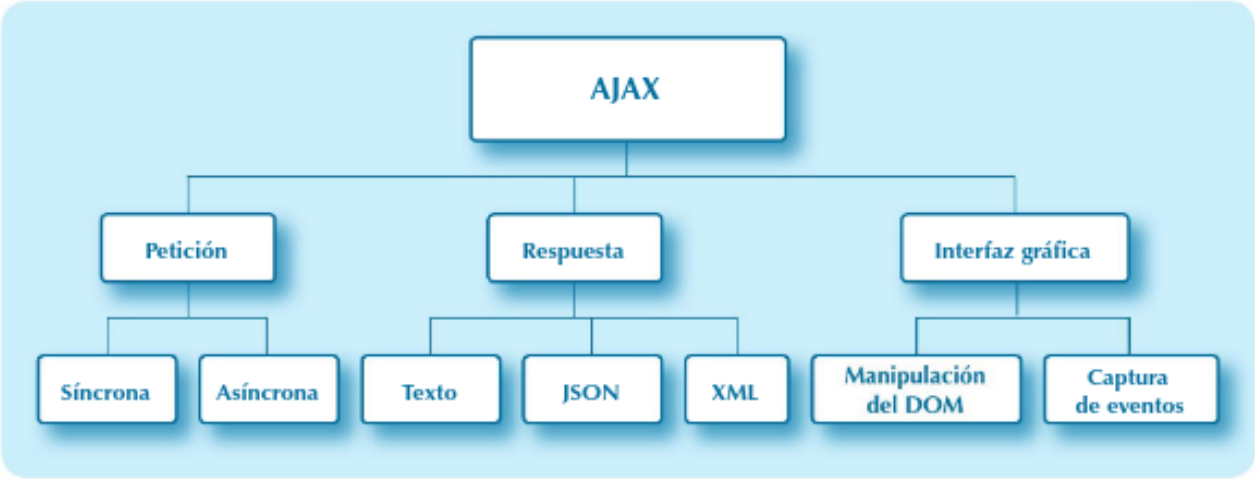
5

Aplicaciones web dinámicas con AJAX

Objetivos

- ✓ Comprender las ventajas de separar la lógica de presentación y la de negocio.
- ✓ Entender el papel de JavaScript dentro de las aplicaciones web.
- ✓ Presentar la técnica AJAX.
- ✓ Entender el funcionamiento de las peticiones asíncronas.
- ✓ Utilizar JSON para las respuestas del servidor.
- ✓ Conocer los principios de las aplicaciones de una sola página.

Mapa conceptual



Glosario

AJAX. *Asyncronous JavaScript and XML*. Técnica de diseño de aplicaciones web que se basa en realizar peticiones al servidor desde JavaScript.

Aplicación web de una sola página. Aplicación en la que todas las peticiones al servidor, salvo la inicial, se hacen de forma dinámica utilizando AJAX.

DOM. Interfaz que permite acceder y manipular el contenido de un documento.

JavaScript. Lenguaje de programación en el lado del cliente. Es uno de los elementos básicos del desarrollo web.

JSON. *JavaScript ObjectNotation*. Lenguaje para intercambio de datos en formato texto muy extendido en desarrollo web.

Lógica de negocio. Es la lógica específica de la aplicación, donde realmente se desarrolla la funcionalidad de esta. Procesa la información que introduce el usuario y maneja la base de datos.

Lógica de presentación. Parte de la aplicación que se ocupa de mostrar la información al usuario.

Petición asíncrona. Cuando se realiza una petición asíncrona, el *script* sigue ejecutándose mientras se espera una respuesta.

Petición síncrona. Cuando se realiza una petición síncrona, la ejecución del *script* se detiene hasta que se recibe la respuesta.

5.1. Separación de la lógica de negocio

Uno de los temas principales en el diseño de aplicaciones es la reusabilidad del código. Para conseguirlo, las aplicaciones se dividen en partes que se relacionan entre sí, pero que también tienen sentido de manera independiente.

A pequeña escala, se crean funciones y clases que, si están bien diseñadas, se pueden reutilizar en otras aplicaciones. A nivel de diseño de aplicaciones, se plantea una arquitectura con varios componentes desacoplados que se ocupan de las diferentes partes de la aplicación.

En los ejemplos realizados hasta ahora, la lógica de negocio y la de presentación están completamente mezcladas. Un ejemplo está en la tabla de productos de la aplicación del capítulo anterior. Hay varios bloques de PHP, etiquetas de HTML en la plantilla y otras etiquetas HTML generadas mediante `echo`. Incluso en un ejemplo sencillo, el código se complica rápidamente.

Para mejorar esta situación, el primer paso sería aislar la salida en funciones específicas que se ocuparan exclusivamente de la salida. Mejor aún, se podría utilizar una librería de plantillas, como se verá en el capítulo 7.

Otro enfoque consiste en llevar la lógica de presentación al cliente. Al fin y al cabo, en el cliente es donde se muestra la información. El lenguaje más extendido en el lado del cliente es JavaScript, que se ejecuta dentro del navegador. Aunque no ha aparecido hasta ahora, es un elemento fundamental en el desarrollo de aplicaciones web. Tiene muchas utilidades, pero en este capítulo se utiliza para realizar peticiones al servidor y mostrar los datos recibidos.

AJAX es una técnica de diseño de aplicaciones web que se basa en hacer peticiones al servidor desde JavaScript. En los capítulos anteriores la salida de la página web se genera por completo en el servidor. Por ejemplo, la tabla de productos de la aplicación de pedidos. En AJAX, simplificando, el servidor devuelve solo los datos que hay que mostrar. Es decir, los datos de los productos. En el cliente, la interfaz se actualiza según los datos recibidos, también utilizando JavaScript.

De esta manera se desacopla la lógica de negocio de la de presentación. Tiene varias ventajas:

- El código es más fácil de mantener y modificar, al no estar mezclado.
- El código del servidor se puede reutilizar con otros clientes, como una aplicación de móvil.
- Se puede trabajar en la parte del cliente y en la del servidor en paralelo.

