# UT8. Almacenamiento de datos en el lado cliente e integración de componentes.

1)localStorage

2)IndexedDB

3)Geolocalización.

# localStorage - definición

**`localStorage`** es una tecnología que permite guardar datos en el navegador de forma persistente, por lo que podrán ser utilizados en la próxima ejecución de la página web. Se utiliza para guardar preferencias del usuario (login, clave, preferencias de uso de la web, ultimas datos utilizados, etc).

Las páginas web solo podrán acceder a los datos guardados en `localStorage` procedentes del mismo sitio web (no válido para subdominios)

Guarda los datos en cadenas de texto, si se desea guardar otro tipo de datos como objetos `JSON`, se han de transformar a string previamente.

Si se desea guardar datos, como imágenes, vídeos, audios, etc, o cualquier dato binario se debe utilizar una base de datos de cliente o de servidor. Una base de datos de cliente sería `indexedDB`, por ejemplo.

Por lo que, `localStorage` es similar a `document.cookie` , con las siguientes diferencias que lo hacen más flexible y eficiente:

- `document.cookie` solo permite un tamaño máximo de cada cookie de 4KB. Con
- este límite depende de cada navegador y de la configuración del usuario.
- `document.cookie` envía todas las cookies de un sitio web en todas las peticiones HTTP a dicho sitio web, lo que es a veces innecesario, y si las cookies ocupan mucho espacio ralentiza la comunicación entre cliente y servidor.

# localStorage - disponibilidad

Para utilizar `localStorage` hay que asegurarse primero que el navegador lo soporta, aunque lo soportan todos los navegadores modernos, hay que hacerlo ya que no sabemos si nuestra aplicación va a ser ejecutada en un navegador antiguo que no lo soporte.

```
if (typeof(Storage)!=="undefined"){

    info.innerHTML+="Tu navegador soporta localStorage"

} else {

    info.innerHTML+="Tu navegador no soporta localStorage"

}
```

**Ejemplo: comprueba que tus navegadores soporte `localStorgae`**

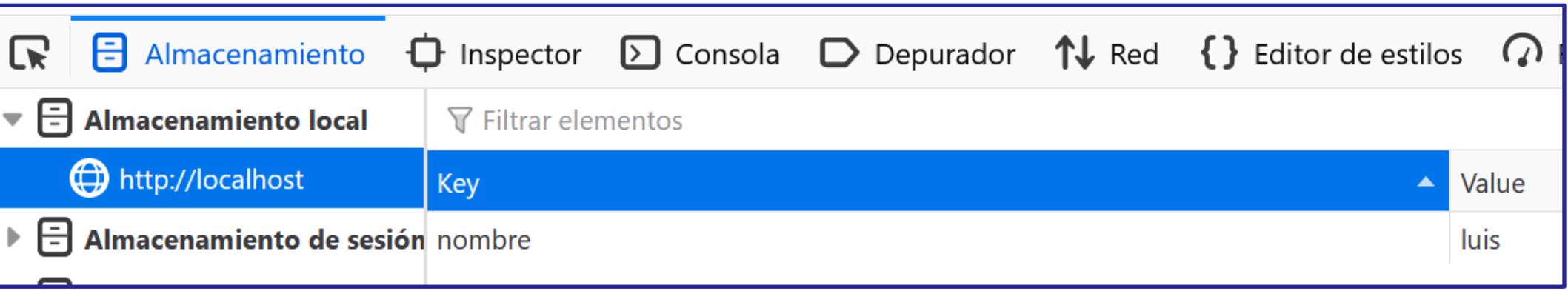
# localStorage – métodos y uso

**`setItem()`:** Para guardar un dato con `localStorage` debes asignarle un nombre que se llama “clave” y guardar en esta clave el valor deseado, de la siguiente forma:

```
localStorage.setItem("nombre","luis");
```

Si ya existiera la clave `nombre` se sobreescribiría su valor.

Se ha asignado a la clave `nombre` el valor `luis`, y podemos comprobar que ha funcionado correctamente utilizando las herramientas del desarrollador del navegador, concretamente en la sección Almacenamiento -> Almacenamiento local




**`getItem()`:** Para leer un dato de `localStorage` se haría de la siguiente forma, recuerda que los valores recuperados siempre van a ser `string`:

```
var valor=localStorage.getItem("nombre");
```

**Ejemplo: crea un pequeño formulario con 2 inputs login y clave, con 2 botones. Uno llamado `Guardar datos` que guardará los datos introducidos en `localStorage`, y otro botón `Ver datos` que mostrará dichos valores.**

# localStorage – métodos y uso

**`removeItem()`:** Para borrar un dato de `localStorage` se procede de la siguiente forma:

```
localStorage.removeItem("nombre");
```

Se borrará la clave `nombre` de `localStorage`. Si se consultara dicha `clave` daría `null`.`removeItem()` no devuelve nada.

