Desarrollo Web en entorno servidor

Unidad 6:
Aplicación de pedidos en AJAX

Tabla de contenido

1. Rediseño en AJAX de la aplicación de pedidos

Vamos a rediseñar la aplicación original de pedidos como una aplicación de una sola página usando AJAX. La nueva aplicación tendrá el mismo aspecto y funcionalidad la anterior, pero la lógica de presentación se pasa al cliente.

El cliente se encargará de:

- Solicitar los datos al servidor y mostrarlos.
- Modificar la estructura de la página para pasar de una pantalla a otra (por ejemplo, de la lista de categorías a la tabla de productos).

La lógica en el lado del servidor se ocupará de las cosas que no se pueden hacer en el cliente:

- Control de sesiones.
- Manejo de la base de datos.
- Envío de correo.

Gran parte del diseño de la aplicación original se mantiene sin cambios:

- La base de datos.
- El mapa de pantallas.
- La variable para el carrito.

Pero una aplicación de una sola página implica cambios importantes:

a) Hay que modificar los ficheros que devuelven HTML para que devuelvan en JSON (o XML) solo los datos que se quieren mostrar. Esto afecta a la lista de categorías, a la tabla de productos y al carrito de la compra.

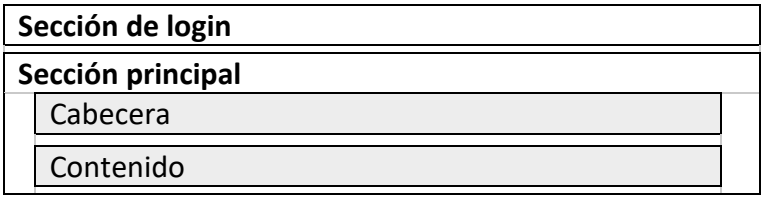
b) El cliente se ocupa de mostrar los datos recibidos o realizar cambios en la estructura de la página web JavaScript.

c) Todas las peticiones al servidor (salvo la inicial) se realizan mediante JavaScript.

2. Estructura de la página web

Aunque el mapa de pantallas no cambia, ahora en realidad solo va a haber un documento HTML que se va modificando. Esto incluye el formulario de login.

La página tendrá dos secciones principales, una con el formulario de login y otra, la principal, para el resto de la aplicación.



La sección principal comienza oculta. Al hacer login con éxito se oculta la sección del formulario y se muestra la principal.

A su vez, la sección principal tendrá dos secciones, como la aplicación original, una cabecera y otra donde se mostrará el contenido que corresponda, es decir, la lista de categorías, la tabla de productos, el carrito y el mensaje de confirmación.

Las transiciones entre pantallas consisten en eliminar el contenido de esta sección y cargar el nuevo.

3. Lado del cliente

En el lado del cliente la principal dificultad está en las transiciones entre pantallas:

- Al hacer login, se oculta el formulario y se muestra la sección principal. El contenido inicial de la sección es la lista de categorías.
- Al cerrar sesión, se oculta la sección principal y se muestra de nuevo la del formulario.
- Al seguir un vínculo en la lista de categorías, se muestra la tabla de productos.
- Al añadir o eliminar un producto, se muestra el carrito.
- Al procesar un pedido, se muestra el mensaje de confirmación.

En todos los casos hay que establecer comunicación con el servidor y modificar el contenido de la página con la respuesta. Habrá una función para cada caso.

Función JavaScript	Descripción
login()	Valida el formulario, muestra la sección principal
cerrarSesionUnaPagina()	Cierra la sesión, muestra el formulario de login
cargarCategorias()	Solicita los datos de las categorías, crea la lista de vínculos
cargarProductos()	Solicita los datos de los productos, crea la tabla
cargarCarrito()	Solicita los datos del carrito, crea la tabla
anadirProductos()	Modifica el carrito y lo muestra
eliminarProductos()	Modifica el carrito y lo muestra
procesarPedido()	Envía la orden de procesar el pedido y muestra el resultado

Para organizar mejor el código, se utilizan algunas funciones auxiliares para la creación de la interfaz.

