Desarrollo web en entorno cliente

Unidad 7: Utilización de mecanismos de comunicación asíncrona

ÍNDICE

INTRODUCCIÓN

Esta unidad trata sobre los **mecanismos de comunicación asíncrona**, los cuales nos permiten que el **emisor** y **receptor** no coincidan necesariamente en el tiempo.

Veremos exactamente de qué se trata la **comunicación asíncrona** y qué aspectos la diferencian de la **comunicación síncrona**. Además, veremos cómo utilizarla, y las **ventajas** e **inconvenientes** que representa.

Más adelante aplicaremos estos conocimientos a la **programación en JavaScript**, viendo cómo podemos pedir datos de forma asíncrona a un servidor web.

Finalmente conoceremos los principales **mecanismos de almacenamiento** de información en el lado del cliente, y cuáles son sus diferencias.

En esta unidad nos alejamos ligeramente de la programación en JavaScript, para centrarnos en otros conceptos, más cercanos a módulos de **programación en servidor** o **base de datos**.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Buscando películas

Ya hemos practicado con la programación asíncrona y tenemos los conocimientos necesarios para entrar en un proyecto más exigente. Desde del departamento de diseño han creado una página web para buscar películas, pero ahora falta dotarla de funcionalidad.

Para ello nos valdremos de dos herramientas que encontramos en internet:

- TMDB API: API de la comunidad The Movie Database (TMDB). Es una API libre que sólo requiere registrarse para conseguir la API KEY.
- IMDB: la mayor base de datos sobre películas que hay en internet.

Se trata de:

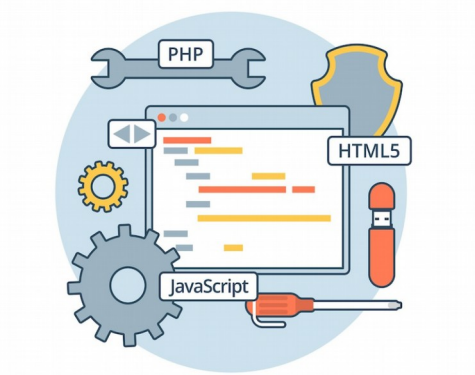
- Usar la API de TMDB para realizar la búsqueda de películas e IMDB para ver la ficha de la película en cuestión.

- La aplicación constará de un sencillo campo de búsqueda. Éste, a medida que vayamos escribiendo el nombre de una película nos debe ir mostrando sugerencias.
- Si clicamos en una de las sugerencias debe llevarnos directamente a la ficha de la película de IMDB.
- Si hacemos clic en el botón de buscar nos debe mostrar un layout con las películas que tienen un nombre parecido al buscado.
- Este layout debe tener, como mínimo, el póster de la película, el nombre, los géneros, la fecha de estreno y los actores principales (al menos 3 actores).
- Posiblemente cuando se consulta la API ésta nos devolverá muchos resultados, haciendo que se tarde un rato en obtenerlos todos. En ese momento se debe buscar algún mecanismo para indicarle al usuario que los datos se están cargando.

Usa el archivo Ejemplo.html adjunto como plantilla inicial. Para descargar la plantilla pulsa aquí: *https://imagenes.contenidosteleformacion.es/ampliar/dewe2021/unidad8/ejemplo.html*

1. MECANISMOS DE COMUNICACIÓN ASÍNCRONA. UTILIDAD, VENTAJAS E INCONVENIENTES

Antes de centrarnos en los mecanismos de **comunicación asíncrona** y ver en detalle su utilidad, ventajas e inconvenientes, deberíamos de conocer qué es la comunicación asíncrona, y, sobre todo, qué la diferencia de la **comunicación síncrona**.



➔ **Comunicación síncrona**:

DESTACADO

Podemos definir la comunicación síncrona como aquella que se da cuando el receptor recibe el mensaje en el mismo momento en que lo emite un emisor, por ejemplo, en una conversación en persona.

