

ACCESO A DATOS DESDE PHP A MYSQL

PHP es uno de los lenguajes de programación más utilizados y de alta vigencia. Por ello es importante su aprendizaje de forma nativa, sobre todo, antes de aventurarnos al uso de algún framework como CodeIgniter o Laravel. Para esta actividad utilizaremos el paradigma de la programación orientado a objeto y la arquitectura Modelo Vista Controlador.

Los procesos por desarrollar	¿Qué debo saber?	¿Qué software necesito?
<ul style="list-style-type: none">• Inserción de datos• Búsqueda• Lista• Paginación• Modificación• Eliminación (lógica a través de estados)	Se requiere conocimientos básicos, fundamentos de: <ul style="list-style-type: none">• HTML5• CSS3• Javascript• jQuery• SQL• PHP	<ul style="list-style-type: none">• XAMPP• Navegador de Internet• Postman (se explicará en su momento)• Visual Studio Code o algún editor similar• SQLYog / MySQL Worbench o similar

Antes de iniciar...

En el apartado anterior sugerí conocimientos de HTML5, CSS3, JS entre otras tecnologías, sin embargo, es muy importante (incluso más que lo señalado en esta lista) la comprensión de PROGRAMACIÓN ORIENTADO A OBJETOS, sino es esta su situación ¡No se preocupe!, existen incontable recursos en la Web que seguro le será de utilidad, a continuación sugiero una revisión obligatoria en caso tenga problemas con esto, caso contrario, si siente que domina muy bien estos conceptos, entonces podrá omitir esta breve capacitación.

Fuente: YouTube
Canal: EDTeam
Duración: 19:57

[Acceder a contenido externo](#)



Fuente: YouTube
Canal: The Coder Cave Esp
Duración: 08:56

[Acceder a contenido externo](#)



¡Ahora sí, ya estamos listos para iniciar!

ÍNDICE DE CONTENIDOS

CAPÍTULO I: Base de datos y estructura del proyecto

Para este proyecto, crearemos una pequeña aplicación que gestione una lista de productos (electrónicos, informáticos o similares). Vamos a crear una base de datos llamada “informática” (aunque puede llamarla de otra forma) y dentro crearemos 3 tablas, tal como se muestra en el siguiente diagrama.

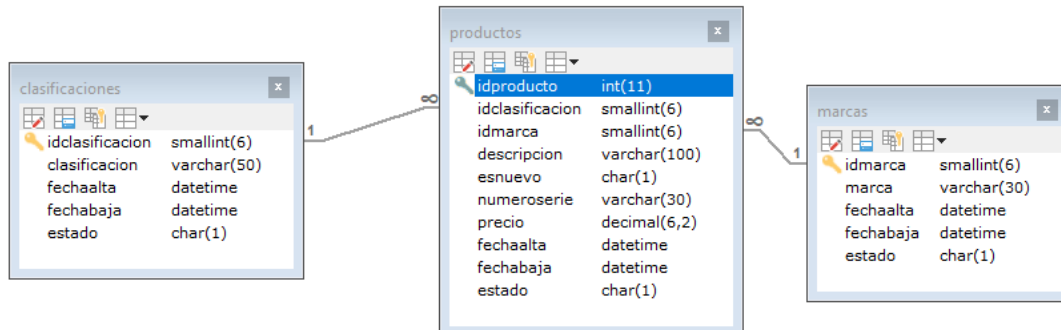


Imagen #01 – Modelo relacional de base de datos del proyecto

Procedimientos:

1. Iniciamos el servidor XAMPP, luego abrimos SQLYog o cualquier otro entorno de base de datos, e ingresamos las siguientes instrucciones:

```
Query | basedatos.sql | +
Autocomplete: [Tab]->Next Tag, [Ctrl+Space]->List All Tags, [Ctrl+Enter]->List Matching Tags, [Ctrl+Shift+]
1  -- 1. Crear BD
2  CREATE DATABASE informatica;
3  USE informatica;
4
5  -- 2 Tablas
6  CREATE TABLE marcas(
7      idmarca      SMALLINT AUTO_INCREMENT PRIMARY KEY,
8      marca        VARCHAR(30) NOT NULL,
9      fechaalta    DATETIME NOT NULL DEFAULT NOW(),
10     fehabaja      DATETIME NULL,
11     estado        CHAR(1) NOT NULL DEFAULT '1',
12     CONSTRAINT uk_marca_mat UNIQUE(marca)
13 )ENGINE = INNODB;
14
```

Imagen #02: Creación de la base de datos y la tabla marcas



¡NO PASES POR ALTO!

- El campo **fechaalta** representa el momento (día y hora de la creación del registro), el sistema lo asigna automáticamente a través de la función **NOW()**
- El campo **fehabaja** representa el momento (día y hora de la desactivación/eliminación lógica del registro), por defecto irá NULL
- El campo **estado** determina si el registro está activo (1) o no (0)
- La restricción **UNIQUE** impide que se creen dos registros iguales

2. Agregamos algunos registros a la tabla marcas

```
16 INSERT INTO marcas (marca) VALUES
17     ('Samsung'),
18     ('HP'),
19     ('Micronics');
20
21 SELECT * FROM marcas;
```

Imagen #03: Inserción de datos en marcas

3. Continuamos con la tabla clasificaciones (no asignar tilde en su creación)

```
24 CREATE TABLE clasificaciones(  
25     idclasificacion SMALLINT AUTO_INCREMENT PRIMARY KEY,  
26     clasificacion VARCHAR(50) NOT NULL,  
27     fechaalta DATETIME NOT NULL DEFAULT NOW(),  
28     fechabaja DATETIME NULL,  
29     estado CHAR(1) NOT NULL DEFAULT '1',  
30     CONSTRAINT uk_clasificacion_cla UNIQUE (clasificacion)  
31 )ENGINE = INNODB;  
32  
33  
34 INSERT INTO clasificaciones (clasificacion) VALUES  
35     ("Memoria RAM"),  
36     ("Fuente de poder"),  
37     ("Mouse"),  
38     ("Monitor");  
39  
40 SELECT * FROM clasificaciones;
```

Imagen #04 – Creación de la tabla clasificaciones e inserción de datos

4. Tabla productos, esta es especial, ya que utilizará (a través de las claves foráneas) los datos de las tablas anteriormente creadas, marcas y clasificaciones.

```
43 CREATE TABLE productos(  
44     idproducto INT AUTO_INCREMENT PRIMARY KEY,  
45     idclasificacion SMALLINT NOT NULL,  
46     idmarca SMALLINT NOT NULL,  
47     descripcion VARCHAR(100) NOT NULL,  
48     esnuevo CHAR(1) NOT NULL DEFAULT 'S',  
49     numeroserie VARCHAR(30) NULL,  
50     precio DECIMAL(6,2) NOT NULL,  
51     fechaalta DATETIME NOT NULL DEFAULT NOW(),  
52     fechabaja DATETIME NULL,  
53     estado CHAR(1) NOT NULL DEFAULT '1',  
54  
55     CONSTRAINT fk_idclasificacion_cla FOREIGN KEY(idclasificacion)  
56         REFERENCES clasificaciones (idclasificacion),  
57  
58     CONSTRAINT fk_idmarca_cla FOREIGN KEY (idmarca) REFERENCES marcas (idmarca)  
59 )ENGINE = INNODB;
```

Imagen #05 – Creación de la tabla productos



¡NO PASES POR ALTO!:

- Esta tabla posee dos claves foráneas (idclasificacion – idmarca), de no haber creada estas tablas anteriormente será imposible la construcción de la tabla productos.
- Al momento de construir la tabla, notará se agregaron tabulaciones entre el nombre del campo, tipo y restricción, si bien, esta forma de escribir el código no es obligatoria, nos brinda mayor legibilidad y facilidad de lectura, por lo que se aconseja hacer.

5. Para verificar que la tabla está creada de manera correcta, procederemos a insertar en ella algunos registros a través de la siguiente instrucción.

```
61 INSERT INTO productos (idclasificacion, idmarca, descripcion, precio) VALUES  
62     (1,1, '16 Gb. DDR5 5000Mhz',570),  
63     (2,2 , '500 Watts real',170);
```

Imagen #06 – Inserción de datos en la tabla productos

6. Verificamos la inserción de datos a través de una consulta

```
65 SELECT * FROM productos;
```

1 Result					
(Read Only)					
idproducto	idclasificacion	idmarca	descripcion	esnuevo	numeroserie
1	1	1	16 Gb. DDR5 5000Mhz	S	(NULL)
2	2	2	500 Watts real	S	(NULL)

Imagen #07 – Verificación de datos

¡Buen trabajo!

Llegado hasta este punto, ya tenemos una base de datos, tablas y registros de prueba con los que podamos trabajar. Ahora, crearemos la carpeta y subdirectorios de nuestro proyecto. Siga atentamente los siguientes procedimientos:

Crea tu proyecto en la carpeta destinada para XAMPP server:

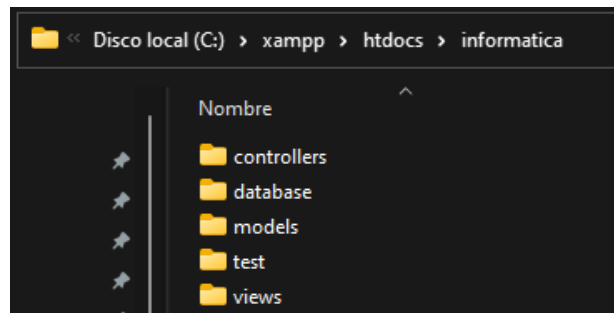


Imagen #08 – Estructura del proyecto



¡NO PASES POR ALTO!:

- Utiliza los nombres de los directorios en minúsculas
- No utilice tilde ni ñ en el nombre de los directorios, ya que luego los proyectos al ser migrados al servidor de producción para su aplicación, tendrán problemas
- El directorio “test” es solo para la realización de algunas pruebas tal como su nombre lo indica.
- Guarda el archivo de base de datos (script SQL) desarrollado en este capítulo en la carpeta [database](#), así mantendrá todo su proyecto bien organizado

CAPÍTULO II: Clase de conexión y nuestro primer modelo

Nuestra aplicación tendrá acceso a un servidor (local), a una base de datos y gestionará los datos contenidos en las tablas. Todo este trabajo sería imposible, sino contamos con un “mecanismo” que nos permita acceder a la información.

Clase Conexión:

Procedemos a crear la clase conexión dentro de la carpeta **models** de nuestro proyecto con el nombre **Conexion.php**.

```
Conexion.php X
> Conexion.php
1  <?php
2
3  class Conexion{
4
5      //Atributo que almacena la conexión
6      protected $pdo;
7
8      //Método que accede al server > db
9      private function Conectar(){
10
11      }
12
13      //Método que retorna / compartirá la conexión
14      public function getConexion(){
15
16      }
17
18  }
19
20  ?>
```

Imagen #09 – clase conexión “un primer vistazo”



¡NO PASES POR ALTO!:

- Las clases se escriben siempre con el primer carácter en mayúscula
- El nombre del archivo y el nombre de la clase por convención siempre deben coincidir.
- Se deben definir primero los atributos, luego los métodos
- El método **Conectar** no podrá ser utilizado desde otros componentes de la aplicación al estar definido como “private” a diferencia de **getConexion** que está “public” (a esto se le conoce como *modificadores de acceso*)
- Utilice comentarios siempre, le ayudará a entender la lógica de su aplicación.

Ahora agregamos las siguientes instrucciones dentro del método **Conectar()**:

```
//Método que accede al server > db
private function Conectar(){
    $cn = new PDO("mysql:host=localhost;port=3306;dbname=informatica;charset=utf8","root","");
    return $cn;
}
```

Imagen #10 – Método Conectar con la cadena de conexión

¿Qué es PDO?

- La extensión Objetos de Datos de PHP (PDO por sus siglas en inglés) define una interfaz ligera para poder acceder a bases de datos en PHP.
- PDO proporciona una capa de abstracción de acceso a datos, lo que significa que, independientemente de la base de datos que se esté utilizando, se emplean las mismas funciones para realizar consultas y obtener datos

Fuente:

<https://www.php.net/manual/es/intro.pdo.php>

Sobre el constructor:

```
$cn = new PDO("cadena_conexion", "nombre_usuario", "clave_acceso");
```

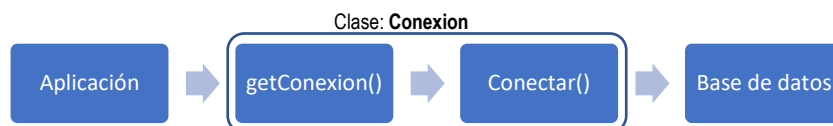
Cadena de conexión PDO	
mysql	Representa al gestor de base de datos al que nos vamos a conectar
host	Dirección de nuestro servidor
port	Número de puerto (rara vez este valor va a cambiar)
dbname	Nombre de nuestra base de datos
charset	Codificación (soporte para español y su conjunto de caracteres)

Método getConnection()

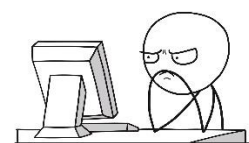
```
15 //Método que retorna / compartirá la conexión
16 public function getConnection(){
17     try{
18         //Almacenamos la conexión en el atributo 'pdo'
19         $pdo = $this->Conectar();
20
21         //Controlar excepciones
22         $pdo->setAttribute(PDO::ATTR_ERRMODE ,PDO::ERRMODE_EXCEPTION);
23
24         //Retornamos la conexión
25         return $pdo;
26     }
27     catch(Exception $e){
28         die($e->getMessage());
29     }
30 }
```

Imagen #11 – Método getConnection() estructura completa

Para simplificar, esto es lo que sucederá...