Función JavaScript auxiliar	Descripción
crearVinculoCategorias()	Crea los vínculos para lista de categorías
crearTablaCarrito()	Crea la tabla del carrito de la compra
crearTablaProductos()	Crea la tabla de productos
crearFila()	Crea una fila de las tablas de productos y carrito
crearFormulario()	Crea los formularios de añadir y eliminar

4. Lado del servidor

La parte del servidor es bastante parecida a la original. Las diferencias son:

- Los ficheros que devolvían HTML ahora devuelven JSON.
- Se eliminan las redirecciones, que no tienen sentido en una aplicación de una sola página. En su lugar, la respuesta será un código indicando si la operación se realizó con éxito o no.

Ruta	Descripción	Parámetros
login_json.php	Valida el formulario de entrada	$_POST['usuario'] $_POST['clave']
logout_json.php	Cierra la sesión	
categorias_json.php	Devuelve un documento JSON con los datos de las categorías	
productos_json.php	Devuelve un documento JSON con los datos de los productos de una categoría	$_GET['categoria']
carrito_json.php	Devuelve un documento JSON con los datos del carrito de la compra	
anadir_json.php	Añade productos al carrito	$_POST['cod'] $_POST['unidades']
eliminar_json.php	Elimina productos del carrito	$_POST['cod'] $_POST['unidades']
procesar_pedido_json.php	Inserta el pedido en la base de datos, envía correos de confirmación y devuelve error o éxito	
una_sola_pagina_json.php	Página principal y única de la aplicación	
sesiones_json.php	Para comprobar que el usuario haya iniciado sesión correctamente	
bd.php	Para agrupar las funciones de la base de datos	
correo.php	Funciones para enviar correo	

Los siete primeros ficheros están pensados para ser accedidos desde JavaScript. Para cada uno de ellos habrá una función de JavaScript que se encargue de llamarlos y procesar la respuesta.

5. Página HTML

El fichero inicial de la aplicación es **una_sola_pagina.php**. Se puede observar:

- Líneas 6-7: incluye los ficheros JavaScript.
- Línea 11: el envío del formulario está asociado a la función JavaScript `login()`.
- Líneas 12-13: los campos del formulario tienen id en lugar de `name`, para usarlos desde JavaScript.
- Línea 17: la sección con id principal comienza oculta.
- Líneas 19-22: la cabecera cambia completamente, los vínculos se sustituyen por llamadas a las funciones correspondientes.

El nombre del usuario no aparece, en su lugar, en la línea 19, aparece una etiqueta **span** que, si se hace login correctamente, se introduce desde JavaScript.

```
 1:  <!DOCTYPE html>
 2:  <html>
 3:  <head>
 4:     <title>Formulario de login</title>
 5:     <meta charset="UTF-8">
 6:     <script type="text/javascript" src="js/cargarDatos.js"></script>
 7:     <script type="text/javascript" src="js/sesion.js"></script>
 8:  </head>
 9:  <body>
10:     <section id="login">
11:         <form onsubmit="return login()" method="POST">
12:             Usuario <input id="usuario" type="text">
13:             Clave <input id="clave" type="password">
14:             <input type="submit">
15:         </form>
16:     </section>
17:     <section id="principal" style="display:none">
18:         <header>
19:             <span id="cab_usuario"></span>
20:             <a href="#" onclick="return cargarCategorias();">Home</a>
21:             <a href="#" onclick="return cargarCarrito();">Carrito</a>
22:            <a href="#" onclick="return cerrarSesionUnaPagina();">Cerrar sesión</a>
23:         </header>
24:         <hr>
25:         <h2 id="titulo"></h2>
26:         <section id="contenido">
27:         </section>
28:     </section>
29:  </body>
30:  </html>
```