➔ **Comunicación asíncrona**:

DESTACADO

Por el contrario, en una comunicación asíncrona, el emisor y el receptor no coinciden en el mismo momento, por lo que la información puede darse de manera intermitente y no como un flujo constante.

Este tipo de comunicación se da a menudo en **informática**, por ejemplo, en el correo electrónico o en la mensajería.

Durante esta unidad usaremos **AJAX** (Asynchronous JavaScript And XML) cuando trabajemos con comunicación asíncrona sobre JavaScript. A continuación, veremos algunos **aspectos relevantes** sobre este:

➔ **Concepto**: **AJAX** no es realmente un lenguaje de programación, sino que es una **técnica** para **acceder a servidores web** desde páginas web.

Ya conocemos JavaScript, así que veamos la otra parte de **AJAX, XML o eXtensible Markup Language**. Es un lenguaje de marcas, que se diseñó para **almacenar y transportar datos**. Éstos están organizados de manera jerárquica, mediante el uso de etiquetas. Al igual que AJAX, XML tampoco es un lenguaje de programación.

➔ **Usos**: así pues, AJAX puede usar XML para transportar datos, pero también puede hacerlo mediante texto plano o JSON.

Además, permite a las páginas web ser **actualizadas** de manera **asíncrona**, intercambiando información con un servidor web de manera transparente. Esto hace que pueda actualizar partes de una página web, sin necesidad de refrescar toda la página.

➔ **Es una combinación de…**

- ✔ **Un objeto XMLHTTPRequest** integrado en el navegador (para solicitar datos de un servidor web).
- ✔ **JavaScript y HTML DOM** (para mostrar o utilizar los datos).

Vídeo: AJAX

Visualiza el siguiente vídeo sobre AJAX.
https://www.youtube.com/embed/z2aipCAq2n8

2. OBJETOS, PROPIEDADES Y MÉTODOS RELACIONADOS

Como hemos dicho, una de las cosas más importantes en el uso de AJAX es **XMLHTTPRequest**. Éste es un **objeto** disponible en los lenguajes de script de cliente, como es JavaScript.

Sirve para realizar **peticiones HTTP o HTTPS** a un servidor web, y cargar las respuestas en la página del cliente.

*Estos **datos** recibidos pueden ser usados para **modificar el DOM**, sin necesidad de recargar la página por completo, cosa que veremos en un capítulo posterior.*

Así pues, para programar con AJAX, necesitamos crear un objeto **XMLHTTPRequest**, que permitirá realizar las peticiones en segundo plano.

Si queremos evitar problemas de dependencias del navegador, podemos usar una función **cross-browser**, como la siguiente:

Código: función cross - browser

```
if (window.XMLHttpRequest) {
    // Código para navegadores modernos
    xmlhttp = new XMLHttpRequest();
} else {
    // Código para antiguos navegadores IE
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Una vez creado este objeto podríamos trabajar con él, por lo que necesitaríamos conocer sus **métodos** y **propiedades**:

➔ Aquí van algunos de los principales **métodos** de **XMLHTTPRequest**:

- ✔ `Abort()`: cancela la **solicitud** actual.
- ✔ `Open()`: crea la solicitud, indicando parámetros como el método **(GET/POST)**, la url o si se trata de una petición síncrona o asíncrona.
- ✔ `Send()`: envía la solicitud al **servidor**.

➔ En cuanto a las **propiedades** principales, nos encontramos con:

- ✔ **ReadyState**: almacena el estado actual de la solicitud **XMLHTTPRequest**. Tiene varios posibles valores, que indican su estado.
- ✔ **Onreadystatechange**: define una función que se ejecutará cuando se cambie la propiedad **readyState**.
- ✔ **Status**: estado numérico devuelta en la petición al servidor. Por ejemplo, **404** para "No encontrado".
- ✔ **ResponseText**: contiene la respuesta, en modo **texto**.
- ✔ **ResponseXML**: contiene la respuesta, en modo **XML**.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué objeto se utiliza para realizar peticiones HTTP o HTTPS a un servidor web en AJAX?

a. XMLHTTPRequest.

b. ActiveXObject.

c. xmlhttp.

Lee el enunciado e indica la opción correcta:

¿Cuál es el método utilizado para cancelar la solicitud actual en XMLHTTPRequest?

a. abort().

b. open().

c. send().

Lee el enunciado e indica la opción correcta:

¿Cuál de las siguientes propiedades almacena el estado actual de la solicitud XMLHTTPRequest?

a. readyState.

b. onreadystatechange.

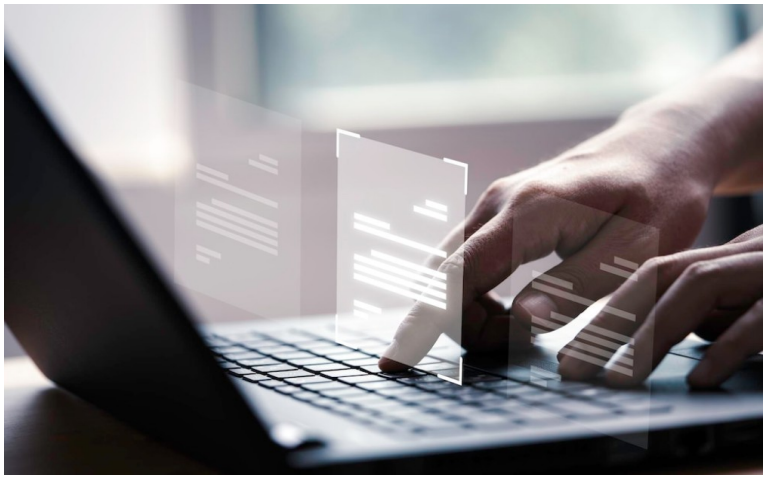
c. status.

3. RECUPERACIÓN REMOTA DE INFORMACIÓN

Una vez hemos creado el objeto **XMLHTTPRequest** es el momento de utilizarlo para la recepción de datos de forma remota.

*Para ello, una de las primeras cosas que podemos hacer, es asignar una función a método **onreadystatechange()** visto anteriormente.*

Esta función será invocada cuando cambie el estado de la petición. Aquí gestionaremos qué hacer con la **respuesta** o si se produce un error.



Una vez asignada la función, se pueden usar los métodos **open**() y **send**() para crear la petición y enviarla, respectivamente.

Veámoslo en el siguiente ejemplo, dónde **url** sería la dirección de la cual queremos obtener los datos:

Ejemplo: obtener datos de una dirección

```
httpRequest.onreadystatechange = procesarCambioEstado;

httpRequest.open('GET', url, true);
httpRequest.send(null);

function procesarCambioEstado() {
    console.log("Ha cambiado el estado de la petición");
}
```

Como se aprecia, una vez se recibe la respuesta, se llama a la función **procesarCambioEstado()**, que, en este caso, sólo muestra un mensaje por la consola. Es aquí donde debemos implementar toda la lógica para procesar la respuesta.

Lo primero que deberíamos hacer cuando el estado cambia, es comprobar si se ha completado o no, para lo cual nos fijaremos en el estado de la propiedad readyState. Hay 5 posibilidades:

- **UNSENT (0)**: aún no se ha abierto la petición con open().
- **OPENED (1)**: se ha abierto la petición, pero aún no se ha enviado.
- **HEADERS_RECEIVED (2)**: se ha enviado la petición.
- **LOADING (3)**: se está descargando la respuesta. La propiedad responseText contiene contenido parcial.
- **DONE (4)**: se ha completado la operación.

Así podríamos hacer:

```
function procesarCambioEstado () {
   if (httpRequest.readyState === XMLHttpRequest.DONE) {
       console.log("Respuesta recibida. Estado:",
httpRequest.readyState);
   } else {
      console.log("Respuesta sin recibir. Estado:",
httpRequest.readyState);
   }
}
```

Si quisiéramos completar la parte en que hemos recibido la respuesta, lo ideal sería crear una función independiente que la gestione, a través de la propiedad **responseXML**.

Ejemplo:

```
procesarRespuesta(httpRequest.responseXML);
```

A partir de aquí, el contenido de la función delimitaría cómo vamos a procesar la respuesta recibida.

Vídeo: XML

Visualiza el siguiente vídeo en el que hablamos sobre XML.

4. PROGRAMACIÓN DE APLICACIONES CON COMUNICACIÓN ASÍNCRONA. NOTIFICACIONES

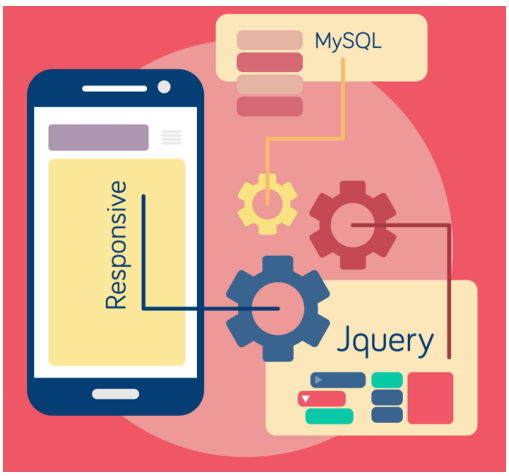
Como ya vimos en unidades anteriores, actualmente el uso de **librerías** de terceros agiliza mucho el proceso de programación por parte del desarrollador, evitando repetición de código y simplificando éste.

DESTACADO

*Una de las librerías más utilizadas es **jQuery**, que además de servir para la gestión de eventos y manipulación del DOM, podemos usar para la gestión de peticiones con **AJAX**.*

Una de las diferencias es que no es necesario realizar ninguna comprobación previa para discriminar entre las versiones antiguas del navegador Internet **Explorer** y los navegadores actuales, la librería se encarga de hacerlo.

Cabe destacar que para poder utilizar estas funcionalidades se debe cargar la librería **jQuery**.



POR EJEMPLO

Ejemplo:

```
<script src="js/librerias/jquery-3.5.1.js"></script>
```

A continuación, veremos la **sintaxis de la librería jQuery** y explicaremos **cómo funciona su código**:

- **Sintaxis**: para crear y enviar peticiones, pues, ahora utilizaremos jQuery, con su **sintaxis**, de la siguiente manera:

```
<script>
   $.ajax('url', {
      success: procesarRespuesta,
      error: procesarError
   });

   function procesarRespuesta (datos, statusText, jqXHR) {
      console.log(datos, statusText);
   }

   function procesarError (jqXHR, statusText, error) {
      console.log(error, statusText);
   }
</script>
```

➔ **Código**: el código es muy simple. Se llama al método **ajax()** del objeto jQuery pasando como parámetros:

- ✔ La **URL** donde se enviará la petición.
- ✔ Un objeto de JavaScript con una propiedad llamada success, a la que se le ha asignado la función **procesarRespuesta** (), y otra propiedad llamada **error**, a la que se le ha asignado la función **procesarError**().

El **objeto jqXHR** es devuelto por el método ajax() y permite realizar o encadenar otras acciones a la invocación mediante una interfaz llamada objeto diferido (deferred object).

Por esta razón, se pasa también como argumento las funciones success y error, de modo que se puede obtener más información o realizar otras operaciones sobre la petición.

5. MODIFICACIÓN DINÁMICA DEL DOCUMENTO UTILIZANDO COMUNICACIÓN ASÍNCRONA

Cuando realizamos una petición al servidor, mediante **AJAX**, se espera que, si no especificamos lo contrario, el formato de la respuesta sea un **XML** válido.

➔ **Crear petición**.

De otra manera deberíamos especificar el formato de datos esperado mediante la propiedad **dataType**:

```
$.ajax({
   url: 'url',
   dataType: 'text',
   success: procesarRespuesta,
   error: procesarError
});
```

➔ **Formatos de repuesta**.

En este caso estaríamos indicando que la respuesta indicada es en formato de **texto plano**. A parte de éste, existen otros posibles formatos de respuesta, que veremos más en profundidad en el capítulo siguiente. Estos son:

- ✔ **Xml** (tipo por defecto): devuelve un documento XML que puede ser procesado por jQuery.
- ✔ **Html**: devuelve el código HTML como texto plano.
- ✔ **Script**: evalúa la respuesta como JavaScript.
- ✔ **Json**: evalúa la respuesta como JSON y devuelve un objeto de JavaScript.
- ✔ **Jsonp**: utilizado para obtener los datos de otro dominio utilizando la técnica JSONP, devuelve un objeto de JavaScript con los datos.

➔ Modificar el código.

Una vez hemos creado la **petición**, y nos hemos asegurado de que el formato de la respuesta es el esperado, podemos utilizar ésta para modificar el documento que estamos utilizando.

Supongamos un **ejemplo** donde recibimos una respuesta en forma de países, que ha de ser añadida, de forma dinámica a nuestra página:

```
<html>
   <h1>Paises</h1>
   <ul>
      <li>España</li>
      <li>Francia</li>
      <li>Italia</li>
      <li>Portugal</li>
   </ul>
</html>
```

Para hacerlo, sólo tendríamos que especificar el tipo como html, y añadirlo utilizando el método **append**() del objeto **jQuery**.

Al final del apartado podrás encontrar el código completo.

Código: modificación del documento

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>
```

```
<body>
   <script>
      $.ajax({
      url: paises.html',
      dataType: 'html',
      success: procesarRespuesta,
      error: procesarError
   });

   function procesarRespuesta (datos, statusText, jqXHR) {
      $ datos = $( datos);
      $('body').append($datos);

   }

   function procesarError (jqXHR, statusText, error) {
      console.log(error, statusText);
   }
   </script>
</body>
</html>
```

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cómo se llama el objeto principal de AJAX?

a. XMLRequest.

b. XMLHTTPRequest.

c. HTTPRequest.

Lee el enunciado e indica la opción correcta:

¿Cómo se llama la librería para gestionar peticiones AJAX de forma sencilla?

a. Moment.js.

b. Chart.js.

c. jQuery.

Lee el enunciado e indica la opción correcta:

¿En jQuery cómo especificamos el tipo de datos de la petición?

a. dataType.

b. JSON.

c. XML.

6. FORMATOS PARA EL ENVÍO Y RECEPCIÓN DE INFORMACIÓN. XML Y JSON

Dentro de los formatos para envío y recepción de datos a través de internet, podemos destacar a dos, que son los más utilizados a día de hoy. Hablamos de **XML** y **JSON**.

En este capítulo vamos a hablar un poco más en profundidad de estos dos formatos de datos, y cuáles son sus principales similitudes y diferencias.

Cómo hemos dicho, ambos sirven para el envío de datos de forma **remota**, y la similitud más evidente es que ambos utilizan una estructuración **jerárquica** de los datos. Se podría decir que los datos están dispuestos en forma de **árbol**.

Veamos el detalle de cada uno de los **formatos**:

- **XML**.

  Son las siglas de **eXtensible Markup Language Éstandar** desarrollado por la W3C y creado a partir del lenguaje **SGML**.

  Es un formato de datos basado en texto, como hemos dicho de forma **jerárquica**, mediante el uso de etiquetas.

  XML es similar a **HTML**, pero más **riguroso**, ya que sirve para representar información de forma fidedigna.