Cuando un componente de la **aplicación** requiera acceder a la información, utilizará nuestra clase **Conexion** a través de su método **getConnection**, el cual solicita internamente al método **Conectar** los accesos a la **base de datos**. Se que esta descripción es bastante básica para lo que realmente está pasando por detrás, pero para iniciar será un buen concepto que deberá tener presente.



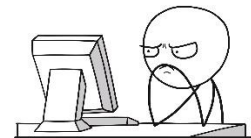
Ahora construiremos nuestro primer MODELO, ¡manos a la obra!

Si revisaste el vídeo que compartí en la primera parte de este material, entenderás lo importante que es el comprender la arquitectura MVC. Construiremos nuestro primer modelo, el cual nos servirá como referencia para todo el proyecto a realizar. No olvides: el modelo contiene toda la lógica que permitirá a la aplicación acceder a nuestros datos, y compartírselos a través del controlador con la vista.

```
Producto.php x
> Producto.php
1 <?php
2
3 //Requerimos acceso a la clase conexión
4 require_once "Conexion.php";
5
6 //La clase Producto será subclase de Conexion
7 class Producto extends Conexion{
8
9     //Objeto que almacena la conexión que obtenemos
10    private $acceso;
11
12    //Constructor
13    public function __construct(){
14        $this->acceso = parent::getConexion();
15    }
16
17    public function listarProductos(){
18        try{
19
20        }
21        catch(Exception $e){
22            die($e->getMessage());
23        }
24    }
25
26 } //Fin de la clase
27
28 ?>
```

Imagen #12 – La clase Producto, un primer vistazo

La herencia (concepto de POO), le permitirá a nuestra clase Producto, utilizar el método `getConexion()` de la clase Conexion. El constructor (concepto de POO), es un método que se desencadena cuando la clase es instanciada. Todos los métodos que iremos construyendo incorporan la estructura `try catch` (manejador de excepciones).



```
17 public function listarProductos(){
18     try{
19         //Preparamos la consulta
20         $consulta = $this->acceso->prepare("SELECT * FROM productos");
21
22         //Ejecutamos la consulta
23         $consulta->execute();
24
25         //Almacenamos el resultado de la consulta en $datos
26         //fetchALL = todos los registros
27         //FETCH_ASSOC = constante que representa a array asociativo
28         $datos = $consulta->fetchALL(PDO::FETCH_ASSOC);
29
30         //Devolver los datos
31         return $datos;
32     }
33     catch(Exception $e){
34         die($e->getMessage());
35     }
36 }
```

Imagen #13 – Instrucciones del método listar (los comentarios describen cada sección)

CAPÍTULO III: Controladores e instalación de aplicación Postman

Iniciaremos con la instalación de la aplicación Postman, la cual es de gran utilidad cuando desarrollamos aplicaciones Web, móviles o servicios en la nube y necesitamos de una herramienta que permita realizar peticiones de una manera simple y organizada.

Procedimientos:

1. Ingresar a la Web oficial <https://www.postman.com/>
2. Seleccione el icono que corresponda con el sistema operativo que está utilizando, en nuestro caso Windows

Download the desktop app



3. Clic en el botón que indica su sistema operativo para descargar (150 Mb.)

The Postman app

Download the app to get started with the Postman API Platform.

Windows 64-bit

4. Luego de la descarga, proceda a ejecutar la aplicación pulsando doble clic sobre su ejecutable. Se solicitará iniciar sesión, puede crear una cuenta en postman.com o bien, utilizar su cuenta de Gmail.

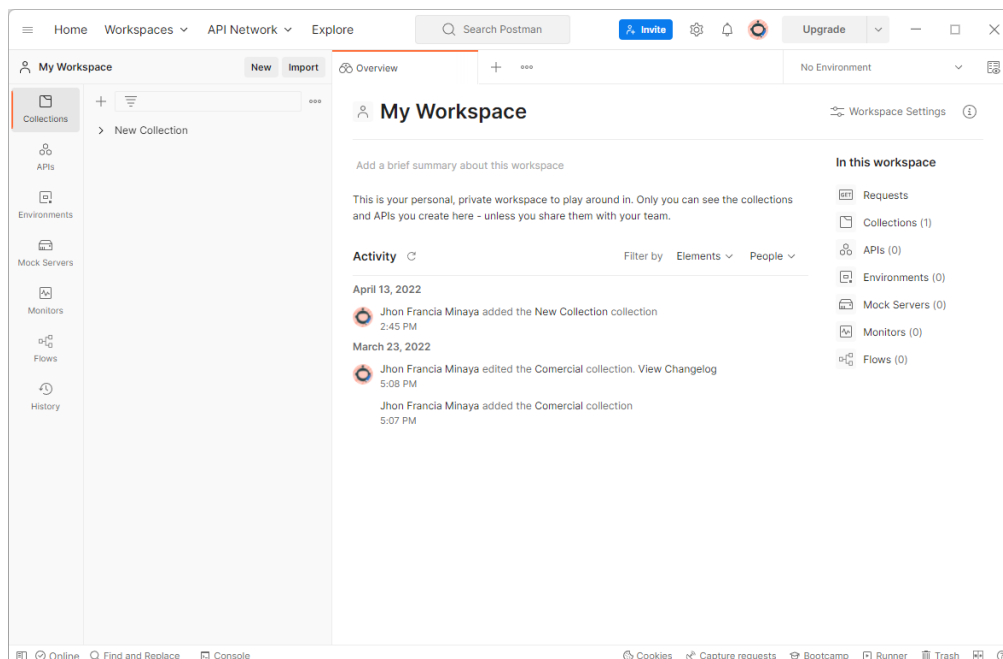


Imagen #14 – Pantalla principal aplicación Postman

Procedemos a crear nuestro primer controlador en la carpeta **controller** de nuestro proyecto, este se llamará **producto.controller.php**

```
producto.controller.php X
> producto.controller.php
1  <?php
2
3  require_once '../models/Producto.php';
4
5  if (isset($_GET['operacion'])){
6
7      $producto = new Producto();
8
9      if ($_GET['operacion'] == 'listarProductos'){
10         echo json_encode($producto->listarProductos());
11     }
12 }
13
14
15 ?>
```

Imagen #15 – Controlador para productos con la funcionalidad de responder una lista de datos



¡NO PASES POR ALTO!

- En el controlador necesitamos del modelo, es por ello iniciamos incorporándolo a nuestro espacio de trabajo.
- La función **isset()** verifica si existe un objeto, en este caso, si hay alguna petición/consulta llamada operación enviada por GET.
- Si se verifica que existe la consulta, se procede a crear la instancia de la clase **Producto**, ahora solo basta saber que operación en curso es la que atenderemos, por ello hacemos una segunda condición y devolvemos como respuesta la lista de productos como JSON.



JSON, cuyas siglas significan JavaScript object notation, en español se traducen como, notación de objetos de JavaScript, es un formato de intercambio de datos que resulta muy fácil de leer y escribir para los programadores y sencillo de interpretar y crear para las máquinas.

Flujo de trabajo (inicia con la solicitud del usuario)

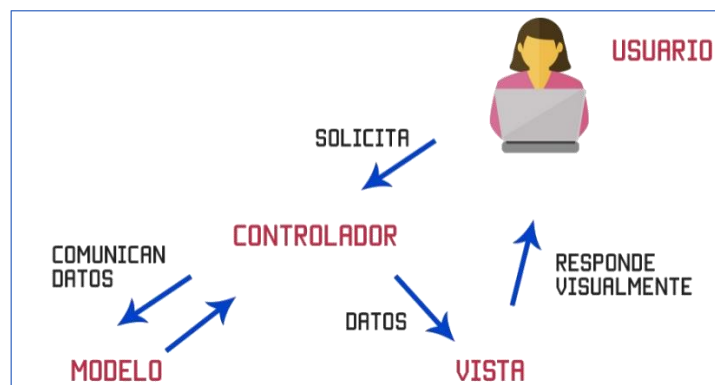


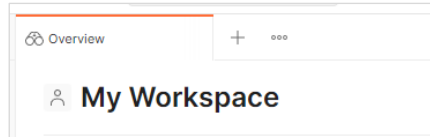
Imagen #16 – Flujo de trabajo de la arquitectura MVC

Verificando controlador

Aun no tenemos diseñada una interfaz donde podamos comprobar la respuesta que el controlador no está sirviendo, por ello utilizaremos la aplicación Postman.

Procedimientos:

- a. Ingrese a postman, cree una nueva pestaña [+] para agregar consultas



- b. Escriba la URL del controlador y verifique que el método de envío sea GET; además agregue como key = **operacion** y value = **listarProductos**, tal como se ve en la siguiente imagen:

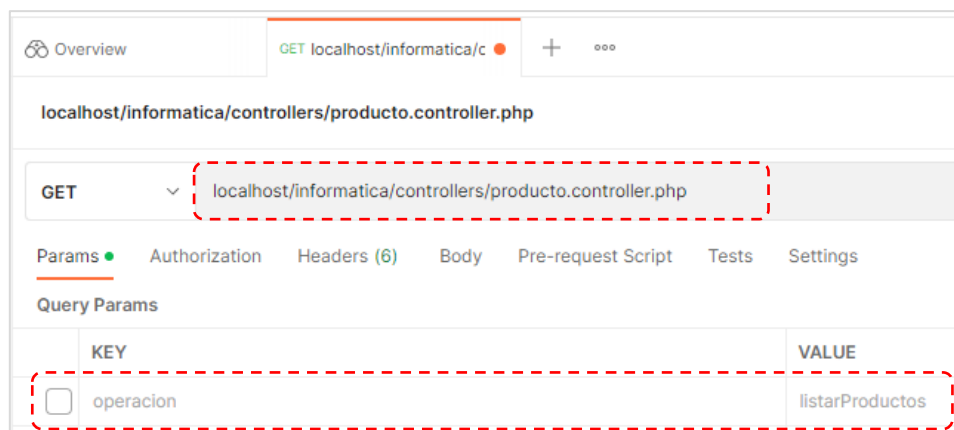


Imagen #17 – Pulse clic a SEND para enviar la consulta

- c. Los resultados se muestran en la parte inferior como texto simple (desordenado), pulse clic en la lista para seleccionar el resultado en formato JSON

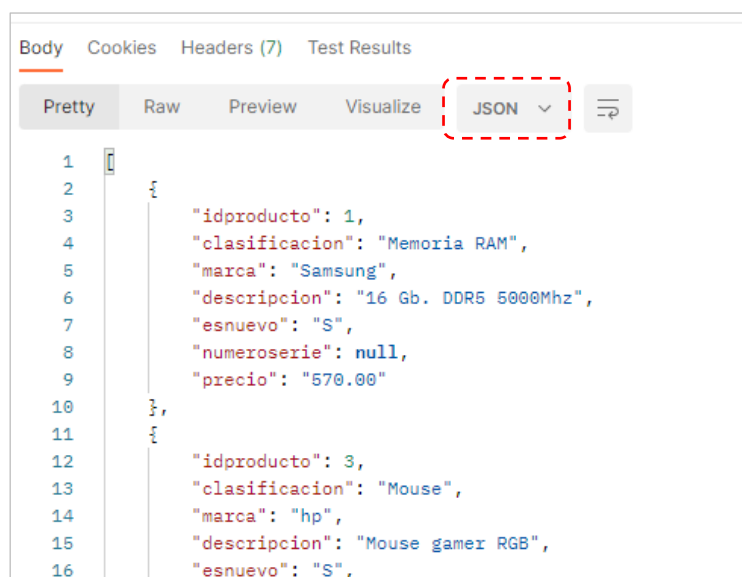


Imagen #18 – Resultado obtenido del controlador verificado a través de Postman

CAPÍTULO IV: Diseñando la vista

Recuerde que la vista es la interfaz con la que el usuario interactúa, así que utilizaremos HTML, CSS y otras tecnologías del FrontEnd. Iniciamos creando un archivo **index.php** en la raíz del proyecto y un **producto.view.php** en la carpeta views.

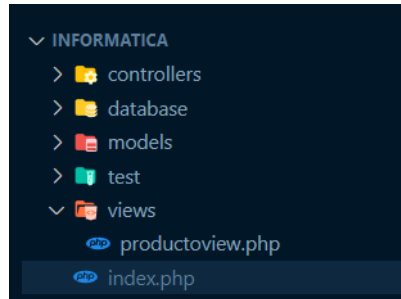


Imagen #19 – Al ingresar a nuestro sitio, el **index.php** será lo primero a mostrar

Utilizaremos Bootstrap 4.6 para este proyecto (soy consciente que existe la versión 5, puede cambiar a esta versión más reciente si lo prefiere). Visite el siguiente enlace para acceder a la librería:

<https://getbootstrap.com/docs/4.6/getting-started/introduction/>

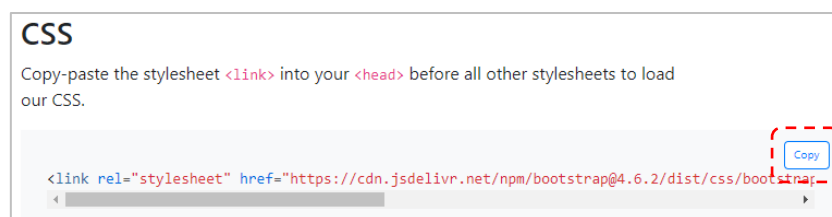


Imagen #20 - Copie el acceso al recurso por CDN de Bootstrap, lo pegará en **index.php**

Nuestro archivo **index.php** mostrará únicamente un título de bienvenida y un enlace a la vista de productos, es en esta última donde se desarrollará todo el proyecto.

