado de los productos y los botones funcionando, el botón borrar será el submit de un formulario cuyo action debe ser el indicado | 2 puntos |
| En "crear.php", generar el formulario funcional de creación del producto | 2 puntos |
| En "update.php" cargar el formulario y que sea funcional con todos los campos del producto en cuestión rellenos. | 2 puntos |
| Página "borrar.php" funcional con mensaje y botón para volver. | 1 Punto |
| Página "detalle.php", mostrando el detalle del producto seleccionado y el botón volver funcionando | 1 Punto |
| La lista de selección **familia** de la página "update.php" tiene seleccionada la familia del producto. | 0.75 Puntos |
| Se utilizan correctamente las excepciones para controlar los posibles errores | 0.5 Puntos |
| Introducir comentarios, legibilidad del código y diseño | 0.75 Puntos |

## RESOLUCIÓN

### Estructura del sitio



* *css*
  + bootstrap.min.css: archivo css de Bootstrap, para tenerlo en local.
* *includes*: directorio para guardar los archivos php que serán llamados desde otros.
  + *conexion.php*: guarda los datos de la conexión con la base de datos.
  + *utilidades.php*: contiene funciones que son usadas en las páginas.
* *js*
  + *bootstrap.bundle.min.js*: archivo javascript de Bootstrap.
* *sql*
  + *proyecto\_datos.sql*: backup con los datos.
  + *proyecto.sql*: backup con la estructura de la base de datos.
* *vistas:* para los archivos que contienen partes de html.
  + *footer.php*: footer común para cada una de las páginas.
  + *head.php*: header para las páginas.
  + *mensaje.php* muestra un mensaje a modo de confirmación en las operaciones o errores.
* *borrar.php*
* *crear.php*
* *detalle.php*
* *listado.php*
* *update.php*

### Conexión con la base de datos

Para la conexión con la base de datos se usa *PDO* y el driver para *mysql*.

Los datos de configuración y conexión se guardan en el archivo *includes/conexión.php*. En este archivo se guardan los datos y se inicializa el *dsn*.

<?php

// La conexión con la base de datos se hace mediante PDO

// Datos de conexión a la base de datos

$host = "localhost";

$port = 3306;

$db = "proyecto";

$user = "gestor";

$pass = "secreto";

// Data Source Name (Nombre de Origen de los Datos)

$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";

Código con los datos de conexión a la base de datos.

La conexión se realiza desde cada uno de los archivos llamando a *conexión.php/getConexion()*

$cnx = getConexion($dsn, $user, $pass);

Llamada a *getConexion()* desde una página.

function getConexion($dsn, $user, $pass)

{

    // Intentar la conexión a la base de datos

    try {

        $cnx = @new PDO($dsn, $user, $pass);

        $cnx->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

        return $cnx;

    } catch (PDOException $e) {

        die(

            "<div class='container my-3'>

            <div class='row justify-content-center'>

                <div class='col text-center'>

                    <h2>No se ha podido conectar con la base de datos</h2>

                </div>

            </div>

        </div>"

        );

    }

}

Código de la función *getConexion()* que establece una conexión con la base de datos.

### Bootstrap

Para la presentación de las páginas se usa el *framewok Bootstrap* en su versión 5.3.  
He decidido incorporar los archivos css y js en local para no depender de una conexión a internet, dado que durante el desarrollo de la tarea no siempre tendré acceso a internet.

### listado.php

Muestra el listado de los productos, tal y como se ve en la propuesta del ejercicio, en una tabla.  
Los datos se cargan en el momento de mostrar las filas, envolviendo el código en un *try-catch* que capture los errores.

