Examen: Conversión de Aplicación de Streaming a React

Introducción

El objetivo de este proyecto es tomar una aplicación existente de visualización de trailers, llamada **Trailerflix**, desarrollada en **HTML, CSS y JavaScript**, y convertirla a **React** utilizando **Vite**. Deberán seguir las mejores prácticas y patrones de diseño de React, aprovechando los hooks y herramientas clave, tales como `useState`, `useEffect`, `react-router`, `useNavigate`, y **Custom Hooks**.

Archivos de Datos

Los archivos JSON con la información de películas y usuarios se encuentran en la carpeta `data/`:

- `data/trailerflix.json` - Contiene el catálogo completo de películas y series
- `data/usuarios.json` - Contiene los usuarios registrados para autenticación

IMPORTANTE: Al convertir a React con Vite, estos archivos deben moverse a la carpeta `public/data/` para que sean accesibles mediante fetch.

Requerimientos Técnicos

La conversión debe implementar lo siguiente:

- **useState**: Para manejar el estado de las películas, usuarios y otros datos relevantes.
- **useEffect**: Para manejar la carga inicial de datos (como las películas desde el archivo JSON) y la actualización del historial.
- **react-router**: Para manejar la navegación entre la página principal y la de detalles de la película.
- **useNavigate**: Para redirigir a los usuarios entre las diferentes páginas (por ejemplo, después de seleccionar una película).
- **useContext**: Para manejar el estado global de autenticación del usuario. Este estado debe mostrarse en todas las páginas, donde el componente de login muestra el nombre de usuario cuando está logueado y el botón de cierre de sesión. No se requiere redirigir después de iniciar sesión.
- **Custom Hooks**: Para encapsular la lógica de autenticación de usuarios.

Pasos para la Conversión

1. Inicializar el Proyecto con Vite

- Crea un nuevo proyecto utilizando **Vite**.
- Organiza la estructura del proyecto en carpetas adecuadas (`components`, `hooks`, `context`, etc.).
- **Crea la carpeta `public/data/`** y copia los archivos `trailerflix.json` y `usuarios.json` dentro de ella.

Ejemplo de Fetch en React:

En React con Vite, los archivos en la carpeta `public/` son accesibles directamente. Para cargar los datos:

```
// Ejemplo de carga de películas
useEffect(() => {
  const fetchMovies = async () => {
    try {
      const response = await fetch('/data/trailerflix.json');
      const data = await response.json();
      setMovies(data);
    } catch (error) {
      console.error('Error al cargar películas:', error);
    }
  };

  fetchMovies();
}, []);

// Ejemplo de validación de usuarios
const fetchUsers = async () => {
  try {
    const response = await fetch('/data/usuarios.json');
    const data = await response.json();
    return data.users;
  } catch (error) {
    console.error('Error al cargar usuarios:', error);
    return [];
  }
};
```

Nota: La ruta comienza con `/data/` (no `./data/` ni `../data/`) porque Vite sirve automáticamente el contenido de `public/` desde la raíz.

2. Crear las Rutas con `react-router`

- Implementa el enrutamiento con **react-router**:
  - Ruta principal (`/`): Donde se mostrarán las películas disponibles.
  - Ruta de detalles de la película (`/movie/:id`): Donde se mostrará la información de la película seleccionada.
  - **Ruta 404 (`*`)**: Para manejar páginas no encontradas. Debe mostrar un mensaje de error y un botón para volver al inicio.

3. Página Principal (Componente `Home`)

- **useState**: Para manejar la lista de películas.
- **useEffect**: Para cargar las películas desde el archivo `trailerflix.json`.
- **useNavigate**: Al hacer clic en una película, navega a la página de detalles usando `useNavigate()`.

4. Página de Detalles de la Película (Componente `MovieDetail`)

- **useEffect**: Para leer el ID de la película desde la URL y cargar los datos correspondientes.
- **useState**: Para manejar los detalles de la película.
- Muestra la información de la película (poster, título, resumen, reparto, y tráiler).
- Incluye un botón para **volver al catálogo**.

