Práctica 2

Explorador de películas con React

Repositorio GitHub:
https://github.com/Mariinaaa/movies-app

Marina Amaya y Cristina Barandiarán

Objetivo del proyecto

El objetivo de la práctica fue desarrollar una aplicación web que permita explorar películas utilizando **React** y la **API de TMDB**, incluyendo funcionalidades de búsqueda, visualización de películas populares y gestión de favoritos con persistencia local.

Implementación de los componentes

1. App.jsx

En este componente central se configuran las **rutas de la aplicación** mediante React Router. Se envolvió toda la app con el MovieProvider para que el estado de favoritos sea accesible desde cualquier componente.

```jsx
function App() {
  return (
    <MovieProvider>
      <Router>
        <NavBar />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/favorites" element={<Favorites />} />
        </Routes>
      </Router>
    </MovieProvider>
  );
}

export default App;
```

Esto permite que la navegación entre Home y Favorites funcione correctamente y que los favoritos se gestionen globalmente.

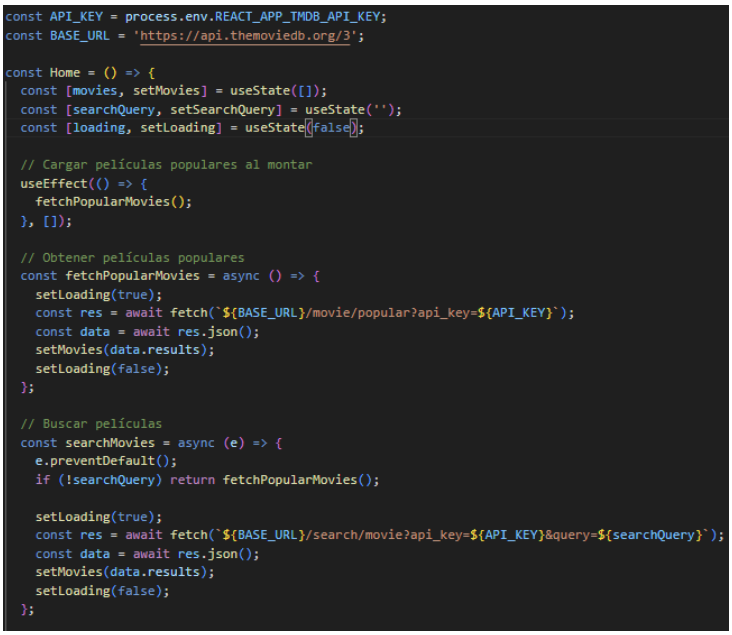
2. NavBar.jsx

Se implementó una **barra de navegación fija** con el título de la app y enlaces a Home y Favorites.
El diseño es responsivo, usando flexbox para mantener los elementos alineados y que se vea bien en móviles y escritorio.
Los enlaces usan <Link> de react-router-dom para cambiar de página sin recargar la app.

```jsx
import { Link } from 'react-router-dom';
import './NavBar.css'; // (opcional, para tus estilos)

const NavBar = () => {
  return (
    <nav className="navbar">
      <h1> MovieApp</h1>
      <div className="links">
        <Link to="/">Home</Link>
        <Link to="/favorites">Favorites</Link>
      </div>
    </nav>
  );
};

export default NavBar;
```

3. Home.jsx

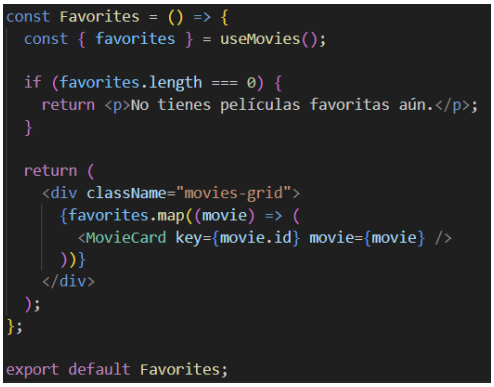
- Se utiliza useEffect para hacer una **petición inicial a la API de TMDB** y mostrar las películas populares al cargar la página.
- Se implementó un **formulario de búsqueda** que actualiza un estado searchQuery y realiza otra petición a la API al enviar el formulario.
- Mientras se cargan los datos, se muestra un **indicador de carga** usando un estado booleano loading.
- Los resultados se muestran en un **grid dinámico de películas** usando el componente MovieCard.



```
const API_KEY = process.env.REACT_APP_TMDB_API_KEY;
const BASE_URL = 'https://api.themoviedb.org/3';

const Home = () => {
  const [movies, setMovies] = useState([]);
  const [searchQuery, setSearchQuery] = useState('');
  const [loading, setLoading] = useState(false);

  // Cargar películas populares al montar
  useEffect(() => {
    fetchPopularMovies();
  }, []);

  // Obtener películas populares
  const fetchPopularMovies = async () => {
    setLoading(true);
    const res = await fetch(`${BASE_URL}/movie/popular?api_key=${API_KEY}`);
    const data = await res.json();
    setMovies(data.results);
    setLoading(false);
  };

  // Buscar películas
  const searchMovies = async (e) => {
    e.preventDefault();
    if (!searchQuery) return fetchPopularMovies();

    setLoading(true);
    const res = await fetch(`${BASE_URL}/search/movie?api_key=${API_KEY}&query=${searchQuery}`);
    const data = await res.json();
    setMovies(data.results);
    setLoading(false);
  };
```