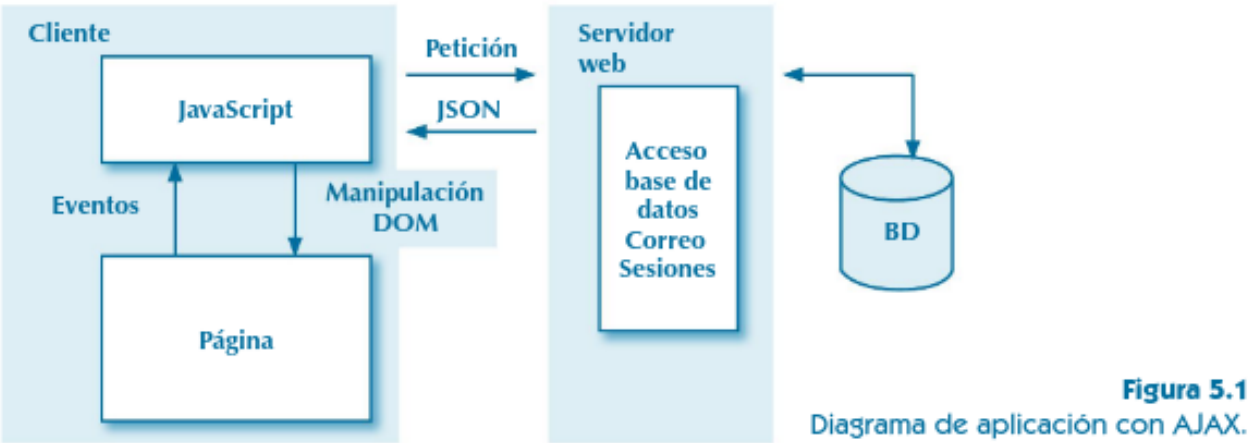


Figura 5.1
Diagrama de aplicación con AJAX.

5.2. Tecnologías y librerías asociadas

El código JavaScript también llega al cliente desde el servidor incluido en páginas HTML. Se ejecuta en el navegador y puede manipular la página para cambiar su aspecto, mostrar mensajes de alerta y establecer comunicaciones con el servidor.

Para modificar la página web, JavaScript utiliza el DOM, un estándar del W3C. Es una interfaz que permite acceder y modificar el contenido, la estructura y el estilo de un documento HTML. Los documentos se representan mediante una estructura de árbol en la que los nodos son los elementos HTML. Los elementos forman parte de una jerarquía y se puede hablar de elementos padres, hijo, descendientes y antecesores

Por ejemplo, para una página sencilla:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo DOM</title>
    </head>
    <body>
    </body>
</html>
```

El elemento `html` es el nodo raíz, y es el padre de `head` y `body`. También es antecesor del resto de elementos de la página. De la misma manera, `title` es hijo de `head` y descendiente de `html`.

Desde JavaScript se puede modificar el contenido de los elementos, sus atributos y su estilo. En el cuadro 5.1 se resumen las funciones que se usan a lo largo del capítulo para trabajar con el DOM.

CUADRO 5.1
Funciones y atributos básicos para manipular el DOM

Función	Descripción
document.createElement(<etiqueta>)	Crea un nuevo elemento
document.getElementById(id)	Devuelve el elemento de la página que tiene ese id
elem.appendChild(elem_hijo)	Inserta elem_hijo como hijo de elem
elem.innerHTML	Propiedad que representa el contenido de un elemento

TOMA NOTA

Aunque el módulo Desarrollo web en entorno servidor, y por lo tanto este libro, se centran en el lado del servidor, para completar los ejemplos de este capítulo es necesario utilizar JavaScript. Los ejemplos de JavaScript están pensados para alcanzar la funcionalidad de manera sencilla, para que puedan entenderse sin necesidad de entrar en los detalles del lenguaje.

5.3. Obtención remota de información

Hasta ahora, se ha visto que se solicita una página al servidor:

- Cuando se accede a una página introduciendo la URL en el navegador.
- Al seguir un vínculo.

- Al enviar un formulario.
- Como consecuencia de una redirección.

En todos los casos, el navegador cambia el contenido que esté mostrando con la respuesta que obtiene del servidor. Este modelo es bastante restrictivo y condiciona el diseño de las páginas, ya que cada interacción con el servidor requiere que este acabe devolviendo una página web completa.

Es posible superar esta limitación utilizando JavaScript, en concreto AJAX. Por ejemplo, una página web puede incluir un *script* que cada minuto consulte las últimas noticias y muestre los titulares en una sección lateral. El *script* se encarga de obtener los datos y de modificar el contenido solo de esa sección, sin tener que recargar la página.

El elemento central de AJAX es el objeto `XMLHttpRequest`, que representa una petición al servidor. Utilizándolo es posible hacer una petición al servidor sin que su respuesta se muestre automáticamente como una nueva página en el navegador. En lugar de eso, la respuesta se procesa mediante JavaScript.

5.3.1. Peticiones síncronas y asíncronas

Las peticiones pueden ser *síncronas* o *asíncronas*. En el primer caso, la ejecución se detiene hasta que se recibe la respuesta. En el segundo, la ejecución continúa y, cuando se recibe una respuesta, se procesa.

Los métodos básicos para enviar una petición son:

1. open(método, destino, asíncrona). Recibe el método HTTP con el que se realiza la petición, la ruta que se solicita y si la petición es asíncrona o no.
2. send([params]). Realiza el envío. Tiene un argumento opcional para añadir parámetros a la petición.
3. setRequestHeader(cabecera). Para establecer las cabeceras de la petición. Se usa, si es necesario, entre `open()` y `send()`.

Por ejemplo, para pedir la ruta **hora_servidor.php** usando el método GET de manera síncrona, se utilizaría:

```
var xhttp = new XMLHttpRequest();
xhttp.open("GET", "hora_servidor.php", false);
xhttp.send();
```

Al ser una petición síncrona, la ejecución del *script* se detiene hasta que se recibe una respuesta o pasa el tiempo máximo de espera. Una vez recibida, se almacena en la propiedad `response`. El código HTTP de la respuesta se almacena en `status`.

```
if (xhttp.status == 200){
    alert("OK");
}else {
    alert("Error");
}
alert(xhttp.response);
```

RECUERDA

Si la petición utiliza el método POST, hay que incluir una cabecera específica y añadir una cadena de parámetros.

