Ejercicio Integrador: Sistema de Gestión de Biblioteca Musical

Desarrollo Web en Entorno Cliente - 2º DAW

Instrucciones Generales

Este ejercicio integrador está diseñado para que apliques de forma práctica todos los conceptos trabajados durante el curso: **arrays, objetos, Map, Set y LocalStorage**.

Duración estimada: 2 horas

Requisitos técnicos:

- Crea un archivo `bibliotecaMusical.js` con todas tus funciones
- Exporta las funciones: `export { funcion1, funcion2, ... }`
- Crea un archivo `app.js` para probar tu código
- Usa `console.log()`, `console.table()`, `console.group()` para mostrar resultados
- NO uses DOM, solo funcionalidades JavaScript puras

Contexto del Ejercicio

Eres un **melómano apasionado** que quiere organizar su colección de música digital. Necesitas crear un sistema que te permita:

- Catalogar tus canciones
- Gestionar playlists personalizadas
- Buscar música de forma inteligente
- Analizar tus gustos musicales
- Llevar registro de tus reproducciones
- Obtener recomendaciones basadas en tus preferencias

Todo el sistema debe **persistir los datos** usando LocalStorage para que no pierdas tu información al cerrar el navegador.

Datos Iniciales

Utiliza el siguiente array de canciones como datos de partida. Cópialo en tu archivo `data.js`:

```
// data.js
export const canciones = [
  {
```

```
    id: 1,
    titulo: "Bohemian Rhapsody",
    artista: "Queen",
    album: "A Night at the Opera",
    duracion: 354,
    genero: "Rock",
    año: 1975,
    reproducciones: 0
  },
  {
    id: 2,
    titulo: "Imagine",
    artista: "John Lennon",
    album: "Imagine",
    duracion: 183,
    genero: "Pop",
    año: 1971,
    reproducciones: 0
  },
  {
    id: 3,
    titulo: "Stairway to Heaven",
    artista: "Led Zeppelin",
    album: "Led Zeppelin IV",
    duracion: 482,
    genero: "Rock",
    año: 1971,
    reproducciones: 0
  },
  {
    id: 4,
    titulo: "Smells Like Teen Spirit",
    artista: "Nirvana",
    album: "Nevermind",
    duracion: 301,
    genero: "Grunge",
    año: 1991,
    reproducciones: 0
  },
  {
    id: 5,
    titulo: "Billie Jean",
    artista: "Michael Jackson",
    album: "Thriller",
    duracion: 294,
    genero: "Pop",
    año: 1982,
    reproducciones: 0
  },
  {
    id: 6,
```

```
    titulo: "Hotel California",
    artista: "Eagles",
    album: "Hotel California",
    duracion: 391,
    genero: "Rock",
    año: 1976,
    reproducciones: 0
  },
  {
    id: 7,
    titulo: "Sweet Child O' Mine",
    artista: "Guns N' Roses",
    album: "Appetite for Destruction",
    duracion: 356,
    genero: "Rock",
    año: 1987,
    reproducciones: 0
  },
  {
    id: 8,
    titulo: "Wonderwall",
    artista: "Oasis",
    album: "(What's the Story) Morning Glory?",
    duracion: 258,
    genero: "Rock",
    año: 1995,
    reproducciones: 0
  },
  {
    id: 9,
    titulo: "Lose Yourself",
    artista: "Eminem",
    album: "8 Mile Soundtrack",
    duracion: 326,
    genero: "Hip-Hop",
    año: 2002,
    reproducciones: 0
  },
  {
    id: 10,
    titulo: "Rolling in the Deep",
    artista: "Adele",
    album: "21",
    duracion: 228,
    genero: "Pop",
    año: 2010,
    reproducciones: 0
  }
];
```

Nota: La duración está en segundos.

PARTE 1: Catálogo Musical con Map

Estructura de datos esperada:

```
Map {
  1 => {
    id: 1,
    titulo: "Bohemian Rhapsody",
    artista: "Queen",
    album: "A Night at the Opera",
    duracion: 354,
    genero: "Rock",
    año: 1975,
    reproducciones: 0,
    historialReproduccion: []
  },
  // ... más canciones
}
```

Función 1: `crearCatalogo()`

Qué debe hacer:

1. Leer el array de canciones importado de `data.js`
2. Crear un **Map** donde:
   - **Clave:** el `id` de la canción
   - **Valor:** objeto completo de la canción
3. Añadir a cada canción la propiedad `historialReproduccion: []` (array vacío)
4. Guardar el Map en **LocalStorage** con la clave `"catalogo"`
5. Devolver el Map creado

