

Browser objects

Els Browser Object Model (BOM), o Objectes del Navegador, són jerarquies d'objectes proporcionades pel navegador web (no l'estàndard ECMAScript) que permeten a JavaScript interactuar i manipular la finestra del navegador i el seu entorn, anomenat entorn host.

Objecte window: El Global

L'objecte window és l'objecte global del BOM i representa la finestra o pestanya oberta del navegador. Tots els altres objectes del BOM, com location, history, screen, i navigator, són propietats o sub-objectes de window. Les variables i funcions globals de JavaScript esdevenen propietats de l'objecte window.

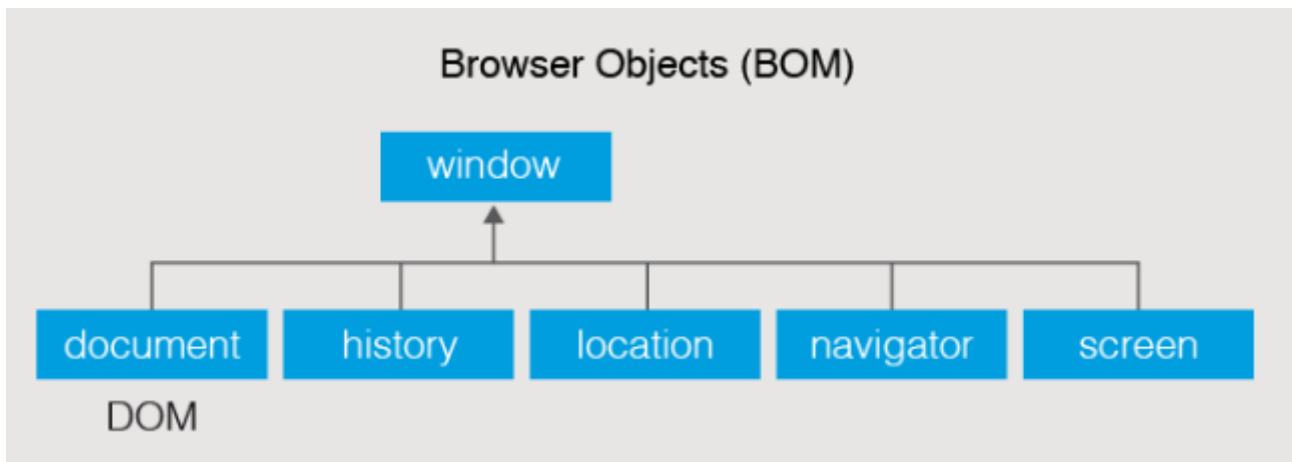
Objectes Clau del BOM

Els objectes principals i les seves funcionalitats són:

- **window**: La pròpia finestra del navegador. Propietats com window.innerWidth (amplada de la finestra) o mètodes com alert(), setTimeout(), i open()
- **location**: Proporciona informació detallada sobre la URL actual del document (protocol, host, path, etc.)
- **history**: Permet navegar cap enrere i endavant a través de l'històric de sessions de la pestanya. És fonamental per a les Single Page Applications (SPAs) el control del state
- **navigator**: Conté dades sobre el navegador de l'usuari i el sistema operatiu (SO), altres com la cadena d'identificació, l'estat de la connexió o accés a APIs del dispositiu com la geolocalització o la càmera/micròfon
- **screen**: Proporciona informació sobre la pantalla física de l'usuari, com la resolució total, la profunditat de color...

- [Qué són els browser objects?](#)
- [Objecte window](#)
- [Objecte document](#)
- [Objecte location](#)
- [Objecte history](#)
- [Objecte navigator](#)
- [Objecte screen](#)
- [Objectes storage](#)

Qué són els browser objects?



Els **Browser Object Model (BOM)** són un conjunt d'objectes que el navegador crea per interactuar amb la finestra del navegador i la seva funcionalitat, a més del contingut del document (que es gestiona mitjançant el DOM, a través de l'objecte `document`).

`window` (L'Objecte Global)

- **Què és:** És l'objecte **arrel (o global)** del BOM. Totes les variables i funcions globals de JavaScript esdevenen propietats i mètodes de l'objecte `window`. Representa la finestra oberta del navegador o el frame que conté el document.
- **Rol Principal:** Proporciona accés a les funcions del navegador i serveix de contenidor per a altres objectes del BOM, com `document`, `location`, `history`, etc. També gestiona **temporitzadors** (`setTimeout`, `setInterval`) i la creació de **subfinestres** (`pop-ups`).

Exemples:

Acció	Codi JavaScript	Descripció
Alerta	<code>window.alert('Hola món!');</code>	Mostra una finestra d'alerta. (<code>window.</code> és sovint opcional).
Temporitzador	<code>setTimeout(function() { console.log('1s passat'); }, 1000);</code>	Executa una funció després d'1 segon.
Mida de la finestra	<code>console.log(window.innerWidth);</code>	Retorna l'amplada de l'àrea de visualització (viewport) en píxels.

location (La URL)

- **Què és:** Conté informació detallada sobre la **URL (Uniform Resource Locator)** actual de la finestra (el document que s'està mostrant).
- **Rol Principal:** Permet **llegir i modificar** la direcció web, cosa que provoca que el navegador navegui a una nova pàgina o recarregui l'actual.

Exemples:

Acció	Codi JavaScript	Descripció
URL actual	<code>console.log(location.href);</code>	Mostra la URL completa (p. ex., <code>http://example.com/pagina.html</code>).
Redirecció	<code>location.href = 'https://google.com';</code>	Carrega la pàgina de Google.
Recàrrega	<code>location.reload();</code>	Recarrega la pàgina actual.

history (Historial de Navegació)

- **Què és:** Conté informació sobre les **direccions URL** que l'usuari ha visitat en la **sessió actual** (la pestanya o finestra).
- **Rol Principal:** Permet al codi JavaScript **navegar cap enrere o endavant** en l'historial del navegador, simulant els botons 'Enrere' i 'Endavant'.

Exemples:

Acció	Codi JavaScript	Descripció
Enrere	<code>history.back();</code>	Equivalent a fer clic al botó 'Enrere' del navegador.
Endavant	<code>history.forward();</code>	Equivalent a fer clic al botó 'Endavant' del navegador.
Moure's	<code>history.go(-2);</code>	Torna dues pàgines enrere en l'historial.

document (El DOM)

- **Què és:** Conté tota la **informació i l'estructura del contingut web** que es mostra actualment a la finestra (el fitxer HTML i els seus elements). És la porta d'entrada al **Document Object Model (DOM)**.
- **Rol Principal:** Permet accedir, manipular i interactuar amb tots els elements de la pàgina (etiquetes, textos, imatges, formularis, estils, etc.). També conté objectes per a dades globals com `cookies` o el `títol` de la pàgina.

Exemples:

Acció	Codi JavaScript	Descripció
Títol	<code>console.log(document.title);</code>	Retorna el text de l'etiqueta <code><title></code> .
Accés a element	<code>let boto = document.getElementById('btn');</code>	Obté un element HTML per la seva ID.
Canvi de fons	<code>document.body.style.backgroundColor = 'lightblue';</code>	Canvia el color de fons del cos de la pàgina.

Els Events (Interactivitat)

És important subratllar que, a més de propietats (dades) i mètodes (funcions), tots aquests objectes són fonamentals per a la **interactivitat** a través dels **events**.

Un **event** és una acció que té lloc a la pàgina web, sovint provocada per l'usuari (p. ex., un clic, escriure, moure el ratolí, carregar la pàgina). Els objectes del BOM (especialment `window` i `document`) permeten **escoltar** i **respondre** a aquests esdeveniments.

Exemple d'Event:

JavaScript

```
// L'event es dispara quan la pàgina (document)
// ha acabat de carregar tots els seus elements.
window.onload = function() {
    console.log("La pàgina s'ha carregat completament!");
};

// L'event es dispara quan l'usuari fa clic
// en un element (suposant que existeix un amb ID 'elMeuBoto')
document.getElementById('elMeuBoto').onclick = function() {
    alert("Has fet clic!");
};
```

Objecte window

L'Objecte `window`: L'Arrel del Navegador

L'objecte `window` és l'objecte **global** de JavaScript quan s'executa en un navegador web. No només representa la finestra oberta del navegador, sinó que també és el contenidor per a **tots** els altres objectes del BOM (com `document`, `location`, `history`, etc.) i per a les variables i funcions globals de JavaScript.

La Jerarquia (BOM i DOM)

La jerarquia del navegador i el document es construeix a partir de `window`:

1. **`window` (BOM):** L'arrel. Representa la finestra/pestanya sencera.
2. **`document` (DOM):** Fill de `window`. Representa el contingut HTML carregat dins d'aquesta finestra.
3. **Elements HTML:** Fills de `document`. Representen etiquetes concretes (com `<img>`, `<button>`, etc.).

L'objecte `window` es pot **ometre** en la majoria de referències als seus fills (p. ex., `window.document` és el mateix que `document`, i `window.alert()` és `alert()`).

Exemple de Jerarquia (DOM)

L'accés directe a elements com `document.img1` (utilitzant l'atribut `name`) és un mètode antic i no es recomana actualment. El mètode estàndard per accedir a elements és utilitzar el **DOM (Document Object Model)**.