```
xhttp.open("POST", ruta, true);
xhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xhttp.send("param1=value1&param2=value2");
```

En las peticiones asíncronas, que son las más habituales, la situación se complica un poco. Al realizar la petición, el *script* sigue ejecutándose sin esperar a la respuesta. Por lo tanto, hay que indicar de alguna manera qué código se encargará de gestionar la respuesta cuando llegue.

Para hacerlo se utiliza la propiedad `onreadystatechange`, que permite asociar una función a un cambio en el estado de la petición. El estado de la petición se representa mediante el atributo `readystate`, que puede tomar los valores recogidos en el cuadro 5.2.

CUADRO 5.2
Estado de una petición

Valor	Nombre	Descripción
0	UNSET	No se ha llamado a open()
1	OPENED	Se ha llamado a open(), pero no a send()
2	HEADERS_RECEIVED	Se han recibido las cabeceras de la respuesta
3	LOADING	Se está recibiendo el cuerpo de la respuesta
4	DONE	Se ha recibido la respuesta

El estado de la petición va cambiando según se desarrolla la comunicación con el servidor. En cada cambio, se llama a la función indicada en `onreadystatechange`.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        alert(this.response);
    }
};
xhttp.open("GET", "hora_servidor.php", true);
xhttp.send();
```

Es bastante habitual que estas funciones comiencen comprobando que el estado es 4, comunicación finalizada, y que el servidor ha devuelto el código 200, que indica que no ha habido errores. La primera comprobación es necesaria para que no se ejecute el código cada vez que cambia el estado.

TOMA NOTA

Si la petición se resuelve sin problemas, el servidor devuelve el código 200 (*OK*). Otros códigos habituales son 404 (*Not Found*), 400 (*Bad Request*) o 500 (*Internal Server Error*).

Actividad propuesta 5.1

Modifica el código precedente para que compruebe el código de error 404 y muestre una alerta cuando ocurra.

5.4. Respuesta del servidor

Como no se espera que la respuesta sea una página, no tiene por qué consistir en HTML. En casos sencillos puede ser texto sin formato. En los ejemplos anteriores se llama a **hora_servidor.php**. Este fichero devuelve simplemente una cadena de texto con la hora.

```
<?php
echo date("h:m:s");
```

Para devolver datos estructurados (como una lista de productos) se puede utilizar JSON o XML. Cuando se popularizó AJAX era habitual devolver los datos como XML, de ahí el nombre. Actualmente, en el desarrollo web está muy extendido JSON porque, en el lado del cliente, se pueden transformar en una variable de JavaScript con la función `JSON.parse()`.

El ejemplo **datos_categorias_json.php** devuelve los datos de un *array* codificados como JSON.

```
<?php
$cat1 = array("cod" => 1, "nombre" => "Comida");
$cat2 = array("cod" => 2, "nombre" => "Bebida");
$array = array($cat1, $cat2);
$json = json_encode($array);
echo $json;
```

Si se accede al fichero directamente desde el navegador, se obtiene:

[{"cod":1,"nombre":"Comida"},{"cod":2,"nombre":"Bebida"}]

Actividad propuesta 5.2

Modifica el ejemplo **datos_categorias_json.php** para que cargue las categorías de la base de datos de la aplicación de pedidos (capítulo 4).

5.5. Modificación de la estructura de la página web

En muchos casos, la respuesta del servidor consiste en una serie de datos que hay que mostrar de una forma u otra en la página. Puede ser que simplemente haya que actualizar una sección o que haya que modificar la estructura de la página, introduciendo y eliminando elementos.

En el ejemplo **hora.php** se muestra un caso básico. Se trata de una página que simplemente muestra la hora del servidor. En lugar de usar una alerta, como en el caso anterior, modifica el contenido de la página utilizando el DOM. Cada cinco segundos solicita el fichero **hora _servidor.php**.

- La hora del servidor se muestra en la sección con `id="hora"`, línea 24. Inicialmente está vacía.
- En las líneas 7-18 se declara la función `loadDoc()`. Se encarga de solicitar **hora_servidor.php** al servidor y de poner la respuesta en la página (líneas 11-12).
- Para seleccionar la sección utiliza `document.getElementById("hora")`. El contenido se modifica con la propiedad `innerHTML`.
- La petición se realiza en la línea 15. Los parámetros son el método HTTP mediante el que se realizará la petición, la ruta solicitada y si la petición es asíncrona o no. Con `true` se especifica que la petición es asíncrona.
- En la línea 19, la función `setInterval()` hace que se llame a `loadDoc()` cada 5 segundos.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Hora en el servidor</title>
5          <meta charset = "UTF-8">
6          <script>
7          function loadDoc() {
8              var xhttp = new XMLHttpRequest();
9              xhttp.onreadystatechange = function() {
10                 if (this.readyState == 4 && this.status == 200) {
11                     document.getElementById("hora").innerHTML =
12                     "Hora en el servidor:" + this.responseText;
13                 }
14             };
15             xhttp.open("GET", "hora_servidor.php", true);
16             xhttp.send();
17             return false;
18         }
19         setInterval(loadDoc, 5000);
20         </script>
21     </head>
22     <body>
23         <h1>Hora en el servidor</h1>
24         <section id="hora"></section>
25     </body>
26 </html>
```

Si se accede a **hora.php** desde el navegador, al principio solo se verá el título. Cinco segundos después aparecerá la hora.

El siguiente ejemplo muestra cómo procesar datos en JSON. Solicita al servidor **datos_categorias_json.php** y, con los datos que recibe, crea una lista de categorías.

```
1  function categorias(){
2     var xhttp = new XMLHttpRequest();
3     xhttp.onreadystatechange = function() {
4        if (this.readyState == 4 && this.status == 200) {
5           // crear lista
6           var lista = document.createElement("ul");
7           // meter los datos de la respuesta en un array
8           var cats = JSON.parse(this.response);
9           // para cada elemento del array
10          for(var i = 0; i < cats.length; i++){
11             //se crea un elemento ul con el campo nombre
12             var elem = document.createElement("li");
13             elem.innerHTML = cats[i]["nombre"];
14             // se añade a la lista
15             lista.appendChild(elem);
16          }
17          var body = document.getElementById("principal");
18          // eliminar el contenido actual
19          body.innerHTML = "";
20          body.appendChild(lista);
21       }
22    };
23    xhttp.open("GET", "datos_categorias_json.php", true);
24    xhttp.send();
25    // para que no se siga el link que llama a esta función
26    return false;
27 }
```

La función solicita el fichero y cuando se recibe la respuesta:

- Línea 6: se crea un elemento `ul`.
- Línea 8: se transforma la respuesta en un *array*.
- Líneas 10-16: se recorre la respuesta creando un elemento `li` por cada elemento del *array*. El texto será el campo nombre. Estos elementos se añaden a la lista.
- Línea 19: se elimina el contenido que pueda haber en la sección principal.
- Línea 20: inserta la lista dentro del elemento principal.

La página es muy sencilla:

- Incluye el fichero en el que está la función con la etiqueta `script`.
- Define la sección en la que irá la lista.
- Llama a la función.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AJAX</title>
        <meta charset = "UTF-8">
        <script type = "text/JavaScript" src = "funciones.js"></script>
    </head>
    <body>
        <section id = "principal"></section>
        <script type = "text/JavaScript" >categorias();</script>
    </body>
</html>
```

Actividad propuesta 5.3

Modifica el ejemplo anterior para que los elementos de la lista sean vínculos. El texto de los vínculos tiene que ser el nombre de cada categoría. Tienen que apuntar a *productos.php?categoria?<código>*, donde código es el código de la categoría.