**`clear()`:** Borra todos los datos.

**Ejemplo:** añade al formulario anterior otro botón llamado `Borrar datos` que borrará de `localStorage`, la clave `nombre`.

**Ejercicio:**

Crea un formulario con los siguientes elementos:

- Un select para elegir entre 4 tipos de letra que te gusten.
- Un input type color para elegir el color del texto.
- Un textarea.

Cuando el usuario elija un color o un tipo de letra se va a:

- Guardar en el navegador utilizando localStorage
- Se va a aplicar al textarea

Cuando la aplicación se cargue se va a consultar el color y el tipo de letra en localStorage.

- Si existen alguno de estos datos: se van a aplicar al textarea.
- Si no existe cualquiera de ellos: se va a aplicar al textarea los valores por defecto respectivos y se va a almacenar en `localStorage`

**Ejercicio:** Adapta el ejercicio guardando los datos en `localStorage` con JSON.

# sessionStorage vs localStorage

**sessionStorage** tiene los mismos métodos que `localStorage`.

La diferencia entre ellos es que los datos guardados en `sessionStorage` solo van a permanecer mientras esté abierta la aplicación web, cuando cierre van a desaparecer.

Entonces, ¿cuál es la utilidad de `sessionStorage`?

1. Que guardará los datos entre las distintas páginas de una aplicación.
2. Podemos tener la aplicación abierta varias veces y cada una de las sesiones tendrá sus propios datos guardados de forma independiente. Esto no ocurre en `localStorage`, con el que los datos serán compartidos entre todas las sesiones.

Si una aplicación no permite guardar datos en el navegador del cliente lo mejor es utilizar `sessionStorage` para guardar datos temporales entre páginas de la aplicación, ya que todos los datos se eliminarán del navegador cuando la aplicación termine.

**Ejemplo:** haz la prueba de ello con cualquiera de los ejercicios propuestos para `localStorage`

# IndexedDB - definición

**IndexedDB es una base de datos en el lado del cliente, que permite almacenar una gran cantidad de datos.**

La principal ventaja de usar IndexedDB es que los datos se almacenan en el navegador del usuario, lo que significa que las aplicaciones web pueden acceder a los datos incluso cuando el usuario no está conectado a Internet.

Esto es útil por ejemplo para aplicaciones web que gestionan datos como correos electrónicos, documentos, imágenes, etc.

**Es una base de datos NoSQL** (no solo SQL)**:**

- La información no se guarda en estructuras fijas como tablas (filas y columnas), sino en "Object Store", que contienen objetos JSON.
- Su estructura no está normalizada.
- Pueden soportar lenguaje SQL.

**Sus operaciones son asíncronas.** No se sabe cuándo van a terminar, en lugar de esperar a que terminen el sistema nos avisará, mediante eventos, de cuando ha acabado la operación solicitada a la Base de datos. Cuando se hayan producido los eventos correspondientes podremos trabajar con los resultados.

**Almacena objetos JSON, parejas clave: valor.**

# IndexedDB - conceptos básicos

**Se pueden crear tantas Bases de datos (IDBDatabase) como se quiera, identificadas por un nombre y una versión.**

**La información se guarda en contenedores llamados almacén de objetos (IDBObjectStore),** es un conjunto de datos, lo que equivaldría a una tabla en una base de datos relacional. La clave podrá ser una propiedad de ese objeto.

**Permite crear índices (IDBIndex) para mejorar las búsquedas.**

**Las lecturas y escrituras se realizan mediante transacciones (IDBTransaction):** si una operación es compleja y está compuesta por varias operaciones, o bien se ejecutan todas las operaciones, o ninguna de ellas, es decir, si una de las operaciones falla se cancelan/deshacen el resto.

**Las consultas (IDBRequest) lanzan el evento success** para indicar que ha sido realizada. Las búsquedas se pueden realizar a partir de propiedades de los objetos.

**Permite el uso de cursores (IDBCursor)** para recorrer un conjunto de objetos devueltos por la base de datos.

# IndexedDB - conceptos básicos

**El funcionamiento es el siguiente:**

1. **Se crea una base de datos o se abre una ya existente**, con el método open()
2. **Crear los almacenes de objetos**. Cada almacén tiene un nombre y una clave única.
3. **Agregar datos**: Se crea el objeto que se va a añadir al almacén de objetos correspondiente, y se llama al método `add()` de dicho almacén de objetos.
4. **Consultar datos**: se utiliza el método `get()` del almacén de objetos que se desea consultar introduciendo la clave única del dato consultado.
5. **Actualizar datos**: se utiliza el método `put()` del almacén de objetos correspondiente para actualizar un objeto existente.
6. **Eliminar datos**: se utiliza el método `delete()` del almacén de objetos correspondiente para eliminar un objeto existente.
7. **Utilización de transacciones**: es conveniente utilizarlas para garantizar la consistencia de los datos. Las transacciones permiten que varias operaciones de base de datos se agrupen en una sola unidad lógica de trabajo, de forma que o se realizan todas las operaciones de las que se compone o no se realiza ninguna.