  Tiene una etiqueta **raíz**, y a partir de ahí, con el mismo formato van apareciendo los **hijos** y **sucesivos**. Cada etiqueta, como sucede con HTML debe tener su pareja de cierre.

  Un ejemplo podría ser:

```
<países>
   <país>
      <nombre> España </nombre>
      <capital> Madrid </capital>
   </país>
   <país>
      <nombre> Italia </nombre>
      <capital> Roma </capital>
   </país>
</países>
```

➔ **JSON**.

Siglas de **JavaScript Object Notation**. Se trata de un formato, igualmente **jerárquico**, pero más legible que el XML, ya que prescinde de alguna manera de etiquetas de inicio y fin.

Este estándar está mantenido por **ECMAInternational**, responsable también de JavaScript, por lo que su uso va muy ligado.

En lugar de etiquetas, aquí tenemos parejas de clave-valor. El mismo ejemplo de antes, en formato JSON sería:

```
{
   "pais": {
      "nombre": "España",
      "capital": "Madrid"
   },
   "pais": {
      "nombre": "Italia",
      "capital": "Roma"
   },
```

Vídeo: JSON

Visualiza este vídeo en el que hablamos sobre JSON.

Actividad de aprendizaje 1: programación asíncrona

Nuestro periodo en la empresa está llegando a su fin, pero antes nos proponen una serie de retos relacionados con la programación asíncrona.

El objetivo de este primer mini reto es poder cargar datos desde un archivo de texto en nuestra página html.

Para ello es necesario:

- Crear un archivo .txt con el contenido que desees.
- Crear un archivo .html con un botón, de manera que cuando lo pulsemos se cargue el contenido del fichero de texto en un párrafo.

Entrega los archivos en el espacio habilitado para ello.

7. LIBRERÍAS DE ACTUALIZACIÓN DINÁMICA

En algunos casos necesitamos recoger datos de forma continua, para poder actualizar nuestra página web de forma **dinámica**. En ocasiones se usan **APIs** o **datos abiertos**.

DESTACADO

Los datos abiertos son conjuntos de datos puestos a disposición del público y que pueden utilizarse sin restricciones. Habitualmente estos datos pueden ser consultados en el mismo sitio web, descargados o consumidos por otros servicios.

Esta última opción permite el desarrollo de aplicaciones que utilicen **datos de terceros**; por ejemplo, para mostrar los próximos eventos en una ciudad o el estado del tráfico.

A continuación, veremos algunos **aspectos relevantes** sobre los **conjuntos de datos**:

- **Conjunto de dato abierto**.

  Un conjunto de datos, para que se consideren abiertos, deben poder ser **reutilizables**, **adaptables** y **distribuibles** por los usuarios.

- **Acceder a los datos**.

Debe tenerse en cuenta que para poder acceder a estos datos los servicios web que los exponen deben admitir la utilización de **JSONP** o de **CORS**, ya que de lo contrario serán bloqueados por la política del mismo origen de los navegadores.

➔ **Comprobar si admite JSONP**.

Para comprobar si una fuente de datos admite **JSONP** se debe consultar la documentación de la API del servicio web que se desea utilizar.

➔ **Comprobar si admite CORS**.

Comprobar si un sitio admite la transmisión de datos mediante el mecanismo **CORS** básico (sin ningún requisito) es mucho más simple: basta con consultar en las herramientas de desarrollador la cabecera de la respuesta al acceder directamente a la dirección.

➔ **Access-Control-Allow-Origin="*"**.

Si la cabecera incluye el parámetro **Access-Control-Allow-Origin="*"** (el asterisco significa que acepta cualquier origen), querrá decir que se puede acceder utilizando **AJAX** directamente, porque el mecanismo **CORS** está habilitado en el servidor.