5.6. Captura de eventos

Muchas veces se llama a una función JavaScript cuando el usuario sigue un vínculo o envía un formulario. Como ya se ha comentado, en estos casos el navegador por defecto carga la respuesta como una nueva página.

Para evitarlo, lo más sencillo es utilizar el atributo `onclick` (para los vínculos). Con este atributo se asocia el evento *click* con una función. Si esta función devuelve FALSE, no se sigue el vínculo. En los formularios se utiliza el atributo `onsubmit`.

El siguiente ejemplo es similar al anterior. En lugar de mostrar la lista de categorías directamente, muestra un vínculo que llama a la función.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AJAX</title>
        <meta charset = "UTF-8">
        <script type = "text/JavaScript" src = "funciones.js"></script>
    </head>
        <body>
        <section id = "principal"></section>
        <a href = "#" onclick = "return categorias();">Categorías</a>
    </body>
</html>
```

Actividad propuesta 5.4

Modifica el ejemplo anterior para añadir un vínculo que cargue en la sección principal la hora en el servidor (utilizando **hora_servidor.php**).

5.7. Aplicaciones de una sola página

Una aplicación web, como la aplicación de pedidos, implica una serie de intercambios entre cliente y servidor. En las aplicaciones de una sola página todas estas comunicaciones, salvo la inicial, se hacen utilizando JavaScript.

En estas aplicaciones la página nunca se recarga por completo. Se va actualizando desde JavaScript como se ha visto a lo largo del capítulo. Desde el punto de vista del usuario, como no hay recargas, estas páginas son más parecidas a aplicaciones de escritorio.

Este enfoque se ha popularizado en los últimos años y se utiliza en varios *frameworks* muy extendidos, como AngularJs o React.

Actividad propuesta 5.5

Investiga qué librerías de JavaScript pueden ser útiles para AJAX y para manipular el DOM. Una de ellas es JQuery: https://jquery.com/.

Resumen

- Repartir la lógica entre cliente y servidor permite obtener código más reutilizable.
- La lógica de presentación se desplaza al cliente, donde se utiliza JavaScript.
- El servidor devuelve los datos en JSON, XML o cualquier otro formato.
- Las peticiones al servidor se realizan utilizando el objeto XMLHttpRequest.
- Para peticiones POST hay que añadir una cabecera y pasar una cadena de parámetros.
- Las peticiones pueden ser síncronas o asíncronas.
- En las peticiones asíncronas hay una función encargada de procesar la respuesta.
- El código en el cliente se encarga de mostrar los datos recibidos. También modifica la estructura de la página.
- JavaScript realiza cambios en la página utilizando el DOM.
- En las aplicaciones de una sola página, todas las solicitudes al servidor, salvo la primera, se realizan desde JavaScript.

Ejercicios propuestos

1. Utilizando AJAX y PHP escribe una página que muestre un número aleatorio. Cada cinco segundos, la página tiene que solicitar al servidor un nuevo número y mostrarlo.
2. Escribe una página web con un formulario que permita sumar dos números. Utiliza AJAX para enviar el formulario y mostrar el resultado.
3. Escribe un fichero PHP que devuelva en JSON los datos de la tabla de productos de la aplicación de pedidos.

4. A partir del ejercicio anterior, escribe una página que cree una tabla con los productos.
5. Añade un vínculo al ejercicio anterior para recargar la página.
6. Escribe un formulario de *login* para la tabla de Restaurantes del capítulo anterior. Muestra el resultado en una alerta.

ACTIVIDADES DE AUTOEVALUACIÓN

1. El DOM es:
   - ☐ *a)* Un lenguaje de programación en el lado del cliente.
   - ☐ *b)* Un lenguaje de programación en el lado del servidor.
   - ☐ *c)* Una interfaz para manipular documentos desde un lenguaje de programación.

2. El código JavaScript:
   - ☐ *a)* Se ejecuta en el servidor.
   - ☐ *b)* Se ejecuta en el cliente.
   - ☐ *c)* Se ejecuta en ambos.

3. La diferencia entre una petición síncrona y una asíncrona es:
   - ☐ *a)* Que las asíncronas bloquean la ejecución del *script* hasta que se recibe la respuesta y las otras no.
   - ☐ *b)* Que las síncronas bloquean la ejecución del *script* hasta que se recibe la respuesta y las otras no.
   - ☐ *c)* Que las síncronas generan una excepción si no se recibe respuesta y las otras no.

4. Las peticiones AJAX utilizan el objeto:
   - ☐ *a)* XMLHTTPRequest
   - ☐ *b)* HTTPRequest
   - ☐ *c)* DOM

5. El formato de las respuestas a las peticiones AJAX es:
   - ☐ *a)* XML
   - ☐ *b)* JSON
   - ☐ *c)* HTML, XML, JSON o cualquier otro formato.

6. Cuando la respuesta implica modificar la estructura de la página, en general:
   - ☐ *a)* El nuevo HTML se genera en el servidor y se envía al cliente
   - ☐ *b)* El nuevo HTML se genera en el cliente a partir de la respuesta del servidor.
   - ☐ *c)* El nuevo HTML tiene que estar presente ya en la página, pero oculto.

7. Una de las siguientes afirmaciones sobre las aplicaciones web de una sola página es incorrecta:
   - ☐ *a)* Realizan todas las peticiones al servidor, salvo la inicial, mediante JavaScript.