# IndexedDB – abrir una BD

**Para abrir una BD haremos:**

```
let petición=indexedDB.open(“nombreBD”,”versión”)
```

**Si existe una base de datos con el nombre indicado y la versión indicada se producirá el evento success, y dentro de él tendremos una referencia a la BD para operar con ella.**

```
peticion.addEventListener(“success", function (e) {
        bd=e.target.result;
        // operaciones con la referencia bd        })
```

**Si no existe esa Base de datos, o existe con una versión inferior se producirán estos eventos y en este orden:**

- **`upgradeneeded`:** que indica que ha habido una actualización o se ha creado por 1ª vez.
- **`success`:** que se ha abierto con éxito. Este evento también facilitará un referencia a la BD.

```
peticion.addEventListener("upgradeneeded", function (e) {
    bd=e.target.result;
    // operaciones con la referencia bd
})
```

**Ejemplo:** crea un botón llamado “Crear/abrir BD” que cree una base de datos nueva si no existe o que abra la BD si existe. La versión de la BD se obtendrá de la clase “`version`” de `localStorage`. Si no existe dicha clave se utilizará la versión “`1`”. Captura los eventos `upgradeneeded` y `success`. Pulsa el botón 1 vez y observa cuál de los eventos se producen. Pulsa de nuevo.

# IndexedDB – abrir una BD

**Ejemplo:**

Crea un botón llamado “Actualizar BD”. Este botón requiere que la BD ya exista, se abrirá nuevamente pero con una versión superior. Obtén la versión de `localStorage`, increméntala y guárdala la nueva versión en `localStorage`. Captura los eventos `upgradeneeded` y `success`. Observa cuál de ellos se ejecuta.

**IMPORTANTE:** Cuando se crea una Base de datos, por primera vez, normalmente en el momento de crearla se construye también su estructura inicial, es decir se crean los almacenes de objetos de la misma. Y se aprovecha directamente el evento “`upgradeneeded`” que se produce al crear la Base de datos para crear los almacenes de la Base de datos. Por lo que no es necesario tras crear la BD, cerrarla y volverla a abrir. Esto solo será necesario si la función manejadora del evento `upgradeneeded` ha concluido, como ocurre en este último ejemplo.

# IndexedDB – cerrar y borrar una BD

**Para cerrar una BD haremos:**

```
bd.close("nombreBD") //siendo bd una referencia a la Base de datos
```

Al cerrar la Base de datos ya no se podrán realizar operaciones sobre ella, y todas las transacciones, peticiones, y referencias a almacenes de objetos se cancelarán.

`bd` no se convierte en `null` pero no funciona.

**Para borrar una BD haremos (primero cerraremos la BD):**

```
bd.close("nombreBD")

var peticion = indexedDB.deleteDatabase("nombreBD");

peticion.addEventListener("success",function() {
      console.log("La BD ha sido eliminada exitosamente");
})

request.addEventListener("error",function() {
      console.log("No se pudo eliminar la BD");
})
```

**Ejemplo:** añade a la aplicación un botón llamado "Cerrar BD" que compruebe que la referencia a la BD que se maneja (en este caso `bd`) existe, y si es así que cierre la base de datos. Ejecuta el programa abriendo la BD y pulsando al botón "Cerrar BD"

# IndexedDB – crear un almacén de objetos

**Para crear un almacén de objetos en la Base de datos haremos:**

```
let almacenAlumnos = bd.createObjectStore("alumnos", { autoIncrement: true })
```

Con el método `createObjectStore` de la base de datos, `bd`, se ha creado un almacén de objetos llamado "`alumnos`", cuya clave será un índice que se autoincrementa automáticamente.