Pistas:

- Usa `JSON.stringify()` para guardar el Map en LocalStorage (convierte Map a array primero)
- Para recuperarlo, usa `JSON.parse()` y luego reconstruye el Map

Ejemplo de llamada:

```
const catalogo = crearCatalogo();
console.log(`Catálogo creado con ${catalogo.size} canciones`);
```

Salida esperada:

```
Catálogo creado con 10 canciones
```

Función 2: `reproducirCancion(idCancion)`

Qué debe hacer:

1. Recuperar el catálogo desde LocalStorage
2. **Validar** que la canción con ese ID exista
   - Si no existe, lanzar un error con mensaje: `"La canción con ID X no existe"`
3. Incrementar el contador `reproducciones` en 1
4. Añadir un registro al array `historialReproduccion` con:

   ```
   {
     fecha: new Date().toISOString(),
     timestamp: Date.now()
   }
   ```

5. Actualizar el catálogo en LocalStorage
6. Devolver el objeto canción actualizado

Ejemplo de llamada:

```
try {
  const cancion = reproducirCancion(1);
  console.log(`Reproduciendo: ${cancion.titulo} - ${cancion.artista}`);
  console.log(`Total reproducciones: ${cancion.reproducciones}`);

  reproducirCancion(1); // Segunda reproducción
  reproducirCancion(999); // ID que no existe
} catch (error) {
  console.error(`Error: ${error.message}`);
}
```

Salida esperada:

```
Reproduciendo: Bohemian Rhapsody - Queen
Total reproducciones: 1
Reproduciendo: Bohemian Rhapsody - Queen
Total reproducciones: 2
Error: La canción con ID 999 no existe
```

PARTE 2: Sistema de Playlists con Set

Estructura de datos esperada:

```
Map {
  "Rock Classics" => Set {1, 3, 6, 7},
  "Favoritas" => Set {1, 2, 5},
  "Para Estudiar" => Set {2, 8, 10}
}
```

Función 3: `gestionarPlaylists()`

Qué debe hacer: Esta función debe devolver un **objeto con métodos** para gestionar playlists. Internamente usará un Map donde:

- **Clave:** nombre de la playlist (string)
- **Valor:** Set con IDs de canciones

Los datos deben persistir en LocalStorage con la clave `"playlists"`.

Métodos que debe incluir el objeto retornado:

3.1. `crear(nombrePlaylist)`

- Valida que la playlist no exista ya
- Crea una nueva playlist vacía
- Guarda en LocalStorage
- Devuelve `true` si se creó, `false` si ya existía

3.2. `agregar(nombrePlaylist, idCancion)`

- Valida que la playlist exista
- Valida que la canción exista en el catálogo
- Añade el ID al Set de la playlist
- Actualiza LocalStorage
- Devuelve `true` si se agregó, `false` si ya estaba

3.3. `eliminar(nombrePlaylist, idCancion)`

- Valida que la playlist exista
- Elimina el ID del Set
- Actualiza LocalStorage
- Devuelve `true` si se eliminó, `false` si no estaba

3.4. `obtener(nombrePlaylist)`

- Valida que la playlist exista
- Devuelve **array de objetos canción** (no solo IDs)
- Busca los datos completos en el catálogo

3.5. `listar()`

- Devuelve array con los nombres de todas las playlists existentes

Ejemplo de llamada:

```
const playlists = gestionarPlaylists();

// Crear playlists
playlists.crear("Rock Classics");
playlists.crear("Favoritas");

// Agregar canciones
playlists.agregar("Rock Classics", 1);  // Bohemian Rhapsody
playlists.agregar("Rock Classics", 3);  // Stairway to Heaven
playlists.agregar("Rock Classics", 6);  // Hotel California

playlists.agregar("Favoritas", 1);
playlists.agregar("Favoritas", 2);

// Listar playlists
console.log("Playlists disponibles:", playlists.listar());

// Obtener canciones de una playlist
const misRocks = playlists.obtener("Rock Classics");
console.log(`\nPlaylist "Rock Classics" tiene ${misRocks.length}
canciones:`);
misRocks.forEach(cancion => {
  console.log(`  - ${cancion.titulo} (${cancion.artista})`);
});

// Eliminar canción de playlist
playlists.eliminar("Rock Classics", 1);
console.log(`\nDespués de eliminar, quedan ${playlists.obtener("Rock
Classics").length} canciones`);
```