- ☐ *b)* Tienen un aspecto más parecido a las aplicaciones de escritorio.
- ☐ *c)* Se utilizan para aplicaciones pequeñas.

8. ¿Cuál de las siguientes afirmaciones sobre las aplicaciones con AJAX es incorrecta?
   - ☐ *a)* Se separa la lógica de negocio de la de presentación.
   - ☐ *b)* La lógica de presentación se desplaza al cliente.
   - ☐ *c)* El cliente se conecta directamente a la base de datos usando JavaScript.

9. En la función que procesa la respuesta a una petición asíncrona, en general:
   - ☐ *a)* Se comprueba el código de respuesta del servidor.
   - ☐ *b)* Se comprueba el estado de la respuesta del servidor.
   - ☐ *c)* Se comprueban ambos.

10. Si se quiere asociar una función JavaScript a un vínculo:
    - ☐ *a)* La función tiene que devolver TRUE para que el navegador no cargue el vínculo.
    - ☐ *b)* La función tiene que devolver FALSE para que el navegador no cargue el vínculo.
    - ☐ *c)* Al utilizar el atributo onclick el comportamiento por defecto del navegador se anula, da igual lo que devuelva la función.

SOLUCIONES:

1. a b **c**	5. a b **c**	
2. a **b** c	6. a **b** c	9. a b **c**
3. a **b** c	7. a b **c**	10. a **b** c
4. **a** b c	8. a b **c**	

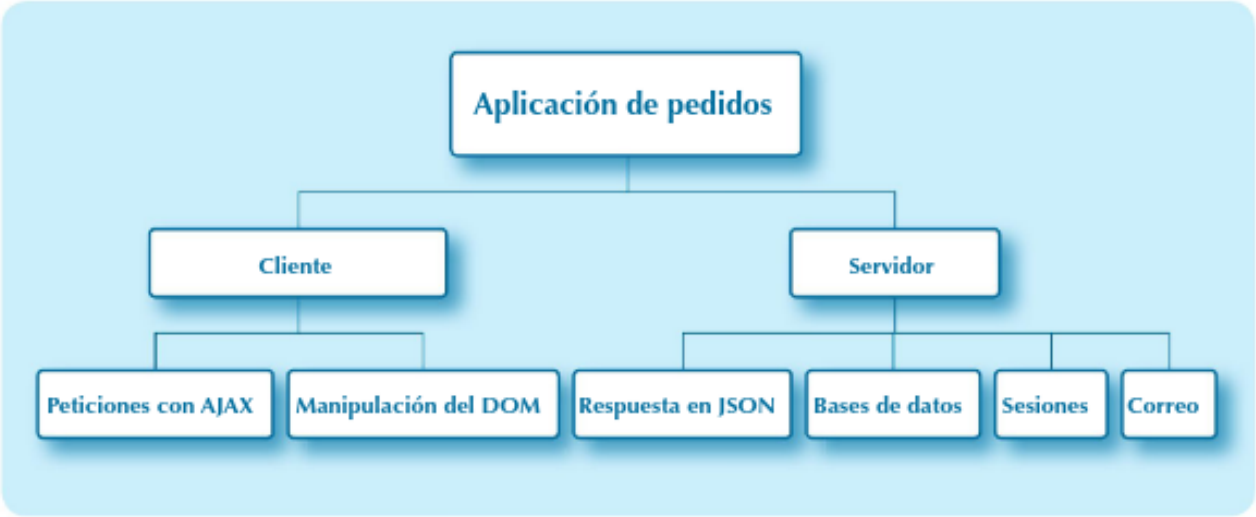
6

Aplicación de pedidos con AJAX

Objetivos

- ✓ Rediseñar la aplicación de pedidos como una aplicación de una sola página.
- ✓ Utilizar AJAX para las comunicaciones con el servidor y la interfaz gráfica.
- ✓ Identificar los cambios necesarios en el lado del servidor.
- ✓ Valorar las mejoras de este enfoque.
- ✓ Analizar las posibilidades de reutilización por separado del código de cliente y servidor.
- ✓ Reutilizar en lo posible el proyecto anterior.

Mapa conceptual



Glosario

Evento. Los eventos se usan, entre otras cosas, para representar las acciones del usuario. Por ejemplo, que pase el ratón por encima de un elemento, o que haga *click*.

iterator_to_array(). Función de PHP para convertir a un *array* cualquier objeto que implemente la interfaz Iterator.

json_encode(). Función de PHP para convertir un *array* en un documento JSON.

JSON.parse(). Función de JavaScript para convertir un documento JSON a un objeto.

onclick. Atributo HTML que permite asociar código al evento *click* (sobre el elemento que tiene el atributo) a una función JavaScript

onsubmit. Atributo para formularios HTML que permite asociar código JavaScript al evento *submit*.

6.1. Diseño de la aplicación

En este capítulo se rediseña la aplicación de pedidos del capítulo 4 como una aplicación de una sola página utilizando AJAX. La nueva aplicación tendrá el mismo aspecto y funcionalidad que la anterior, pero la lógica de presentación se pasa al cliente. Este 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.

RECUERDA

✓ Si no recuerdas bien la aplicación, revisa el apartado 4.3.

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

1. La base de datos. Utiliza la misma que las otras aplicaciones
2. El mapa de pantallas.
3. 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.

Actividad propuesta 6.1

Al utilizar AJAX, manejar la interfaz gráfica es más sencillo. ¿Qué cambios harías en la aplicación original para que fuera más cómoda para el usuario? Modifica el mapa de pantallas si es necesario.

6.2. 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*. Hay que pensar la estructura de este documento.

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 sección en la que se mostrarán 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.

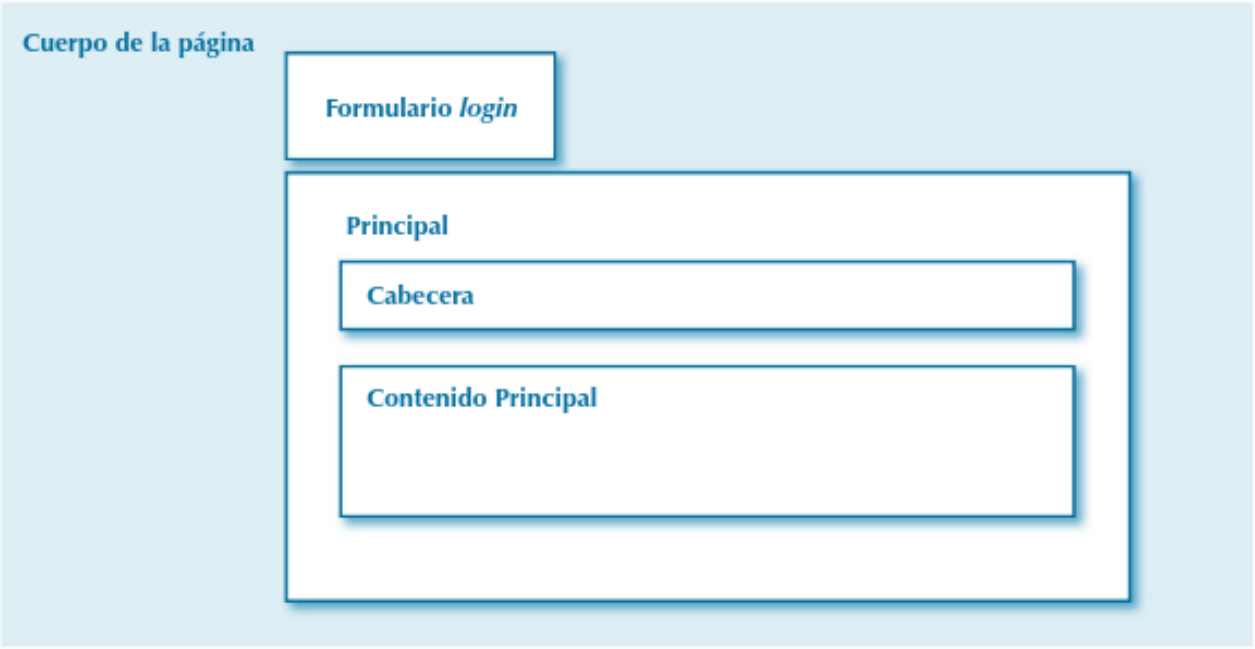


Figura 6.1
Estructura base de la página.

6.3. Cambios en la estructura

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 (cuadro 6.1).

CUADRO 6.1
Funciones más importantes en el cliente

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

[.../...]

CUADRO 6.1 (CONT.)

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 (cuadro 6.2).

CUADRO 6.2
Funciones auxiliares para la interfaz gráfica

Función	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

6.4. En el servidor

La parte del servidor es bastante parecida a la del capítulo 4. 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.

CUADRO 6.3
Ficheros en el servidor

Ruta	Descripción	Parámetros
login_json.php	Valida el formulario de entrada	$_POST['usuario'] $_POST['clave']
logout_json.php	Cierra la sesión	
categorías_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'], el código de la categoría

[.../...]

CUADRO 6.3 (CONT.)

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.html	Página principal y única de la aplicación	
cabecera_json.php	Cabecera con vínculos para ver el carrito, las categorías o cerrar sesión	
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.

Actividad propuesta 6.2

¿Crees que JSON es el mejor formato para devolver los datos de categorías y productos? ¿Sería mejor devolver directamente la lista o la tabla HTML? ¿Y devolver los datos en XML? Analiza las ventajas e inconvenientes de cada opción.

6.5. Implementación

El fichero inicial de la aplicación es **una_sola_pagina.php**.