**Para poder crear un almacén nuevo** ha de producirse el evento `upgradeneeded`, y dentro de la función manejadora de este evento crearse el almacén. Por lo que si ya ha terminado la función manejadora del evento `upgradeneeded` tras su creación o última actualización, ésta ha de cambiar de versión. Por tanto, hay que cerrarla, abrirla de nuevo con una versión superior y crear el almacén de objetos nuevo en la función manejadora del evento `upgradeneeded`. **Si la base de datos está recién creada no hay que cerrarla y volverla a abrir con una versión superior, sino aprovechar el evento "`upgradeneeded`" que se produce al crear la BD por 1ª vez para crear el almacén.**

```
bd.close();
let peticion=indexedDB.open("BD","2")
peticion.addEventListener("upgradeneeded",function (e){
     info.innerHTML+="BD Actualizado para crear almacen<br>"
     bd=e.target.result
    let almacen= bd.createObjectStore("alumnos", { autoIncrement: true })
     almacen.transaction.addEventListener("complete",()=>{
                console.log("Almacen creado");
    })
     almacen.transaction.addEventListener("error",()=>{
                console.log("Error en la creación del almacén")
     })
})
```

# `IndexedDB` – crear un almacén de objetos

**Ejercicio:** añade un botón llamado “Crear almacén alumnos” que cree el almacén de alumnos, para ello ha de actualizar antes la BD (ya que la función manejadora del evento “`upgradeneeded`” que se produjo tras su creación ó última actualización ya ha terminado). Recuerda recoger la última versión de `localStorage` (que previamente se ha guardado) e incrementarla en `1`. Esta nueva versión se ha de guardar en `localStorage`

# IndexedDB – borrar un almacén de objetos

**Para poder borrar un almacén** en una base de datos, ésta ha de actualizar su versión, por lo que si la base de datos, hay que cerrarla, abrirla de nuevo con una versión superior y borrar el almacén de objetos.

**Una vez que tenemos la referencia a la BD actualizada, antes de borrar el almacén de objetos comprobaremos que dicho almacén existe con** `bd.objectStoreNames.contains("alumnos")`

**Un almacén de objetos se borra con** `bd.deleteObjectStore("nombreAlmacen")`

```
bd.close();

let peticion=indexedDB.open("BD","2")

peticion.addEventListener("upgradeneeded",function (e){
     bd=e.target.result
    //BORRAR EL ALMACÉN DE DATOS
    if (bd.objectStoreNames.contains("alumnos")){
            info.innerHTML+="Existe el almacén y se va a borrar<br>"
        bd.deleteObjectStore("alumnos")
    }
})
```

**NOTA:** si se borra un almacén para crear otro en su lugar con otra estructura, hay que migrar los datos del antiguo almacén al nuevo.

**Ejercicio:** añade un botón "Borrar almacén alumnos" que borre el almacén de alumnos, para ello se ha de actualizar antes la BD. Recuerda recoger la última versión de `localStorage` (que previamente se ha guardado) e incrementarla en `1`, y guardarla de nuevo en `localStorage`

# IndexedDB – añadir datos en un almacén de objetos

**Para añadir datos en un almacén de objetos de una BD:**

```
let transac=bd.transaction("alumnos","readwrite")

let almacenAlumnos=trans.objectStore("alumnos")
transac.addEventListener("complete",()=>info.innerHTML+="transacción ok")
transac.addEventListener("error",()=>info.innerHTML+="transacción nok")

let registro={"nombre":nombre.value,"edad":edad.value}

let peticAdd=almacenAlumnos.add(registro)
peticAdd.addEventListener("success",()=> console.log("Inserción ok"))
peticAdd.addEventListener("error",()=> console.log("Inserción nok"))
```

- **Con `db.transaction` se crea una transacción**, `transac`, en la que va a estar afectado el almacén de objetos `alumnos` para `lectura/escritura`. Es necesario crear una transacción para cualquier operación de lectura, escritura sobre la base de datos, aunque solo sea una operación que afecta a una única tabla.
- **De esta transacción obtenemos una referencia al almacén** "`alumnos`" con el método `objectStore` de la transacción**.**
- **Controlamos la transacción con los eventos `complete` y `error`**, para manejar el hecho de que vaya bien la transacción y el hecho de que vaya mal la transacción.
- **Por último hacemos inserción del objeto en el almacén de objetos,** con el método `add()`, también se puede utilizar el método `put()` de la misma manera que en este ejemplo se utiliza el `add()`. Controlamos la operación de inserción con los eventos `success` y `error`.

# `IndexedDB` – añadir datos en un almacén de objetos

**Ejemplo:** añade a la aplicación anterior lo siguiente:

- Un formulario con los siguientes inputs "`nombre`", "`edad`" e "`indice`".
- Un botón llamado "Añadir alumno" que añada un nuevo objeto al almacén "`alumnos`" con los datos de los inputs anteriores.

