CARGA Y DESCARGA

Con archivos json

Explicación de la Descarga (Exportar JSON)

Concepto: Convertir los datos de la memoria RAM (variables) en un archivo físico en el disco duro del usuario.

El proceso ocurre en la función descargarJSON().

Lo podemos dividir en **3 pasos lógicos**:

Paso A: Serialización (Empaquetado)

Primero, tenemos el array alumnos que es código JavaScript puro. Necesitamos convertirlo a texto plano para que pueda vivir en un archivo.

- **Código:** JSON.stringify(alumnos, null, 2).
- **Explicación :** Usamos stringify. El truco aquí son los parámetros extra (..., null, 2). Esto no cambia los datos, pero le dice a JS que use **2 espacios de sangría**. Sin esto, el archivo se vería en una sola línea ilegible; con esto, se ve ordenado ("Pretty Print").

Paso B: Creación del Objeto Binario (El "Archivo Virtual")

El navegador no puede guardar un simple string como archivo directamente. Necesita un contenedor de datos tipo fichero.

- **Código:** new Blob([datosStr], { type: "application/json" }).
- **Explicación:** Un **Blob** (Binary Large Object) es como un archivo en memoria. Le decimos: "Toma este texto y trátalo como un archivo de tipo JSON". Ahora tenemos un archivo, pero vive "flotando" en la memoria del navegador, no tiene URL ni ubicación.

Paso C: El Enlace Fantasma (El truco de descarga)

Para descargar algo en la web, normalmente necesitas un enlace (<a href="...">). Como nuestro archivo está en memoria, creamos una URL temporal y un enlace invisible.

1. **Crear URL:** URL.createObjectURL(blob). Genera una dirección interna (tipo blob:http://...) que apunta a nuestro archivo en memoria.
2. **Crear Enlace:** Creamos una etiqueta <a> en el aire (no se añade al HTML visible).
3. **Clic Automático:** Asignamos el nombre del archivo (a.download = "notas.json") y forzamos un clic con a.click().
4. **Limpieza:** URL.revokeObjectURL(url). Liberamos la memoria. Si no hacemos esto, el navegador se quedará con basura acumulada.

Explicación de la Carga (Importar JSON)

Concepto: Leer un archivo del ordenador del usuario, leer su contenido de texto y convertirlo de nuevo en variables de JavaScript.

El proceso ocurre mediante un "listener" en el input #inputArchivo.

Paso A: Selección del Archivo

Usamos un <input type="file">. Cuando el usuario selecciona algo, se dispara el evento change.

- **Código:** const archivo = e.target.files[0];.
- **Explicación :** files es un array porque un input podría aceptar múltiples archivos. Nosotros tomamos el primero ([0]).

Paso B: El Lector Asíncrono (FileReader)

Leer un archivo del disco duro es "lento" para el procesador. Por eso no podemos hacerlo instantáneamente; usamos un objeto FileReader que trabaja con eventos (similar a fetch) que veremos en la próxima lección de clase.

1. **Instanciar:** const lector = new FileReader();.
2. **Ordenar la lectura:** lector.readAsText(archivo);. Le decimos "Empieza a leer esto como si fuera texto".
3. **Esperar (El evento onload):** No podemos acceder a los datos en la línea siguiente. Tenemos que definir una función que se ejecute **solo cuando la lectura termine**. Esto es lector.onload.

Paso C: Deserialización y Actualización

Una vez que el lector termina, tenemos el contenido en evento.target.result.

- **Parseo:** JSON.parse(contenido) convierte ese texto de vuelta a un Array de Objetos.
- **Protección:** Usamos un bloque try...catch. ¿Por qué? Porque si el usuario sube una foto o un PDF en lugar de un JSON, JSON.parse fallará y rompería la aplicación. El catch nos permite mostrar un error.
- **Limpieza:** e.target.value = ''. Esto es un truco importante. Si el usuario carga "notas.json", lo borra, e intenta cargar "notas.json" de nuevo, el evento change no se dispararía la segunda vez porque el nombre del archivo no cambió. Limpiarlo permite recargar el mismo archivo.