```
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                <?php require 'cabecera_json.php' ?>
20            </header>
21            <h2 id = "titulo"></h2>
22            <section id = "contenido">
23            </section>
24        </section>
25    </body>
26 </html>
```

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.

Actividad propuesta 6.3

En lugar de utilizar una página con una estructura base, se podría crear toda la página desde JavaScript. ¿Qué ventajas e inconvenientes tiene esta opción?

6.6. Lado del servidor

En esta versión de la aplicación los ficheros del servidor son más sencillos porque no hay que ocuparse de la salida.

6.6.1. *Login*

Del *login* se encarga **login_json.php**. Es parecido al fichero de *login* original, pero solo tiene la parte de procesamiento. El formulario está en página principal (y única). Además, no redirige. Si hay error, devuelve la cadena FALSE; y si no crea las variables de sesión, devuelve la cadena TRUE.

```php
<?php
require_once '..\cap4\bd.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
    if($usu===FALSE){
        echo "FALSE";
    }else{
        session_start();
        // $usu tiene campos correo y codRes, correo
        $_SESSION['usuario'] = $usu;
        $_SESSION['carrito'] = [];
        echo "TRUE";
    }
}
```

6.6.2. Control de acceso

Hay cambios en el control de acceso, que antes se basaba en una redirección. Ahora, la función `comprobar_sesion()` devuelve TRUE o FALSE según haya sesión abierta o no.

```php
<?php
    function comprobar_sesion(){
        session_start();
        if(!isset($_SESSION['usuario'])){
        return FALSE;
        }else return TRUE;
    }
```

Los demás ficheros hacen esta comprobación:

```php
require_once '..\cap4\bd.php';
if(!comprobar_sesion()) return;
```

6.6.3. La cabecera

La cabecera, aunque corta, cambia completamente. Los vínculos se sustituyen por llamadas a las funciones correspondientes. El nombre del usuario no aparece, en su lugar hay un elemento **span** con id = "nombre". Esto es así porque la cabecera se sirve antes de que se haya hecho *login*, así que no puede saberse el usuario. Si se hace *login* correctamente, se introduce el nombre desde JavaScript.

```html
<header>
    <span id="cab_usuario"></span>
    <a href="#" onclick="loadCategorias();">Home</a>
    <a href="#" onclick="cargarCarrito();">Carrito</a>
    <a href="#" onclick="cerrarSesionUnaPagina();">Cerrar sesión</a>
</header>
<hr>
```

6.6.4. Las categorías

El fichero **categorias_json.php** devuelve los datos de la tabla de categorías en formato JSON. Utiliza la función `cargar_categorias()`, la misma del capítulo 4. La función `json_enconde()` transforma un *array* a formato JSON. Como `cargar_categorias()` devuelve un cursor, se utiliza `iterator_to_array()` para convertirlo.

```
<?php
require_once 'sesiones_json.php';
require_once '..\cap4\bd.php';
if(!comprobar_sesion()) return;
$categorias = cargar_categorias();
$cat_json = json_encode(iterator_to_array($categorias);
echo $cat_json;
```

Si se accede directamente desde el navegador, se obtendrá:

[{"codCat":"3","0":"3","nombre":"Bebidas con","1":"Bebidas con"},
{"codCat":"2","0":"2","nombre":"Bedidas sin","1":"Bedidas sin"},
{"codCat":"1","0":"1","nombre":"Comida","1":"Comida"}]

6.6.5. 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.

```
<?php
require_once '..\cap4\bd.php';
/*comprueba que el usuario haya abierto sesión o devuelve*/
require 'sesiones_json.php';
if(!comprobar_sesion()) return;
$productos_array = [];
$productos = cargar_productos_categoria($_GET['categoria']);
$cat_json = json_encode(iterator_to_array($productos);
echo $cat_json;
```

6.6.6. 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 a **carrito.php** del capítulo 4.

```
<?php
require 'sesiones_json.php';
require_once '..\cap4\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);
```

6.6.7. 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.

```
<?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;
}
<?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]);
    }
}
```

6.6.8. Cerrar la sesión

Lo mismo ocurre con **logut_json.php**.

```
<?php
if(!comprobar_sesion()) return;
$_SESSION = array();
session_destroy(); // eliminar la sesion
setcookie(session_name(), 123, time() - 1000); // eliminar la cookie
```

6.6.9. 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 del capítulo 4.