Salida esperada:

```
Playlists disponibles: ["Rock Classics", "Favoritas"]

Playlist "Rock Classics" tiene 3 canciones:
```

```
  - Bohemian Rhapsody (Queen)
  - Stairway to Heaven (Led Zeppelin)
  - Hotel California (Eagles)

Después de eliminar, quedan 2 canciones
```

PARTE 3: Búsqueda Inteligente con Map de Índices

Estructura del índice invertido:

```
Map {
  "queen" => Set {1},
  "rock" => Set {1, 3, 6, 7, 8},
  "bohemian" => Set {1},
  "rhapsody" => Set {1},
  "1975" => Set {1},
  // ... más términos
}
```

Función 4: `construirIndiceBusqueda()`

Qué debe hacer:

1. Recuperar el catálogo de canciones
2. Crear un **Map de índice invertido** donde:
   - **Clave:** término de búsqueda (en minúsculas)
   - **Valor:** Set con IDs de canciones que contienen ese término
3. Extraer términos de los siguientes campos:
   - `titulo` (dividir por espacios)
   - `artista` (dividir por espacios)
   - `album` (dividir por espacios)
   - `genero` (como término completo)
   - `año` (convertido a string)
4. Guardar el índice en LocalStorage con clave `"indiceBusqueda"`
5. Devolver el Map creado

Pistas:

- Convierte todo a minúsculas antes de indexar
- Elimina caracteres especiales y paréntesis
- Usa `split(" ")` para dividir en palabras
- Usa Set para evitar duplicados de IDs

Ejemplo de llamada:

```
const indice = construirIndiceBusqueda();
console.log(`Índice construido con ${indice.size} términos únicos`);
console.log(`El término "rock" aparece en ${indice.get("rock").size}
canciones`);
console.log(`El término "queen" aparece en ${indice.get("queen").size}
canciones`);
```

Salida esperada:

```
Índice construido con 75 términos únicos
El término "rock" aparece en 5 canciones
El término "queen" aparece en 1 canciones
```

Función 5: `buscarCanciones(termino, filtros = {})`

Qué debe hacer:

1. Recuperar el índice de búsqueda desde LocalStorage
2. Buscar el término (convertido a minúsculas) en el índice
3. Obtener el Set de IDs de canciones
4. Convertir los IDs en objetos completos usando el catálogo
5. **Aplicar filtros opcionales** (si se proporcionan):
   - `filtros.genero`: filtrar por género exacto (string)
   - `filtros.añoMin`: año mínimo (número)
   - `filtros.añoMax`: año máximo (número)
   - `filtros.duracionMax`: duración máxima en segundos (número)
6. Ordenar resultados por `reproducciones` de mayor a menor
7. Devolver array de canciones que coinciden

Ejemplo de llamada:

```
// Búsqueda simple
const resultados1 = buscarCanciones("rock");
console.log(`Búsqueda "rock": ${resultados1.length} resultados`);
resultados1.forEach(c => console.log(`  - ${c.titulo} (${c.genero})`));

// Búsqueda con filtros
const resultados2 = buscarCanciones("rock", {
  añoMin: 1970,
  añoMax: 1980
});
console.log(`\nBúsqueda "rock" años 70: ${resultados2.length} resultados`);
resultados2.forEach(c => console.log(`  - ${c.titulo} - ${c.año}`));
```

```
// Búsqueda con múltiples filtros
const resultados3 = buscarCanciones("rock", {
  genero: "Rock",
  duracionMax: 400
});
console.log(`\nBúsqueda "rock" género Rock, max 400s: ${resultados3.length}
resultados`);
```

Salida esperada:

```
Búsqueda "rock": 5 resultados
  - Bohemian Rhapsody (Rock)
  - Stairway to Heaven (Rock)
  - Hotel California (Rock)
  - Sweet Child O' Mine (Rock)
  - Wonderwall (Rock)

Búsqueda "rock" años 70: 3 resultados
  - Bohemian Rhapsody - 1975
  - Stairway to Heaven - 1971
  - Hotel California - 1976

Búsqueda "rock" género Rock, max 400s: 4 resultados
```

PARTE 4: Análisis y Estadísticas

Función 6: `generarEstadisticasMusicales()`

Qué debe hacer:

1. Recuperar el catálogo completo desde LocalStorage
2. Analizar todos los datos y calcular:

   a) `totalCanciones`: cantidad total de canciones en el catálogo

   b) `duracionTotal`: suma de todas las duraciones convertida a minutos (con 2 decimales)

   c) `cancionMasReproducida`: objeto con la canción que tiene más reproducciones

   d) `generosPorCantidad`: objeto con formato `{ "Rock": 5, "Pop": 3, "Hip-Hop": 1 }`

   e) `artistasUnicos`: cantidad de artistas diferentes (usa Set para contar)

   f) `añoPromedio`: año promedio de todas las canciones (redondeado)

g) `distribucionDecadas`**:** objeto que agrupa canciones por década

```
{
  "1970s": 3,
  "1980s": 2,
  "1990s": 2,
  "2000s": 2,
  "2010s": 1
}
```

3. Devolver un objeto con todas estas estadísticas

Pistas:

- Para las décadas: `Math.floor(año / 10) * 10 + "s"`
- Usa `reduce()` para calcular sumas y promedios
- Usa `Math.max()` con spread operator para encontrar máximos

Ejemplo de llamada:

```
const stats = generarEstadisticasMusicales();

console.log("=== ESTADÍSTICAS MUSICALES ===");
console.log(`Total de canciones: ${stats.totalCanciones}`);
console.log(`Duración total: ${stats.duracionTotal} minutos`);
console.log(`Canción más reproducida:
${stats.cancionMasReproducida.titulo}`);
console.log(`Artistas únicos: ${stats.artistasUnicos}`);
console.log(`Año promedio: ${stats.añoPromedio}`);

console.log("\nGéneros:");
console.table(stats.generosPorCantidad);

console.log("\nDistribución por década:");
console.table(stats.distribucionDecadas);
```

Salida esperada:

```
=== ESTADÍSTICAS MUSICALES ===
Total de canciones: 10
Duración total: 55.47 minutos
Canción más reproducida: Bohemian Rhapsody
Artistas únicos: 10
Año promedio: 1985
```

```
Géneros:
┌─────────┬────────┐
│ (index) │ Values │
├─────────┼────────┤
│  Rock   │   5    │
│   Pop   │   3    │
│ Grunge  │   1    │
│ Hip-Hop │   1    │
└─────────┴────────┘

Distribución por década:
┌─────────┬────────┐
│ (index) │ Values │
├─────────┼────────┤
│ 1970s   │   3    │
│ 1980s   │   2    │
│ 1990s   │   2    │
│ 2000s   │   1    │
│ 2010s   │   1    │
└─────────┴────────┘
```

PARTE 5: Sistema de Recomendaciones

Función 7: `generarRecomendaciones(idCancionBase, cantidad = 3)`

Qué debe hacer:

1. Recuperar el catálogo desde LocalStorage
2. Validar que la canción base exista
3. Buscar canciones similares usando un **sistema de puntuación**:
   - **+5 puntos:** si es del mismo artista
   - **+3 puntos:** si es del mismo género
   - **+2 puntos:** si el año de lanzamiento está en un rango de ±5 años
   - **+1 punto:** si la duración es similar (±60 segundos)
4. Excluir la canción base de los resultados
5. Ordenar canciones por puntuación descendente
6. Limitar resultados a la cantidad solicitada
7. Devolver array de objetos con formato:

```
{
  cancion: { /* objeto completo de la canción */ },
  puntuacion: 8,
  razones: ["Mismo artista", "Mismo género"]
}
```

Pistas:

- Guarda las razones de similitud para mostrarlas al usuario
- Si dos canciones tienen la misma puntuación, ordena por valoración o reproducciones

Ejemplo de llamada:

```
// Primero reproducimos algunas canciones para simular historial
reproducirCancion(1); // Bohemian Rhapsody
reproducirCancion(3); // Stairway to Heaven

// Generamos recomendaciones basadas en Bohemian Rhapsody (ID 1)
const recomendaciones = generarRecomendaciones(1, 5);

console.log("Si te gustó 'Bohemian Rhapsody', te recomendamos:\n");
recomendaciones.forEach((rec, index) => {
  console.log(`${index + 1}. ${rec.cancion.titulo} -
${rec.cancion.artista}`);
  console.log(`   Puntuación: ${rec.puntuacion} puntos`);
  console.log(`   Razones: ${rec.razones.join(", ")}`);
  console.log();
});
```