8. ALMACENAMIENTO WEB. APLICACIONES EN CACHÉ

Pese a que la principal utilidad del **desarrollo web en entorno cliente** es crear **páginas dinámicas**, que se comunican con una parte de **servidor**, para recoger información y presentarla, en muchas ocasiones es útil poder **almacenar** información directamente en el **cliente**.

Esto nos permite a menudo agilizar la velocidad de la página, mejorar la interacción con el cliente o reducir el número de peticiones web hacia el servidor, cuando es posible mantener estos datos en la caché de la página.

Hoy en día existen **tres formas de almacenar datos en el lado del cliente**, cada una con sus diferencias:

- **Cookies**.
- **LocalStorage o SessionStorage**.
- **IndexedDB**.

➔ **Cookies**.

Ya hemos hablado ampliamente de las **cookies** en unidades anteriores, pero recordaremos que:

*Básicamente se basan en parejas **clave-valor** que permiten almacenar información del usuario para no tener que volver a introducirla en cada visita.*

➔ **LocalStorage o SessionStorage**.

El principal problema de las **cookies** es su **tamaño** y **estructura**, que apenas permite enviar pequeños trozos de texto, por lo que a menudo necesitamos otro tipo de soluciones, para enviar, por ejemplo, imágenes.

Algunas de las características son:

- ✔ Permite datos de hasta **5 MB**.
- ✔ Los datos no se transmiten, si no que se **almacenan** en el navegador.
- ✔ No tienen **fecha de caducidad**.
- ✔ Al igual que las cookies, se guarda una pareja **clave-valo**r, que, en este caso, están asociadas a un dominio.

La principal diferencia entre **LocalStorage y SessionStorage**, es que, en este último, los datos se eliminan cuando **finaliza la sesión** en el navegador.

Veamos un **ejemplo** de uso de algunas de sus funciones:

```
// Crear una clave-valor
localStorage.setItem('key', 'value');

// Ontener el valor de una clave
let value = localStorage.getItem('key');

// Eliminar una clave
localStorage.removeItem('key');

// Guardar un valor en formato JSON
value = JSON.stringify({"key1": true, "key2": 42, "key3": "Hello World!"});
localStorage.setItem('key', value);
```

➔ **IndexedDB**.

Hablaremos de IndexedDB en el siguiente capítulo.

Vídeo: JavaScript

Visualiza este video en el que se habla sobre qué es JavaScript, cómo crear y obtener una sesión sessionStorage con Lynn.
https://www.youtube.com/embed/Kgbu-_C98bU

9. BASES DE DATOS SQL (STANDARD QUERY LANGUAGE) EN ENTORNO CLIENTE

Como hemos dicho, existen tres grandes alternativas para el almacenamiento web en el lado del cliente:

- Cookies.
- LocalStorage o SessionStorage.
- IndexedDB.

Durante este capítulo nos centraremos en IndexedDB, que es el mecanismo para utilizar bases de datos estructuradas en entorno cliente.

Su principal característica es que permite almacenar grandes cantidades de datos.

Además, otras de sus características son:

- ➔ Permite **almacenar datos estructurados**, como haríamos en cualquier base de datos.
- ➔ Los datos se almacenan en **parejas clave-valor**, pero además permite búsquedas por varias claves e índices.
- ➔ Se basa en un **modelo de base de datos transaccional**, por lo que cada una de las operaciones siempre se realiza dentro de una transacción.
- ➔ **Trabaja de forma asíncrona**, utilizando muchas solicitudes, que pueden ser gestionadas mediante eventos y listeners.
- ➔ Está **orientada a objetos**. No se trata de una base de datos relacional al uso, que contiene tablas y registros.

Trabaja mediante **objectStore**, para almacenar **datos relacionados**, a los cuales se accede por una clave primaria. Por el contrario, la gran diferencia es que puede haber varios **índices**, cada uno de los cuales indexa los datos con una clave distinta.