Mètode Antic (Desaconsellat)	Mètode Modern (Recomanat)
<code>console.log(document.img1.width);</code>	<code>document.getElementById('id_de_la_imatge').width;</code>

Codi HTML/JavaScript Modern:

HTML

```
<!DOCTYPE html>
<html>
  <body>
```

```


<script>
  // Accedim a l'objecte document a través de window
  const imatge = document.getElementById('muntanyaImg');

  // Accés a propietats de l'element <img>
  console.log("Amplada de la imatge:", imatge.width);

  // Accés a una propietat de window (sense window.)
  console.log("URL de la pàgina:", location.href);
</script>
</body>
</html>

```

Propietats Clau de l'Objecte window

Les propietats de `window` permeten obtenir informació sobre l'estat del navegador i del document.

Propietat	Descripció	Exemple Pràctic
<code>document</code> , <code>location</code> , <code>history</code>	Són els objectes fonamentals del BOM que ja coneixes.	<code>console.log(document.title);</code>
<code>innerWidth</code> , <code>innerHeight</code>	Retornen l' amplada i altura de l'àrea visible (viewport) del navegador en píxels, excloent les barres d'eines, etc.	<code>console.log(window.innerWidth);</code>
<code>screenX</code> , <code>screenY</code>	Retornen la posició horitzontal i vertical de la finestra del navegador respecte a la pantalla de l'usuari. (També <code>screenLeft</code> / <code>screenTop</code>).	<code>console.log(window.screenX);</code>
<code>parent</code> , <code>top</code> , <code>self</code>	Utilitzades principalment per treballar amb <code>iframes</code> (marcs). <code>parent</code> és el pare immediat; <code>top</code> és la finestra superior (l'arrel de totes les subfinestres); <code>self</code> és la finestra actual (<code>window</code>).	Permet trencar l'iframe: <code>if (window !== window.top) { window.top.location.href = location.href; }</code>
<code>localStorage</code> , <code>sessionStorage</code>	Ofereixen un mecanisme per emmagatzemar dades de forma permanent (<code>localStorage</code>) o durant la sessió actual (<code>sessionStorage</code>) al navegador de l'usuari.	<code>localStorage.setItem('usuari', 'Jordi');</code>
<code>closed</code>	Retorna un valor booleà (<code>true</code> / <code>false</code>) indicant si la finestra ha estat tancada.	<code>if (myWindow.closed) { ... }</code>
<code>status</code> , <code>defaultStatus</code>	(Nota d'ús: Aquestes propietats estan àmpliament obsoletes o ignorades pels navegadors)	

Propietat	Descripció	Exemple Pràctic
	moderns per raons de seguretat i usabilitat).	

Exemple de Propietats (BOM):

JavaScript

```
// La mida interior del viewport (la part de la web)
console.log(`Mida del viewport: ${window.innerWidth}x${window.innerHeight}`);

// La posició de la finestra a la pantalla
console.log(`Posició a la pantalla (X, Y): ${window.screenX}, ${window.screenY}`);

// Emmagatzemant una dada
window.localStorage.setItem('tema', 'fosc');
console.log(`Tema guardat: ${localStorage.getItem('tema')}`);
```

Mètodes Fonamentals de l'Objecte `window`

Els mètodes de `window` s'utilitzen per gestionar la finestra del navegador i la interacció amb l'usuari.

Mètodes d'Interacció amb l'Usuari

Mètode	Propòsit	Exemple Pràctic
<code>alert(missatge)</code>	Mostra un missatge a l'usuari i espera que faci clic a OK.	<code>alert('Benvingut!');</code>
<code>confirm(missatge)</code>	Mostra un missatge i botons OK/Cancel. Retorna <code>true</code> o <code>false</code> .	<code>const volEsborrar = confirm('Estàs segur?');</code>
<code>prompt(missatge, [textDefecte])</code>	Mostra un missatge i un camp de text per introduir dades. Retorna la dada introduïda o <code>null</code> si es cancel·la.	<code>const nom = prompt('Introdueix el teu nom:', 'Anònim');</code>

Mètodes de Finestra i Temporització

Mètode	Propòsit	Exemple Pràctic
<code>open(url, name, params)</code>	Crea una nova finestra (o pestanya) amb una URL i paràmetres de mida i aparença. Retorna una referència a la nova finestra.	<code>const popup = window.open('/avis.html', 'avis', 'width=400,height=200');</code>
<code>close()</code>	Tanca la finestra actual o una finestra creada per l' <code>open()</code> .	<code>popup.close();</code>
<code>setTimeout(funció, msec)</code>	Executa la <code>funció</code> una sola vegada després que passin els mil·lisegons (<code>msec</code>).	<code>const idTimer = setTimeout(() => { alert('Temps esgotat!'); }, 5000);</code>
<code>setInterval(funció, msec)</code>	Executa la <code>funció</code> repetidament cada <code>msec</code> mil·lisegons.	<code>const idInterval = setInterval(() => { console.log('Tick'); }, 1000);</code>
<code>clearTimeout(id)</code> / <code>clearInterval(id)</code>	Cancel·la l'execució pendent de <code>setTimeout</code> o <code>setInterval</code> , utilitzant la ID que retornen.	<code>clearTimeout(idTimer);</code>

Exemple de Mètodes (Temporització):

JavaScript

```
let comptador = 0;

// Configurar una funció que s'executa cada segon
const intervalId = setInterval(() => {
  comptador++;
  console.log(`Segons passats: ${comptador}`);

  // Aturar l'interval després de 5 segons
  if (comptador >= 5) {
    clearInterval(intervalId); // Atura el temporitzador
    console.log("Interval aturat.");
  }
}, 1000);

// Una funció que s'executa només una vegada després de 100 mil·lisegons
setTimeout(() => {
  alert("Benvingut a la pàgina (amb setTimeout)");
}, 100);
```

window.open

El mètode `window.open()` és una funció de JavaScript que s'utilitza per **obrir una nova finestra o una nova pestanya** al navegador. Et permet especificar la URL a carregar, el nom de la finestra i una sèrie de característiques de visualització, com les dimensions i les barres d'eines.

Sintaxi i Paràmetres

La sintaxi completa de `window.open()` és:

JavaScript

```
let novaFinestra = window.open(url, nomFinestra, especificacions, reemplaçarEntradaHistorial);
```

url (String)

- **Descripció:** L'URL del document a carregar a la nova finestra.
- **Opcionalitat:** Opcional. Si s'omet (o és una cadena buida `""`), la nova finestra o pestanya es crea amb una pàgina en blanc (`about:blank`).
- **Exemple:** `'https://www.exemple.cat/pagina.html'`

nomFinestra (String)

- **Descripció:** Un nom per a la nova finestra o pestanya.
- **Opcionalitat:** Opcional.
- **Casos d'Ús:**
 - Si el nom coincideix amb una finestra ja oberta, el contingut es **recarrega** en aquesta finestra existent en lloc d'obrir-ne una de nova.
 - Pots utilitzar valors especials com `_blank` (sempre obre una nova finestra o pestanya, per defecte) o `_self` (carrega l'URL a la finestra actual, actuant com una navegació normal).
- **Exemple:** `'popupLogin'`

especificacions (String)

- **Descripció:** Una cadena separada per comes que defineix les dimensions i característiques visuals de la nova finestra. **Aquest paràmetre només té efecte real si el navegador obre una finestra nova**, no una pestanya.
- **Opcionalitat:** Opcional.
- **Exemples Comuns de Característiques:**
 - `width=W`: Amplada de la finestra en píxels.
 - `height=H`: Alçada de la finestra en píxels.
 - `left=X`, `top=Y`: Posició de la finestra a la pantalla.
 - `resizable=yes/no`: Si l'usuari pot redimensionar la finestra (sovint ignorat).
 - `toolbar=yes/no`: Mostra o amaga la barra d'eines del navegador (sovint ignorat).
- **Exemple:** `'width=600,height=400,left=100,top=100,resizable=no'`

reemplaçarEntradaHistorial (Boolean)

- **Descripció:** Indica si l'URL que s'està obrint ha de reemplaçar l'entrada actual a l'historial de la nova finestra.
- **Opcionalitat:** Opcional (valor per defecte: `false`).
- **Cas d'Ús:** S'utilitza rarament. Si es posa a `true`, la funció funciona de manera similar a `location.replace()` dins de la nova finestra.

Valor de retorn

El mètode `window.open()` retorna una **referència a la nova finestra (objecte `window`)**.

- Si l'obertura té èxit, la variable `novaFinestra` contindrà l'objecte de la finestra emergent, permetent-te interactuar amb ella des del script original (p. ex., cridar funcions o tancar-la).
- Si l'obertura és bloquejada per un pop-up blocker, retorna `null`.

Restriccions importants (bloquejadors de Pop-ups)

La majoria dels navegadors tenen estrictes **polítiques de bloqueig de pop-ups** per millorar l'experiència d'usuari:

1. **Activació per Usuari:** Perquè `window.open()` funcioni correctament i no sigui bloquejat, ha de ser cridat **directament dins d'una funció que es desencadena per una interacció de l'usuari** (p. ex., un `click` d'un botó o una pulsació de tecla).
2. **Temps d'Espera:** Si intentes cridar `window.open()` després d'un `setTimeout` o d'una promesa asíncrona, serà molt probablement bloquejat.

Exemples i casos d'ús

Obrir una nova pestanya

Aquesta és la forma més comuna d'usar `window.open()` avui dia, ja que la majoria de navegadors ignoren les especificacions i obren una pestanya.

JavaScript

```
document.getElementById('obrirPestanya').addEventListener('click', () => {
    // La majoria de navegadors ignoren el 'pop-up' i obren una pestanya
    window.open('https://www.google.com', '_blank');
});
```

Obrir una finestra centrada amb dimensions específiques

En alguns navegadors i configuracions, això pot obrir una finestra petita sense barra de navegació, ideal per a diàlegs o logotypes.