**Ejemplo:** añade a la aplicación anterior lo siguiente:

- Un botón llamado "Crear BD/Crear almacén/Añadir alumno" que cree la BD por primera vez, cree el almacén y añada el alumno incluido en el formulario. Al probar este botón, antes has de pulsar al botón "Borrar BD"

**NOTA IMPORTANTE:** Crea la BD, y cuando se produzca el evento `upgradeneeded` de la orden `open()` de la BD, es decir, en la función manejadora de este evento, crea el almacén, ya que estarás en el proceso de actualización y en él podrás cambiar la estructura de la Base de datos.

**NOTA IMPORTANTE:** Ten en cuenta que todos los métodos de la base de datos `indexedDB` son asíncronos, es decir se lanzan, y continúa la ejecución del programa. Teniendo en cuenta que la orden de crear el almacén es asíncrona, no podremos hacer la inserción de datos en el almacén de objetos hasta que la orden de creación del almacén haya concluido. De lo contrario puede ocurrir que la orden de inserción de datos se ejecute antes de que la operación de creación del almacén donde estás intentando hacer la inserción no haya terminado. Por lo que haremos la inserción de datos en el almacén cuando se produzca el evento `complete`, de la transacción de la creación del almacén, es decir, en la función manejadora de este evento, se añadirá el dato.

# IndexedDB – leer un dato del almacén de objetos

**Para leer un dato del almacén de datos hay que utilizar el campo clave del objeto:**

```
let transac=bd.transaction("alumnos","readonly")

let almacenAlumnos=transac.objectStore("alumnos")
transac.addEventListener("complete",()=>{ console.log("transacción ok")})
transac.addEventListener("error",()=>{ console.log("transacción nok")})

Let ordenLect=almacenAlumnos.get(2)  //pedimos el objeto con clave 2

ordenLect.addEventListener("success",(e)=>{
     let datoLeido=e.target.result;
     if (datoLeido)  console.log(datoLeido)
     else console.log("No existe ningún dato con clave 2")}
ordenLect.addEventListener("error",()=>console.log("error en lectura"))
```

- **Con `db.transaction` se crea una transacción**, `transac,` de solo lectura. Es necesario aunque se trata de una operación de lectura únicamente. **Controlamos si la transacción ha ido bien con sus eventos `complete y error`.**
- **De esta transacción obtenemos una referencia al almacén** "`alumnos`" con el método `objectStore` de la transacción**.**
- **Sobre esa referencia al almacén obtenemos el objeto con clave igual a 2 con el método `get()`.** La clave numérica autoincremental.
- **Si la operación de lectura, lanzada por `get()`, ha ido bien se dispara el evento "`success`", que tiene en `target.result` el dato si existe, y si no, `result`, será `null`. Si la operación de lectura ha ido mal, se lanzará el evento `error`.**

# `IndexedDB` – leer un dato del almacén de objetos

**Ejercicio:** añade a la aplicación un botón llamado “Leer alumno” que devuelva el objeto del almacén “`alumnos`” cuya clave es la introducida en el input “`índice`” del formulario.

# IndexedDB – actualizar un dato del almacén de objetos

**Para actualizar un dato del almacén de datos hay que utilizar el campo clave del objeto:**

```
let transac=bd.transaction("alumnos","readwrite")
let almacenAlumnos=transac.objectStore("alumnos")
transac.addEventListener("complete",()=>console.log("Transacción ok"))
transac.addEventListener("error",()=>console.log("Transacción nok"))
let registro={"nombre":nombre.value,"edad":edad.value}
var petAct=almacenAlumnos.put(registro,parseInt(indice.value))
petAct.addEventListener("success",()=>console.log("Dato actualizado"))
petAct.addEventListener("error",()=>console.log("Error en la actualización")
```

- **Con `db.transaction` se crea una transacción**, `transac`, de escritura y lectura ya que vamos a actualizar datos. **De esta transacción obtenemos una referencia al almacén de objetos** "`alumnos`" con el método `objectStore` de la transacción**. Controlamos si la transacción ha ido bien con los eventos `complete` y `error`.**
- **Sobre esa referencia al almacén ejecutamos el método `put()`, que recibe 2 argumentos:**
  - El segundo es el valor de la clave del objeto que hay que actualizar. ¡Cuidado! Si la clave es un número hay que pasarla a entero. ¡Cuidado!: Si no se utiliza se agregará el objeto a los existentes, pero no sustituirá a ninguno. Si la clave no existe también se agregará el objeto a los existentes.
  - El primero es el objeto que va a ser sustituido por el existente en el almacén de objetos.
- **Si la operación de actualización, lanzada por `put()`, ha ido bien se dispara el evento "`success`", y si no se lanza el evento `error`.**

**Ejercicio:** añade a la aplicación un botón llamado "Actualizar alumno" que actualice el objeto del almacén "`alumnos`" cuya clave es la introducida en el input "`índice`" del formulario. Los nuevos datos del objeto serán los introducidos en los inputs "`nombre`" y "`edad`".