Permite **operaciones de inserción**, **consulta**, **actualización**, **eliminación** y **búsqueda**.

```
// Crea un objectStore
let objectStore = db.createObjectStore(notas, {keyPath: 'id',
autoIncrement: true});


// Define qué elementos de datos contendrá el objectStore
objectStore.createIndex('title', 'title', { unique: false });
objectStore.createIndex('body', 'body', { unique: false });
```

Actividad de aprendizaje 2: peticiones AJAX

Siguiendo con el periodo de aprendizaje de programación asíncrona, nos proponen un nuevo mini proyecto.

El objetivo de esta actividad es aprender a enviar peticiones AJAX y procesar la respuesta recibida en formato JSON utilizando la biblioteca jQuery.

Nos piden que implementemos una aplicación que lea el archivo de datos JSON de la siguiente URL mediante una llamada AJAX y lo procese para añadir una fila a una tabla con esta información en su página web.

No es necesario preocuparse por el estilo, porque vendrá dado por la página donde se incrustará. Puedes añadir la tabla directamente al body del documento HTML.

Lo principal es fijarse bien en la información que se devuelve y cómo conseguirla.

Investigad conjuntamente el contenido del JSON, para poder realizar el script.

Entrega el archivo .html en el espacio habilitado para ello.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cuál de los siguientes no es un método de almacenamiento web en el cliente?

a. Cookies.

b. MySQL.

c. IndexedDB.

Lee el enunciado e indica la opción correcta:

¿Cuál de los siguientes sistemas de almacenamiento no es jerárquico?

a. XML.

b. JSON.

c. SQL.

Lee el enunciado e indica la opción correcta:

¿Cuál de los siguientes sistemas de almacenamiento web en cliente permite mayor capacidad?

a. IndexedDB.

b. Cookies.

c. LocalStorage.

RESUMEN

Esta quizás ha sido una de las unidades más teóricas que hemos visto, pero que nos ha permitido conocer conceptos interesantes que aplicaremos en nuestra carrera como desarrolladores.

Empezamos viendo lo que es la **comunicación síncrona**, **asíncrona**, sus **diferencias** y cuándo es mejor utilizar cada cual.

Además, hemos visto cómo podemos acceder mediante **JavaScript** a datos remotos de forma asíncrona, lo que nos permite más **independencia** del servidor.

Hemos conocido dos de los grandes formatos actuales para el **envío** y **recepción** de información a través de internet, como son **JSON** y **XML**.

Hemos visto ejemplos, diferencias y capacidades de uso de cada uno de los dos formatos. Parecen muy similares, pero en el fondo pueden tener usos distintos.

Hemos conocido más en profundidad una **librería** que ya habíamos visto por encima anteriormente, **jQuery**, y más en concreto para su uso junto a **AJAX**, la principal tecnología de comunicación asíncrona.

Finalmente hemos visto las posibilidades de almacenamiento web en entorno cliente, centrándonos, sobre todo en **IndexedDB**.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- IndexedDB - Referencia de la API Web | MDN: *https://developer.mozilla.org/es/docs/Web/API/IndexedDB_API* [Consulta septiembre 2022].
- Window.localStorage - Referencia de la API Web | MDN: *https://developer.mozilla.org/es/docs/Web/API/Window/localStorage* [Consulta septiembre 2022].
- Window.sessionStorage - Referencia de la API Web | MDN: *https://developer.mozilla.org/es/docs/Web/API/Window/sessionStorage* [Consulta septiembre 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- JSON - JavaScript | MDN: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON* [Consulta septiembre 2022].
- W3C: *https://www.w3.org/* [Consulta septiembre 2022].

GLOSARIO

- **Cross-browser**: técnica que permite que una página se vea de manera idéntica en varios navegadores.
- **GET/POST**: métodos para el envío de información entre páginas web.
- **jqXHR**: es una versión ampliada del objeto XMLHttpRequest de los navegadores.
- **JSON**: JavaScript Object Notation, permite almacenar datos mediante parejas clave-valor, de forma jerárquica.