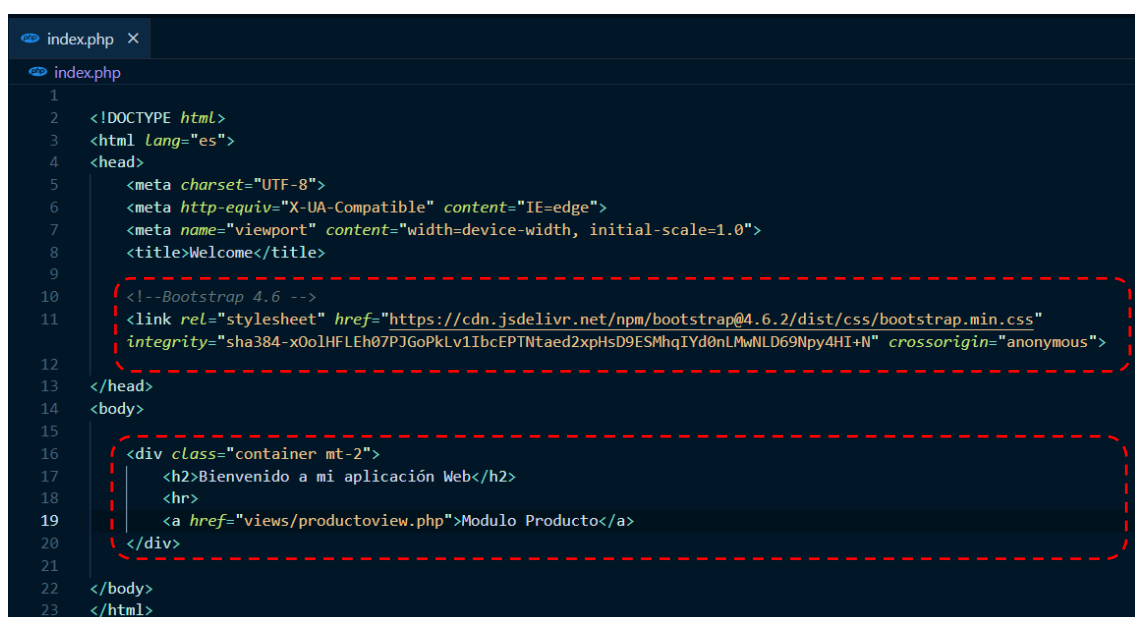


Imagen #21 – Configuración de nuestro archivo **index.php**

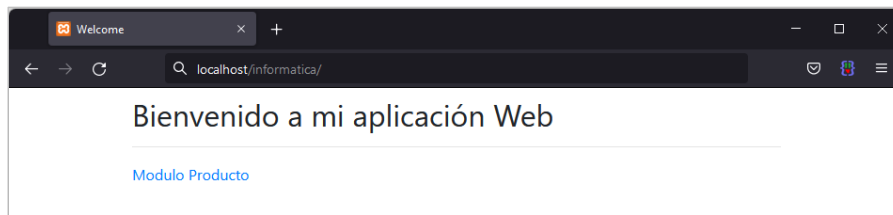


Imagen #22 – Nuestra página principal dándonos la bienvenida

Ahora procederemos a codificar nuestra vista de productos (producto.view.php), para ello requerimos 3 librerías: Bootstrap (nos ayudará con los estilos y maquetación de nuestra Web), jQuery (acceso a datos de forma síncrona) y DataTable (crea una tabla con muchas funcionalidades tales como buscador, Paginador, botones para ordenar alfabéticamente, etc.).

Las librerías las encuentra aquí:

Bootstrap 4
https://getbootstrap.com/docs/4.6/getting-started/introduction/
jQuery
https://developers.google.com/speed/libraries#jquery
DataTable
https://datatables.net/

Hasta este punto seguro te estás preguntando qué es un CDN (lo comenté en la página anterior), pues aquí una breve explicación antes de poder incorporarlos a nuestra Web.



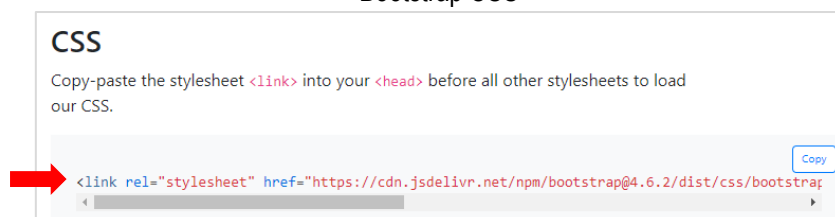
Una red de distribución de contenido (CDN) es un grupo de servidores repartidos en distintas zonas geográficas que aceleran la entrega del contenido web al acercarlo a los usuarios.

Agregando librerías:

jQuery



Bootstrap CSS



Bootstrap JS

Separate

If you decide to go with the separate scripts solution, Popper must come first (if you're using tooltips or popovers), and then our JavaScript plugins.

```
<script src="https://cdn.jsdelivr.net/npm/jquery@3.4.1/dist/jquery.min.js" integrity="sha384-...>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-...>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.min.js" integrity="sha384-...>
```

DataTable CSS y JS

1 - Include these two files

CSS //cdn.datatables.net/1.12.1/css/jquery.dataTables

JS //cdn.datatables.net/1.12.1/js/jquery.dataTables

```
productview.php X
views > productview.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Producto</title>
8
9   <!-- Bootstrap 4.6 -->
10  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
11    integrity="sha384-x0o1HFLhEh07PJGoPklv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">
12
13  <!-- css datatable -->
14  <link rel="stylesheet" href="//cdn.datatables.net/1.12.1/css/jquery.dataTables.min.css">
15 </head>
16 <body>
```

Imagen #22 – Recuerde que entre <head> solo debe importar librerías de estilos (CSS)

```
16 <body>
17
18 <!-- Zona para componentes HTML -->
19
20
21 <!-- jquery -->
22 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
23
24 <!-- Bootstrap 4.6 -->
25 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/
26   reFTGAW83EW2RDu2S0VKAizap3H661ZH81PoY1FhbGU+6BZp6G7niu735Sk71N" crossorigin="anonymous"></script>
27
28 <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.min.js" integrity="sha384-
29   +sLI0odYLS7CIrQpBjl+C7nPvqq+FbNUBDunl/OZv93DB7Ln/533i8e/mZXLi/P+" crossorigin="anonymous"></script>
30
31 <!-- Datatable -->
32 <script src="//cdn.datatables.net/1.12.1/js/jquery.dataTables.min.js"></script>
33
34 <!-- Mis funciones y eventos javascript -->
35 <script>
36
37 </script>
38 </body>
39 </html>
```

Imagen #23 – Las etiquetas <script> van siempre al final de nuestras Web.

En la siguiente imagen podrá apreciar el contenido para la sección HTML.

```
18 <!-- Zona para componentes HTML -->
19 <div style="width:90%; margin: auto" class="mt-2">
20 <h2 class="text-center mb-4">Módulo de productos </h2>
21 <button class="btn btn-primary" type="button">Registrar un Producto</button>
22 <hr>
23 <div class="table-responsive" >
24 <table class="table" id="tabla-productos">
25 <thead class="table-dark">
26 <tr>
27 <th>#</th>
28 <th>Clasificación</th>
29 <th>Marca</th>
30 <th>Descripción</th>
31 <th>Nuevo</th>
32 <th>Serie</th>
33 <th>Precio</th>
34 </tr>
35 </thead>
36 <tbody>
37
38 </tbody>
39 </table>
40 </div>
41 </div> <!-- Fin de capa principal -->
```

Imagen #24 – Todo el contenido se alojará en una capa que ocupa 90% espacio disponible, además creamos un botón que más adelante nos permitirá registrar un producto y la tabla donde se mostrarán los datos.



Imagen #25 – Resultado obtenido al cargar la vista en el navegador

RECOMENDACIÓN

¡Ya están puestas todas las cartas sobre la mesa!, tenemos nuestras tablas, clase de conexión, primer modelo, controlador y vista. Es probable que de aquí en adelante puedan presentarse dificultades, así que, para evitarnos un terrible dolor de cabeza, vamos a inspeccionar la vista y revisar la consola a fin de que esta no muestre error alguno, tal como se ve en la siguiente imagen:

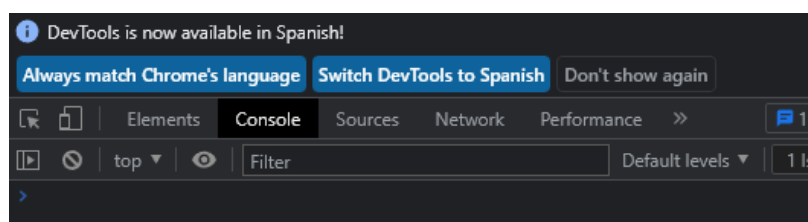


Imagen #26 – Consola sin reporte de problema alguno

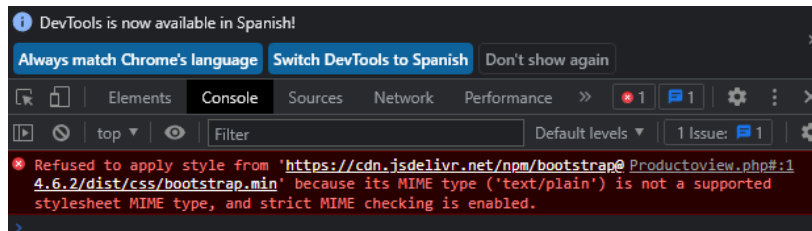


Imagen #27 – Consola reportándonos que tenemos problemas

Para solucionar cualquier dificultad, revise el error mostrado en pantalla y determine el procedimiento en donde se está generando la excepción. El manual le será de gran ayuda siempre que lo lea con paciencia y en orden.

Ahora agregaremos un modal a nuestro HTML, para ello visite:

<https://getbootstrap.com/docs/4.6/components/modal/>

Seleccione y copie todo el código correspondiente al modal

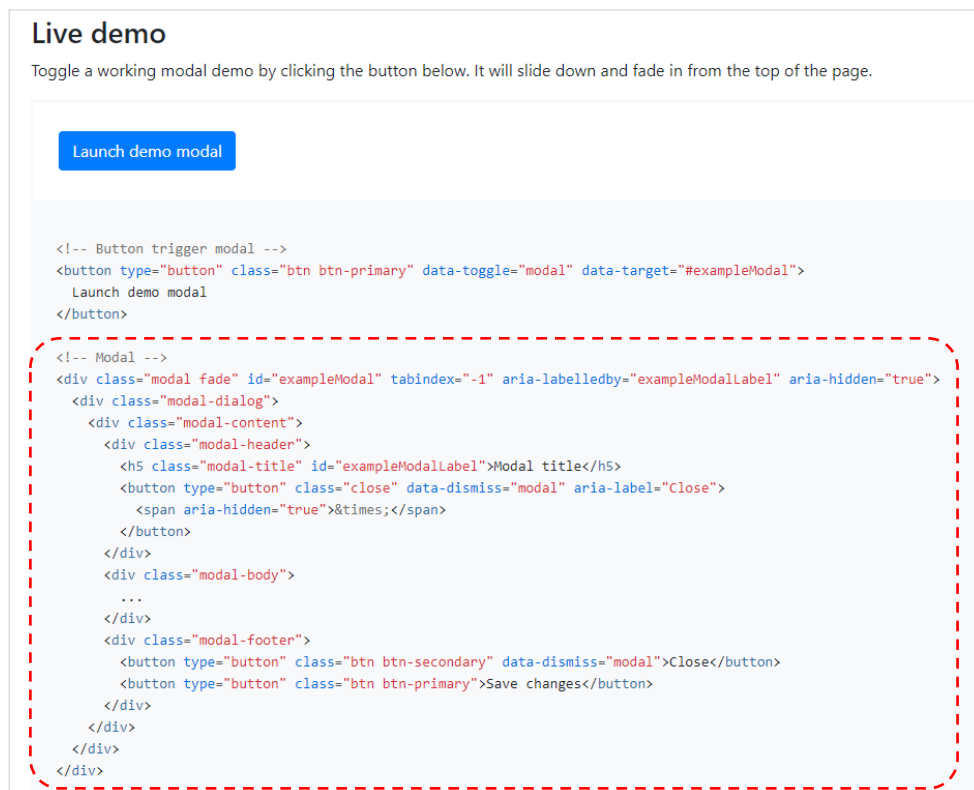


Imagen #28 – Estructura del componente Modal de Bootstrap



Imagen #29 – Pegue el código copiado anteriormente después de la capa principal pero antes de la sección de librerías javascript.


```

41 </div> <!-- Fin de capa principal -->
42
43 <!-- Zona modales -->
44 <div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">
45   <div class="modal-dialog">
46     <div class="modal-content">
47       <div class="modal-header">
48         <h5 class="modal-title" id="exampleModalLabel">Modal title</h5>
49         <button type="button" class="close" data-dismiss="modal" aria-label="Close">
50           <span aria-hidden="true">&times;</span>
51         </button>
52       </div>
53       <div class="modal-body">
54         ...
55       </div>
56       <div class="modal-footer">
57         <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
58         <button type="button" class="btn btn-primary">Save changes</button>
59       </div>
60     </div>
61   </div>
62 </div>
63
64 <!-- jquery -->
65 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
66