# IndexedDB – borrar un dato del almacén de objetos

**Para borrar un dato del almacén de datos hay que utilizar el campo clave del objeto:**

```
let transac=bd.transaction("alumnos","readwrite")
let almacenAlumnos=transac.objectStore("alumnos")
transac.addEventListener("complete",()=>{ console.log("transacción ok")})
transac.addEventListener("error",()=>{ console.log("transacción NOK")})
let petDel=almacenAlumnos.delete(2)  //pedimos el dato con clave 2
petDel.addEventListener("success",()=>console.log("borrado ok"))
petDel.addEventListener("error",()=>console.log("borrado NOK"))
```

- **Con `db.transaction` se crea una transacción**, `transac`, de lectura/escritura.
- **De esta transacción obtenemos una referencia al almacén** "`alumnos`" con el método `objectStore` de la transacción**. Controlamos si la transacción ha ido bien con sus eventos `complete y error`**
- **Sobre esa referencia al almacén borramos, con el método `delete()` el objeto con clave igual a `2`** que es la clave numérica autoincremental.
- **Si la operación de borrado, lanzada por `delete()`, ha ido bien se dispara el evento "`success`", y si no se lanza el evento `error`.**

**Ejercicio:** añade a la aplicación un botón llamado "Borrar alumno" que borre el objeto del almacén "`alumnos`" cuya clave es la introducida en el input "`índice`" del formulario.

# IndexedDB – leer todos los datos de un almacén de objetos

**Para leer todos los objetos de un almacén de objetos, que se llaman `values`, se puede hacer de 2 formas:**

1. Creando un cursor con el método `openCursor()` del almacén de objetos. Este método te va a devolver la clave de cada objeto y los valores (objetos)
2. Utilizando el método `getAll()` del almacén de objetos. Este método solo te va a devolver los valores, es decir, los objetos, no la clave. Ésta es la diferencia con el método `openCursor()` Todos los objetos son devueltos en un array.

---

**PRIMERA FORMA** - `getAll()`

Si no hubiera registros en el almacén devolverá un array vacío (con `length=0`)

```
let transac=bd.transaction("alumnos","readonly")

let almacenAlumnos=transac.objectStore("alumnos")
transac.addEventListener("complete",()=>{ console.log("transacción ok")})
transac.addEventListener("error",()=>{ console.log("transacción NOK")})

let petGetAll=almacenAlumnos.getAll()
petGetAll.addEventListener("error",()=>console.log("GetAll NOK"))
petGetAll.addEventListener("success",(e)=>{
  let valores=e.target.result;
  if (valores)
    valores.forEach(item=> // se trabaja con cada objeto del almacén)
  if (valores.length==0) console.log("No hay valores en el almacén") Con
})
```

# IndexedDB – leer todos los datos de un almacén de objetos

**PRIMERA FORMA** - `getAll()`

- **Con db.transaction se crea una transacción, `transac`, de lectura.**
- **De esta transacción obtenemos una referencia al almacén** "`alumnos`" con el método `objectStore` de la transacción**. Controlamos si la transacción ha ido bien con sus eventos `complete y error`**
- **Sobre esa referencia al almacén obtenemos todos sus objetos, con el método `getAll()`**
- **Si la operación lanzada por `getAll()`, ha ido bien se dispara el evento "`success`", y si no se lanza el evento `error`.**
- **Si se lanza el evento "`success`", podemos obtener de la propiedad `target.result` del evento todos los objetos del almacén en un array. Si no hubiera ningún registro en el almacén el array se devolverá igualmente pero vacío.**

**Ejercicio:** añade a la aplicación un botón llamado "Recorrer almacén alumnos con getAll" que muestre todos los objetos del almacén.

# IndexedDB – leer todos los datos de un almacén de objetos

**SEGUNDA FORMA** - openCursor()

```
let transac=bd.transaction("alumnos","readonly")

let almacenAlumnos=transac.objectStore("alumnos")
transac.addEventListener("complete",()=>{ console.log("transacción ok")})
transac.addEventListener("error",()=>{ console.log("transacción NOK")})

let petCursor=almacenAlumnos.openCursor()
petGetAll.addEventListener("error",()=>console.log("OpenCursor NOK"))

petGetAll.addEventListener("success",(e)=>{
    let cursor=e.target.result;
    if (cursor){
          console.log("Clave: "+cursor.key)
          console.log("Objeto: "+JSON.stringify(cursor.value))
          cursor.continue()
    }
})
```

# IndexedDB – leer todos los datos de un almacén de objetos

**SEGUNDA FORMA** - `openCursor()`

- **Sobre la referencia al almacén creamos un cursor** que recorrerá todos los objetos con el método **`openCursor()`**
- **Si la operación lanzada por `openCursor()`, ha ido bien se dispara el evento "`success`", y si no se lanza el evento `error`.**
- **Si se lanza el evento "`success`", podemos obtener el cursor de la propiedad `target.result` del evento.**
- **En el propio cursor obtenido están los datos del objeto,** que son `key` (la clave) y `value` (el propio objeto).
- **Este cursor tiene un método `continue()`,** que si va bien lanzará un evento `success` en la petición ejecutándose de nuevo la función manejadora de dicho evento repitiéndose el proceso hasta llegar al final de los objetos del almacén.