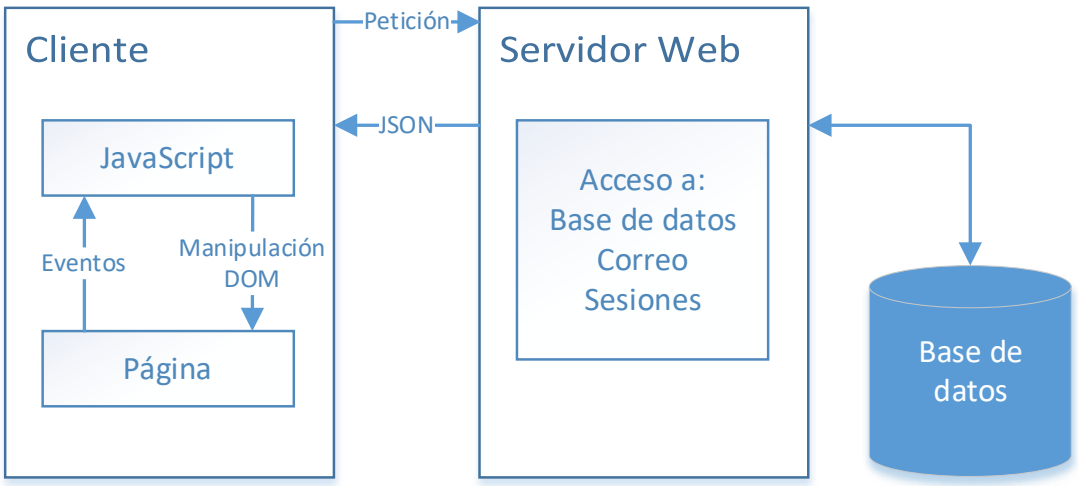
6. Código del lado del servidor

En esta versión de la aplicación los ficheros del servidor son más sencillos porque no hay que ocuparse de la salida. Tenemos dos tipos de archivos en el lado de servidor:

- Archivos que gestionan la base de datos. En nuestro caso, una única página: **bd.php**
- Archivos que procesan las peticiones desde el cliente y las respuestas de la base de datos: páginas **_json.php** y **correo.php**

Las páginas **bd.php** y **correo.php** son las originales de la aplicación pedidos, salvo la función `comprobar_usuarios()`, qué está modificada para poder validar las claves hasheadas de la base de datos `pedidos`.

La extensión PHPmailer puede agregarse a la carpeta del proyecto utilizando composer.



6.1. Login

Del login se encarga **login_json.php**. Es parecido al fichero de login original, pero solo tiene la parte de procesamiento. Además, no redirige. Si hay error, devuelve la cadena FALSE. Si no, crea las variables de sesión y devuelve la cadena TRUE.

```
1:  <?php
2:  require_once 'bd.php';
3:  /*formulario de login habitual
4:  si va bien abre sesión, guarda el nombre de usuario y redirige a principal.php
5:  si va mal, mensaje de error */
6:  if ($_SERVER["REQUEST_METHOD"] == "POST") {
7:     $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
8:     if($usu===FALSE){
9:         echo "FALSE";
10:    }else{
11:        session_start();
12:        // $usu tiene campos correo y codRes, correo
13:        $_SESSION['usuario'] = $usu;
14:        $_SESSION['carrito'] = [];
15:        echo "TRUE";
16:    }
17: }
```

6.2. Control de acceso

Del control de acceso se encarga **sesiones_json.php.** Originalmente se basaba en una redirección si el usuario no había iniciado sesión. Ahora, la función `comprobar_sesion()` devuelve TRUE o FALSE según haya sesión abierta o no.

```
1:  <?php
2:  function comprobar_sesion(){
3:      session_start();
4:      if(!isset($_SESSION['usuario'])){
5:          return false;
```

```
6:      }else return true;
7:  }
```

Los demás ficheros **_json.php** hacen la siguiente comprobación:

```
require 'sesiones_json.php';
if(!comprobar_sesion()) return;
```

Estás páginas no redirigen a **una_sola_pagina.php** puesto que estamos bajo el esquema de AJAX, en el que el servidor solo resuelve peticiones enviadas desde Javascript.