```

Imagen #30 – Estructura del modal

El componente MODAL se divide en 3 secciones:

- Cabecera `<div class = "modal-header">` `</div>`
- Cuerpo `<div class = "modal-body">` `</div>`
- Pié de modal `<div class = "modal-footer">` `</div>`

Haremos algunos cambios en el modal que hemos tomado de la Web de Bootstrap, ajustándolo a nuestras necesidades, observe y actualice los siguientes datos:

```

43 <!-- Zona modales -->
44 <div class="modal fade" id="modal-productos" tabindex="-1" aria-labelledby="titulo-modal-productos" aria-hidden="true">
45   <div class="modal-dialog">
46     <div class="modal-content">
47       <div class="modal-header bg-dark text-light">
48         <h5 class="modal-title" id="titulo-modal-productos">Registro de productos</h5>
49         <button type="button" class="close" data-dismiss="modal" aria-label="Close">
50           <span aria-hidden="true">&times;</span>
51         </button>
52       </div>
53       <div class="modal-body">
54         <form action="" id="formulario-productos" autocomplete="off">
55           <!-- Creación de controles -->
56         </form>
57       </div>
58       <div class="modal-footer">
59         <button type="button" id="cancelar-modal" class="btn btn-sm btn-secondary" data-dismiss="modal">Cerrar</button>
60         <button type="button" id="guardar-producto" class="btn btn-sm btn-primary">Guardar</button>
61       </div>
62     </div>
63   </div>
64 </div> <!-- Fin de modal -->
65

```

Imagen #31 – Personalizando cada sección del modal

Para poder mostrar lo trabajado hasta el momento, ubíquese en las primeras líneas de este archivo HTML y agregue dos atributos al botón, tal como se muestra en la siguiente imagen:

```

17
18 <!-- Zona para componentes HTML -->
19 <div style="width:90%; margin: auto" class="mt-2">
20   <h2 class="text-center mb-4">Módulo de productos</h2>
21   <button class="btn btn-primary" type="button" data-toggle="modal" data-target="#modal-productos">Registrar un Producto</button>
22   <hr>

```

Imagen #32 – Estos atributos permitirán mostrar el modal en pantalla

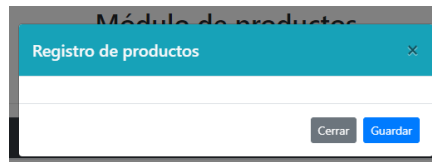


Imagen #33 – Vista previa del modal una vez ejecutado

Ahora procederemos a crear los controles que van en el cuerpo del modal.

```

53     <div class="modal-body">
54       <form action="" id="formulario-productos" autocomplete="off">
55
56         <!-- Creación de controles -->
57         <div class="form-group">
58           <label for="clasificaciones">Clasificaciones:</label>
59           <select name="clasificaciones" id="clasificaciones" class="form-control form-control-sm">
60             <option value="">Seleccione</option>
61           </select>
62         </div>
63
64         <div class="form-group">
65           <label for="marcas">Marcas:</label>
66           <select name="marcas" id="marcas" class="form-control form-control-sm">
67             <option value="">Seleccione</option>
68           </select>
69         </div>
70
71         <div class="form-group">
72           <label for="descripcion">Descripción:</label>
73           <input type="text" id="descripcion" class="form-control form-control-sm">
74         </div>
75
76         <div class="form-group">
77           <label for="numeroserie">Número de serie:</label>
78           <input type="text" id="numeroserie" class="form-control form-control-sm">
79         </div>

```

Los últimos dos controles van en una misma fila, por ello debemos crear una capa con clase row

```

80
81     <!-- Los dos últimos componentes comporten una misma fila -->
82     <div class="row">
83       <div class="col-md-6 form-group">
84         <label for="esnuevo">Producto nuevo:</label>
85         <select name="esnuevo" id="esnuevo" class="form-control form-control-sm">
86           <option value="S">Sí</option>
87           <option value="N">No</option>
88         </select>
89       </div>
90       <div class="col-md-6 form-group">
91         <label for="precio">Precio:</label>
92         <input type="text" class="form-control form-control-sm" id="precio">
93       </div>
94     </div>
95     <!-- Fin de controles -->
96   </form>
97
98 </div>

```

El resultado obtenido se muestra en la siguiente página:

Registro de productos ✕

Clasificaciones:

Marcas:

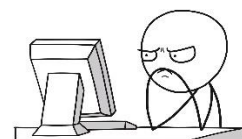
Descripción:

Número de serie:

Producto nuevo: Precio:

Imagen #34 – Vista previa del modal con formulario terminado ejecutado

Asegúrese que antes de continuar con el desarrollo de este manual, todo lo realizado hasta el momento, está acorde a lo explicado. Adelantarse puede parecer una buena idea en principio, pero si luego el trabajo está “trabado” en algún error por la prisa innecesaria en terminar antes, pasaremos largas horas revisando el código. **¡Primero aprenda a hacerlo bien, luego a hacerlo rápido!**



CAPÍTULO V: Procedimientos almacenados

Un procedimiento almacenado (stored procedure en inglés) es un programa (o procedimiento) almacenado físicamente en una base de datos. Su implementación varía de un gestor de bases de datos a otro. La ventaja de un procedimiento almacenado es que al ser ejecutado, en respuesta a una petición de usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente corre en un servidor separado. Como tal, posee acceso directo a los datos que necesita manipular y sólo necesita enviar sus resultados de regreso al usuario, deshaciéndose de la sobrecarga resultante de comunicar grandes cantidades de datos salientes y entrantes.

Fuente:

https://es.wikipedia.org/wiki/Procedimiento_almacenado

¿Cómo crear un Store Procedure en MySQL?

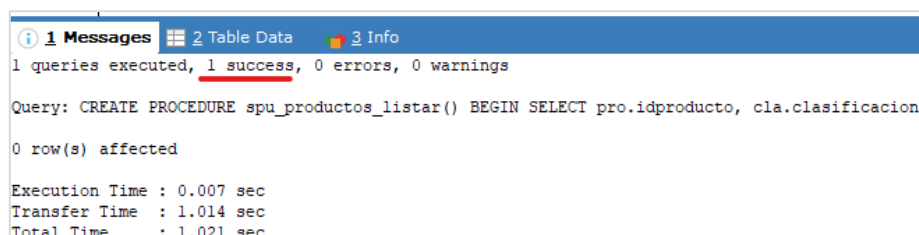
```
DELIMITER $$
CREATE PROCEDURE spu_nombreTabla_accionRealizar
(
    -- Variables (opcional)
)
BEGIN
    -- Instrucciones a ejecutar
END $$
```

Imagen #35 – El delimitador \$\$ se agrega para que en el cuerpo del procedimiento las instrucciones a ejecutar puedan finalizar con ; sin que esto detenga por completo la ejecución del programa.

Procederemos a crear 5 procedimientos almacenados de las tareas más importantes con nuestra tabla productos, todo esto en el mismo archivo de scripts que utilizó para crear las tablas, al final de capítulo verificamos la funcionalidad de estos:

```
67 -- CREACIÓN DE PROCEDIMIENTOS ALMACENADOS
68 -- Procedimiento almacenado para listar productos
69 DELIMITER $$
70 CREATE PROCEDURE spu_productos_listar()
71 BEGIN
72     SELECT pro.idproducto, cla.clasificacion, mar.marca, pro.descripcion,
73            pro.esnuevo, pro.numeroserie, pro.precio
74     FROM productos pro
75     INNER JOIN clasificaciones cla ON pro.idclasificacion = cla.idclasificacion
76     INNER JOIN marcas mar ON pro.idmarca = mar.idmarca
77     WHERE pro.estado = '1';
78 END $$
```

Imagen #36 - Seleccione todo el procedimiento y ejecute el programa pulsando F9



The screenshot shows the MySQL Messages window with the following content:

```
1 Messages 2 Table Data 3 Info
1 queries executed, 1 success, 0 errors, 0 warnings

Query: CREATE PROCEDURE spu_productos_listar() BEGIN SELECT pro.idproducto, cla.clasificacion,
0 row(s) affected

Execution Time : 0.007 sec
Transfer Time : 1.014 sec
Total Time : 1.021 sec
```

Imagen #37 – El mensaje de confirmación de creación de tablas, deberá indicar **1 success, 0 errors, 0 warning**, tal como se muestra en la imagen.

```

80 -- Los datos se enviarán desde la vista, entonces requerimos
81 -- variables de entrada, por eso de declaran con "IN" (INPUT)
82 -- Además las variables deben iniciar con @ o _ y su tipo debe
83 -- coincidir con el definido en la tabla
84 DELIMITER $$
85 CREATE PROCEDURE spu_productos_registrar
86 (
87     IN _idclasificacion SMALLINT,
88     IN _idmarca          SMALLINT,
89     IN _descripcion      VARCHAR(100),
90     IN _esnuevo          CHAR(1),
91     IN _numeroserie      VARCHAR(30),
92     IN _precio           DECIMAL(6,2)
93 )
94 BEGIN
95     -- No se pasan los campos definidos con DEFAULT, a excepción de "esnuevo" ya que el usuario tiene
96     -- la posibilidad de elegir el valor desde la vista/view
97     INSERT INTO productos (_idclasificacion, _idmarca, _descripcion, _esnuevo, _numeroserie, _precio) VALUES
98     (_idclasificacion, _idmarca, _descripcion, _esnuevo, _numeroserie, _precio);
99
100 END $$

```

```

103 DELIMITER $$
104 CREATE PROCEDURE spu_productos_eliminar(IN _idproducto INT)
105 BEGIN
106     UPDATE productos
107     SET estado = '0', fechabaja = NOW() WHERE idproducto = _idproducto;
108 END $$
109

```

```

110 DELIMITER $$
111 CREATE PROCEDURE spu_productos_obtener(IN _idproducto INT)
112 BEGIN
113     SELECT idproducto, idclasificacion, idmarca, descripcion, esnuevo, numeroserie, precio
114     FROM productos
115     WHERE estado = '1' AND idproducto = _idproducto;
116 END $$
117

```

```

119 DELIMITER $$
120 CREATE PROCEDURE spu_productos_actualizar
121 (
122     IN _idproducto      INT,
123     IN _idclasificacion SMALLINT,
124     IN _idmarca         SMALLINT,
125     IN _descripcion     VARCHAR(100),
126     IN _esnuevo         CHAR(1),
127     IN _numeroserie     VARCHAR(30),
128     IN _precio          DECIMAL(6,2)
129 )
130 BEGIN
131     UPDATE productos SET
132     idclasificacion = _idclasificacion,
133     idmarca = _idmarca,
134     descripcion = _descripcion,
135     esnuevo = _esnuevo,
136     numeroserie = _numeroserie,
137     precio = _precio
138     WHERE idproducto = _idproducto;
139 END $$
140

```

Para ejecutar un procedimiento y así verificar su funcionamiento, utilizaremos el siguiente formato:

```
call spu_tabla_accion(variables_opcional);
```

Procedimiento para listar datos:

```

141 -- TEST PROCEDIMIENTOS TABLA PRODUCTOS
142 CALL spu_productos_listar();
143

```

idproducto	clasificacion	marca	descripcion	esnuevo	numeroserie	precio
1	Memoria RAM	Samsung	16 Gb. DDR5 5000Mhz	S	(NULL)	570.00
3	Mouse	hp	Mouse gamer RGB	S	HN-7SA521	78.50

Imagen #38 – Seleccione toda la línea y pulse F9 para ejecutar un procedimiento almacenado

```

140
141 -- TEST PROCEDIMIENTOS TABLA PRODUCTOS
142 CALL spu_productos_listar();
143
144 → CALL spu_productos_obtener(1);
145

```

idproducto	idclasificacion	idmarca	descripcion	esnuevo	numeroserie	precio
1	1	1	1 16 Gb. DDR5 5000Mhz	S	(NULL)	570.00

Imagen #39 – Procedimiento obtener, retorna los datos de un único registro

```

141 -- TEST PROCEDIMIENTOS TABLA PRODUCTOS
142 CALL spu_productos_listar();
143
144 CALL spu_productos_obtener(1);
145
146 → CALL spu_productos_eliminar(2);
147 → SELECT * FROM productos;
148

```

idproducto	idclasificacion	idmarca	descripcion	esnuevo	numeroserie	precio	fechaalta	fechabaja	estado
1	1	1	1 16 Gb. DDR5 5000Mhz	S	(NULL)	570.00	2022-10-01 11:41:32	(NULL)	1
2	2	2	2 500 Watts real	S	(NULL)	170.00	2022-10-01 11:41:32	(NULL)	0

Imagen #40 – El procedimiento eliminar, cambia el estado del registro a 0, no realiza una eliminación física real (borrado de la base de datos)

```

147
148 → CALL spu_productos_registrar(3, 2, 'Mouse', 'N', 'ABC', 100);
149 → SELECT * FROM productos;
150

```

idproducto	idclasificacion	idmarca	descripcion	esnuevo	numeroserie	precio	fechaalta	fechabaja	estado
1	1	1	1 16 Gb. DDR5 5000Mhz	S	(NULL)	570.00	2022-10-01 11:41:32	(NULL)	1
2	2	2	2 500 Watts real	S	(NULL)	170.00	2022-10-01 11:41:32	(NULL)	0
3	3	3	2 Mouse	N	ABC	100.00	2022-10-04 01:44:13	(NULL)	1

Imagen #41 – Se guardó un nuevo registro en la tabla Productos con idproducto = 3

```

149
150 CALL spu_productos_actualizar(3, 3, 2, 'Mouse gamer RGB', 'S', 'HN-7SA521', 78.5);
151 SELECT * FROM productos;
152

```

idproducto	idclasificacion	idmarca	descripcion	esnuevo	numeroserie	precio	fechaalta	fechabaja	estado
1	1	1	1 16 Gb. DDR5 5000Mhz	S	(NULL)	570.00	2022-10-01 11:41:32	(NULL)	1
2	2	2	2 500 Watts real	S	(NULL)	170.00	2022-10-01 11:41:32	(NULL)	0
3	3	3	2 Mouse gamer RGB	S	HN-7SA521	78.50	2022-10-04 01:44:13	(NULL)	1

Imagen #42 – Comprobamos que la actualización de datos haya tenido resultado.