5. Página 404 (Componente `NotFound`)

- Crea un componente para manejar rutas no encontradas.
- Debe mostrar:
  - El logo de **TRAILERFLIX**.
  - Un mensaje de error **404 - Página no encontrada**.
  - Una descripción amigable del error.
  - Un botón para **volver al inicio** usando `useNavigate()`.
- Mantén la estética consistente con el resto de la aplicación (fondo negro, textos en rojo y blanco).

6. Sistema de Autenticación

- **useContext**: Crea un contexto para manejar el estado de autenticación del usuario.
  - El formulario de inicio de sesión debe mostrarse si el usuario no ha iniciado sesión.
  - Una vez autenticado, debe actualizarse el componente de login para mostrar el nombre y el botón de cerrar sesión.

7. Implementación de Custom Hooks

- **useAuth**: Un custom hook para manejar la lógica de autenticación de usuarios.

Requerimientos Funcionales

1. **Visualización de películas**: La lista de películas debe cargarse dinámicamente desde el archivo `trailerflix.json` ubicado en `public/data/`.
2. **Autenticación de usuarios**: El sistema debe permitir iniciar sesión usando los datos de `usuarios.json` ubicado en `public/data/`. Debe mostrar el formulario de login o la información del usuario según corresponda. Al iniciar sesión, solo debe actualizarse el componente de login, sin redirecciones.
3. **Navegación**: Implementar navegación entre la página principal, detalles de película y página 404.
4. **Página 404**: Debe mostrar un mensaje de error amigable cuando se acceda a una ruta inexistente, con opción de volver al inicio.
5. **Hooks personalizados**: Se deben implementar hooks personalizados para la lógica de autenticación.
6. **Responsive**: La aplicación debe verse correctamente en dispositivos móviles y desktop.

Estructura Sugerida del Proyecto React

```
proyecto-react/
├── public/
│   └── data/
│       ├── trailerflix.json
│       └── usuarios.json
├── src/
│   ├── components/
│   │   ├── Header.jsx
│   │   ├── Login.jsx
│   │   ├── MovieCard.jsx
│   │   └── ...
│   ├── pages/
│   │   ├── Home.jsx
│   │   ├── MovieDetail.jsx
│   │   └── NotFound.jsx
│   ├── context/
│   │   └── AuthContext.jsx
│   ├── hooks/
│   │   └── useAuth.js
│   ├── App.jsx
│   └── main.jsx
└── package.json
```

Evaluación

1. **Funcionalidad**: La aplicación debe funcionar correctamente en React.
2. **Uso de Hooks**: Se evaluará el uso adecuado de `useState`, `useEffect`, `useNavigate`, `useContext`, y custom hooks.
3. **Modularidad**: El código debe estar bien organizado y estructurado en componentes reutilizables.
4. **Estado Global**: El estado de autenticación debe manejarse correctamente con `useContext` y reflejarse en todas las páginas.
5. **Manejo de errores**: Implementación correcta de la página 404 y manejo de errores en las peticiones fetch.
6. **Carga de datos**: Correcta implementación del fetch a los archivos JSON desde la carpeta `public/data/`.

Entrega

Los alumnos deberán subir el proyecto a un repositorio de **GitHub**, con instrucciones claras para ejecutarlo, y enviar el enlace antes de la fecha límite.