6.3. Las categorías

El fichero **categorias_json.php** devuelve los datos de la tabla de categorías en formato JSON. Utiliza la función `cargar_categorias()` que devuelve un cursor (un objeto, en particular uno de la clase PDOStatement). Utilizamos la función `iterator_to_array()` para convertirlo en un array.

```
1:  <?php
2:  require_once 'sesiones_json.php';
3:  require_once 'bd.php';
4:  if (!comprobar_sesion()) return;
5:  $categorias = cargar_categorias();
6:  $cat_json = json_encode(iterator_to_array($categorias));
7:  echo $cat_json;
```

6.4. Los productos

El fichero **productos_json.php** devuelve los datos de los productos de una categoría en formato JSON. El código de la categoría se pasa en la URL. Funciona de manera análoga al anterior.

```
1:  <?php
2:  require_once 'bd.php';
3:  /*comprueba que el usuario haya abierto sesión o devuelve*/
4:  require 'sesiones_json.php';
5:  if (!comprobar_sesion()) return;
6:  $productos_array = [];
7:  $productos = cargar_productos_categoria($_GET['categoria']);
8:  $cat_json = json_encode(iterator_to_array($productos));
9:  echo $cat_json;
```

6.5. El carrito de la compra

Para obtener los productos del carrito se usa **carrito_json.php**. Devuelve un array JSON con los datos de los productos presentes en el carrito y las unidades pedidas. Es muy parecido al **carrito.php** original.

En la línea 8 escribimos `&` delante de `$producto`. Esto nos permite modificar directamente los elementos del array dentro de bucle.

```php
<?php
require 'sesiones_json.php';
require_once 'bd.php';
if (!comprobar_sesion()) return;
$productos = cargar_productos(array_keys($_SESSION['carrito']));
// hay que añadir las unidades
$productos = iterator_to_array($productos);
foreach ($productos as &$producto) {
   $cod = $producto['CodProd'];
   $producto['unidades'] = $_SESSION['carrito'][$cod];
}
echo json_encode($productos);
```

6.6. Añadir y eliminar productos

Los ficheros **anadir_json.php** y **eliminar_json.php**, son prácticamente iguales a los de la aplicación anterior. El único cambio es que no redirigen.

anadir_json.php

```php
<?php
/*comprueba que el usuario haya abierto sesión o devuelve*/
require 'sesiones_json.php';
if (!comprobar_sesion()) return;
$cod = $_POST['cod'];
$unidades = (int)$_POST['unidades'];
/*si existe el código sumamos las unidades*/
if (isset($_SESSION['carrito'][$cod])) {
   $_SESSION['carrito'][$cod] += $unidades;
} else {
   $_SESSION['carrito'][$cod] = $unidades;
}
```

eliminar_json.php

```php
<?php
/*comprueba que el usuario haya abierto sesión*/
require_once 'sesiones_json.php';
if(!comprobar_sesion()) return;
$cod = $_POST['cod'];
$unidades = $_POST['unidades'];
/*si existe el código restamos las unidades, con mínimo de 0*/
if(isset($_SESSION['carrito'][$cod])){
   $_SESSION['carrito'][$cod] -= $unidades;
   if($_SESSION['carrito'][$cod] <= 0){
       unset($_SESSION['carrito'][$cod]);
   }

}
```

6.7. Cerrar la sesión

Lo mismo ocurre con **logut_json.php**, es similar al original, solo que no redirige.

```
1:  <?php
2:  session_start();
3:  $_SESSION = array();
4:  session_destroy();  // eliminar la sesion
5:  setcookie(session_name(), 123, time() - 1000); // eliminar la cookie
```