Queda prohibido continuar con el siguiente capítulo del manual, si no ha podido desarrollar y testear correctamente sus procedimientos almacenados.

CAPÍTULO VI: Módulos complementarios

Para poder trabajar con los productos: registrarlos, actualizarlos, buscarlos; vamos a necesitar de otras dos tablas: **marcas** y **clasificaciones**, las mismas que fueron explicadas al inicio de esta guía. Por lo que procederemos a crear los procedimientos almacenados, modelos y controladores para ambas. De esta manera y cuando corresponda, nuestro módulo principal – PRODUCTOS - podrá hacer uso de ellas.

Procedimientos almacenados:

```
153 -- Solo se utiliza la PK y el campo a mostrar en la lista
154 DELIMITER $$
155 CREATE PROCEDURE spu_clasificaciones_listar()
156 BEGIN
157     SELECT idclasificacion, clasificacion
158     FROM clasificaciones
159     WHERE estado = '1'
160     ORDER BY clasificacion;
161 END $$
162
163 -- Solo se utiliza la PK y el campo a mostrar en la lista
164 DELIMITER $$
165 CREATE PROCEDURE spu_marcas_listar()
166 BEGIN
167     SELECT idmarca, marca
168     FROM marcas
169     WHERE estado = '1'
170     ORDER BY marca;
171 END $$
```

Imagen #43 – Recuerde ejecutar un procedimiento a la vez y ejecutar (seleccionar desde DELIMITER hasta END)

Agregue dos nuevas **clases** en el directorio models, tal como se indica a continuación:

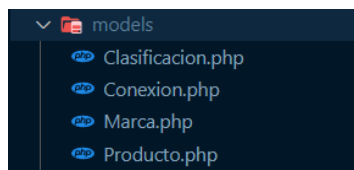


Imagen #44 – Agregar **Clasificacion.php** y **Marca.php**

```
Clasificacion.php X
models > Clasificacion.php
1  <?php
2
3  require_once "Conexion.php";
4
5  class Clasificacion extends Conexion{
6
7      private $acceso;
8
9      public function __construct(){
10         $this->acceso = parent::getConexion();
11     }
12
13     public function listarClasificaciones(){
14         try {
15             $consulta = $this->acceso->prepare("CALL spu_clasificaciones_listar()");
16             $consulta->execute();
17             $datos = $consulta->fetchAll(PDO::FETCH_ASSOC);
18
19             return $datos;
20         } catch (Exception $e) {
21             die($e->getMessage());
22         }
23     }
24 }
25
```

Imagen #45 – Clase y método para operar con las clasificaciones

```

Marca.php X
models > Marca.php
1  <?php
2
3  require_once "Conexion.php";
4
5  class Marca extends Conexion{
6
7      private $acceso;
8
9      public function __construct(){
10         $this->acceso = parent::getConexion();
11     }
12
13     public function listarMarcas(){
14         try {
15             $consulta = $this->acceso->prepare("CALL spu_marcas_listar()");
16             $consulta->execute();
17             $datos = $consulta->fetchAll(PDO::FETCH_ASSOC);
18
19             return $datos;
20         }
21         catch (Exception $e) {
22             die($e->getMessage());
23         }
24     }
25 }
26
27 ?>

```

Imagen #46 – Clase y método para operar con las marcas

Proceda agregar dos nuevos controladores, tal como se muestra a continuación:

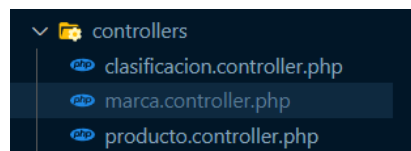


Imagen #47 - Agregue: **clasificación.controller.php** y **marca.controller.php**

```

clasificacion.controller.php X
controllers > clasificacion.controller.php
1  <?php
2
3  require_once '../models/Clasificacion.php';
4
5  if (isset($_GET['operacion'])){
6
7      $clasificacion = new Clasificacion();
8
9      if ($_GET['operacion'] == 'listarClasificaciones'){
10         $dataClasificacion = $clasificacion->listarClasificaciones();
11         echo json_encode($dataClasificacion);
12     }
13 }
14
15
16 ?>

```

Imagen #48 – Instrucciones para el controlador de las clasificaciones

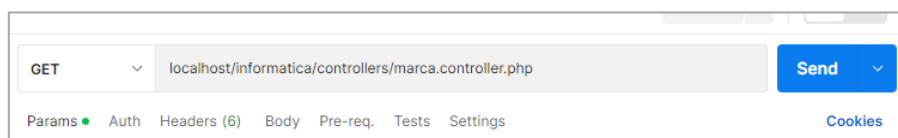

```

marca.controller.php x
controllers > marca.controller.php
1  <?php
2
3  require_once '../models/Marca.php';
4
5  if (isset($_GET['operacion'])){
6
7      $marca = new Marca();
8
9      if ($_GET['operacion'] == 'listarMarcas'){
10         $dataMarcas = $marca->listarMarcas();
11         echo json_encode($dataMarcas);
12     }
13 }
14
15
16 ?>

```

Imagen #49 – Instrucciones para el controlador de las marcas.

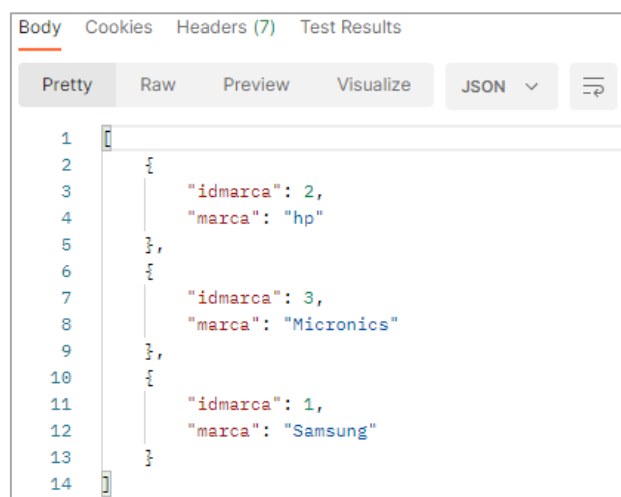
Puede verificar la funcionalidad de sus controladores recién creados desde Postman, solo (Paso 1) escriba el nombre del controlador, (Paso 2) pase las variables necesarias como claves y valores y (Paso 3) muestra el resultado en formato JSON.



Paso 1

Query Params		
	KEY	VALUE
<input checked="" type="checkbox"/>	operacion	listarMarcas

Paso 2



Paso 3

Repita este procedimiento para comprobar clasificaciones.

CAPÍTULO VII: Introducción AJAX – primera parte

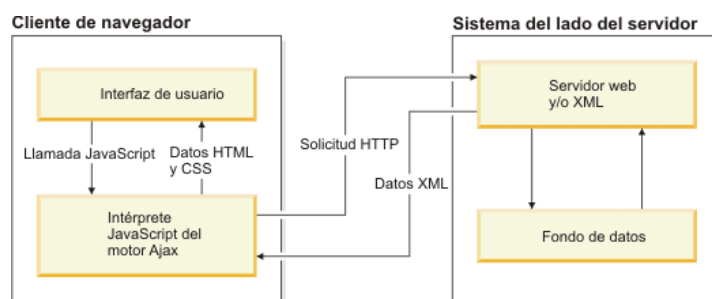
Un poco de teoría...

Ajax (Asynchronous JavaScript and XML) se refiere a un grupo de tecnologías que se utilizan para desarrollar aplicaciones web. Al combinar estas tecnologías, las páginas web parecen que son más receptivas puesto que los paquetes pequeños de datos se intercambian con el servidor y las páginas web no se vuelven a cargar cada vez que un usuario realiza un cambio de entrada.

Ajax permite que un usuario de la aplicación web interactúe con una página web sin la interrupción que implica volver a cargar la página web. La interacción del sitio web ocurre rápidamente sólo con partes de la página de recarga y renovación.

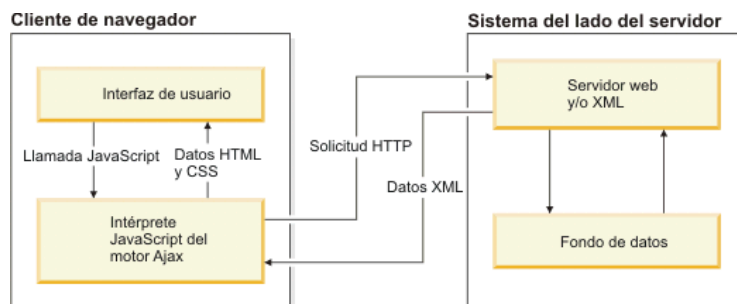
Aplicación Web tradicional (sin Ajax)

En una aplicación web tradicional, las solicitudes HTTP, que se inician mediante la interacción del usuario con la interfaz web, se realizan a un servidor web. El servidor web procesa la solicitud y devuelve una página HTML al cliente. Durante el transporte HTTP, el usuario no puede interactuar con la aplicación web.



Aplicación con Ajax

En una aplicación web Ajax, no se interrumpe el usuario en interacciones con la aplicación web. El motor de Ajax o el intérprete JavaScript permite que el usuario interactúe con la aplicación web independientemente del transporte HTTP procedente del servidor o que tenga el servidor como destino representando la interfaz y gestionando las comunicaciones con el servidor en nombre del usuario.



Fuente:

<https://www.ibm.com/docs/es/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview>

DEBES SABER ESTO...

Para la utilización de AJAX utilizaremos la librería jQuery que simplifica tremendamente nuestro trabajo, aunque queda abierta la posibilidad de realizar toda la rutina utilizando vanillaJS (Javascript "puro"), para ello nos valdremos del objeto XMLHttpRequest o con una tercera y reciente API integrada con las promesas, llamada Fetch, podrá encontrar más información al respecto siguiendo este link:

<https://es.stackoverflow.com/questions/25798/c%C3%B3mo-realizar-una-llamada-ajax-sin-bibliotecas>

LISTANDO DATOS DE PRODUCTOS EN UTILIZANDO DATATABLE

Llegamos a un punto importante del sistema, recuerde que en capítulos anteriores construimos procedimientos almacenados en la base de datos, los mismos que facilitarán enormemente nuestro desarrollo. Vamos a abrir nuestro modelo de productos, el cual es el archivo Producto.php almacenado en Models.

Antes de iniciar:

```
16 public function listarProductos(){
17     try {
18
19         //Preparamos la consulta
20         $consulta = $this->acceso->prepare("CALL spu_productos_listar()");
21     }
```

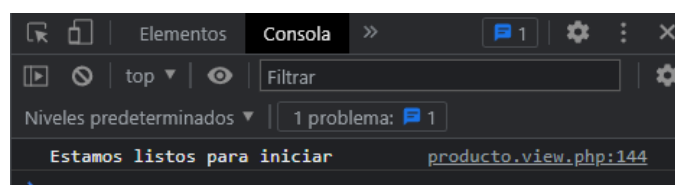
Imagen #50 – Actualice el método cambiando la consulta SELECT por el nombre del procedimiento almacenado

Procedimientos:

- Abra el archivo producto.view.php almacenado en la carpeta views, ubíquese en la sección inferior, donde están las instrucciones javascript y defina algunas de las funciones que utilizaremos.

```
116
117 <!-- Mis funciones y eventos javascript -->
118 <script>
119     $(document).ready(function (){
120
121         //Mostrará los registros en el DataTable
122         function mostrarProductos(){
123         }
124
125         //Poblará de datos el control <select>
126         function listarMarcas(){
127         }
128
129         //Poblará de datos el control <select>
130         function listarClasificaciones(){
131
132         }
133
134         //Reiniciará los valores del formulario
135         function reiniciarFormulario(){
136
137         }
138
139         //Registro-actualización de productos
140         function registrarProducto(){
141
142         }
143
144         console.log("Estamos listos para iniciar");
145     });
146 </script>
```

Imagen #51 – Las funciones no tienen instrucciones, sin embargo, se irán construyendo y explicando una a la vez. El mensaje de la línea 144 deberá ser visible en la consola si todo marchó correctamente, luego, puede eliminarlo.