4. Favorites.jsx

- Recupera la lista de películas favoritas desde el contexto (useContext).
- Si la lista está vacía, muestra un mensaje indicando que no hay favoritos.
- Si hay favoritos, se renderizan usando el mismo MovieCard que en Home, manteniendo coherencia visual.



```
const Favorites = () => {
  const { favorites } = useMovies();

  if (favorites.length === 0) {
    return <p>No tienes películas favoritas aún.</p>;
  }

  return (
    <div className="movies-grid">
      {favorites.map((movie) => (
        <MovieCard key={movie.id} movie={movie} />
      ))}
    </div>
  );
};

export default Favorites;
```

5. MovieCard.jsx

- Cada tarjeta muestra la **imagen del póster**, el **título** y el **año de lanzamiento** de la película.
- Se implementó un **botón de favorito interactivo** con un corazón que cambia según si la película está en favoritos (❤️ / 🤍).
- Se añadió un **efecto hover** con overlay, haciendo que el botón se destaque sobre la imagen.
- Para determinar si una película está en favoritos, se usa useContext para acceder al estado global.

```jsx
const MovieCard = ({ movie }) => {
  const { favorites, toggleFavorite } = useMovies();
  const isFavorite = favorites.some(m => m.id === movie.id);

  return (
    <div className="movie-card">
      <img
        src={`https://image.tmdb.org/t/p/w500${movie.poster_path}`}
        alt={movie.title}
      />
      <div className="info">
        <h3>{movie.title}</h3>
        <p>{movie.release_date ? new Date(movie.release_date).getFullYear() : 'N/A'}</p>
        <button onClick={() => toggleFavorite(movie)}>
          {isFavorite ? '❤️' : '🤍'}
        </button>
      </div>
    </div>
  );
};

export default MovieCard;
```

6. MovieContext.jsx

```jsx
const MovieContext = createContext();

export const MovieProvider = ({ children }) => {
  // Inicializa directamente leyendo de localStorage
  const [favorites, setFavorites] = useState(() => {
    const stored = localStorage.getItem("favorites");
    return stored ? JSON.parse(stored) : [];
  });

  // Guarda cada cambio en localStorage
  useEffect(() => {
    localStorage.setItem("favorites", JSON.stringify(favorites));
  }, [favorites]);

  // Alterna favoritos (añade o quita)
  const toggleFavorite = (movie) => {
    if (!movie || !movie.id) return;

    setFavorites((prevFavorites) => {
      const exists = prevFavorites.some((fav) => fav.id === movie.id);
      return exists
        ? prevFavorites.filter((fav) => fav.id !== movie.id)
        : [...prevFavorites, movie];
    });
  };

  // Devuelve contexto global
  return (
    <MovieContext.Provider value={{ favorites, toggleFavorite }}>
      {children}
    </MovieContext.Provider>
  );
};

// Hook personalizado para acceder al contexto
export const useMovies = () => useContext(MovieContext);
```

- Se creó un contexto global para almacenar los favoritos (createContext).
- Se utiliza useState para manejar la lista de favoritos.
- La lista de favoritos se inicializa leyendo directamente desde localStorage, lo que permite que al refrescar la página los favoritos se mantengan sin perderse.
- useEffect se encarga de guardar los cambios en localStorage cada vez que se actualiza la lista, asegurando la persistencia de los datos.
- Se expone un método toggleFavorite que agrega o quita películas de la lista, verificando si ya existen mediante su ID.
- Se creó un custom hook useMovies para facilitar el consumo del contexto desde cualquier componente.

Hooks de React utilizados

- **useState**: gestión de búsqueda, lista de películas, estado de carga y lista de favoritos.
- **useEffect**: carga inicial de películas populares, sincronización con localStorage.
- **useContext**: acceso a favoritos y funciones de gestión desde MovieCard y Favorites.
- **createContext**: contexto global de favoritos.

Gestión de la API de TMDB

- Primero nos registramos en TMBD.
- En el perfil en la sección de API generamos la API Key.
- Se almacenó la **API Key** en un archivo .env como: REACT_APP_TMDB_API_KEY.
- Se realizan peticiones a:
    - /movie/popular para cargar películas al inicio.
    - /search/movie para buscar por título.
- La respuesta de la API se procesa y se guarda en el estado movies, que luego se renderiza en el grid.

Problemas y dificultades encontradas

Errores de CSS

- Inicialmente faltaban los archivos MovieCard.css y NavBar.css, generando errores de compilación.
- Se solucionó creando los archivos y añadiendo estilos básicos para el grid y overlay.

Persistencia de favoritos

- Al recargar la página, los favoritos se perdían.
- Se solucionó usando useEffect para sincronizar localStorage con el estado de favoritos.

Configuración de la API Key

- Fue necesario reiniciar el servidor de React para que la variable de entorno se reconociera correctamente.

Responsive design

- Ajustes en CSS para que las tarjetas se vean bien en distintos tamaños de pantalla y que el overlay funcione correctamente.

Conclusión

La aplicación permite explorar películas, buscar por título, gestionar favoritos y navegar entre secciones de forma fluida.
Se implementó una estructura modular con **React**, **Context API** y **hooks**, cumpliendo todos los requisitos de la práctica.