6.8. Procesar el pedido

El fichero **procesar_pedido.php** también es muy parecido al de la primera versión. En lugar de devolver un mensaje, devuelve las cadenas TRUE o FALSE según se haya podido insertar el pedido o no. Utiliza las funciones de correo originales, **correo.php**.

```
 1:  <?php
 2:  /*comprueba que el usuario haya abierto sesión o redirige*/
 3:  require 'correo.php';
 4:  require_once 'bd.php';
 5:  /*comprueba que el usuario haya abierto sesión o devuelve*/
 6:  require 'sesiones_json.php';
 7:  if (!comprobar_sesion()) return;
 8:  $resul = insertar_pedido($_SESSION['carrito'], $_SESSION['usuario']['CodRes']);
 9:  if ($resul === FALSE) {
10:     echo "FALSE";
11:  } else {
12:     $correo = $_SESSION['usuario']['Correo'];
13:     $conf = enviar_correos($_SESSION['carrito'], $resul, $correo);
14:     echo "TRUE";
15:     //vaciar carrito
16:     $_SESSION['carrito'] = [];
17:  }
```

7. Código del lado del cliente

Las funciones de JavaScript se reparten en dos ficheros:

- **sesion.js**: Con las funciones para hacer login y cerrar sesión.
- **cargarDatos.js**: Con las funciones que piden los datos al servidor y modifican la estructura de la página.

Veamos las funciones de login y cerrar sesión

7.1. Login

El formulario de login está asociado con la función **`login()`** de **sesion.js**, que envía a login_json.php los datos del formulario. Si los datos enviados son correctos, entonces:

- Muestra la sección principal (línea 25).
- Oculta la sección del formulario (línea 26).
- Introduce el nombre del usuario en la parte apropiada de la cabecera (línea 28).
- Si no son correctos, muestra un mensaje de error con una alerta (línea 23).

```
18: function login(formu) {
19:     var xhttp = new XMLHttpRequest();
20:     xhttp.onreadystatechange = function () {
21:         if (this.readyState == 4 && this.status == 200) {
22:             if (this.responseText === "FALSE") {
23:                 alert("Revise usuario y contraseña");
24:             } else {
25:                 document.getElementById("principal").style.display = "block";
26:                 document.getElementById("login").style.display = "none";
27:                 /*ponemos el usuario devuelto en el hueco correspondiente*/
28:                 document.getElementById("cab_usuario").innerHTML = "Usuario: " +
                        usuario;
29:                 cargarCategorias();
30:             }
31:         }
32:     }
33:     var usuario = document.getElementById("usuario").value;
34:     var clave = document.getElementById("clave").value;
35:     var params = "usuario=" + usuario + "&clave=" + clave;
36:     xhttp.open("POST", "login_json.php", true);
37:     // envío con  POST requiere cabecera y cadena de parámetros
38:     xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
39:     xhttp.send(params);
40:     return false;
41: }
```

La petición se realiza en la línea 39.

Como se trata de enviar un formulario se utiliza el método POST. Por tanto, hay que añadir la cabecera específica (línea 38) y enviar una cadena con los parámetros.

7.2. Cerrar sesión

El enlace para cerrar sesión está asociado a la función **`cerrarSesionUnaPagina()`** de **sesión.js**. Oculta las secciones principales y borra su contenido. Además, muestra de nuevo la sección de login.

```
1: function cerrarSesionUnaPagina() {
2:     /*cerrar sesión*/
3:     var xhttp = new XMLHttpRequest();
4:     xhttp.onreadystatechange = function () {
5:         if (this.readyState == 4 && this.status == 200) {
```

```
 6:             /*cambiar visibilidades de las secciones*/
 7:             document.getElementById("principal").style.display = "none";
 8:             document.getElementById("login").style.display = "block";
 9:             document.getElementById("contenido").innerHTML = "";
10:             alert("Sesion cerrada con éxito");
11:         }
12:     };
13:     xhttp.open("GET", "logout_json.php", true);
14:     xhttp.send();
15:     return false;
16: }
```