- b. Construimos la función **mostrarProductos()** paso a paso. Indicar, que casi en su totalidad, todas las funciones harán uso de asincronismo (AJAX), utilizando para ello la librería jQuery, por eso, el primer paso es definir la estructura base, tal como se indica a continuación:

```
120
121 //Mostrará los registros en el DataTable
122 function mostrarProductos(){
123     $.ajax({
124         url: '',
125         type: 'GET',
126         data: '',
127         success: function(result){
128
129         }
130     });
131 }//Fin función mostrarProductos
132
```

PASO 1: estructurando objeto AJAX para mostrarProductos

```
121 //Mostrará los registros en el DataTable
122 function mostrarProductos(){
123     $.ajax({
124         url: '../controllers/producto.controller.php',
125         type: 'GET',
126         data: 'operacion=listarProductos',
127         success: function(result){
128             let registros = JSON.parse(result);
129             console.log(registros);
130         }
131     });
132 }//Fin función mostrarProductos
133
```

PASO 2: Con los atributos asignados y convirtiendo el resultado (**result**) a un objeto mapeable de tipo JSON (**registros**) podemos ver el resultado en la consola.

```
153 //console.log("Estamos listos para iniciar");
154
155 //Funciones de carga automática
156 mostrarProductos();
157
158 });
159 </script>
```

PASO 3: Antes de probar la función no olvide invocarla (utilice la parte inferior para el llamado a la función)



Resultado obtenido en consola

- c. Ahora vamos a agregar toda la lógica dentro de la sección **success** de nuestro objeto **Ajax**, recuerde que seguimos trabajando en la función **mostrarProductos()** y siempre GUARDE y vaya verificando los cambios en la consola del navegador.

```

success: function(result){
  let registros = JSON.parse(result);
  let nuevaFila = ``;

  //Reiniciar todas las filas de la tabla
  $("#tabla-productos tbody").html("");

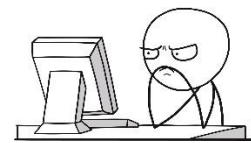
  //Recorremos toda la colección
  registros.forEach(registro => {

    //Creamos una nueva fila HTML
    nuevaFila = `
      <tr>
        <td>${registro['idproducto']}

```

Imagen #52 – Construcción de filas para la tabla Productos

Explicación: Luego de recibir los datos (**result**), estos se convertirán en un objeto mapeable de tipo JSON con el nombre **registros**, seguidamente eliminamos el contenido de la tabla para que a través de ajax pueda poblarse con la información obtenida desde la base de datos. La forma de recoger los datos es utilizando la estructura **forEach**, la cual recorrerá cada posición de la matriz asociativa **registros** y permitirá la construcción de una **nuevafila**, la misma, que se agrega al cuerpo de la tabla al final.



Registrar un Producto						
#	Clasificación	Marca	Descripción	Nuevo	Serie	Precio
1	Memoria RAM	Samsung	16 Gb. DDR5 5000Mhz	S	null	570.00
3	Mouse	hp	Mouse gamer RGB	S	HN-7SA521	78.50
4	Monitor	Samsung	Samsung curso 165Hz 1ms	S	F55S574W8	950.00

Imagen #53 –Resultado obtenido

- d. Vamos a integrar una columna adicional a nuestra tabla, en ella encontraremos dos botones, uno que permita la eliminación del registro, y el otro la edición. Entonces, vuelva a la parte superior de su código y agregue una nueva columna en la sección <thead> de su tabla.

```

<table class="table table-striped table-sm" id="tabla-productos">
  <thead class="table-dark">
    <tr>
      <th>#</th>
      <th>Clasificación</th>
      <th>Marca</th>
      <th>Descripción</th>
      <th>Nuevo</th>
      <th>Serie</th>
      <th>Precio</th>
      <th>Comandos</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>Memoria RAM</td>
      <td>Samsung</td>
      <td>16 Gb. DDR5 5000Mhz</td>
      <td>S</td>
      <td>null</td>
      <td>570.00</td>
      <td>
        <a href="#" class="btn btn-sm btn-danger">Eliminar</a>
        <a href="#" class="btn btn-sm btn-info">Editar</a>
      </td>
    </tr>
  </tbody>
</table>

```

Imagen #54 – Actualizando nuestra tabla productos (note que se agregó dos clases más) y una columna

```

//Recorremos toda la colección
registros.forEach(registro => {

  //Creamos una nueva fila HTML
  nuevaFila = `
    <tr>
      <td>${registro['idproducto']}</td>
      <td>${registro['clasificacion']}</td>
      <td>${registro['marca']}</td>
      <td>${registro['descripcion']}</td>
      <td>${registro['esnuevo']}</td>
      <td>${registro['numeroserie']}</td>
      <td>${registro['precio']}</td>
      <td>
        <a href="#" class="btn btn-sm btn-danger">Eliminar</a>
        <a href="#" class="btn btn-sm btn-info">Editar</a>
      </td>
    </tr>
  `;

  //Agregamos la nueva fila al cuerpo de la tabla
  $("#tabla-productos tbody").append(nuevaFila);
});

```

Imagen #55 – Agregamos una nueva fila y dos enlaces con clases de botones

Módulo de productos							
<div>Registrar un Producto</div>							
#	Clasificación	Marca	Descripción	Nuevo	Serie	Precio	Comandos
1	Memoria RAM	Samsung	16 Gb. DDR5 5000Mhz	S	null	570.00	<div>Eliminar</div> <div>Editar</div>
3	Mouse	hp	Mouse gamer RGB	S	HN-7SA521	78.50	<div>Eliminar</div> <div>Editar</div>
4	Monitor	Samsung	Samsung curso 165Hz 1ms	S	F55S574W8	950.00	<div>Eliminar</div> <div>Editar</div>

Imagen #56 – Ya tenemos lista nuestra tabla, muestra los datos obtenidos y es capaz de generar dos comandos de acción (eliminar – editar) para cada registro. El siguiente paso será integrar DataTable

- e. **DataTable** es un poderoso componente Web, que permite convertir a nuestras típicas y poco productivas tablas HTML, en componentes más sofisticados y de gran utilidad y valor para nuestras aplicaciones. Siga atentamente los pasos para integrarlo a nuestro proyecto. **NOTA:** todo lo explicado a continuación se hará en el método **mostrarProductos()** el cual hemos venido trabajando.

```

success: function(result){
    let registros = JSON.parse(result);
    let nuevoFila = ``;

    //Destruimos el DataTable actual (puede existir uno)
    let tabla = $("#tabla-productos").DataTable();
    tabla.destroy();

    //Reiniciar todas las filas de la tabla
    $("#tabla-productos tbody").html("");

    //Recorremos toda la colección
    registros.forEach(registro => {

```

Imagen #57 – Agregamos las siguientes instrucciones dentro del **success** del objeto Ajax. Se está destruyendo el DataTable, porque con cada nuevo registro agregado, será necesario reconstruir todo el componente.

```

        <td>${registro['numeroserie']}</td>
        <td>${registro['precio']}</td>
        <td>
            <a href='#' class='btn btn-sm btn-danger'>Eliminar</a>
            <a href='#' class='btn btn-sm btn-info'>Editar</a>
        </td>
    </tr>
    `;
    //Agregamos la nueva fila al cuerpo de la tabla
    $("#tabla-productos tbody").append(nuevaFila);

    }); //Fin forEach

    //Construimos el DataTable con los nuevos datos
    $('#tabla-productos').DataTable({
        language:{
            url: '//cdn.datatables.net/plug-ins/1.12.1/i18n/es-MX.json'
        }
    }); //Fin DataTable
    } //Fin success
}); //Fin de $.ajax
} //Fin función mostrarProductos

```

Imagen #58 – luego del recorrido de datos que realiza `forEach`, construimos nuestro DataTable, estamos además configurando su idioma a español, comparto el enlace al CDN de la traducción:

```
//cdn.datatables.net/plug-ins/1.12.1/i18n/es-MX.json'
```

El resultado obtenido:

<div>Registrar un Producto</div>							
Mostrar 10 entradas				<div>Buscar:</div>			
#	Clasificación	Marca	Descripción	Nuevo	Serie	Precio	Comandos
1	Memoria RAM	Samsung	16 Gb. DDR5 5000Mhz	S	null	570.00	<div>Eliminar</div> <div>Editar</div>
3	Mouse	hp	Mouse gamer RGB	S	HN-7SA521	78.50	<div>Eliminar</div> <div>Editar</div>
4	Monitor	Samsung	Samsung curso 165Hz 1ms	S	F55S574W8	950.00	<div>Eliminar</div> <div>Editar</div>
Mostrando 1 a 3 de 3 entradas				<div>Anterior</div> <div>1</div> <div>Siguiente</div>			

Imagen #59 – Procedimiento listar para productos terminado

REGISTRO DE PRODUCTOS

Hay que recordar el modelo de BD propuesto, y notar que un producto requiere de una clasificación y una marca. Por lo que para la siguiente actividad, debemos tener esto en consideración.

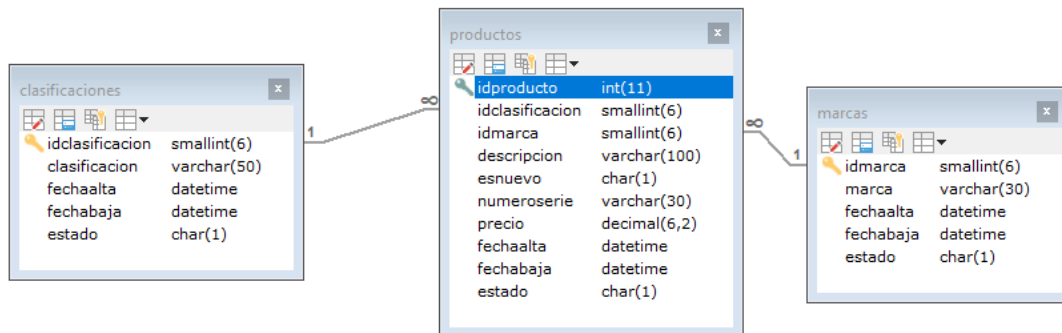


Imagen #60 – Modelo relacional original del proyecto

Cuando pretendemos registrar un producto, la lista desplegable donde se deberían mostrar las clasificaciones y marcas no muestran datos:

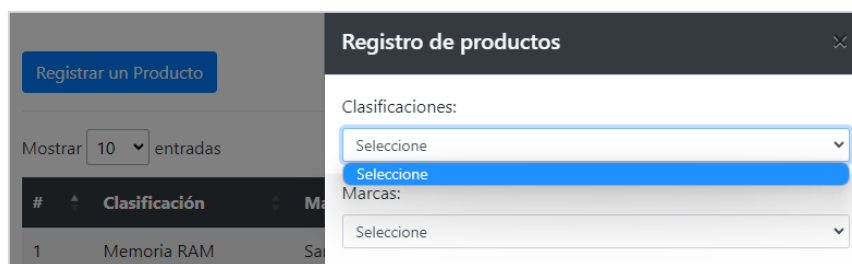


Imagen #61 – Los controles desplegables deberán acceder a los datos de sus tablas

Procedimientos:

- En el **capítulo VI – módulos complementarios**, ya se crearon los procedimientos almacenados, modelos y controladores para clasificaciones y marcas. Por lo que solo nos falta por construir las funciones en Javascript para poder acceder a esta información y poder mostrarla en el formulario. Abrimos el `producto.view.php` y nos ubicamos en la función **listarClasificaciones**

```
//Poblará de datos el control <select>
function listarClasificaciones(){
    $.ajax({
        url: '',
        type: 'GET',
        data: '',
        success: function(result){

        }
    });
} //Fin listarClasificaciones
```

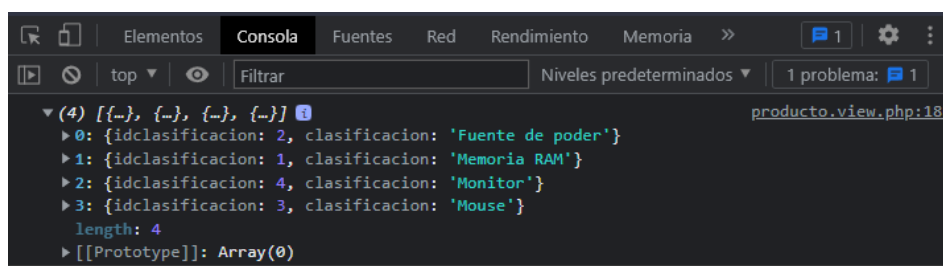
Imagen #62 – Construcción del método listarClasificaciones
PASO 1: Estructurar el objeto AJAX


```
//Poblará de datos el control <select>
function listarClasificaciones(){
  $.ajax({
    url: '../controllers/clasificacion.controller.php',
    type: 'GET',
    data: 'operacion=listarClasificaciones',
    success: function(result){
      let registros = JSON.parse(result);
      console.log(registros);
    }
  });
} //Fin listarClasificaciones
```

PASO 2: agregamos controlador y operación, luego recibimos los datos, los mapeamos como JSON y los mostramos en la consola.

```
//Funciones de carga automática
mostrarProductos();
listarClasificaciones();
});
</script>
```

PASO 3: agregamos la función en la parte inferior de nuestras instrucciones Javascript para que se ejecute automáticamente.



```
▼ (4) [{-}, {-}, {-}, {-}]
  ▶ 0: {idclasificacion: 2, clasificacion: 'Fuente de poder'}
  ▶ 1: {idclasificacion: 1, clasificacion: 'Memoria RAM'}
  ▶ 2: {idclasificacion: 4, clasificacion: 'Monitor'}
  ▶ 3: {idclasificacion: 3, clasificacion: 'Mouse'}
  length: 4
  ▶ [[Prototype]]: Array(0)
```

Resultado obtenido en la consola, es posible que Ud. no tenga los mismos datos, pero si la misma estructura de datos, un array asociativo, con dos claves: **idclasificacion** y **clasificación**

CONSEJO: Ya no estamos utilizando la función **mostrarProductos()** por lo que es una buena idea, ocultarla, para que no desordene la revisión de nuestro código, lo mismo puede hacer con la sección HTML donde construyó su tabla y modal. Mientras más ordenado, más eficiente, menos errores, pero sobre todo **¡más divertido!**.

```
119 <script>
120 $(document).ready(function (){
121
122 //Mostrará los registros en el DataTable
123 > function mostrarProductos(){...
172 }//Fin función mostrarProductos
```

Imagen #63 – oculte las funciones y secciones de su vista que no utilice ahora mismo

b. El método `listarClasificaciones()` tendrá la siguiente forma terminada:

```

//Poblará de datos el control <select>
function ListarClasificaciones(){
$.ajax({
url: '../controllers/clasificacion.controller.php',
type: 'GET',
data: 'operacion=listarClasificaciones',
success: function(result){
let registros = JSON.parse(result);
let elementosLista = ``;

if (registros.length > 0){
//Asignamos un primer elemento, que será el que se muestre por defecto
elementosLista = `<option selected>Selecione</option>`;

//Recorremos toda la colección
registros.forEach(registro => {
//Creamos la etiqueta <option> con el valor requerido
elementosLista += `<option value=${registro['idclasificacion']}>${registro['clasificacion']}</option>`;
});
}else{
//En caso no tengamos datos
elementosLista = `<option>No hay datos asignados</option>`;
}

//Agregando los elementos al <select>
$("#clasificaciones").html(elementosLista);
} //Fin success
}); //Fin ajax
} //Fin ListarClasificaciones

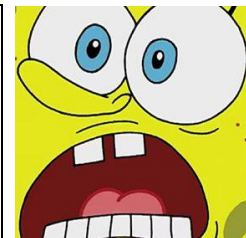
```

Se explica líneas más abajo detalladamente

Imagen #64 – Agregue las instrucciones en la sección success de Ajax

¿Qué pasa dentro de la estructura forEach?