JavaScript

```
function obrirLogin() {
    const ample = 400;
    const altura = 500;
    const esquerra = (screen.width / 2) - (ample / 2);
    const superior = (screen.height / 2) - (altura / 2);

    const especificacions = `width=${ample},height=${altura},left=${esquerra},top=${superior},resizable=no,toolbar=no,menubar=no`;

    // Intentem obrir la finestra de login, donant-li el nom 'loginWin'
    const loginWindow = window.open('login.html', 'loginWin', especificacions);

    if (loginWindow === null) {
        alert("El bloquejador de pop-ups ha impedit obrir la finestra de login.");
    }
}
```

Comunicació entre finestres

Utilitzant la referència retornada (`novaFinestra`), el script original pot interactuar amb el contingut de la nova finestra (si ambdues estan al mateix **origen**).

JavaScript

```
let popup;

document.getElementById('obrir').addEventListener('click', () => {
    popup = window.open('salutacio.html', 'salutacio', 'width=300,height=200');
});

document.getElementById('enviarMissatge').addEventListener('click', () => {
    if (popup && !popup.closed) {
        // Crida una funció definida dins de salutacio.html (si tenen el mateix origen)
        popup.mostrarMissatge('Hola des de la finestra principal!');
    }
});
```

Objecte document

L'Objecte `document`: La Porta al DOM

L'objecte `document` és la propietat més important de l'objecte `window`. La seva funció principal és servir com a **node arrel** de l'estructura **DOM (Document Object Model)**, permetent al codi JavaScript accedir, inspeccionar i manipular el contingut, l'estructura i els estils del document HTML que es mostra.

Es pot visualitzar com l'arbre de nodes que representa el document HTML, on cada etiqueta, text o atribut és un node.

Propietats Clau de l'Objecte `document`

Aquestes propietats permeten obtenir informació general sobre el document.

Propietat	Descripció	Exemple Pràctic
<code>title</code>	Cadena de text que conté el contingut de l'etiqueta <code><title></code> . És editable .	<code>document.title = "Nou Títol ";</code>
<code>location</code>	Un objecte que conté la URL completa del document. (És idèntic a <code>window.location</code>).	<code>console.log(document.location.href);</code>
<code>lastModified</code>	Una cadena de text que indica la data i hora de l'última modificació del document.	<code>console.log(document.lastModified);</code>
<code>cookie</code>	Permet llegir i escriure les galetes (cookies) associades al document.	<code>document.cookie = "tema=fosc; expires=Fri, 31 Dec 2024 12:00:00 UTC";</code>
<code>body</code>	Retorna l'element <code><body></code> del document. Molt útil per canvis d'estil o afegir elements.	<code>document.body.style.backgroundColor = 'lightgray';</code>
<code>documentElement</code>	Retorna el node arrel del DOM, que normalment és l'element <code><html></code> .	<code>console.log(document.documentElement.tagName); // "HTML"</code>

Mètodes Clau per a la Manipulació del DOM

Si bé `document.write()` existeix, els mètodes següents són la base de la manipulació moderna del DOM, ja que permeten la interacció amb elements específics sense sobreescriure tot el contingut.

Mètode	Propòsit	Exemple Pràctic
<code>getElementById(id)</code>	El més comú. Retorna el primer element l'atribut <code>id</code> del qual coincideix amb el valor especificat.	<pre>const btn = document.getElementById('boto_enviar');</pre>
<code>querySelector(selector)</code>	Retorna el primer element que coincideixi amb un selector CSS (p. ex., <code>.classe</code> , <code>#id</code> , <code>p > a</code>).	<pre>const primerEnllac = document.querySelector('.menu a');</pre>
<code>querySelectorAll(selector)</code>	Retorna una NodeList (una llista) de tots els elements que coincideixen amb el selector CSS.	<pre>const totsElsParagrafs = document.querySelectorAll('p');</pre>
<code>createElement(nomEtiqueta)</code>	Crea un nou node d'element amb el nom d'etiqueta especificat (p. ex., 'div', 'p').	<pre>const nouDiv = document.createElement('div');</pre>
<code>createTextNode(text)</code>	Crea un nou node de text .	<pre>const textNode = document.createTextNode('Hola!');</pre>
<code>appendChild(node)</code>	Afegeix un node com a últim fill de l'element. (Aquest mètode és en realitat de l'element, no del document).	<pre>document.body.appendChild(nouDiv);</pre>

Exemple de Manipulació del DOM

Aquest exemple mostra com canviar el text d'un element i afegir un nou element a la pàgina.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <h1 id="titol-principal">Benvingut</h1>
  <ul id="llista-tasques">
    <li>Tasca 1</li>
  </ul>

  <script>
    // 1. Canviar el contingut d'un element existent
    const titol = document.getElementById('titol-principal');
    titol.textContent = "Manipulació del DOM OK "; // textContent és millor que
    innerHTML per text simple
```

```
// 2. Crear i afegir un nou element
const llista = document.getElementById('llista-tasques');

// Crear un nou <li>
const novaTasca = document.createElement('li');

// Crear el text per a aquest <li>
novaTasca.textContent = "Nova tasca dinàmica";

// Afegir el nou <li> a la llista <ul>
llista.appendChild(novaTasca);
</script>
</body>
</html>
```

Mètodes d'Escriptura (`write` / `open` / `close`)

Els mètodes `write()`, `writeln()`, `open()` i `close()` s'utilitzaven històricament per generar contingut de manera dinàmica.

- **Advertència:** L'ús de `document.write()` després que el document s'hagi carregat completament pot **sobreescriure tot el contingut de la pàgina**, ja que implícitament crida a `document.open()`. El seu ús modern és gairebé exclusivament dins de la fase de càrrega o dins d'una finestra creada per `window.open()`.

Exemple d'ús de `write()`

JavaScript

```
function reemplaçarContingut() {
  // Obre el document per a escriptura, esborrant tot l'anterior
  document.open("text/html");

  document.write("<h2>Nou contingut generat dinàmicament</h2>");
  document.write("<p>La data d'avui és: " + new Date().toLocaleDateString() + "</p>");

  // Tanca el document i mostra el contingut (obligatori després de open)
  document.close();
}
```

```
// Si cridem aquesta funció, l'HTML original desapareixerà.  
// Per això, la manipulació amb createElement és l'opció preferida.
```

 **Atenció.** Mètode no recomanat

Objecte location

L'objecte `location` de JavaScript és una propietat del objecte `window` (o del objecte `document`) i representa la **ubicació actual (URL) del document** que s'està mostrant al navegador. Permet llegir i modificar les diferents parts de l'URL i, per tant, navegar a altres pàgines o recarregar la pàgina actual.

Pots accedir-hi mitjançant `window.location` o simplement `location` (ja que `window` és l'objecte global).

Propietats de `location`

L'objecte `location` ofereix diverses propietats de només lectura o lectura/escriptura que permeten accedir a les parts individuals de l'URL. Si modifiques una d'aquestes propietats (excepte `hash`, `search`, i `pathname` en certs contextos), el navegador intentarà navegar a la nova URL.

Suposem que l'URL actual és: `https://usuari:contrasenya@www.exemple.cat:8080/caminet/fitxer.html?id=123&nom=prova#seccio1`

Propietat	Descripció	Exemple de Valor	Modificable
<code>href</code>	Tota l'URL completa.	<code>https://usuari:contrasenya@www.exemple.cat:8080/caminet/fitxer.html?id=123&nom=prova#seccio1</code>	
<code>protocol</code>	El protocol de l'URL, inclòs el caràcter <code>:</code> .	<code>https:</code>	
<code>host</code>	El nom del host (servidor) i el número de port (si no és el port per defecte).	<code>www.exemple.cat:8080</code>	
<code>hostname</code>	El nom del host (servidor), sense el port.	<code>www.exemple.cat</code>	
<code>port</code>	El número de port de l'URL. Si no s'especifica, és una cadena buida o el port per defecte (80 per HTTP, 443 per HTTPS).	<code>8080</code>	
<code>pathname</code>	El camí (path) de l'URL, que comença amb <code>/</code> .	<code>/caminet/fitxer.html</code>	
<code>search</code>	La cadena de consulta (query string), que comença amb <code>?</code> .	<code>?id=123&nom=prova</code>	
<code>hash</code>	El fragment o anclatge de l'URL, que comença amb <code>#</code> .	<code>#seccio1</code>	
<code>origin</code>			

Propietat	Descripció	Exemple de Valor	Modificable
	L'origen de l'URL (protocol, host, i port). De només lectura.	<code>https://www.exemple.cat:8080</code>	

Mètodes de `location`

L'objecte `location` té tres mètodes principals que s'utilitzen per a la navegació i la recàrrega de la pàgina.

Mètode	Descripció	Efecte
<code>assign(url)</code>	Carrega el recurs a l'URL especificat.	Afegeix una nova entrada a l'historial de navegació, permetent a l'usuari tornar enrere. És equivalent a assignar un valor a <code>location.href</code> .
<code>replace(url)</code>	Carrega el recurs a l'URL especificat.	Substitueix l'URL actual a l'historial de navegació, evitant que l'usuari pugui fer servir el botó "Enrere" per tornar a la pàgina anterior.
<code>reload(forceReload)</code>	Recarrega el document actual.	Recarrega la pàgina des del cau del navegador (per defecte). Si es passa <code>true</code> com a argument (<code>location.reload(true)</code>), força la recàrrega del servidor, ignorant la memòria cau.

Exemples, casos d'ús

1. Lectura de l'URL (Obtenir Informació)

Pots utilitzar les propietats per analitzar l'URL actual i extreure'n dades.

JavaScript