7.3. Las categorías

La función **cargarCategorias()** solicita al servidor los datos de las categorías (categorias_json.php) y crea la lista de vínculos.

```
47: function cargarCategorias() {
48:     var xhttp = new XMLHttpRequest();
49:     xhttp.onreadystatechange = function () {
50:         if (this.readyState == 4 && this.status == 200) {
51:             var cats = JSON.parse(this.responseText);
52:             var lista = document.createElement("ul");
53:             for (var i = 0; i < cats.length; i++) {
54:                 var elem = document.createElement("li");
55:                 vinculo = crearVinculoCategorias(cats[i].nombre,
                    cats[i].codCat);
56:                 elem.appendChild(vinculo);
57:                 lista.appendChild(elem);
58:             }
59:             var contenido = document.getElementById("contenido");
60:             contenido.innerHTML = "";
61:             var titulo = document.getElementById("titulo");
62:             titulo.innerHTML = "Categorías";
63:             contenido.appendChild(lista);
64:         }
65:     };
66:     xhttp.open("GET", "categorias_json.php", true);
67:     xhttp.send();
68:     return false;
69: }
```

Cuando recibe la respuesta del servidor:

- La convierte en un objeto JavaScript, que será un array (línea 51). Cada elemento del array será un objeto con campos `codCat` y `nombre` (los nombres de los campos en el array JSON que devuelve el servidor).
- Crea un elemento `ul`, la lista que tiene que crear (línea 52).
- Por cada elemento del array crea un vinculo usando el nombre y el código (línea 55). Ese elemento se introduce en un elemento `li` (línea 56).

- El elemento `li` se introduce en la lista (línea 57).
- Elimina el contenido de la sección "contenido" y luego introduce la lista en ella (líneas 59-63).

Para crear los vínculos se utiliza una función auxiliar `crearVinculoCategorias()` (línea 55).

```
157: function crearVinculoCategorias(nom, cod) {
158:     var vinculo = document.createElement("a");
159:     var ruta = "productos_json.php?categoria=" + cod;
160:     vinculo.href = ruta;
161:     vinculo.innerHTML = nom;
162:     vinculo.onclick = function () { return cargarProductos(this); }
163:     return vinculo;
164: }
```

Mediante el atributo `onclick`, estos vínculos están asociados a la función `cargarProductos()`. En `href` está la ruta del vínculo.

7.4. Los productos

La función **`cargarProductos()`** es muy parecida a `cargarCategorias()`.

- Pide los datos al servidor según el destino (productos_json.php`?categoria=" + cod;`).
- Crea la tabla de productos.
- Elimina el contenido de la sección "contenido" y luego introduce la tabla en ella.

Si la categoría no tuviera productos (líneas 83-88) capturaríamos el error y mostraríamos el contenido de la línea 85

```
71: function cargarProductos(destino) {
72:     var xhttp = new XMLHttpRequest();
73:     xhttp.onreadystatechange = function () {
74:         if (this.readyState == 4 && this.status == 200) {
75:             var prod = document.getElementById("contenido");
76:             var titulo = document.getElementById("titulo");
77:             titulo.innerHTML = "Productos";
78:             try {
79:                 var filas = JSON.parse(this.responseText);
80:                 var tabla = crearTablaProductos(filas);
81:                 prod.innerHTML = "";
82:                 prod.appendChild(tabla);
83:             } catch (e) {
84:                 var mensaje = document.createElement("p");
85:                 mensaje.innerHTML = "Categoría sin productos";
86:                 prod.innerHTML = "";
87:                 prod.appendChild(mensaje);
88:             }
89:         }
90:     };
91:     xhttp.open("GET", destino, true);
92:     xhttp.send();
93:     return false;
94: }
```

Para crear la tabla utiliza la función auxiliar `crearTablaProductos(filas)` (línea 80).