**Ejercicio:** añade a la aplicación un botón llamado "Recorrer almacén alumnos con cursor" que muestre todos los objetos del almacén.

# Geolocalización

**Se pueden utilizar diferentes APIs de geolocalización. Nosotros veremos las siguientes:**

- IPStack.
- Geolocation API.
- API de Google Maps.

## IPStack

**Sitio web que mediante peticiones HTTP nos da en tiempo real y en formato JSON, datos de una IP**.

Si conocemos la IP del navegador cliente que ejecuta nuestra aplicación web, podemos saber datos sobre él como: ubicación (latitud, longitud), continente, país, región, ciudad, código postal, idioma, huso horario, bandera, etc... De esta forma podemos ajustar nuestra aplicación a los intereses del usuario.

**Para utilizar IPStack es necesario registrarse previamente** a través de su página web **`https://ipstack.com/`** y obtener así una clave de acceso. Puedes registrarte para obtener una clave o utilizar la siguiente: **`64f577375e0e46d08e984187f795ee6d`**

**Para obtener la información sobre una IP concreta haremos una petición HTTP a:**

```
Let access_key="64f577375e0e46d08e984187f795ee6d"
Let ip="2.2.2.2"   //un ejemplo de ip
http://api.ipstack.com/' + ip + '?access_key=' + access_key;  //JSON
```

**Ejercicio 1:** Crea una página web con: un `input` llamado "`IP`" para que el usuario introduzca una IP, y un botón "`Mostrar`" que mostrará los datos de dicha IP facilitados por IPStack: latitud, longitud, ciudad, región, país, idioma (en inglés) .

# Geolocalización – API Geolocation

**La API Geolocation es una interfaz de programación de aplicaciones (API) que permite a los desarrolladores de software obtener la ubicación geográfica del navegador del usuario.** Esta API se utiliza para proporcionar servicios basados en la ubicación, como direcciones, mapas y recomendaciones para mejorar la experiencia del usuario.

**La API Geolocation utiliza diferentes métodos para determinar la ubicación del dispositivo, incluyendo la dirección IP, la señal de Wi-Fi, la torre de telefonía móvil y el GPS.** Dependiendo del método utilizado, la precisión de la ubicación puede variar desde unos pocos metros hasta varios kilómetros.

**Para utilizar la API Geolocation,** no es necesario incluir ningún framework, ni obtener ninguna clave de uso, ya que lo tienen incluido la gran mayoría de navegadores.

**Para que funcione la API Geolocation,** es necesario:

- Acceso a Internet: ya que la API hará uso de servicios web de internet.
- Navegador compatible: la mayoría de los navegadores modernos lo son.
- Permiso del usuario: el usuario debe dar permiso para compartir su ubicación.
- HTTPS: las peticiones a los servicios de internet que se utilizan se hacen mediante el protocolo HTTPS.
- Una clave de acceso: a veces es necesaria una clave de acceso. No para el uso que vamos a dar nosotros.

**USO:**

```
navigator.geolocation.getCurrentPosition(fPosition,fError)
```

**NOTA MUY IMPORTANTE:** Este método es asíncrono, por lo que se seguirá ejecutando la instrucción siguiente a la misma sin esperar a que este método acabe.

`fPosition`: función que recibirá en un argumento un objeto con las coordenadas la ubicación.

`fError`: función que recibirá como argumento un objeto con la información sobre el error que se haya producido en caso de que la petición haya ido mal.

# Geolocalización – API Geolocation - uso

```
function fPosition(posicion) {  //función manejadora de la ubicación
    // posicion es el objeto recibido con las coordenadas de la ubicación
    // posicion.coords.longitude-> es la longitud de la ubicación.
    // posicion.coords.latitude-> es la latitud de la ubicación.
}

function fError (error){  //función manejadora del error, en su caso

  // error es el objeto recibido en caso de que la petición haya ido mal.
  // error.message guarda el mensaje informativo del error ocurrido
  // error.code guarda el código del error ocurrido, que puede ser:
  // error.PERMISSION_DENIED (1)-> El usuario ha denegado la peticion de geolocalización
  // error.POSITION_UNAVAILABLE (2)-> La informacion de localizacion no esta disponible
  // error.TIMEOUT (3)-> Se ha terminado el tiempo para aceptar la peticion
  // error.UNKNOWN_ERROR-> Error desconocido

}
```

- `error.PERMISSION_DENIED`, `error.POSITION_UNAVAILABLE`, etc son constantes definidas dentro del objeto `error`.