Salida esperada:

```
Si te gustó 'Bohemian Rhapsody', te recomendamos:

1. Hotel California - Eagles
   Puntuación: 5 puntos
   Razones: Mismo género, Año similar, Duración similar

2. Stairway to Heaven - Led Zeppelin
   Puntuación: 4 puntos
   Razones: Mismo género, Año similar

3. Sweet Child O' Mine - Guns N' Roses
   Puntuación: 4 puntos
   Razones: Mismo género, Duración similar

4. Wonderwall - Oasis
   Puntuación: 3 puntos
   Razones: Mismo género

5. Smells Like Teen Spirit - Nirvana
   Puntuación: 1 punto
   Razones: Duración similar
```

Archivo de Prueba (app.js)

Crea un archivo `app.js` para probar todas las funcionalidades. Ejemplo:

```
import { canciones } from './data.js';
import {
  crearCatalogo,
  reproducirCancion,
  gestionarPlaylists,
  construirIndiceBusqueda,
  buscarCanciones,
  generarEstadisticasMusicales,
  generarRecomendaciones
} from './bibliotecaMusical.js';

console.clear();
console.log("🎵 === SISTEMA DE BIBLIOTECA MUSICAL === 🎵\n");

// 1. CREAR CATÁLOGO
console.log("1️⃣ Creando catálogo...");
const catalogo = crearCatalogo();

// 2. REPRODUCIR CANCIONES
console.log("\n2️⃣ Reproduciendo canciones...");
reproducirCancion(1);
reproducirCancion(1);
reproducirCancion(3);
reproducirCancion(5);

// 3. GESTIONAR PLAYLISTS
console.log("\n3️⃣ Gestionando playlists...");
const playlists = gestionarPlaylists();
playlists.crear("Rock Classics");
playlists.agregar("Rock Classics", 1);
playlists.agregar("Rock Classics", 3);
playlists.agregar("Rock Classics", 6);

// 4. CONSTRUIR ÍNDICE Y BUSCAR
console.log("\n4️⃣ Construyendo índice de búsqueda...");
construirIndiceBusqueda();
const resultados = buscarCanciones("rock", { añoMin: 1970, añoMax: 1980 });

// 5. ESTADÍSTICAS
console.log("\n5️⃣ Generando estadísticas...");
const stats = generarEstadisticasMusicales();

// 6. RECOMENDACIONES
console.log("\n6️⃣ Generando recomendaciones...");
```

```
const recs = generarRecomendaciones(1, 3);

console.log("\n✅ Todas las funcionalidades probadas correctamente");
```

Criterios de Evaluación

Distribución de puntos:

1. **Funcionalidad correcta (40%)**
   - Todas las funciones devuelven los resultados esperados
   - Manejo correcto de casos límite y errores
2. **Uso adecuado de estructuras de datos (30%)**
   - Map usado correctamente para catálogo, índices y playlists
   - Set usado para colecciones únicas (IDs en playlists, artistas únicos)
   - Arrays usados cuando corresponde
3. **Persistencia con LocalStorage (15%)**
   - Datos se guardan correctamente
   - Datos se recuperan correctamente
   - Se mantiene la estructura de datos al guardar/recuperar
4. **Calidad del código (15%)**
   - Código limpio y bien organizado
   - Validaciones y manejo de errores
   - Nombres descriptivos de variables y funciones
   - Comentarios en partes complejas

Consejos Importantes

✅ **Lee todo el enunciado antes de empezar a programar**

✅ **Implementa y prueba las funciones en orden** (son progresivas) aunque yo empezaría con el guardar y cargar map y set en localstorage para tener eso listo desde el principio

✅ **Valida SIEMPRE los datos de entrada** (IDs que existen, playlists que existen, etc.)

✅ **Usa console.log() generosamente** para debuggear tu código

✅ **Guarda cambios en LocalStorage después de cada modificación**

✅ **Convierte Map/Set a Array antes de guardar en LocalStorage** usando:

- `Array.from(map)` o `[...map]`
- `JSON.stringify()` y `JSON.parse()`

✓ **Recuerda reconstruir Map/Set al recuperar de LocalStorage**

✓ **Si te bloqueas en una función, pasa a la siguiente** y vuelve después

---

Entrega

Archivos requeridos:

- `data.js` (con el array de canciones proporcionado)
- `bibliotecaMusical.js` (con todas las funciones implementadas)
- `app.js` (con las pruebas de todas las funcionalidades)

Formato de entrega:

- Carpeta comprimida (.zip) con tu nombre: `Apellido_Nombre_BibliotecaMusical.zip`
- Incluye comentarios explicando las partes más complejas