<tbody class="table-group-divider">

<?php

try {

  $listado = $cnx->query("SELECT id, nombre FROM productos ORDER BY nombre");

$producto = $listado->fetch(PDO::FETCH\_OBJ);

  while ($producto != null):

?>

<tr>

<?php set\_error\_handler("gestionarError"); ?>

  <td>

    <a href="detalle.php?id=<?=$producto->id?>" class="btn btn-success bg-info">Detalle</a>

   </td>

<td><?= $producto->id ?></td>

  <td><?= $producto->nombre ?></td>

    <td class="d-flex justify-content-around">

    <a href="update.php?id=<?=$producto->id?>" class="btn btn-warning">Actualizar</a>

      <form action="borrar.php" method="post">

      <input type="hidden" name="id" id="id" value="<?=$producto->id?>">

        <button type="submit" class="btn btn-danger">Borrar</button>

     </form>

    </td>

<?php restore\_error\_handler(); ?>  
  </tr>

<?php

$producto = $listado->fetch(PDO::FETCH\_OBJ);

 endwhile;

} catch (PDOException $e) {

  $mensaje = 'Ocurrió algo inesperado y no se han podido recuperar los articulos';

require\_once('vistas/mensaje.php');

}

?>

</tbody>

Código que muestra el resultado de una consulta capturando las excepciones.

Para ver el detalle o actualizar un producto, se pasa el id por GET y, para borrar, usa un formulario que llama a *borrar.php* pasándole el id del producto por PSOT.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Detalle del producto

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Actualizar un producto

Captura de pantalla de un videojuego

Descripción generada automáticamente

Detalle del listado

#### Errores

En el listado puede darse el caso de que la consulta no esté bien formada y se arroje una excepción o un *warning*.  
Forzamos un error en la declaración de la consulta sql.

$listado = $cnx->query("SELECT id nombre FROM productos ORDER BY nombre");

Y vemos el resultado

Captura de pantalla de un celular

Descripción generada automáticamente

Especificamos el mensaje con *set\_error\_handler()* para lo cual he creado una función *gestionarError()* en *includes/utilidades.php*

Pantalla de un video juego

Descripción generada automáticamente con confianza media

Personalización de errores mediante set\_error\_handler

$producto = $listado->fetch(PDO::FETCH\_OBJ);

  while ($producto != null): ?>

  <tr>

  <?php set\_error\_handler("gestionarError"); ?>

. . .

  <?php restore\_error\_handler(); ?>

  </tr>

  <?php

      $producto = $listado->fetch(PDO::FETCH\_OBJ);

  endwhile;

Forzamos ahora una excepción:

$listado = $cnx->query("SELECT id, nombre\_ FROM productos ORDER BY nombre");

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Error generado en la consulta sql

Y la gestionamos con *try-catch*

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

El mismo error gestionando la excepción

### crear.php

A crear se accede mediante un botón en *listado.php*.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Botón para ir a "crear.php" en la página que muestra el listado

Al ejecutarse el archivo se comprueba si se ha recibido algún dato por POST.  
Si POST está vacío, simplemente se muestra el formulario.  
El formulario se valida primero en el propio cliente.

El *select* que recoge las familias de productos se obtiene mediante una consulta, capturando los posibles errores mediante *try-catch*.

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

Formulario para crear productos

En caso de recibir datos por POST, el primer paso es comprobar que estén todos los datos y que no estén en blanco.  
Esta operación se realiza en la función *validarDatosCompletos()* dentro del archivo *includes/utilidades.php*.

En caso de que la función encuentre errores, sale de la aplicación mostrando un mensaje:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Error al guardar un dato, en este caso se ha dejado el PVP en 0.

Si los datos se han recibido correctamente, el siguiente paso es validar el formato.   
Igualmente, la función que lo comprueba está en *includes/utilidades.php* y es la función *validarFormato()*.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

El error está en los guiones del nombre corto.

Finalmente, si todo es correcto, se ejecuta la consulta y se muestra un feedback de confirmación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Datos correctos

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Pantalla de confirmación

### detalle.php

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Pantalla con los detalles de un producto

El funcionamiento de esta pantalla es bastante simple: se realiza la consulta a la base de datos y se muestran.