Recuerda que los controles <select></select> o listas desplegables, requieren de una etiqueta <option></option> por cada elemento, estas etiquetas a su vez llevan un valor que es lo que realmente se considera para la aplicación, así por ejemplo: <option value = "2">Fuente de Poder</option>, lo que se muestra en el navegador es "Fuente de Poder", pero lo que se guarda en la base de datos será el value = 2. Es por ello asignamos como value a **idclasificacion** y como valor mostrado **clasificación**, ambos obtenidos desde el array registro[].



¡Qué es esoooooo!

Imagen #65 – La lista cargó correctamente

- La función listarMarcas() es muy parecida a la que acabamos de terminar. Sugiero hacer este trabajo paso a paso como hemos venido desarrollando las funcionalidades anteriores. A continuación, las instrucciones base para utilizar Ajax.

```
//Poblará de datos el control <select>
function listarMarcas(){
  $.ajax({
    url: '',
    type: 'GET',
    data: '',
    success: function(result){

    }
  });
}
//Fin función listarMarcas
```

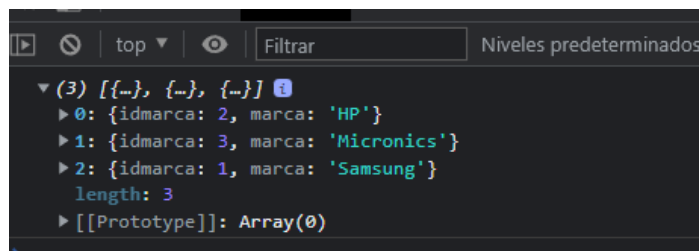
Imagen #65 – Construcción del método listarMarcas
PASO 1: Estructurar el objeto

```
//Poblará de datos el control <select>
function listarMarcas(){
  $.ajax({
    url: '../controllers/marca.controller.php',
    type: 'GET',
    data: 'operacion=listarMarcas',
    success: function(result){
      let registros = JSON.parse(result);
      console.log(registros);
    }
  });
}
//Fin función listarMarcas
```

PASO 2: agregamos controlador y operación, luego recibimos los datos, los mapeamos como JSON y los mostramos en la consola.

```
//Funciones de carga automática
mostrarProductos();
listarClasificaciones();
listarMarcas();
});
</script>
```

PASO 3: agregamos la función en la parte inferior de nuestras instrucciones Javascript para que se ejecute automáticamente.



```
▼ (3) [{...}, {...}, {...}] ⓘ
  ▶ 0: {idmarca: 2, marca: 'HP'}
  ▶ 1: {idmarca: 3, marca: 'Micronics'}
  ▶ 2: {idmarca: 1, marca: 'Samsung'}
  length: 3
  ▶ [[Prototype]]: Array(0)
```

Resultado obtenido

- d. El método listarMarcas() tendrá la siguiente forma terminada:

```

//Poblará de datos el control <select>
function listarMarcas(){
    $.ajax({
        url: '../controllers/marca.controller.php',
        type: 'GET',
        data: 'operacion=listarMarcas',
        success: function(result){
            let registros = JSON.parse(result);
            let elementosLista = ``;

            if (registros.length > 0){
                //Asignamos un primer elemento, que será el que se muestre por defecto
                elementosLista = `<option selected>Selecione</option>`;

                //Recorremos toda la colección
                registros.forEach(registro => {
                    //Creamos la etiqueta <option> con el valor requerido
                    elementosLista += `<option value=${registro['idmarca']}>${registro['marca']}</option>`;
                });
            }else{
                elementosLista = `<option>No hay datos asignados</option>`;
            }

            $("#marcas").html(elementosLista);
        }
    });
}
//Fin función listarMarcas

```

Imagen #66 – Método listarMarcas terminado, como notará, es muy parecido a listarClasificaciones

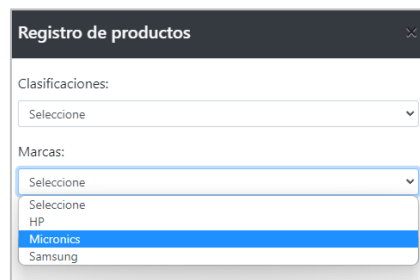


Imagen #67 – Resultado obtenido

- e. Ya tenemos el procedimiento almacenado que permite, a través de sus variables de entrada, recibir valores externos y poder agregarlos dentro de la tabla (spu_productos_registrar). Actualizamos el modelo con la función correspondiente:

```

public function registrarProducto($datosGuardar){
    try {
        //Cada ? (comodín), representa una variable que requiere el SPU
        $consulta = $this->acceso->prepare("CALL spu_productos_registrar (?, ?, ?, ?, ?, ?)");

        //Ya no haremos solo esto: $comando->execute(); porque necesitamos pasar 6 variables, entonces...
        $consulta->execute(array(
            $datosGuardar['idclasificacion'],
            $datosGuardar['idmarca'],
            $datosGuardar['descripcion'],
            $datosGuardar['esnuevo'],
            $datosGuardar['numeroserie'],
            $datosGuardar['precio']
        ));

        //Este método no retorna valor alguno
    }
    catch (Exception $e) {
        die($e->getMessage());
    }
}

```

Imagen #68 – Método para registrar producto



¡NO PASES POR ALTO!

- Este método requiere cada valor solicitado por el procedimiento almacenado, es decir 6 valores. No utilizamos una variable para cada valor, porque sería muy poco “elegante” el código, imagine esto:
registrarProducto(\$variable1, \$variable2, \$variable3, \$variable...)
- La solución, es que, en lugar de enviar cada variable por separado, podamos enviar un solo array asociativo (**\$datosGuardar**) conteniendo toda la información solicitada.
- En el método execute(), debemos pasarle el arreglo con todos los datos.

Ciclo de operaciones que estaremos realizando:



Imagen #69 – Flujo de trabajo para el proceso de inserción (válido para otros como eliminar, modificar, etc.)

- f. Ahora debemos dotar a nuestro controlador, la capacidad de recibir todos los parámetros necesarios para poder registrar un nuevo Producto. Abra el archivo **producto.controller.php** y proceda con su actualización tal como se indica a continuación:

```

if ($_GET['operacion'] == 'registrarProducto'){

    /*
    En un array asociativo guardamos los datos que recibimos por $_GET[] (es decir la vista)
    RECUERDA:
    1. La Vista, es decir, el formulario envía todo por AJAX
    2. El controlador recibe lo enviado por la vista así: $_GET['algo']
    3. Los datos recibidos, se deberán "empaquetar" en un ARRAY ASOCIATIVO
    4. Enviamos el array asociativo como único valor a nuestro método registrarProducto()
    */

    $datosSolicitados = [
        "idclasificacion" => $_GET['idclasificacion'],
        "idmarca"         => $_GET['idmarca'],
        "descripcion"      => $_GET['descripcion'],
        "esnuevo"          => $_GET['esnuevo'],
        "numeroserie"      => $_GET['numeroserie'],
        "precio"           => $_GET['precio']
    ];

    $producto->registrarProducto($datosSolicitados);
}
  
```

Imagen #70 – Los comentarios no son obligatorios, sin embargo, agréguelos para saber qué está sucediendo

- g. Regresando a la vista (**producto.view.php**), iniciamos la creación de la función javascript que nos permitirá registrar un nuevo producto. Para ello, agregamos las siguientes instrucciones, que servirán como base, para el proceso a realizar. Considere lo siguiente:

```
//Registro-actualización de productos
function registrarProducto(){

    //Array que contiene los datos a enviar
    let datos = {};

    if (confirm("¿Está seguro de guardar este registro?")){
        $.ajax({
            url: '',
            type: '',
            data: datos,
            success: function(result){

                //Confirmar envío

                //Reiniciar interfaz a su estado original

            }
        }); //Fin ajax
    } //Fin confirm
} //Fin registrarProducto
```

Imagen #71 – Los comentarios describen perfectamente lo que sucederá cuando procedamos a guardar. El atributo data de ajax ya no recibirá la información requerida como cadena de texto, lo hará a través de un array asociativo (se declara **asignando llaves** a datos).

```
//Registro-actualización de productos
function registrarProducto(){

    //Array que contiene los datos a enviar
    let datos = {
        'operacion'      : 'registrarProducto',
        'idclasificacion' : $("#clasificaciones").val(), //Control <select>
        'idmarca'        : $("#marcas").val(),           //Control <select>
        'descripcion'    : $("#descripcion").val(),
        'esnuevo'        : $("#esnuevo").val(),
        'numeroserie'     : $("#numeroserie").val(),
        'precio'         : $("#precio").val()
    };

    if (confirm("¿Está seguro de guardar este registro?")){
        $.ajax({
            url: '../controllers/producto.controller.php',
            type: 'GET',
            data: datos,
            success: function(result){

                //Confirmar envío

                //Reiniciar interfaz a su estado original

            }
        }); //Fin ajax
    } //Fin confirm
} //Fin registrarProducto
```

Imagen #72 – Actualizamos nuestra función, el array asociativo datos no solo llevará los valores que requiere el procedimiento almacenado para funcionar, sino también, la operación a realizar (primer parámetro).

- h. Ya puede probar la función **registrarProducto** (aunque está por terminar). Para ello, ubíquese en la parte baja de código, cerca al llamado de las funciones y en el evento click del botón **guardar-producto** (botón creado en el footer del modal), agregue el llamado a esta función javascript.

```
//Eventos
$("#guardar-producto").click(registrarProducto);

//Funciones de carga automática
mostrarProductos();
listarClasificaciones();
listarMarcas();
});
</script>
```

Imagen #73 – Enlazando función a evento click, no olvide guardar antes de continuar

- i. Al momento de registrar un nuevo producto, por ahora: **no encontrará ninguna respuesta por parte de la aplicación** (recuerde que aun está por terminar), sin embargo, intente lo siguiente: llenar el formulario, pulsar el botón guardar, confirmar, y luego actualizar manualmente el navegador pulsando F5 para ver el registro en el DataTable.

Registro de productos

Clasificaciones:

Fuente de poder

Marcas:

Micronics

Descripción:

Certificada 80 Gold 500Watts

Número de serie:

A5F989W

Producto nuevo:

Sí

Precio:

295

Cerrar

Guardar

Registrar un Producto

Mostrar 10 entradas

Buscar:

#	Clasificación	Marca	Descripción	Nuevo	Serie	Precio	Comandos
1	Memoria RAM	Samsung	16 Gb. DDR5 5000Mhz	S	null	570.00	Eliminar Editar
3	Mouse	HP	Mouse gamer RGB	S	HN-7SA521	78.50	Eliminar Editar
4	Fuente de poder	Micronics	Certificada 80 Gold 500Watts	S	A5F989W	295.00	Eliminar Editar

Mostrando 1 a 3 de 3 entradas

Anterior

1

Siguiente

Imagen #73 – El proceso debería funcionar perfectamente, aunque no recarga los datos de forma automática, sino que hasta este punto se requiere una actualización manual (F5) por parte del usuario.