**Ejercicio 2:** Crea una página web con un botón "`Mostrar`" que mostrará la longitud y la latitud de las coordenadas de tu ubicación. Dicha aplicación ha de controlar los errores que se puedan producir en la petición de la ubicación. Haz la prueba del control de errores denegando a la aplicación el acceso a tu ubicación.

# Geolocalización – API Google Maps

En la URL `https://developers.google.com/maps/documentation` puedes encontrar todas las funcionalidades que ofrece esta API.

**Para utilizar esta API es necesario registrarse previamente** a través de su página web y obtener una clave de acceso. Puedes registrarte para obtener una clave o utilizar la siguiente: **`AIzaSyBDaeWicvigtP9xPv919E-RNoxfvC-Hqik`**

**Incluye lo siguiente en tu página web** para poder utilizar la API con esta clave mencionada**:**

```
<script
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBDaeWicvigtP9xPv91
9E-RNoxfvC-Hqik"></script>
```

## ¿Cómo crear un mapa?

```
<p id=“mapa” style="height: 400px;width: 100%"></p>
Let coord={lat: 1, lng: 3}   //han de ser números
var map = new google.maps.Map(mapa, {
        zoom: 10,
        center: coord
});
```

- **Se crea un elemento HTML para cargar en él el mapa con los estilos `height` y `width` definidos.**
- **Crea un mapa y lo dibuja en el elemento HTML anterior,** con un `zoom` de `10` (para ver a nivel de región), y centrado en las coordenadas `coord (center).coord` será un objeto `JSON` con 2 propiedades: `lat` y `lng` que guardarán la latitud y la longitud de las coordenadas en `entero` o `float`.
- Además **devuelve una referencia al mapa en la variable `map`**.

# Geolocalización – API Google Maps

## ¿Cómo marcar una ubicación en un mapa?

```
var marker = new google.maps.Marker({
        position: coord,
        map: map
});
```

- **Se crea un marcador indicando la posición donde hay que poner el marcador y en qué mapa.**

**Ejercicio 3:** Crea una página web con: dos `inputs` llamados "`Longitud`" y "`Latitud`" para que el usuario introduzca las coordenadas de una ubicación, y un botón "`Mostrar`" que mostrará dicha ubicación en un mapa.

**Ejercicio 4:** Modifica el ejercicio de IPStack. Añade un botón llamado "`Mostrar en mapa`" que al pulsarlo marque en el mapa la ubicación de dicha IP.

**Ejercicio 5:** Modifica el ejercicio de API Geolocation. Añade un botón llamado "`Mostrar en mapa`" que al pulsarlo marque en el mapa tu ubicación.

## ¿Cómo calcular distancias entre 2 ubicaciones?

**Incluye lo siguiente en tu página web** para poder utilizar la API con esta clave mencionada (Igual que antes, pero incluyendo la librería `geometry`)**:**

```
<script
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBDaeWicvigtP9xPv919E-RNoxfvC-Hqik&libraries=geometry"></script>
```

# Geolocalización – API Google Maps

Se utiliza el método siguiente:

```
let distancia=google.maps.geometry.spherical.computeDistanceBetween(coord1, coord2)
```

- `coord1` y `coord2` son objetos `JSON`, con las propiedad "`lat`" y "`lng`" que son números.
- `distancia` es un número que expresa la distancia entre `coord1` y `coord2` en metros.

**Ejercicio 6:** Crea una página web con:

- 4 `inputs` llamados "`Longitud 1`" y "`Latitud 1`", "`Longitud 2`" y "`Latitud 2`" para que el usuario introduzca las coordenadas de 2 ubicaciones
- 1 botón llamado "`Mostrar`" que marque en un mapa estas 2 ubicaciones y la del usuario.
- 1 botón llamado "Más cercano" que indique cuáles de las 2 ubicaciones (1 ó 2) está más cerca del usuario, expresando las distancias en km.

**Ejercicio 7:** Crea una página web con 1 botón llamado "`Mostrar`" que marque en un mapa tu ubicación y la de la tienda Carrefour más cercana a ti, así como la distancia a la misma. **NOTA IMPORTANTE: para que funcione has de darte de alta en la API de Google Maps que te permita utilizar este servicio**