```
// Si l'URL és: https://localhost:3000/productes/detall.html?sku=P45#reviews

console.log(location.protocol); // Resultat: "https:"
console.log(location.hostname); // Resultat: "localhost"
console.log(location.port);     // Resultat: "3000"
console.log(location.pathname); // Resultat: "/productes/detall.html"
console.log(location.search);   // Resultat: "?sku=P45"
console.log(location.hash);     // Resultat: "#reviews"
```

```
// Cas d'ús: Obtenir un paràmetre de consulta (cal una petita lògica addicional o usar
URLSearchParams)
const urlParams = new URLSearchParams(location.search);
const sku = urlParams.get('sku');
console.log(sku); // Resultat: "P45"
```

2. Navegació i Redirecció

Redirecció standard (`assign` o `href`)

Afegeix l'URL a l'historial. L'usuari pot tornar a la pàgina anterior.

JavaScript

```
// Mètode 1: Assignar a href
location.href = 'https://www.novaweb.cat/inici';

// Mètode 2: Usar el mètode assign() (equivalent)
location.assign('https://www.novaweb.cat/inici');

// Cas d'ús: Redirigir l'usuari després d'una acció (ex: login)
if (usuariHaFetLogin) {
    location.href = '/dashboard';
}
```

Redirecció sense historial (`replace`)

S'utilitza per evitar que l'usuari torni a la pàgina anterior, per exemple, després d'un formulari de login o una pàgina de càrrega temporal.

JavaScript

```
// La pàgina actual (ex: login.html) serà reemplaçada a l'historial pel dashboard.
location.replace('/dashboard');

// Cas d'ús: Evitar que l'usuari torni a la pàgina de login després d'entrar.
```

3. Recàrrega de pàgina (`reload`)

Per refrescar la pàgina actual.

JavaScript

```
// Recàrrega normal (pot usar el cau del navegador)
location.reload();
```

```
// Recàrrega forçada (ignora el cau, forçant una nova petició al servidor)
location.reload(true);

// Cas d'ús: Actualitzar una llista de dades després d'esborrar un element.
```

4. Modificació de part de l'URL

Pots canviar components individuals per recarregar la pàgina a la mateixa base, però amb un nou valor.

JavaScript

```
// Canviar el hash per anar a una altra secció de la mateixa pàgina
location.hash = 'contacte';
// L'URL canvia i el navegador es desplaça al nou anclatge, sense recarregar.

// Canviar la cadena de consulta per filtrar contingut
location.search = '?filtre=actiu&ordre=asc';
// El navegador recarregarà la pàgina amb la nova query string.

// Canviar el camí (pathname)
location.pathname = '/nous-productes';
// El navegador recarregarà la pàgina amb el nou camí.
```

Objecte history

L'objecte `history` de JavaScript és l'altre component clau per gestionar la navegació i l'experiència de l'usuari dins d'una aplicació web. `History` forma part del **BOM (Browser Object Model)** i proporciona accés a les sessions de l'historial del navegador, permetent-te navegar cap endavant i cap enrere dins de les pàgines visitades per l'usuari a la pestanya actual.

Es pot accedir-hi mitjançant `window.history` o simplement `history`.

Propietats de `history`

L'objecte `history` té dues propietats principals, ambdues de només lectura.

Propietat	Descripció	Exemple de Valor
<code>length</code>	Retorna el nombre d'entrades a l'historial de sessions del navegador per a la pestanya actual (incloent-hi la pàgina actual).	<code>3</code> (Si l'usuari ha visitat dues pàgines abans de l'actual).
<code>state</code>	Retorna l'estat que es va passar a l'última entrada de l'historial mitjançant els mètodes <code>pushState()</code> o <code>replaceState()</code> . L'estat pot ser qualsevol valor o objecte serialitzable.	<code>{ filtre: 'actiu', pagina: 2 }</code>

Mètodes de `history`

Els mètodes de `history` es divideixen en dues categories: navegació tradicional i manipulació de l'historial (clau per a les **Single Page Applications - SPAs**).

Mètodes de Navegació

Aquests mètodes simulen la pulsació dels botons "Enrere" i "Endavant" del navegador.

Mètode	Descripció	Cas d'Ús
<code>back()</code>	Simula el botó "Enrere". Carrega l'URL anterior a l'historial.	Crear un botó de "Tornar" personalitzat dins de la pàgina.
<code>forward()</code>	Simula el botó "Endavant". Carrega l'URL següent a l'historial.	Utilitzat menys freqüentment, però permet avançar a la pàgina posterior.

Mètode	Descripció	Cas d'Ús
<code>go(delta)</code>	Carrega una pàgina específica a l'historial, on <code>delta</code> és un nombre.	<code>history.go(0)</code> : Recarrega la pàgina actual (similar a <code>location.reload()</code>). <code>history.go(-2)</code> : Torna dues pàgines enrere.

Mètodes de manipulació historial (HTML5 History API)

Aquests són els mètodes més potents i s'utilitzen per canviar l'URL (i, per tant, l'estat) **sense recarregar la pàgina**, cosa essencial per a les SPAs.

Mètode	Descripció	Conseqüència
<code>pushState(state, title, url)</code>	Afegeix una nova entrada a l'historial.	Afegeix una nova entrada a l'historial i canvia l'URL a la barra d'adreces sense recarregar la pàgina. Permet l'ús dels botons d'historial.
<code>replaceState(state, title, url)</code>	Modifica l'entrada actual de l'historial.	Canvia l'URL actual i el seu estat sense recarregar la pàgina. No afegeix una nova entrada a l'historial, sinó que sobreescriu la que hi ha.

Paràmetres per a `pushState` i `replaceState`:

- `state` (objecte):** Un objecte de dades que conté l'estat de la sessió. Quan l'usuari navega a aquesta entrada de l'historial (usant Enrere/Endavant), l'objecte `state` és restaurat i accessible mitjançant `history.state` o l'esdeveniment `popstate`.
- `title` (string):** Actualment ignorat per la majoria de navegadors, però es pot passar una cadena buida (``) o el títol de la nova pàgina.
- `url` (string):** La nova URL de la pàgina. Opcionalment, pot ser només el camí (`/productes/45`) sense l'origen.

Exemples i casos d'ús

1. Navegació Bàsica

Simplement movent-se per l'historial.

JavaScript

```
// Retorna a la pàgina anterior
function anarEnrere() {
  history.back();
}
```

```
}

// Recarrega la pàgina actual amb un truc
function recarregar() {
  history.go(0);
}

// Torna dues pàgines enrere (útil si vols forçar l'usuari a saltar una transició)
setTimeout(() => {
  history.go(-2);
}, 5000);
```

2. Single Page Applications (SPAs) amb `pushState()`

Aquest és el cas d'ús més important de l'API History. Quan un usuari fa clic a un enllaç en una SPA, normalment carregues contingut nou amb AJAX sense recarregar. Però vols que la barra d'adreces mostri la URL correcta (`/usuari/perfil`) i que els botons Enrere/Endavant funcionin.

JavaScript

```
function carregarPerfilUsuari(id) {
  // 1. Carregar les dades de l'usuari amb AJAX o Fetch
  // ... (Codi per actualitzar la interfície d'usuari) ...

  const nouEstat = {
    userId: id,
    vista: 'perfil'
  };
  const novaURL = `/usuari/${id}`;

  // 2. Afegir una nova entrada a l'historial sense recarregar la pàgina
  history.pushState(nouEstat, 'Pàgina de Perfil', novaURL);

  console.log(`Nova entrada a l'historial: ${novaURL}`);
}

// Exemple d'ús:
// L'URL actual és: https://meuweb.cat/inici
carregarPerfilUsuari(101);
// La nova URL a la barra d'adreces és: https://meuweb.cat/usuari/101
// La pàgina NO s'ha recarregat.
```

3. Modificació de l'estat actual amb `replaceState()`

S'utilitza per actualitzar l'estat de l'entrada actual (per exemple, un filtre de cerca) sense crear una nova entrada a l'historial.

JavaScript

```
function aplicarFiltre(criteri) {
  // 1. Carregar les dades filtrades
  // ... (Codi per actualitzar la llista de productes) ...

  const nouEstat = {
    criteriFiltre: criteri,
    timestamp: Date.now()
  };
  const novaURL = `/?filtre=${criteri}`;

  // 2. Modificar l'entrada ACTUAL de l'historial.
  history.replaceState(nouEstat, 'Productes Filtrats', novaURL);

  console.log(`Estat de l'historial actualitzat a: ${criteri}`);
}

// L'usuari fa una cerca:
aplicarFiltre('ofertes');
// La URL canvia a: /?filtre=ofertes. No es crea una nova entrada a l'historial.
```

4. Gestió de l'esdeveniment `popstate`

Quan l'usuari fa clic als botons "Enrere" o "Endavant" del navegador, es dispara l'esdeveniment `popstate`. Aquest esdeveniment és vital en SPAs per carregar el contingut correcte.

JavaScript

```
window.addEventListener('popstate', (event) => {
  // L'objecte state conté les dades que vas passar amb pushState() o replaceState()
  const estatAnterior = event.state;

  if (estatAnterior && estatAnterior.vista === 'perfil') {
    console.log(`Tornant al perfil de l'usuari ${estatAnterior.userId}`);
    // Codi per carregar el perfil usant les dades de l'estat.
  } else {
    console.log("Tornant a una pàgina bàsica (o la pàgina d'inici)");
  }
});
```

