Ejemplo con callbacks

El archivo JavaScript (`baseDatos.js`) se encarga de la lógica de negocio, mientras que el archivo HTML maneja la interacción con la interfaz de usuario.

Aquí te muestro cómo quedaría el código reorganizado:

Asincronia_cb.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Callbacks</title>
 </head>
 <body>
  <select name="" id="pares_o_nones">
   <option value="pares">Pares</option>
   <option value="nones">Impares</option>
  </select>
  <button id="disparador">Llama BaseDatos</button>
  <table id="tabla" border="1"></table>

  <script type="module">
   import { baseDatos } from "./js/baseDatos.js";

   // Función para actualizar la tabla (manipulación del DOM)
   function actualizarTabla(datos) {
    const tabla = document.getElementById("tabla");
    tabla.innerHTML = ""; // Limpiar la tabla

    datos.forEach((numero) => {
     const fila = document.createElement("tr");
     const celda = document.createElement("td");
     celda.textContent = numero;
     fila.appendChild(celda);
     tabla.appendChild(fila);
    });
   }

   // Evento para llamar a la base de datos
   document.getElementById("disparador").addEventListener("click", () => {
    let eleccion = document.getElementById("pares_o_nones").value;

    // Llamamos a baseDatos y le pasamos actualizarTabla como callback
    baseDatos(eleccion, actualizarTabla);
```

```
      console.log("Petición realizada");
      console.log("Voy haciendo otras cosas");
    });
  </script>
 </body>
</html>
```

2. Modificar `baseDatos.js` para aceptar un callback

Ahora que `actualizarTabla` está en el archivo HTML, necesitamos modificar `baseDatos` para que acepte un callback como argumento. Este callback será la función `actualizarTabla` que se pasa desde el archivo HTML.

`baseDatos.js`

```
const consulta = function (entrada, callback) {
 let listado = [];
 if (entrada == "pares") {
  listado = Array.from({ length: 10 }, (_, i) => i + 2).filter(
   (i) => i % 2 === 0
  );
 } else {
  listado = Array.from({ length: 10 }, (_, i) => i + 1).filter(
   (i) => i % 2 === 1
  );
 }
 callback(listado); // Ejecutamos el callback con los resultados
};

export const baseDatos = function (peticion, callback) {
 console.log("Petición recibida " + peticion);

 setTimeout(() => {
  consulta(peticion, (listado) => {
   console.log("Consulta completada:", listado);
   callback(listado); // Pasamos los resultados al callback
  });
 }, 6000);
};
```

El flujo de trabajo queda así:

1. El usuario selecciona "Pares" o "Impares" en el `<select>` y hace clic en el botón.
2. Al hacer clic, se llama a `baseDatos` desde el archivo HTML, pasando la elección del usuario (`peticion`) y la función `actualizarTabla` como callback.
3. La función `baseDatos` simula una consulta asincrónica durante 6 segundos.

4. Una vez completada la consulta, los resultados se pasan al callback (`actualizarTabla`), que actualiza la tabla en el DOM.

Ventajas de esta organización

1. **Separación de responsabilidades**:
   - El archivo JavaScript (`baseDatos.js`) se enfoca en la lógica de negocio (generar números pares o impares).
   - El archivo HTML maneja la interacción con el DOM y la presentación de los datos.
2. **Reutilización**:
   - La función `baseDatos` puede ser reutilizada en otros contextos sin depender de una implementación específica de la interfaz de usuario.