- j. Busque la función **reiniciarFormulario()** y agregue la siguiente instrucción:

```
//Reiniciará los valores del formulario
function reiniciarFormulario(){
  $("#formulario-productos")[0].reset();
}
```

Imagen #74 – Más adelante agregaremos algunas instrucciones más, por ahora le indicamos que todos los elementos (controles del formulario), regresarán a su estado original

- k. Terminamos de construir la función `registrarProducto()`, añadiendo las líneas de cierre en la sección `success` del objeto Ajax.

```
success: function(result){  
  
    if (result == ""){  
        //Confirmar envío  
        alert("Proceso terminado correctamente");  
        //Reconstruir el datatable  
        mostrarProductos();  
  
        //Reiniciar interfaz a su estado original  
        reiniciarFormulario();  
        //Cerrar el modal  
        $("#modal-productos").modal("hide");  
    }  
}
```

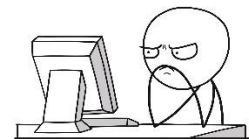
Imagen #75 – Asegúrese que el nombre del modal que está utilizando coincida con el referenciado en la imagen

ELIMINACIÓN DE PRODUCTOS

Este proceso se realizará de forma lógica, es decir, realmente el registro NO se eliminará de la BD, sino que cambiaremos el valor del campo **estado** de 1 a 0, de esta manera no se mostrarán en pantalla, ya que el procedimiento almacenado para listarlos agrega una condición al final de la consulta donde indica **WHERE estado = '1'** y al no cumplir con este requisito quedarán fuera.

NOTA IMPORTANTE

Antes de iniciar, cierre todas las pestañas de VisualStudio Code o del editor que esté utilizando, guarde, actualice y revise que todo lo trabajado hasta el momento esté **100% operativo** y **sin errores en consola**. De no ser así, por favor, corregir antes de continuar.



Procedimientos:

- a. Cierre todas las funciones que no esté utilizando, luego abra `mostrarProductos()`, esta función sufrirá algunos cambios, haga los cambios de acuerdo a lo indicado en la siguiente imagen:

```
<td>  
    <a href='#' class='btn btn-sm btn-danger'>Eliminar</a>  
    <a href='#' class='btn btn-sm btn-info'>Editar</a>  
</td>
```

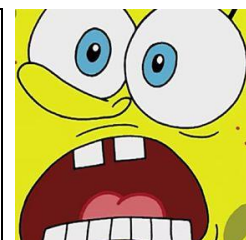
El antes y el después a continuación...

```
<td>  
    <a href='#' data-idproducto='${registro[idproducto]}' class='btn btn-sm btn-danger eliminar'>Eliminar</a>  
    <a href='#' class='btn btn-sm btn-info'>Editar</a>  
</td>
```

Imagen #76 – Estas instrucciones se encuentran dentro de la estructura **forEach**, donde se genera una nueva **Fila**

¿Qué está sucediendo?

Necesitamos que cada botón eliminar pueda identificar el id del producto que le corresponde eliminar. Por ello, estamos creando un nuevo atributo HTML utilizando **data-idproducto**, y almacenando en él la clave primaria. Además, estamos agregando a las clases CSS de las que ya teníamos una nueva llamada **"eliminar"**, esta clase realmente NO la utilizaremos para personalizar los estilos (apariencia) del componente, sino, para poder controlarlo más adelante con Javascript.



¡Qué es esoooooo!

- b. Si lo anterior es correcto, guarde y actualice su navegador, luego pulse clic derecho inspeccionar sobre cualquier botón eliminar y revise el HTML generado, encontrará el ID del producto dentro de un atributo que nosotros hemos definido.

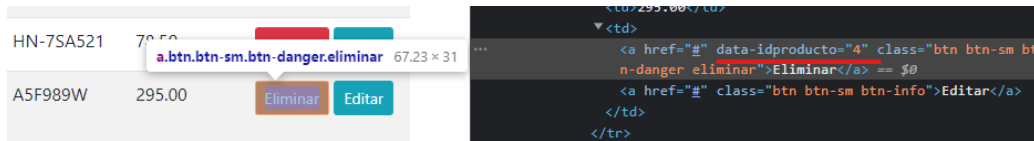


Imagen #77 – Así es como se ve el **idproducto** asignado en cada botón eliminar.

- c. Antes de continuar, diseñemos la función PHP en el **modelo (Producto.php)** para poder eliminar de forma lógica un registro. La instrucción es sumamente sencilla.

```
public function eliminarProducto($idproducto){
    try{
        $consulta = $this->acceso->prepare("CALL spu_productos_eliminar(?)");
        $consulta->execute(array($idproducto));
    }
    catch(Exception $e){
        die($e->getMessage());
    }
}
```

Imagen #78 – El método solo requiere el **idproducto**, el mismo que pasa al procedimiento almacenado.

- d. Agreguemos también un controlador que se pueda comunicar con el modelo recién construido

```
controllers > producto.controller.php
1  <?php
2  require_once '../models/Producto.php';
3
4  if (isset($_GET['operacion'])){
5
6      $producto = new Producto();
7
8      if ($_GET['operacion'] == 'listarProductos'){ ...
12 }
13
14 if ($_GET['operacion'] == 'registrarProducto'){ ...
35 }
36
37 if ($_GET['operacion'] == 'eliminarProducto'){
38     $producto->eliminarProducto($_GET['idproducto']);
39 }
40
41 }
42 ?>
```

Imagen #79 – Esta condición se deberá crear dentro de la estructura **if (isset())** al mismo nivel jerárquico de otras operaciones que ya tenemos funcionando. No olvide que todo lo que llega por **\$_GET[algo]** ha sido enviado desde el view a través de Ajax.

- e. Ya puede cerrar el controlador y modelo (siempre mantenga el orden de su proyecto), ahora debemos crear una función para eliminar un registro, pero... **si son muchos registros, N filas (indeterminado), ¿cómo sabremos a cuál de todos estos botones eliminar debemos asignarle la función?** Otra pregunta sería: y si los botones son creados de forma asíncrona (Ajax), **¿cómo le asignamos un evento (Click) si el control no existía cuando la página fue renderizada/cargada en el navegador?**

La solución viene a continuación...

Utilizaremos el evento **on** de jQuery. Este evento nos permitirá solucionar los dos problemas planteados en la página anterior. Primero, asignarle un evento a N botones, asociándolo a través de su clase y segundo, poder operar con estos elementos que se crearon de forma asíncrona (Ajax). La sintaxis es la siguiente:

```
$("#componente-padre").on("evento", "componentes-hijos", function(){
    //Acciones a realizar
});
```

Imagen #80 – **NO TRANSCRIBIR**. Esto es solo la forma como funcionará nuestro evento on de jQuery, en la siguiente imagen recién aplicaremos este concepto en el proceso de eliminación.

```
> //Registro-actualización de productos
function registrarProducto(){ ...
} //Fin registrarProducto

✓ $( "#tabla-productos tbody" ).on( "click", ".eliminar", function(){
  console.log( "Pulsaste clic en un botón eliminar" );
});

//Eventos
$( "#guardar-producto" ).click( registrarProducto );

//Funciones de carga automática
mostrarProductos();
listarClasificaciones();
listarMarcas();
});
</script>
```

Imagen #81 – Agregue el evento ON en el código Javascript de su vista de Productos

Iniciamos referenciando al componente_padre (**#tabla-productos tbody**), es decir, el componente HTML donde se alojarán los objetos que deseamos manipular componentes_hijos (**.eliminar**), no sin antes indicarle que el evento asociado a es **"click"**. Como tercer argumento, se crea una **función anónima** con las instrucciones que deseamos ejecutar.

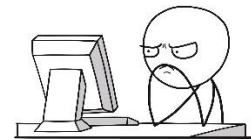


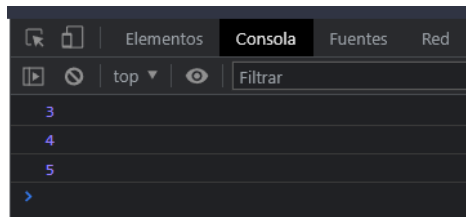
Imagen #82 - Guarde el archivo, y verifique utilizando la consola, la salida del mensaje al hacer click sobre cualquier botón eliminar

- f. Ya tenemos asociado el evento click a cada botón eliminar mostrado en pantalla, pero ahora, ¿cómo sabemos qué registro deberá eliminar?; la respuesta es simple, no olvide que cada botón lleva también un **data-idproducto** como atributo, conteniendo la clave primaria que se desea eliminar. Para obtener este valor y mostrarlo en pantalla, procedemos:

```
$( "#tabla-productos tbody" ).on( "click", ".eliminar", function(){
    //Almacenamos La PK en una variable
    let idproducto = $(this).data("idproducto");

    //Comprobamos el valor en la consola
    console.log(idproducto);
});
```

Imagen #83 – Obteniendo el idproducto para la eliminación



En la consola vamos a ir visualizando los ID que correspondan

- g. Ya tenemos todo listo, ahora solo preguntamos al usuario si está seguro de proceder con la eliminación, de responder afirmativamente, le indicamos al controlador de producto, que necesitamos que procese las variables enviadas en el atributo data (le estamos pasando un array asociativo). De confirmar que el resultado no retornó mensaje alguno (está vacío), entonces reiniciamos el valor de idproducto y reconstruimos todo el DataTable con la función mostrarProductos().

```
$("#tabla-productos tbody").on("click", ".eliminar", function(){
    //Almacenamos la PK en una variable
    let idproducto = $(this).data("idproducto");

    if (confirm("¿Está seguro de eliminar el registro?")){
        $.ajax({
            url: '../controllers/producto.controller.php',
            type: 'GET',
            data: {'operacion': 'eliminarProducto', 'idproducto': idproducto},
            success: function(result){
                if (result == ""){
                    idproducto = ``;
                    mostrarProductos();
                }
            } //Fin success
        }); //Fin ajax
    } //Fin confirm
}); //Fin evento ON
```

Imagen #84 – Eliminación de registros completa

Recuerde:

Los registros no han sido eliminados, solo actualizaron su estado de 1 (activo) a 0 (inactivo). Bastaría con que el administrador de la base de datos haga un **UPDATE productos SET estado = '1' WHERE idproducto = N** para volverlo a activar, y que este, se muestre nuevamente en el DataTable.

```
175 SELECT * FROM productos;
176
```

idproducto	idclasificacion	idmarca	descripcion	esnuevo	numeroserie	precio	fechaalta	fechabaja	estado
1	1	1	16 Gb. DDR5 5000Mhz	S	(NULL)	570.00	2022-10-01 11:41:32	2022-10-06 08:58:04	0
2	2	2	500 Watts real	S	(NULL)	170.00	2022-10-01 11:41:32	(NULL)	1
3	3	3	2 Mouse gamer RGB	S	HN-7SA521	78.50	2022-10-04 01:44:13	2022-10-06 08:58:15	0
4	2	3	Certificada 80 Gold 500Watts	S	ASFS989W	295.00	2022-10-05 16:17:54	2022-10-06 08:58:07	0
5	4	1	Curvo 27" 165Hz Gamer	N	T8FS5664	550.00	2022-10-05 16:28:13	(NULL)	1
6	1	2	DDR4 - 8Gb 3330 Mhz.	S	8G665G43	260.00	2022-10-05 16:29:43	2022-10-06 08:58:08	0

Imagen #85 – Los registros "eliminados" aun existen fisicamente en la BD

EDICIÓN DE REGISTROS

Llegamos al final de esta guía, no de todo el programa. No es casualidad que esta operación queda al final, ya que de los 4 procesos básicos (CRUD – Crear, leer, actualizar y eliminar), el proceso de edición / actualización / modificación de registros, es uno de los menos utilizados, en comparación con listar y registrar, por ejemplo. Antes de iniciar con la codificación debemos entender lo siguiente.

1. Se va a necesitar traer los datos del producto desde la tabla al formulario para que el usuario pueda modificarlos a través de la interfaz Web.
2. Además de los campos editables como clasificación, marca, descripción, numeroserie, etc; será necesario enviar el idproducto para que la instrucción UPDATE identifique al registro.

3. El flujo de todo este proceso será el siguiente:



Procedimientos:

- a. Iniciamos el trabajo con la construcción de dos métodos en el modelo (Producto.php), el primero permitirá recoger la información de un registro `getProducto()` y el segundo la actualización del mismo.

```
//Retorna una colección de datos correspondiente a un registro de producto
public function getProducto($idproducto){
    try{
        $consulta = $this->acceso->prepare("CALL spu_productos_obtener(?)");
        $consulta->execute(array($idproducto));

        return $consulta->fetch(PDO::FETCH_ASSOC);
    }
    catch(Exception $e){
        die($e->getMessage());
    }
}

public function actualizarProducto($datosGuardar){
    try {
        $consulta = $this->acceso->prepare("CALL spu_productos_actualizar (?, ?, ?, ?, ?, ?, ?)");

        $consulta->execute(array(
            $datosGuardar['idproducto'],
            $datosGuardar['idclasificacion'],
            $datosGuardar['idmarca'],
            $datosGuardar['descripcion'],
            $datosGuardar['esnuevo'],
            $datosGuardar['numeroserie'],
            $datosGuardar['precio']
        ));
    }
    catch (Exception $e) {
        die($e->getMessage());
    }
}
```

- b. El segundo paso será la inclusión de los controladores. Recuerde que la solicitud / request viene de la vista (la misma que se envía por Ajax), estos datos son tomados por el controlador y ejecutados gracias a la lógica del modelo. Agregue las siguientes instrucciones en **producto.controller.php**

```
if ($_GET['operacion'] == 'getProducto'){
    echo json_encode($producto->getProducto($_GET['idproducto']));
}

if ($_GET['operacion'] == 'actualizarProducto'){

    $datosSolicitados = [
        "idproducto"      => $_GET['idproducto'],
        "idclasificacion" => $_GET['idclasificacion'],
        "idmarca"         => $_GET['idmarca'],
        "descripcion"     => $_GET['descripcion'],
        "esnuevo"         => $_GET['esnuevo'],
        "numeroserie"     => $_GET['numeroserie'],
        "precio"          => $_GET['precio']
    ];

    $producto->actualizarProducto($datosSolicitados);
}
```