```
<?php
/*comprueba que el usuario haya abierto sesión o redirige*/
require '..\cap4\correo.php';
require_once '..\cap4\bd.php';
/*comprueba que el usuario haya abierto sesión o devuelve*/
require 'sesiones_json.php';
if(!comprobar_sesion()) return;
$resul = insertar_pedido($_SESSION['carrito'], $_SESSION['usuario']['co-
dRes']);
if($resul === FALSE){
    echo "FALSE";
}else{
    $correo = $_SESSION['usuario']['correo'];
    $conf = enviar_correos($_SESSION['carrito'], $resul, $correo);
    echo "TRUE";
    //vaciar carrito
    $_SESSION['carrito'] = [];
}
```

6.6.10. Funciones auxiliares

Los ficheros **bd.php** y **correo.php** son los del capítulo 4. Se reutilizan sin cambios.

6.7. El 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.

6.7.1. *Login*

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

- Muestra la sección principal (línea 8).
- Oculta la sección del formulario (línea 9).
- Introduce el nombre del usuario en la parte apropiada de la cabecera (línea 11).

Si no son correctos, muestra un mensaje de error con una alerta (línea 6).

```
1  function login(formu){
2     var xhttp = new XMLHttpRequest();
3     xhttp.onreadystatechange = function() {
4        if (this.readyState == 4 && this.status == 200) {
5           if(this.responseText==="FALSE"){
6              alert("Revise usuario y contraseña");
7           }else{
8              document.getElementById("principal").style.display= "block";
9              document.getElementById("login").style.display= "none";
10             /*ponemos el usuario devuelto en el hueco correspondiente*/
11             document.getElementById("cab_usuario").innerHTML = "Usua-
               rio: " + usuario;
12             cargarCategorias();
13          }
14       }
15    }
16    var usuario = document.getElementById("usuario").value;
17    var clave = document.getElementById("clave").value;
18    var params = "usuario="+usuario+"&clave="+clave;
19    xhttp.open("POST", "login_json.php", true);
20    // envío con POST requiere cabecera y cadena de parámetros
21    xhttp.setRequestHeader("Content-type","application/
         x-www-form-urlencoded");
22    xhttp.send(params);
23    return false;
24 }
```

La petición se realiza en la línea 22. Como se trata de enviar un formulario se utiliza el método POST. Por tanto, hay que añadir la cabecera específica (línea 21) y enviar una cadena con los parámetros.

6.7.2. Las categorías

La función `cargarCategorias()` solicita al servidor los datos de las categorías (**categorias_json.php**) y crea la lista de vínculos. Cuando recibe la respuesta del servidor:

- La convierte en un objeto JavaScript, que será un *array* (línea 5). 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 6).
- Por cada elemento del *array* crea un vínculo usando el nombre y el código (línea 9).
- Ese elemento se introduce en un elemento `li` (línea 10).
- El elemento `li` se introduce en la lista (línea 11).
- Elimina el contenido de la sección "contenido" y luego introduce la lista en ella (líneas 13-17).

```
1  function cargarCategorias() {
2     var xhttp = new XMLHttpRequest();
3     xhttp.onreadystatechange = function() {
4     if (this.readyState == 4 && this.status == 200) {
5        var cats = JSON.parse(this.responseText);
6        var lista = document.createElement("ul");
7        for(var i = 0; i < cats.length; i++){
8           var elem = document.createElement("li");
9           vinculo = crearVinculoCategorias(cats[i].nombre, cats[i].codCat);
10          elem.appendChild(vinculo);
11          lista.appendChild(elem);
12       }
13       var contenido = document.getElementById("contenido");
14       contenido.innerHTML = "";
15       var titulo = document.getElementById("titulo");
16       titulo.innerHTML ="Categorías";
17       contenido.appendChild(lista);
18    }
19 };
20 xhttp.open("GET", "categorias_json.php", true);
21 xhttp.send();
22 return false;
```

Para crear los vínculos utiliza una función auxiliar.

```
function crearVinculoCategorias(nom, cod){
   var vinculo = document.createElement("a");
   var ruta = "productos_json.php?categoria=" +cod;;
   vinculo.href = ruta;
   vinculo.innerHTML = nom;
   vinculo.onclick = function(){return cargarProductos(this);}
   return vinculo;
}
```

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

Actividad propuesta 6.4

Investiga la función on() de jQuery(), que permite asociar eventos a funciones.

6.7.3. Los productos

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

- Pide los datos al servidor (**productos_json.php**).
- Crea la tabla de productos.

- Elimina el contenido de la sección "contenido" y luego introduce la tabla en ella.

```
function cargarProductos(destino){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var prod = document.getElementById("contenido");
            var titulo = document.getElementById("titulo");
            titulo.innerHTML ="Productos";
            try{
                var filas = JSON.parse(this.responseText);
                var tabla = crearTablaProductos(filas);
                prod.innerHTML = "";
                prod.appendChild(tabla);
            }catch(e){
                var mensaje = document.createElement("p");
                mensaje.innerHTML = "Categoría sin productos";
                prod.innerHTML = "";
                prod.appendChild(mensaje);
            }
        }
    };
    xhttp.open("GET", destino, true);
    xhttp.send();
    return false;
}
```

Para crear la tabla utiliza la función auxiliar `crearTablaProductos(filas)`. Esta función recibe el *array* de productos y devuelve un elemento `<table>` con una fila por producto.

```
function crearTablaProductos(productos){
    var tabla = document.createElement("table");
    var cabecera = crear_fila(["Código", "Nombre", "Descripción",
        "Stock", "Comprar"], "th");
    tabla.appendChild(cabecera);
    for(var i = 0; i < productos.length; i++){
        /*formulario*/
        formu = crearFormulario( "Añadir",
            productos[i]['CodProd'],anadirProductos);
        fila = crear_fila([productos[i]['CodProd'],
            productos[i]['Nombre'],
            productos[i]['Descripcion'],
            productos[i]['Stock']], "td");
        celda_form = document.createElement("td");
        celda_form.appendChild(formu);
        fila.appendChild(celda_form);
        tabla.appendChild(fila);
    }
```

```
    return tabla;
}
```

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

1. El texto del botón del formulario.
2. El código del producto.
3. 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 envío del formulario, se llamará a la función.