Rúbrica de Evaluación

Puntaje Total: 100 puntos

1. React Router - Navegación (10 puntos)

Criterio	Excelente (9-10)	Bueno (7-8)	Regular (4-6)	Insuficiente (0-3)
Implementación de rutas	Todas las rutas funcionan correctamente: `/` (Home), `/movie/:id` (Detalles), `*` (404). Navegación fluida con `useNavigate`	Rutas principales funcionan, puede faltar 404 o tener errores menores en navegación	Rutas implementadas pero con errores significativos en la navegación	No implementa react-router o rutas no funcionales

Puntos clave a evaluar:

- ✅ Instalación de `react-router-dom`
- ✅ Configuración de `BrowserRouter`
- ✅ Ruta principal `/` funcional
- ✅ Ruta dinámica `/movie/:id` funcional
- ✅ Ruta 404 `*` implementada
- ✅ Uso correcto de `useNavigate()` para navegación programática

---

2. useState - Manejo de Estado Local (10 puntos)

Criterio	Excelente (9-10)	Bueno (7-8)	Regular (4-6)	Insuficiente (0-3)
Uso de useState	`useState` usado correctamente en múltiples componentes: lista de películas, detalles de película, estados de carga, etc.	`useState` implementado en componentes principales con algunos errores menores	`useState` usado pero de forma incorrecta o incompleta	No usa `useState` o su uso es completamente erróneo

Puntos clave a evaluar:

- ✅ Estado para lista de películas en Home
- ✅ Estado para detalles de película en MovieDetail
- ✅ Estados de carga (loading) implementados
- ✅ Manejo correcto de la actualización del estado

---

3. useEffect - Efectos y Carga de Datos (10 puntos)

Criterio	Excelente (9-10)	Bueno (7-8)	Regular (4-6)	Insuficiente (0-3)
Implementación de useEffect	`useEffect` usado correctamente para cargar datos de películas y usuarios. Manejo adecuado de dependencias y cleanup cuando necesario	`useEffect` implementado para carga de datos con errores menores en dependencias	`useEffect` usado pero con problemas significativos (bucles infinitos, dependencias incorrectas)	No usa `useEffect` o su uso genera errores críticos

Puntos clave a evaluar:

- ✅ Carga de películas desde `/data/trailerflix.json`
- ✅ Carga de película individual en MovieDetail
- ✅ Array de dependencias correcto `[]` para carga inicial
- ✅ Manejo de errores en peticiones fetch
- ✅ Estados de carga mientras se obtienen datos

---

4. useContext - Estado Global de Autenticación (15 puntos)

Criterio	Excelente (13-15)	Bueno (10-12)	Regular (6-9)	Insuficiente (0-5)
Context API implementado	`AuthContext` creado y usado correctamente. Estado de autenticación disponible en toda la app. Login y logout funcionan perfectamente. Se muestra nombre de usuario o formulario según corresponda	Context implementado, funciona pero con errores menores en la sincronización del estado	Context creado pero con problemas significativos en su implementación o uso	No implementa `useContext` o no funciona

Puntos clave a evaluar:

- ✅ Archivo `AuthContext.jsx` creado
- ✅ Provider envolviendo la aplicación
- ✅ Estado de usuario compartido globalmente
- ✅ Funciones `login` y `logout` disponibles en el contexto
- ✅ Componente Login actualizado según estado (formulario vs info usuario)
- ✅ Estado persiste durante la navegación

---

5. Custom Hook - useAuth (10 puntos)

Criterio	Excelente (9-10)	Bueno (7-8)	Regular (4-6)	Insuficiente (0-3)
Custom Hook implementado	`useAuth` hook personalizado implementado correctamente. Encapsula lógica de autenticación, validación de usuarios contra `usuarios.json`, retorna funciones y estados necesarios	Hook creado y funcional con errores menores	Hook creado pero no encapsula correctamente la lógica o tiene errores	No implementa custom hook

Puntos clave a evaluar:

- ✅ Archivo `useAuth.js` en carpeta `hooks/`
- ✅ Lógica de login encapsulada
- ✅ Validación contra `usuarios.json`
- ✅ Retorna funciones y estados necesarios
- ✅ Reutilizable y siguiendo convenciones de hooks

---

6. Página Principal - Home (10 puntos)

Criterio	Excelente (9-10)	Bueno (7-8)	Regular (4-6)	Insuficiente (0-3)
Componente Home funcional	Muestra todas las películas correctamente, cards interactivas, navegación a detalles funciona perfectamente, diseño atractivo	Muestra películas, navegación funciona con errores menores de UI	Muestra películas pero con problemas en navegación o diseño	No muestra películas o no funciona