```
// Codi per gestionar la navegació a una URL que no té un estat personalitzat.  
}  
  
console.log(`L'URL actual és: ${location.pathname + location.search}`);  
});
```

Aquest cicle de `pushState()` + `popstate` és el que permet que les aplicacions modernes (com Gmail, Twitter, o qualsevol framework de frontend com React, Vue o Angular) gestionin les rutes sense recàrregues completes, oferint una experiència d'usuari ràpida i fluida.

La funció `addEventListener()` és un mètode fonamental de JavaScript que permet registrar una funció (**el listener o gestor d'esdeveniments**) que serà cridada cada vegada que un **esdeveniment** especificat (com un clic de ratolí, una pulsació de tecla, o una càrrega de pàgina) succeeixi sobre un **objecte concret del DOM** (Document Object Model), com un botó, un formulari, o l'objecte `window`. Aquesta funció pren típicament tres arguments: el **tipus d'esdeveniment** (p. ex., `'click'`), la **funció a executar** quan l'esdeveniment passa, i opcionalment, un booleà o objecte per definir si la captura s'ha de fer durant la fase de captura o de bombolleig (sent la fase de bombolleig l'opció per defecte), permetent una programació no intrusiva i mantenint la lògica de JavaScript separada de l'estructura HTML.

Objecte navigator

L'objecte `navigator` de JavaScript és un objecte del **BOM (Browser Object Model)** que proporciona informació sobre el **navegador** de l'usuari i l'**estat del sistema** on s'executa. És el teu punt d'accés per obtenir dades sobre quin programari està utilitzant l'usuari (el nom i la versió del navegador) i quines capacitats té el dispositiu (geolocalització, connexió a Internet, etc.).

Pots accedir-hi mitjançant `window.navigator` o, simplement, `navigator`.

Propietats de `navigator`

Les propietats principals de `navigator` es divideixen en informació bàsica (compatibilitat) i informació avançada (estat del sistema/capacitats).

Propietats d'informació i compatibilitat (Legacy)

Aquestes propietats solien ser molt usades, però avui dia estan desaconsellades per a la detecció de navegadors específica a causa de la suplantació (spoofing) i la manca de precisió. Són de només lectura.

Propietat	Descripció	Exemple de Valor
<code>userAgent</code>	Retorna la cadena completa de l'agent d'usuari (user agent) enviada pel navegador.	<code>'Mozilla/5.0 (Windows NT 10.0; Win64; x64) ... Chrome/117.0.0.0 Safari/537.36'</code>
<code>appName</code>	El nom oficial de l'aplicació. Històricament, retorna gairebé sempre <code>'Netscape'</code> per raons de compatibilitat.	<code>'Netscape'</code>
<code>appVersion</code>	La versió del navegador, sovint poc fiable.	<code>'5.0 (Windows)'</code>
<code>platform</code>	La plataforma del sistema operatiu on s'està executant el navegador.	<code>'Win32'</code> , <code>'MacIntel'</code> , <code>'Linux x86_64'</code>
<code>cookieEnabled</code>	Un booleà que indica si les cookies estan habilitades al navegador (<code>true</code> o <code>false</code>).	<code>true</code>
<code>language</code>	La configuració de llengua preferida del navegador (p. ex., <code>'ca'</code> , <code>'en-US'</code>).	<code>'ca-ES'</code>

Propietats de capacitat i estat del sistema (Modernes)

Aquestes propietats accedeixen a funcionalitats i estats específics del dispositiu o el navegador.

Propietat	Descripció	Cas d'Ús
<code>onLine</code>	Un booleà que indica l'estat de la connexió a Internet (<code>true</code> si està online).	Comprovar si es pot sincronitzar dades amb un servidor.
<code>geolocation</code>	Un objecte que conté mètodes per obtenir la ubicació geogràfica de l'usuari.	Sol·licitar coordenades GPS per mostrar un mapa.
<code>mediaDevices</code>	Un objecte que proporciona accés a càmeres, micròfons i altres dispositius multimèdia.	Desenvolupar aplicacions de videoconferència o càmera web.
<code>clipboard</code>	API per llegir i escriure contingut al porta-retalls.	Implementar un botó de "Copiar URL".
<code>storage</code>	API per gestionar la quota d'emmagatzematge persistent.	Comprovar l'espai disponible per a dades offline.
<code>vendor</code>	El venedor del navegador (sovint buit o poc fiable, excepte per a Chrome).	<code>'Google Inc.'</code>
<code>maxTouchPoints</code>	El nombre màxim de punts de contacte simultanis (tocs) compatibles amb el dispositiu.	Detectar si el dispositiu és tàctil per ajustar la interfície.

Mètodes de `navigator`

A diferència de `location` i `history`, `navigator` té pocs mètodes directes a l'objecte base. La majoria de les funcionalitats s'implementen mitjançant mètodes dels sub-objectes (com `navigator.geolocation` o `navigator.mediaDevices`).

Mètode	Descripció	Cas d'Ús
<code>javaEnabled()</code>	Retorna un booleà que indica si Java està habilitat. (Moltes vegades retorna <code>false</code> o no està suportat).	Comprovació de compatibilitat de components legacy (desaconsellat).
<code>getBattery()</code>	Mètode que permet accedir a l'estat de la bateria (obsolet/no disponible en tots els navegadors).	Utilitzat per reduir el consum d'energia de la pàgina si la bateria és baixa.
<code>vibrate(pattern)</code>	Fes vibrar el dispositiu de l'usuari (si és compatible).	Notificacions tàctils en aplicacions mòbils.

Exemples i casos d'ús

1. Detecció de connexió Online/Offline

Aquest és un cas d'ús molt important per a les aplicacions offline.

JavaScript

```
function comprovarEstatConnexio() {
  if (navigator.onLine) {
    console.log("Estàs connectat a Internet. Sincronitzant dades...");
  } else {
    console.log("Estàs fora de línia. Les dades es desaran localment.");
  }
}

// També pots utilitzar esdeveniments per reaccionar als canvis
window.addEventListener('online', () => {
  console.log("Connexió RESTABLIDA.");
});
window.addEventListener('offline', () => {
  console.log("Connexió PERDUDA.");
});
```

2. Geolocalització (navigator.geolocation)

Permet obtenir la posició de l'usuari amb el seu consentiment.

JavaScript

```
function obtenirPosicio() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      // Funció d'èxit
      (posicio) => {
        const latitud = posicio.coords.latitude;
        const longitud = posicio.coords.longitude;
        console.log(`Posició: ${latitud}, ${longitud}`);
      },
      // Funció d'error
      (error) => {
        console.error(`Error de geo-localització: ${error.code}`);
      }
    );
  }
}
```

```
    );  
  } else {  
    console.error("El teu navegador no suporta la geolocalització.");  
  }  
}
```

3. Detecció de llengua i configuració

Per oferir la versió correcta de la pàgina.

JavaScript

```
const idiomaPreferit = navigator.language; // Ex: 'ca-ES'  
const idiomaBase = idiomaPreferit.split('-')[0]; // Ex: 'ca'  
  
if (idiomaBase === 'ca') {  
  console.log("Mostrant contingut en català.");  
} else if (idiomaBase === 'es') {  
  console.log("Mostrant contingut en castellà.");  
} else {  
  console.log("Utilitzant l'anglès com a idioma per defecte.");  
}
```

4. Detecció de dispositius tàctils

Per oferir controls més grans o una interfície diferent.

JavaScript

```
// La propietat 'maxTouchPoints' és un bon indicador, però hi ha altres mètodes  
const esTactil = navigator.maxTouchPoints > 0;  
  
if (esTactil) {  
  document.body.classList.add('interficie-tactil');  
  console.log("Interfície optimitzada per a tàctil.");  
} else {  
  console.log("Interfície per a ratolí/teclat.");  
}
```

5. Accedir càmera

`navigator.mediaDevices` (Càmera i Micròfon).

L'objecte `mediaDevices` proporciona accés als dispositius d'entrada multimèdia (càmera i micròfon) mitjançant l'API `MediaDevices`. El mètode clau és `getUserMedia()`, que sol·licita permís a l'usuari.

JavaScript

```
// Obté l'element <video> del DOM
const videoElement = document.getElementById('myVideo');

function activarCamera() {
  if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
    // Sol·licita accés al vídeo
    navigator.mediaDevices.getUserMedia({ video: true, audio: false })
      .then(stream => {
        // Connecta el flux de vídeo a l'element <video>
        videoElement.srcObject = stream;
        videoElement.play();
        console.log("Càmera activada correctament.");
      })
      .catch(error => {
        // Gestió d'errors (p. ex., usuari denega el permís)
        console.error("No s'ha pogut accedir a la càmera: ", error);
      });
  } else {
    console.error("El navegador no suporta MediaDevices API.");
  }
}

// Cas d'ús: En una aplicació de videoconferència o un escàner de codis QR.
// Però caldria afegir un `<button onclick="activarCamera()">Activar Càmera</button>`
```

6. Porta-retalls `navigator.clipboard`

L'API `Clipboard` permet llegir i escriure text al porta-retalls del sistema de manera asíncrona, requerint el permís de l'usuari en la majoria dels casos per raons de seguretat.

Example: Escriure (Copiar) al Porta-retalls

JavaScript

```
const textACopiar = "Aquesta és l'URL que vull compartir.";

function copiarAlPortaRetalls(text) {
```

```

if (navigator.clipboard) {
  // Utilitza el mètode asíncron writeText
  navigator.clipboard.writeText(text)
    .then(() => {
      alert(`"${text}" copiat al porta-retalls!`);
    })
    .catch(err => {
      console.error('No s\'ha pogut copiar el text: ', err);
    });
} else {
  // Fallback per a navegadors antics (utilitzant document.execCommand)
  console.warn("Clipboard API no suportada. Utilitzant mètode antic.");
}