```
1  function crearFormulario(texto, cod, funcion){
2      var formu = document.createElement("form");
3      var unidades = document.createElement("input");
4      unidades.value = 1;
5      unidades.name = "unidades";
6      var codigo = document.createElement("input");
7      codigo.value = cod;
8      codigo.type = "hidden";
9      codigo.name = "cod";
10     var bsubmit = document.createElement("input");
11     bsubmit.type = "submit";
12     bsubmit.value = texto;
13     formu.onsubmit = function(){return funcion(this);}
14     formu.appendChild(unidades);
15     formu.appendChild(codigo);
16     formu.appendChild(bsubmit);
17     return formu;
18 }
```

6.7.4. El carrito

Esta función 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()`.

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

```
function crearTablaCarrito(productos){
    var tabla = document.createElement("table");
    var cabecera = crear_fila(["Código", "Nombre", "Descripción",
        "Unidades", "Eliminar"], "th");
    tabla.appendChild(cabecera);
    for(var i = 0; i < productos.length; i++){
```

```
            /*formulario*/
            formu = crearFormulario("Eliminar",productos[i]['CodProd'],elimi-
            narProductos);
            fila = crear_fila([productos[i]['CodProd'],
                productos[i]['Nombre'],
                productos[i]['Descripcion'],
                productos[i]['unidades']], "td"
            );
            celda_form = document.createElement("td");
            celda_form.appendChild(formu);
            fila.appendChild(celda_form);
            tabla.appendChild(fila);
        }
        return tabla;
    }
```

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

Actividad propuesta 6.5

Las funciones para crear la tabla de productos y la del carrito son muy parecidas. ¿Crees que sería buena idea unificarlas? ¿Qué cambios habría que hacer?

6.7.5. Añadir y eliminar

La función de añadir productos 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**.

```
function anadirProductos(formulario){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            alert("Producto añadido con éxito");
            cargarCarrito();
        }
    };
    var params = "cod="+formulario.elements['cod'].value+
    "&unidades="+formulario.elements['unidades'].value;
    xhttp.open("POST", "anadir_json.php", true);
```

```
xhttp.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
xhttp.send(params);
return false;
}
function eliminarProductos(formulario){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        alert("Producto eliminado con éxito");
        cargarCarrito();
    }
    };
    var params = "cod="+formulario.elements['cod'].value+
        "&unidades="+formulario.elements['unidades'].value;
    xhttp.open("POST", "eliminar_json.php", true);
    xhttp.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");
    xhttp.send(params);
    return false;
}
```

6.7.6. Realizar el pedido

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

```
function procesarPedido(){
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        var contenido = document.getElementById("contenido");
        contenido.innerHTML = "";
        var titulo = document.getElementById("titulo");
        titulo.innerHTML ="Estado del pedido";
        if(this.responseText=="TRUE"){
            contenido.innerHTML = "Pedido realizado";
        }else{
            contenido.innerHTML = "Error al procesar el pedido";
        }
    }};
    xhttp.open("GET", "procesar_pedido_json.php", true);
    xhttp.send();
    return false;
}
```

Resumen

- La aplicación tiene el mismo aspecto y funcionalidad que la del capítulo 4, pero está elaborada utilizando AJAX.
- Después de la petición inicial, todas las peticiones al servidor se realizan a través de JavaScript.
- La página inicial contiene el formulario de *login* y una sección principal para el resto de la aplicación. Incluye los ficheros JavaScript.
- Al hacer *login* con éxito, se oculta el formulario y se muestra la sección principal.
- La lógica de presentación se desplaza al cliente. El servidor se ocupa del control de sesiones, base de datos y envío de correo.
- El servidor devuelve los datos en JSON en lugar de utilizar HTML.
- El código en el cliente se encarga de mostrar los datos recibidos. También modifica la estructura de la página.
- Hay que asociar los eventos de envío de formularios con las funciones de JavaScript correspondientes.
- Lo mismo ocurre con los vínculos de la lista de categorías, la cabecera y realizar pedido.
- Se reutilizan sin cambios las funciones de bases de datos y correo del capítulo anterior.

Ejercicios propuestos

Modifica la aplicación para que:

1. No redirija al carrito de la compra al añadir o eliminar productos.
2. No muestre los productos sin *stock*.
3. Pida confirmación al usuario antes de realizar el pedido.
4. El carrito de la compra esté siempre visible.
5. Muestre un vínculo "Zona admin" solo a los usuarios con rol 1.
6. Para reflexionar: ventajas e inconvenientes de manejar el carrito de la compra desde JavaScript en lugar de desde el servidor.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Al hacer *login* con éxito, se pasa a la pantalla principal:
   - ☐ *a)* Solicitando la página al servidor por JavaScript.
   - ☐ *b)* Haciendo visible la sección correspondiente.
   - ☐ *c)* El servidor envía una redirección.

3. El correo electrónico:
   - ☐ *a)* Se envía desde el cliente.
   - ☐ *b)* Se envía desde el servidor.
   - ☐ *c)* El correo se genera en el cliente, que lo pasa al servidor para que lo envíe.

4. Los datos de conexión a la base de datos se introducen:
   - ☐ *a)* En un fichero XML.
   - ☐ *b)* En un fichero de texto plano.
   - ☐ *c)* Directamente en las funciones que usan la base de datos.

5. Los datos del servidor de correo electrónico se introducen:
   - ☐ *a)* En un fichero XML.
   - ☐ *b)* En un fichero de texto plano.
   - ☐ *c)* Directamente en la función de correo.

6. La cabecera de la página:
   - ☐ *a)* Se inserta en el cliente.
   - ☐ *b)* Se inserta en el servidor.
   - ☐ *c)* Se inserta en el servidor, pero algunos campos se completan en el cliente.

7. La conexión con la base de datos se realiza:
   - ☐ *a)* Desde el cliente.
   - ☐ *b)* Desde el servidor.
   - ☐ *c)* Desde el cliente para el *login*, desde el servidor para los pedidos.

8. ¿Cuál de estos elementos ha cambiado respecto a la aplicación inicial?
   - ☐ *a)* Diseño de la base de datos.
   - ☐ *b)* Carrito de la compra.
   - ☐ *c)* Formato de salida de los *scripts* del servidor.

9. Al cerrar sesión:
   - ☐ *a)* El servidor redirige a la página de *login*.
   - ☐ *b)* Se oculta la sección principal y se vuelve a mostrar la de *login*.
   - ☐ *c)* Se solicita mediante JavaScript la página de despedida.

10. Las sesiones se manejan:
    - ☐ *a)* Desde el servidor.
    - ☐ *b)* Desde el cliente.
    - ☐ *c)* La variable para el carrito en el servidor, la que almacena los datos del usuario en el cliente.

SOLUCIONES:

1.	a **b** c	**5.**	a b **c**		
2.	a **b** c	**6.**	a b **c**	**9.**	a **b** c
3.	a **b** c	**7.**	a **b** c	**10.**	**a** b c
4.	**a** b c	**8.**	a b **c**		