Puntos clave a evaluar:

- ✅ Renderiza lista completa de películas
- ✅ Componentes `MovieCard` reutilizables
- ✅ Click en película navega a detalles
- ✅ Diseño responsive
- ✅ Manejo de estado de carga

---

7. Página de Detalles - MovieDetail (10 puntos)

Criterio	Excelente (9-10)	Bueno (7-8)	Regular (4-6)	Insuficiente (0-3)
Componente MovieDetail funcional	Muestra todos los detalles de la película (poster, título, resumen, reparto, tráiler), obtiene ID de URL correctamente, botón volver funciona	Muestra información principal con detalles menores faltantes	Muestra información pero con problemas significativos	No muestra detalles o no funciona

Puntos clave a evaluar:

- ✅ Uso de `useParams()` para obtener ID
- ✅ Carga de datos de película específica

- ✅ Muestra poster, título, resumen, reparto
- ✅ Integración de tráiler de YouTube
- ✅ Botón "Volver al catálogo" funcional
- ✅ Manejo de película no encontrada

---

8. Página 404 - NotFound (5 puntos)

Criterio	Excelente (5)	Bueno (4)	Regular (2-3)	Insuficiente (0-1)
Componente NotFound	Página 404 completa con logo, mensaje de error, descripción amigable, botón de retorno funcional, estética consistente	Página 404 funcional con elementos menores faltantes	Página 404 básica sin estética o funcionalidad completa	No implementa página 404

Puntos clave a evaluar:

- ✅ Logo de TRAILERFLIX visible
- ✅ Mensaje "404 - Página no encontrada"
- ✅ Descripción amigable del error
- ✅ Botón "Volver al inicio" con `useNavigate()`
- ✅ Estética consistente (fondo negro, colores corporativos)

---

9. Sistema de Autenticación (15 puntos)

Criterio	Excelente (13-15)	Bueno (10-12)	Regular (6-9)	Insuficiente (0-5)
Login funcional	Sistema de login completo: formulario funcional, valida contra `usuarios.json`, muestra errores, actualiza UI mostrando nombre de usuario, botón de logout funciona, no hay redirecciones innecesarias	Login funciona, validación correcta con errores menores en UI	Login implementado pero con problemas de validación o UI	Login no funciona o no implementado

Puntos clave a evaluar:

- ✅ Formulario de login (usuario y contraseña)
- ✅ Validación contra `usuarios.json`
- ✅ Mensajes de error informativos
- ✅ UI actualizada al iniciar sesión (muestra nombre de usuario)
- ✅ Botón de "Cerrar sesión" funcional
- ✅ No redirige después de login (solo actualiza componente)
- ✅ Estado persistente durante navegación

---

10. Modularidad y Organización del Código (5 puntos)

Criterio	Excelente (5)	Bueno (4)	Regular (2-3)	Insuficiente (0-1)
Código limpio y organizado	Componentes bien separados, código reutilizable, nombres descriptivos, comentarios cuando necesario, sin código repetido	Código organizado con mejoras menores posibles	Código funcional pero desorganizado o repetitivo	Código desorganizado y difícil de mantener

Puntos clave a evaluar:

- ✅ Componentes en archivos separados
- ✅ Nombres descriptivos de variables y funciones
- ✅ No hay código duplicado
- ✅ Imports organizados
- ✅ Código legible y mantenible

Criterios de Desaprobación Automática

El proyecto será desaprobado automáticamente si:

- ❌ La aplicación no ejecuta o tiene errores críticos que impiden su funcionamiento

---

Bonus (Hasta 10 puntos adicionales)

Puntos extras por implementaciones adicionales:

- **+5 puntos**: Persistencia de sesión con localStorage
- **+5 puntos**: Filtros de búsqueda por categoría/género