Esta función recibe el array de productos y devuelve un elemento `<table>` con una fila por producto.

```
141: function crearTablaProductos(productos) {
142:     var tabla = document.createElement("table");
143:     var cabecera = crear_fila(["Código", "Nombre", "Descripción", "Stock",
         "Comprar"], "th");
144:     tabla.appendChild(cabecera);
145:     for (var i = 0; i < productos.length; i++) {
146:         /*formulario*/
147:         formu = crearFormulario("Añadir", productos[i]['CodProd'],
             anadirProductos);
148:         fila = crear_fila([productos[i]['CodProd'], productos[i]['Nombre'],
             productos[i]['Descripcion'], productos[i]['Stock']], "td");
149:         celda_form = document.createElement("td");
150:         celda_form.appendChild(formu);
151:         fila.appendChild(celda_form);
152:         tabla.appendChild(fila);
153:     }
154:     return tabla;
155: }
```

Para crear cada fila, se usa la función `crear_fila(campos, tipo)`(líneas 143 y 148) Esta función recibe un array con los campos a insertar en cada fila y el tipo de elemento a crear en la fila.

```
 96: function crear_fila(campos, tipo) {
 97:     var fila = document.createElement("tr");
 98:     for (var i = 0; i < campos.length; i++) {
 99:         var celda = document.createElement(tipo);
100:         celda.innerHTML = campos[i];
101:         fila.appendChild(celda);
102:     }
103:     return fila;
104: }
```

Para crear el formulario “Añadir” de cada fila se usa la función `crearFormulario(texto, cod, funcion)` (línea 147). Esta función recibe tres parámetros:

- El texto del botón del formulario.
- El código del producto.
- La función que se encarga de enviar el formulario.

En la tabla de productos el formulario se envía con la función `anadirProductos()`. El formulario queda asociado con la función a través del atributo `onsubmit`. Es decir, cuando se pulse el botón de envió del formulario, se llamará a la función.

```
106: function crearFormulario(texto, cod, funcion) {
107:     var formu = document.createElement("form");
108:     var unidades = document.createElement("input");
109:     unidades.value = 1;
110:     unidades.name = "unidades";
```

```
111:     var codigo = document.createElement("input");
112:     codigo.value = cod;
113:     codigo.type = "hidden";
114:     codigo.name = "cod";
115:     var bsubmit = document.createElement("input");
116:     bsubmit.type = "submit";
117:     bsubmit.value = texto;
118:     formu.onsubmit = function () { return funcion(this); }
119:     formu.appendChild(unidades);
120:     formu.appendChild(codigo);
121:     formu.appendChild(bsubmit);
122:     return formu;
```

7.5. El carrito

cargarCarrito() solicita al servidor los datos de los productos del carrito (carrito_json.php) y crea la tabla de productos. Elimina el contenido de la sección "contenido" y luego introduce la tabla en ella. También muestra un vínculo para realizar el pedido, que está asociado con la función realizarPedido().

```
16: function cargarCarrito() {
17:     var xhttp = new XMLHttpRequest();
18:     xhttp.onreadystatechange = function () {
19:         if (this.readyState == 4 && this.status == 200) {
20:             var contenido = document.getElementById("contenido");
21:             contenido.innerHTML = "";
22:             var titulo = document.getElementById("titulo");
23:             titulo.innerHTML = "Carrito de la compra";
24:             try {
25:                 var filas = JSON.parse(this.responseText);
26:                 tabla = crearTablaCarrito(filas);
27:                 contenido.appendChild(tabla);
28:                 /*ahora el vínculo de procesar pedio*/
29:                 var procesar = document.createElement("a");
30:                 procesar.href = "#";
31:                 procesar.innerHTML = "Realizar pedido";
32:                 procesar.onclick = function () { return procesarPedido(); };
33:                 contenido.appendChild(procesar);
34:             } catch (e) {
35:                 var mensaje = document.createElement("p");
36:                 mensaje.innerHTML = "Todavía no tiene productos";
37:                 contenido.appendChild(mensaje);
38:             }
39:
40:         }
41:     };
42:     xhttp.open("GET", "carrito_json.php", true);
43:     xhttp.send();
44:     return false;
45: }
```