// Cas d'ús: Implementar un botó de "Copiar enllaç" o "Copiar codi" fàcil.
// copiarAlPortaRetalls(textACopiar);

```

7. Gestió emmagatzematge

navigator.storage

L'**emmagatzemament persistent** en el context d'una aplicació web es refereix a la capacitat d'una pàgina web o una aplicació web progressiva (**PWA**) de sol·licitar al navegador que **protegeixi les dades emmagatzemades localment** (com les bases de dades IndexedDB, l'API Cache o Web Storage) per evitar que siguin eliminades automàticament pel navegador en situacions de baixa memòria o d'espai d'emmagatzematge limitat. A diferència de l'emmagatzemament normal, que es considera "millor esforç" (best effort) i pot ser esborrat sense previ avís, l'emmagatzemament persistent (**navigator.storage.persist()**) garanteix que les dades crucials d'una aplicació (com el contingut offline o la configuració d'usuari) romanguin segures al dispositiu de l'usuari a llarg termini, millorant la fiabilitat de l'experiència offline.

L'objecte `storage` proporciona mètodes per gestionar l'emmagatzematge del lloc web, com ara el **càlcul de la quota d'emmagatzematge** i la **sol·licitud d'emmagatzematge persistent**. **Exemple:** Comprovar la Quota d'Emmagatzematge

JavaScript

```

function comprovarQuota() {
  if (navigator.storage && navigator.storage.estimate) {
    // Utilitza el mètode estimate() per obtenir l'ús actual i la quota total
    navigator.storage.estimate()
      .then(estimacio => {
        const usMB = (estimacio.usage / 1024 / 1024).toFixed(2);
        const quotaMB = (estimacio.quota / 1024 / 1024).toFixed(2);

        console.log(`Ús actual: ${usMB} MB`);
        console.log(`Quota total: ${quotaMB} MB`);
      });
  }
}

```

```
const percentatgeUs = ((estimacio.usage / estimacio.quota) *
100).toFixed(2);
    console.log(`Usat: ${percentatgeUs}%`);
    });
    } else {
        console.warn("El mètode navigator.storage.estimate() no està disponible.");
    }
}

// Cas d'ús: Aplicacions web progressives (PWA) que necessiten emmagatzemar dades
// grans offline (IndexedDB, Cache API) i comprovar si tenen espai per guardar-les.
// comprovarQuota();
```

8. Informació navegador (navigator.userAgent)

La cadena `userAgent` és un identificador enviat pel navegador a cada servidor web, i actua com el "document d'identitat" digital del client. La seva estructura és històricament complexa i sovint poc precisa a causa de les necessitats de compatibilitat (spoofing). En una cadena `userAgent` típica, hi podem trobar diverses seccions clau que proporcionen informació sobre:

Informació del navegador i motor de renderització

La major part de la cadena se centra en identificar el programari utilitzat per l'usuari per visualitzar el contingut web:

- **Identificador de Compatibilitat (Legacy):** Gairebé sempre comença amb `Mozilla/5.0`. Això es manté per raons històriques, ja que molts llocs antics requerien el navegador Netscape (Mozilla) per funcionar.
- **Motor de Renderització:** Indica el motor que utilitza el navegador per interpretar l'HTML i el CSS. Per exemple:
 - `AppleWebKit/XXX`: Utilitzat per navegadors basats en Chromium (Chrome, Edge, Opera) i Safari.
 - `Gecko/XXX`: Utilitzat per Firefox.
- **Nom i Versió del Navegador:** Es pot trobar el nom real del navegador i la seva versió, tot i que sovint s'emmasken:
 - `Chrome/XXX` (per a Google Chrome o Edge basat en Chromium).
 - `Firefox/XXX` (per a Mozilla Firefox).
 - `Safari/XXX` (per a Apple Safari).

Informació del SSOO i Plataforma

Aquesta secció especifica l'entorn de programari i hardware on s'executa el navegador:

- **Sistema Operatiu (OS):** Identifica l'OS. Exemples:
 - `Windows NT 10.0` (per a Windows 10/11).
 - `Macintosh; Intel Mac OS X 10_15` (per a macOS).

- **X11; Linux x86_64** (per a sistemes Linux).
- **Tipus de Dispositiu:** Indica si és un dispositiu mòbil, de taula, o una tauleta:
 - **Mobi** (una etiqueta comú per a dispositius mòbils).
 - **Android XXX** (per a telèfons/tauletes Android).

Exemple `userAgent`

Aquí hi ha un exemple típic de cadena `userAgent` desglossada:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/120.0.0.0 Safari/537.36
```

Segment	Informació	Significat
<code>Mozilla/5.0</code>	Compatibilitat	Identificador històric.
<code>(Windows NT 10.0; Win64; x64)</code>	Sistema Operatiu	Windows 10/11, arquitectura 64-bit.
<code>AppleWebKit/537.36</code>	Motor de Renderització	Usant el motor WebKit.
<code>(KHTML, like Gecko)</code>	Compatibilitat	Més referències històriques per dir "sóc compatible amb tot".
<code>Chrome/120.0.0.0</code>	Navegador Principal	És Google Chrome, versió 120.
<code>Safari/537.36</code>	Identificador Secundari	S'inclou per compatibilitat amb llocs que busquen Safari.

Casos d'Ús: Històricament, es feia servir per oferir contingut específic a cada navegador o dispositiu, però avui dia s'utilitza principalment per a **anàlisi i logging** de trànsit web i errors. Per detectar funcionalitats (com la càmera o la geolocalització), és molt més recomanable utilitzar la **detecció de característiques** (p. ex., comprovant si `navigator.geolocation` existeix).

La propietat `userAgent` retorna la cadena que el navegador envia al servidor per identificar-se. Tot i que està desaconsellada per a la detecció de funcionalitats (és millor fer detecció de característiques, no de navegadors!), segueix sent útil per a registres i certes necessitats. **Exemple:** Analitzar la Cadena `userAgent`

JavaScript

```
const userAgentString = navigator.userAgent;
let missatge = `La cadena de l'agent d'usuari és: ${userAgentString}`;

// Cas d'ús: Detecció simple (i sovint imprecisa) de mòbil.
// NOTA: Per a detecció de mòbil, és millor fer-ho basat en mitjans CSS o altres
// propietats
// com navigator.maxTouchPoints.

if (userAgentString.includes('Mobi') || userAgentString.includes('Android')) {
  missatge += "\nSembla que l'usuari està utilitzant un dispositiu mòbil.";
} else {
  missatge += "\nSembla que l'usuari està utilitzant un escriptori.";
}
```

```
console.log(missatge);
```

```
// Cas d'ús: Enviar aquesta informació al servidor juntament amb els logs d'error  
// per entendre l'entorn on s'ha produït un problema.
```

Objecte screen

L'objecte `screen` de JavaScript és un objecte del **BOM (Browser Object Model)** que proporciona informació sobre la **pantalla o el monitor** de l'usuari que executa el navegador web. A diferència de l'objecte `window`, que informa sobre la finestra del navegador, `screen` informa sobre les dimensions i la profunditat de color del dispositiu de visualització real.

Pots accedir-hi mitjançant `window.screen` o, simplement, `screen`.

Propietats de `screen`

Totes les propietats de l'objecte `screen` són de **només lectura** i s'expressen en **píxels**.

1. Dimensions totals de la pantalla

Aquestes propietats indiquen la resolució total de la pantalla del dispositiu.

Propietat	Descripció	Exemple
<code>width</code>	L'amplada total de la pantalla en píxels.	<code>1920</code>
<code>height</code>	L'alçada total de la pantalla en píxels.	<code>1080</code>

2. Dimensions disponibles (sense barra tasques)

Aquestes propietats indiquen l'espai real de la pantalla que el navegador pot utilitzar, ja que exclouen elements fixos del sistema operatiu (com la barra de tasques o les dreceres).

Propietat	Descripció	Exemple
<code>availWidth</code>	L'amplada de la pantalla disponible (excloent barres del sistema).	<code>1920</code>
<code>availHeight</code>	L'alçada de la pantalla disponible (excloent barres del sistema).	<code>1040</code>

3. Profunditat de color

Aquestes propietats són útils per a la compatibilitat de colors, tot i que avui dia la majoria dels dispositius utilitzen una alta profunditat de color.

Propietat	Descripció	Exemple
<code>colorDepth</code>	Retorna el nombre de bits utilitzats per mostrar un color. Normalment <code>24</code> o <code>32</code> .	<code>24</code>
<code>pixelDepth</code>	Retorna la resolució de color de la pantalla en bits. Normalment és el mateix que <code>colorDepth</code> .	<code>24</code>

Mètodes/Funcions de `screen`

Actualment, l'objecte `screen` no té mètodes propis àmpliament utilitzats o estandarditzats per manipular directament la pantalla. Totes les interaccions es basen en la lectura de les seves propietats. Les úniques funcions que històricament han estat associades o relacionades amb la pantalla (tot i que formen part de l'objecte `window`) són les que gestionen la finestra mateixa:

- `window.moveTo(x, y)`: Mou la finestra actual a les coordenades

especificades a la pantalla.

- `window.resizeTo(ampçada, alçada)`: Redimensiona la finestra actual a les dimensions especificades.

Nota: L'ús d'aquests mètodes per manipular la finestra està **molt restringit** pels navegadors moderns per raons de seguretat i d'experiència d'usuari, especialment en finestres que no han estat creades per l'script mateix (`window.open()`).