La consulta y su tratamiento están envueltas en la cláusula *try-catch* para capturar excepciones. Además, se evita que muestren errores en el caso de que algún campo no esté disponible:

<strong>Nombre:</strong> <?= @$caracteristicas->nombre ?>

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Error al buscar un id no válido

La gestión de la excepción se implementa en *mensaje()*, en el archivo *vistas/vistas.php*.

En el caso de que no se indique un id, se redirige a *listado.php*.

if (!isset($\_GET['id'])) {

header('Location:listado.php');

}

### update.php

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Pantalla de actualización de un producto

La pantalla que se muestra es la misma que para *crear.php* pero, en este caso, se realiza previamente una consulta a la base de datos para obtener el producto recibido por GET.  
Si no se recibe un id, se redirecciona hacia *listado.php*.

Al cargarse el script, se comprueba si existe un id recibido por GET y hay datos en POST.  
En caso de cumplirse las condiciones, se siguen los mismos pasos que se dan en *crear.php*

Todo el bloque que muestra la información está incluido en un bloque *try-catch* para capturar las excepciones.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Feedback tras un error en la conexión a la base de datos

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Mensaje de error para el formato de los datos: 0€

El *select* del formulario ejecuta su propia consulta, también dentro de un *try-catch*, y compara el id del producto leído con el recogido en el campo *id* del producto detallado.

<select name="familia" id="familia" class="form-select" required>

  <?php

  try {

    // Consulta a la base de datos

    $listado = $cnx->query("SELECT \* FROM familias ORDER BY nombre");

    // Obtenemos los productos como objetos

    $familia = $listado->fetch(PDO::FETCH\_OBJ);

  while ($familia != null):

  ?>

  <option value="<?= $familia->cod ?>" <?php if ($familia->cod == $caracteristicas->familia)

    echo 'selected'; ?>>

  <?= $familia->nombre ?>

  </option>

  <?php

  $familia = $listado->fetch(PDO::FETCH\_OBJ);

    endwhile;

  } catch (PDOException $e) { echo '<option value="">Error</option>'; }

  ?>

</select>

Código para cargar el *select* con los datos de las familias de productos.

En el caso de producirse un error, se muestra una única opción en el selector con el texto *“Error”* y un valor vacío.  
Forzamos un error en la consulta a la base de datos:

$listado = $cnx->query("SELECT \* FROM familias\_ ORDER BY nombre");

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Error al cargar los datos de las familias de producto

Si todo es correcto, se actualiza el producto.

Interfaz de usuario gráfica

Descripción generada automáticamente

Producto actualizado correctamente.

### borrar.php

Cuando se carga el script *borrar.php* lo primero que se comprueba es que exista un id que se pueda borrar:

if (empty($\_POST) {

    header("Location:listado.php");

}

Si se produjese cualquier error, se muestra este y se sale de la aplicación.  
Para ello se hace uso del la función *ejecutarConsulta()* en *includes/conexion.php*, que a su vez muestra el mensaje de error a través de *mostrarExcepcion()* que está en el archivo *includes/utilidades.php*.

Forzamos el error:

$consulta = 'DELETE FROM productos WHERE id = AB';

Y tratamos de borrar cualquier producto:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Mensaje de error al tratar de eliminar un producto incorrecto.

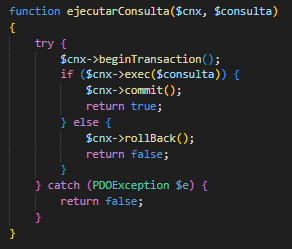
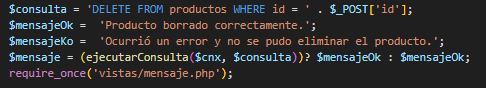
Recuperamos el código correcto y repetimos la operación:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Producto borrado correctamente.

*ejecutarConsulta()* devuelve o *false* dependiendo de si se realizó o no la operación.  
A continuación se carga la vista mostrando el mensaje.



### Directorio includes y archivos

#### conexión.php

Contiene los datos de conexión con la base de datos y funciones que trabajan directamente con ella.