Para crear la tabla utiliza la función auxiliar **crearTablacarrito(filas)**, que es muy parecida a la de crear productos.

```
125: function crearTablaCarrito(productos) {
126:     var tabla = document.createElement("table");
127:     var cabecera = crear_fila(["Código", "Nombre", "Descripción", "Unidades",
         "Eliminar"], "th");
128:     tabla.appendChild(cabecera);
129:     for (var i = 0; i < productos.length; i++) {
130:         /*formulario*/
131:         formu = crearFormulario("Eliminar", productos[i]['CodProd'],
             eliminarProductos);
132:         fila = crear_fila([productos[i]['CodProd'], productos[i]['Nombre'],
             productos[i]['Descripcion'], productos[i]['unidades']], "td");
133:         celda_form = document.createElement("td");
134:         celda_form.appendChild(formu);
135:         fila.appendChild(celda_form);
136:         tabla.appendChild(fila);
137:     }
138:     return tabla;
139: }
```

Para crear el formulario de cada fila también se usa la función `crearFormulario (texto, cod, funcion)`, pero como función se pasa `eliminarProductos()`.

7.6. Añadir y eliminar

La función de añadir productos, **anadirProductos(formulario)**, está asociada al formulario de la tabla de productos. Cuando se envía el formulario se ejecuta la función. El argumento de la función es el propio formulario. La función se encarga de coger los datos del formulario y enviarlos a anadir_json.php. Como es un envío POST hay que crear una cabecera y una cadena con los parámetros. Al recibir la respuesta muestra una alerta y llama a `cargarCarrito()`.

La función eliminar es igual, pero llama a **eliminar_json.php**.

```
 1: function anadirProductos(formulario) {
 2:     var xhttp = new XMLHttpRequest();
 3:     xhttp.onreadystatechange = function () {
 4:         if (this.readyState == 4 && this.status == 200) {
 5:             alert("Producto añadido con éxito");
 6:             cargarCarrito();
 7:         }
 8:     };
 9:     var params = "cod=" + formulario.elements['cod'].value + "&unidades=" +
        formulario.elements['unidades'].value;
10:     xhttp.open("POST", "anadir_json.php", true);
11:     xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
12:     xhttp.send(params);
13:     return false;
14: }
```

7.7. Realizar el pedido

La función **procesarPedido()** solicita procesar_pedido_json.php al servidor. Muestra un mensaje de confirmación o error según reciba las cadenas TRUE o FALSE en la respuesta.

```
183: function procesarPedido() {
184:     var xhttp = new XMLHttpRequest();
185:     xhttp.onreadystatechange = function () {
186:         if (this.readyState == 4 && this.status == 200) {
187:             var contenido = document.getElementById("contenido");
188:             contenido.innerHTML = "";
189:             var titulo = document.getElementById("titulo");
190:             titulo.innerHTML = "Estado del pedido";
191:             if (this.responseText == "TRUE") {
192:                 contenido.innerHTML = "Pedido realizado";
193:             } else {
194:                 contenido.innerHTML = "Error al procesar el pedido";
195:             }
196:         }
197:     };
198:     xhttp.open("GET", "procesar_pedido_json.php", true);
199:     xhttp.send();
200:     return false;
201: }
```

8. Ejercicios

Modifica la aplicación para que:

1. No redirija al carrito de la compra al añadir o eliminar productos.

2. Pida confirmación al usuario antes de realizar el pedido.

3. El carrito de la compra esté siempre visible.

4. Muestre un vínculo "Zona admin" solo a los usuarios con rol 1.

5. Al procesar pedido, se resten las unidades correspondientes del stock permitiendo que queden en negativo.

6. Al hacer clic en "Zona admin" se ha de generar en la sección del contenido un formulario en forma de tabla que permita modificar los datos de los restaurantes, salvo la clave.

La petición de modificación la recibe restaurantes_json.php, la cual, tras recuperar los datos de la base de datos a través de la función `cargar_restaurantes()` de bd.php, devuelve los datos en formato JSON a javascritp.

En javascript, hacer uso de las funciones auxiliares ya definidas en la medida de lo posible.