Exemples i casos d'ús

Detecció de resolució

El cas d'ús més comú és detectar la resolució total o disponible per adaptar funcionalitats o notificar a l'usuari.

JavaScript

```
console.log(`Resolució total del monitor: ${screen.width} x ${screen.height}`);
console.log(`Àrea utilitzable (sense barres del sistema): ${screen.availWidth} x $
{screen.availHeight}`);

// Cas d'ús: Alertar sobre resolucions molt baixes
```

```
if (screen.width < 1024) {
    document.getElementById('avís').textContent = "Aquesta web no està optimitzada per a la teva baixa resolució de pantalla.";
}
```

Optimització de Finestra o Pop-ups

Quan es crea una nova finestra emergent (pop-up), es pot utilitzar `screen.width` i `screen.height` per centrar-la o dimensionar-la de manera òptima.

JavaScript

```
function obrirFinestraCentrada(url) {
    const ampleFinestra = 800;
    const alturaFinestra = 600;

    // Calcula les coordenades per centrar la finestra a l'àrea disponible
    const centreX = (screen.availWidth - ampleFinestra) / 2;
    const centreY = (screen.availHeight - alturaFinestra) / 2;

    const especificacions = `width=${ampleFinestra},height=${alturaFinestra},left=${centreX},top=${centreY}`;

    window.open(url, 'FinestraPopup', especificacions);
}

// Cas d'ús: Obrir una finestra de login o de confirmació amb dimensions fixes i centrada.
// obrirFinestraCentrada('login.html');
```

 [Més informació a](#) `window.open`

Detecció de Profunditat de Color

Encara que poc comú avui dia, en el passat era útil per decidir si mostrar imatges d'alta qualitat o una versió simplificada.

JavaScript

```
console.log(`Profunditat de color: ${screen.colorDepth} bits`);

if (screen.colorDepth < 16) {
    console.warn("Pantalla amb colors limitats. Es desactiven els efectes de color complexos.");
}
```

```
// Codi per carregar imatges en 16 bits en lloc de 24.  
}
```

Detecció d'Orientació (Mòbils)

Tot i que l'objecte `screen` conté les dimensions, l'API per detectar i bloquejar l'orientació del dispositiu (vertical o horitzontal) resideix a `screen.orientation` (un sub-objecte).

JavaScript

```
// Obté l'orientació actual (p. ex., 'portrait-primary' o 'landscape-primary')  
console.log(`Orientació actual: ${screen.orientation.type}`);  
  
// Cas d'ús: Bloquejar l'orientació en un joc web  
function bloquejarOrientacio() {  
    if (screen.orientation && screen.orientation.lock) {  
        screen.orientation.lock('landscape')  
            .then(() => console.log("Orientació bloquejada a horitzontal."))  
            .catch(error => console.error("No es pot bloquejar l'orientació:",  
error));  
    }  
}  
  
// També es pot reaccionar als canvis d'orientació amb un listener  
screen.orientation.addEventListener('change', () => {  
    console.log(`L'usuari ha girat la pantalla a: ${screen.orientation.type}`);  
});
```

Objectes storage

L'objecte `Storage` de JavaScript, accessible mitjançant `localStorage` i `sessionStorage`, forma part de l'API Web Storage i s'utilitza per emmagatzemar dades clau-valor localment al navegador de l'usuari. És una alternativa més moderna i amb més capacitat que les cookies per a l'emmagatzematge de dades persistents de la sessió o a llarg termini.

Tipus d'emmagatzematge web

Hi ha dos tipus d'objectes `Storage`, que comparteixen les mateixes propietats i mètodes, però difereixen en la seva persistència:

`localStorage` (Emmagatzematge persistent)

- **Persistència:** Les dades es mantenen **fins que l'usuari les esborra manualment** (a través de la configuració del navegador o el codi). Romanen disponibles fins i tot després de tancar i reobrir el navegador.
- **Cas d'Ús:** Preferències d'usuari (tema fosc/clar), tokens d'autenticació (amb precaució), dades offline, historial de carrets de compra.

`sessionStorage` (Emmagatzematge de sessió)

- **Persistència:** Les dades es mantenen **només durant la sessió de la pestanya o finestra actual**. Quan l'usuari tanca la pestanya o la finestra, les dades s'esborren automàticament.
- **Cas d'Ús:** Dades temporals de la sessió, estat de formularis incomplets, estat de la navegació en una sessió.

Propietats i mètodes

Tant `localStorage` com `sessionStorage` implementen la mateixa interfície `Storage`. Totes les dades s'emmagatzemen com a **parells clau-valor de tipus string**.

Propietats

Propietat	Descripció	Exemple
<code>length</code>	Retorna el nombre de parells clau-valor emmagatzemats a l'objecte <code>Storage</code> actual.	<code>3</code>

Mètodes

Mètode	Descripció
<code>setItem(clau, valor)</code>	Afegeix el parell clau-valor a l'emmagatzematge. Si la clau ja existeix, el valor antic es sobreescriu.
<code>getItem(clau)</code>	Retorna el valor associat a la clau especificada. Retorna <code>null</code> si la clau no es troba.
<code>removeItem(clau)</code>	Elimina el parell clau-valor amb la clau especificada de l'emmagatzematge.
<code>clear()</code>	Elimina tots els parells clau-valor de l'objecte <code>Storage</code> (tot el <code>localStorage</code> o tot el <code>sessionStorage</code>).
<code>key(index)</code>	Retorna la clau de l'índex especificat (útil per iterar sobre totes les dades).

Exemples d'ús

Emmagatzemar dades simples (Strings)

Totes les dades s'emmagatzemen com a strings.

JavaScript

```
// Utilitzem localStorage
localStorage.setItem('usuari_nom', 'Jordi');
localStorage.setItem('data_login', '2025-09-30');

// Recuperar un valor
const nom = localStorage.getItem('usuari_nom'); // Resultat: "Jordi"

// Eliminar un element
localStorage.removeItem('data_login');
```

```
// Comprovar la mida
console.log(localStorage.length); // Resultat: 1 (només queda 'usuari_nom')
```

2. Emmagatzemar objectes o arrays (Necessita Serialització)

Com que `Storage` només accepta strings, cal convertir els objectes JavaScript a format JSON (string) abans de guardar-los i des-serialitzar-los quan es recuperen.

JavaScript

```
const configuracio = {
  tema: 'fosc',
  notificaciones: true,
  idioma: 'ca'
};

// GUARDAR (Serialització a JSON string)
localStorage.setItem('config_usuari', JSON.stringify(configuracio));

// RECUPERAR (Des-serialització a Objecte JavaScript)
const configGuardadaJSON = localStorage.getItem('config_usuari');

if (configGuardadaJSON) {
  const configObjecte = JSON.parse(configGuardadaJSON);
  console.log(configObjecte.tema); // Resultat: "fosc"
}
```

3. Iterar sobre l'emmagatzematge

Pots recórrer tots els elements utilitzant la propietat `length` i el mètode `key()`.

JavaScript

```
console.log("Elements de la sessió actual:");

for (let i = 0; i < sessionStorage.length; i++) {
  const clau = sessionStorage.key(i);
  const valor = sessionStorage.getItem(clau);
  console.log(`Clau: ${clau}, Valor: ${valor}`);
}

// Cas d'ús: Auditar o depurar el contingut emmagatzemat.
```

Neteja de l'emmagatzematge

S'utilitza per esborrar totes les dades emmagatzemades al domini actual.

JavaScript

```
// ATENCIÓ: Esborra TOT el localStorage
function tancarSessio() {
    localStorage.clear();
    sessionStorage.clear();
    console.log("Tots els elements de storage eliminats.");
    // Redirigir a la pàgina de login o inici
    // location.href = '/login';
}
```

Seguretat i Límits

- **Límits de mida:** La majoria dels navegadors permeten emmagatzemar entre **5 MB i 10 MB** de dades per domini. Si se supera el límit, es llança una excepció.
- **Seguretat (XSS):** Les dades emmagatzemades a `localStorage` **no són segures contra atacs Cross-Site Scripting (XSS)**. Qualsevol script maliciós injectat a la pàgina pot llegir-les. Per això, no s'hauria d'emmagatzemar informació altament sensible com ara contrasenyes o tokens d'autenticació de llarga durada.
- **Àmbit (Scope):** `localStorage` i `sessionStorage` estan restringits al **mateix origen** (protocol, host, i port). Els scripts d'un domini no poden accedir a l'emmagatzematge d'un altre.
- **Sincronia:** Les operacions són síncrones i poden bloquejar el fil d'execució principal (main thread) si es llegeixen o escriuen grans quantitats de dades. Per a dades molt grans o complexes, es recomana utilitzar **IndexedDB**.

IndexedDB (BBDD al navegador)

El Web Storage (com `localStorage` i `sessionStorage`) és excel·lent per a dades clau-valor petites i simples. Tanmateix, per a l'emmagatzematge de dades estructurades, grans volums d'informació, o la necessitat de consultar i indexar registres, JavaScript ofereix una opció molt més robusta: **IndexedDB**.

IndexedDB és una **API asíncrona** de JavaScript per a un emmagatzematge de dades a gran escala i altament estructurat al costat del client. És, essencialment, un sistema de base de dades NoSQL transaccional que resideix al navegador de l'usuari.

Característiques

1. **Emmagatzematge asíncron:** Totes les operacions són asíncrones i basades en esdeveniments (o promeses si s'utilitzen llibreries o wrappers moderns). Això significa que **no bloquegen el fil d'execució principal** de JavaScript (a diferència de `localStorage`), la qual cosa és crucial per mantenir la fluïdesa de la interfície d'usuari.
2. **NoSQL (Orientada a Objectes):** Emmagatzema objectes JavaScript de manera nativa, utilitzant **claus** per a la indexació i la recuperació. No treballa amb taules i SQL, sinó amb **magatzems d'objectes** (Object Stores).
3. **Transaccional:** Totes les modificacions es realitzen dins de **transaccions**. Això garanteix la integritat de les dades: si una part de l'operació falla, tota la transacció es desfà.
4. **Capacitat:** Els límits d'emmagatzematge solen ser molt més alts que els de `localStorage` (sovint entre el 50% i el 80% de l'espai lliure del disc, amb un mínim garantit de 250 MB o més), i és l'emmagatzematge preferit per a les **Progressive Web Apps (PWA)**.

Components

IndexedDB es basa en conceptes que reflecteixen les bases de dades tradicionals:

Component	Concepte	Funció
<code>Database</code>	La base de dades pròpiament dita (l'equivalent al fitxer de BD).	Es defineix amb un nom (p. ex., 'LaMevaAppDB') i una versió.
<code>Object Store</code>	La unitat principal d'emmagatzematge (l'equivalent a una taula).	Conté els registres (objectes).
<code>Index</code>	Un punter a una propietat específica dels objectes.	Permet la cerca i la recuperació ràpida de dades basades en aquesta propietat (no només per la clau primària).
<code>Transaction</code>	Un embolcall que garanteix que un conjunt d'operacions s'executen completament o no s'executen en absolut.	Pot ser de lectura (<code>readonly</code>) o d'escriptura (<code>readwrite</code>).

Mètodes i operacions

L'API IndexedDB és complexa i es basa principalment en mètodes de l'objecte `IDBRequest` (per gestionar les sol·licituds) i l'objecte `IDBTransaction`.

Operació	Mètode / Propietat Clau	Descripció
Obrir BD	<code>indexedDB.open(nom, versió)</code>	Inicia la connexió a la base de dades. Requereix gestionar

Operació	Mètode / Propietat Clau	Descripció
		l'esdeveniment <code>onupgradeneeded</code> per crear o modificar els Object Stores.
Afegir	<code>objectStore.add(objecte, clau)</code>	Afegeix un nou objecte al magatzem.
Recuperar	<code>objectStore.get(clau)</code>	Recupera un objecte pel seu identificador de clau.
Actualitzar	<code>objectStore.put(objecte, clau)</code>	Actualitza un objecte existent o l'afegeix si no existeix.
Esborrar	<code>objectStore.delete(clau)</code>	Elimina un registre pel seu identificador de clau.
Cursor	<code>objectStore.openCursor()</code>	Permet iterar eficientment sobre grans conjunts de dades o utilitzar índexs per a la cerca avançada.

Casos d'ús

A causa de la seva complexitat, IndexedDB s'utilitza principalment quan `localStorage` no és suficient:

1. **Dades Offline Extenses:** Emmagatzematge del cos de missatges o el contingut d'articles sencers per llegir sense connexió.
2. **Cerca ràpida (Indexació):** Aplicacions que necessiten cercar ràpidament registres grans (p. ex., productes en un catàleg o contactes en una llibreta).
3. **Emmagatzematge d'arxius grans:** Capacitat per emmagatzemar objectes binaris grans (`Blob`s), cosa que permet guardar imatges o fitxers de so.
4. **Frameworks i llibreries:** Els frameworks de frontend moderns (com Angular, React) o llibreries de dades offline (com PouchDB) solen utilitzar IndexedDB per sota de la superfície.

Exemple de recuperació de dades

L'API nativa és molt detallada, però conceptualment es veu així:

JavaScript

```
// 1. Obtenir la connexió a la base de dades
const request = indexedDB.open('CatàlegProductes', 1);

request.onerror = function(event) {
  console.error("Error en obrir la base de dades");
};

request.onsuccess = function(event) {
  const db = event.target.result;

  // 2. Iniciar una transacció de lectura
```

```
const transaccio = db.transaction(['productes'], 'readonly');
const store = transaccio.objectStore('productes');

// 3. Demanar la recuperació d'una clau
const getRequest = store.get(123);

getRequest.onsuccess = function(event) {
  console.log("Producte recuperat:", event.target.result);
};

};

// Aquest procés és asíncron i no bloqueja la UI.
```

Exemple complet IndexedDB (catàleg llibres)

```
// Nom i versió de la base de dades
const DB_NAME = 'CatalegLlibresDB';
const DB_VERSION = 1;
const STORE_NAME = 'llibres'; // El nostre magatzem d'objectes (la "taula")

let db; // Variable global per a la connexió a la base de dades

// --- 1. OBRIR I CREAR LA BASE DE DADES (Si no existeix) ---

function obrirDB() {
  return new Promise((resolve, reject) => {
    // Sol·licita obrir la base de dades
    const request = indexedDB.open(DB_NAME, DB_VERSION);

    // Aquest esdeveniment es dispara si la versió de la DB no existeix o és inferior
    request.onupgradeneeded = (event) => {
      db = event.target.result;
      console.log(`Creant o actualitzant magatzem d'objectes: ${STORE_NAME}`);

      // Creem l'Object Store (la "taula")
      // Definim 'isbn' com a clau primària (keyPath)
      if (!db.objectStoreNames.contains(STORE_NAME)) {
        db.createObjectStore(STORE_NAME, { keyPath: 'isbn' });
        // També podríem crear un índex per a cerques per 'autor', per exemple
        // store.createIndex('per_autor', 'autor', { unique: false });
      }
    };

    request.onsuccess = (event) => {
      db = event.target.result;
    };
  });
}
```

```

        console.log("Base de dades oberta amb èxit!");
        resolve();
    };

    request.onerror = (event) => {
        console.error("Error en obrir la base de dades:", event.target.errorCode);
        reject(new Error("No es pot obrir la DB"));
    };
});
}

// --- 2. Afegir ELEMENTS (CRUD: Create) ---

function afegirLlibrellibre(libre) {
    if (!db) {
        console.error("La base de dades no està connectada.");
        return;
    }

    // 1. Iniciar una transacció de lectura/escriptura
    const transaccio = db.transaction([STORE_NAME], 'readwrite');
    const store = transaccio.objectStore(STORE_NAME);

    // 2. Afegir l'objecte
    const request = store.add(libre);

    request.onsuccess = () => {
        console.log(`Llibre afegit amb ISBN ${libre.isbn}`);
    };

    request.onerror = (event) => {
        // Això passa sovint si ja existeix un element amb la mateixa clau (ISBN)
        console.error("Error en afegir el llibre:", event.target.error);
    };
}

// --- 3. CONSULTAR UN ELEMENT PER CLAU (CRUD: Read) ---

function consultarLlibre(isbn) {
    if (!db) {
        console.error("La base de dades no està connectada.");
        return;
    }

    // 1. Iniciar una transacció de només lectura
    const transaccio = db.transaction([STORE_NAME], 'readonly');
    const store = transaccio.objectStore(STORE_NAME);

    // 2. Recuperar l'element per la seva clau primària (ISBN)
    const request = store.get(isbn);

```

```

    request.onsuccess = (event) => {
        const llibre = event.target.result;
        if (llibre) {
            console.log(`\n--- Llibre trobat (ISBN ${isbn}) ---`);
            console.log(llibre);
        } else {
            console.log(`\nNo s'ha trobat cap llibre amb ISBN ${isbn}`);
        }
    };

    request.onerror = (event) => {
        console.error("Error en consultar el llibre:", event.target.error);
    };
}

// --- 4. ELIMINAR UN ELEMENT PER CLAU (CRUD: Delete) ---

function eliminarLlibre(isbn) {
    if (!db) {
        console.error("La base de dades no està connectada.");
        return;
    }

    // 1. Iniciar una transacció de lectura/escriptura
    const transaccio = db.transaction([STORE_NAME], 'readwrite');
    const store = transaccio.objectStore(STORE_NAME);

    // 2. Eliminar l'element per la seva clau primària (ISBN)
    const request = store.delete(isbn);

    request.onsuccess = () => {
        console.log(`\nLlibre amb ISBN ${isbn} eliminat correctament.`);
    };

    request.onerror = (event) => {
        console.error("Error en eliminar el llibre:", event.target.error);
    };
}

// --- FLUX D'EXECUCIÓ (Async/Await per simplificar) ---

const llibre1 = { isbn: '978-8426102604', titol: 'El petit príncep', autor: 'Antoine de Saint-Exupéry', any: 1943 };
const llibre2 = { isbn: '978-8429749103', titol: 'Tirant lo Blanc', autor: 'Joanot Martorell', any: 1490 };

// Esperem a que la base de dades estigui oberta abans de fer operacions
obrirDB()

```

```
.then(() => {  
    // Afegeix el llibre 1  
    afegirLlibrellibre1);  
  
    // Afegeix el llibre 2  
    afegirLlibrellibre2);  
  
    // Consulta el llibre 1 després d'un petit retard (per assegurar-nos que s'ha  
escrit)  
    setTimeout(() => consultarLlibre('978-8426102604'), 500);  
  
    // Intentem consultar un ISBN que no existeix  
    setTimeout(() => consultarLlibre('111-2223334445'), 1000);  
  
    // Elimina el llibre 2  
    setTimeout(() => eliminarLlibre('978-8429749103'), 1500);  
  
    // Comprovem que s'ha eliminat  
    setTimeout(() => consultarLlibre('978-8429749103'), 2000);  
})  
.catch((error) => {  
    console.error("No es pot continuar. La DB no s'ha pogut inicialitzar.");  
});
```