

	ESCUELA SUPERIOR DE ECONOMÍA Y NEGOCIOS INGENIERÍA DE SOFTWARE Y NEGOCIOS DIGITALES
CICLO 01-2025	GUIA DE LABORATORIO Nº 3
	Nombre de la práctica: React Hooks Tiempo estimado: 2 horas Materia: Desarrollo Web II

I. Objetivos

Que el estudiante sea capaz de:

1. Comprender el uso de los React Hooks más comunes para gestionar estado y efectos secundarios en componentes funcionales.
2. Aplicar Hooks en proyectos reales para optimizar la gestión del estado y mejorar la reutilización del código en aplicaciones React.

II. Introducción Teórica

Los Hooks permiten a los desarrolladores utilizar el estado y otras funcionalidades clave de React dentro de los componentes funcionales, una característica que anteriormente era exclusiva de los componentes de clase.

Este avance ha simplificado la arquitectura de React y ampliado su alcance funcional, convirtiendo a los Hooks en un elemento esencial del desarrollo contemporáneo con React.

Pautas para usar Hooks

- Los Hooks deben usarse en el nivel superior de un componente y no dentro de bucles, condiciones o funciones anidadas.
- Deben llamarse desde dentro de componentes funcionales de React, no desde funciones JavaScript normales.

Hooks de estado: useState()

El Hook useState es fundamental para agregar estado a los componentes funcionales. Devuelve un par: el estado actual y una función para actualizarlo.

```

● ● ●

function SampleComponent() {
  const [value, setValue] = useState(0);
  return <h1>{value}</h1>;
}
  
```

Hooks de efectos secundarios: useEffect

El Hook useEffect se utiliza para manejar efectos secundarios en los componentes funcionales. Un **efecto secundario** se refiere a cualquier operación que se necesita ejecutar tan pronto como la página se carga, sin tener que llamar esas operaciones o funcionalidades por separado, como obtener datos de una API, suscribirse a eventos, manipular el DOM o establecer temporizadores. Los efectos secundarios son acciones que ocurren de manera asíncrona y que afectan el estado de la aplicación o la interfaz de usuario (UI).



```
import React, { useState, useEffect } from "react";

function DependencyEffect() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(`El contador cambió a: ${count}`);
  }, [count]); // El efecto solo se ejecuta cuando cambia "count".

  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrementar</button>
    </div>
  );
}

export default DependencyEffect;
```

Custom Hooks

Los Custom Hooks son una herramienta poderosa en React que permite encapsular lógica reutilizable en funciones personalizadas. Al crear Custom Hooks, los desarrolladores pueden mejorar la organización del código, promover la reutilización y mantener componentes más simples y enfocados en la presentación.

Un Custom Hook siempre debe comenzar con el prefijo use (ejemplo: useMyHook). Es posible usar otros Hooks dentro de ellos para construir la lógica.



```
import { useState } from "react";

export function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(initialValue);

  return { count, increment, decrement, reset };
}
```

III. Procedimiento

1. Abre la terminal de tu preferencia y navega hasta la carpeta donde deseas crear el proyecto. Crea el proyecto de React con el comando **npm create vite@latest**. Nombre el proyecto como “practica3”.
2. Accede a la carpeta del proyecto con el comando **cd practica3** e instala las dependencias con **npm install**.
3. Inicia el servidor de desarrollo con el comando **npm run dev** y verifica que el proyecto de prueba cargue correctamente en el navegador.
4. Elimina todo el contenido de la carpeta src a excepción del archivo **main.jsx**
5. Ubica el archivo **index.css**, que se proporciona como recurso, en la carpeta src del proyecto.
6. Crea las carpetas “components” y “hooks” dentro del directorio src de tu proyecto.
7. Crear un archivo con el nombre **useFetchMovies.js** dentro del directorio **hooks**. Este archivo define un **custom hook** llamado **useFetchMovies**, que se encarga de buscar películas en la API de **OMDb** según un término de búsqueda (query). El hook maneja el estado de los resultados, la carga y los posibles errores.

Digita el siguiente código fuente dentro del archivo:

```
import { useEffect, useState } from "react";

// Clave de API para acceder a OMDB
export const API_KEY = "COLOCA TU API KEY ACA";

/**
 * Hook personalizado para obtener películas desde la API de OMDB.
 * @param {string} query - Término de búsqueda ingresado por el usuario.
 * @returns {Object} - Retorna un objeto con:
 *   - movies: Lista de películas encontradas.
 *   - isLoading: Estado de carga de la solicitud.
 *   - error: Mensaje de error en caso de fallo.
 */
export function useFetchMovies(query) {
  // Estado para almacenar las películas obtenidas
  const [movies, setMovies] = useState([]);

  // Estado para indicar si la solicitud está en curso
  const [isLoading, setIsLoading] = useState(false);

  // Estado para manejar errores
  const [error, setError] = useState("");

  useEffect(() => {
    // Si la búsqueda tiene menos de 3 caracteres, limpiar resultados y error
  })
}
```

```

        if (query.length < 3) {
            setMovies([]);
            setError("");
            return;
        }
        /**
         * Función asíncrona que obtiene las películas de la API.
         */
        async function fetchMovies() {
            try {
                setIsLoading(true); // Inicia el estado de carga
                setError(null); // Reinicia errores previos

                // Petición a la API de OMDB con la clave de acceso y la consulta
                const response = await
fetch(`http://www.omdbapi.com/?apikey=${API_KEY}&s=${query}`);

                // Verifica si la respuesta HTTP es correcta
                if (!response.ok)
                    throw new Error("Error al cargar películas");

                const data = await response.json();

                // Si la API responde con un error, lanzar una excepción
                if (data.Response === "False")
                    throw new Error("No se encontraron resultados");

                // Guardar las películas obtenidas en el estado
                setMovies(data.Search);
            } catch (err) {
                // Manejo de errores: guardar el mensaje de error y limpiar la lista
                setError(err.message);
                setMovies([]);
            } finally {
                setIsLoading(false); // Finaliza el estado de carga
            }
        }

        // Llamar a la función para obtener los datos
        fetchMovies();
    }, [query]); // Se ejecuta cada vez que cambia la consulta (query)

    // Retornar los valores necesarios para su uso en componentes
    return { movies, isLoading, error };
}

```

8. Crear un archivo con el nombre **useFetchMovieDetails.js** dentro del directorio **hooks**. Este archivo define un **custom hook** que se encarga de obtener los detalles de una película específica desde la API de **OMDb**, utilizando su identificador (`selectedId`).

Digita el siguiente código fuente dentro del archivo:

```
import { useEffect, useState } from "react";
import { API_KEY } from "./useFetchMovies"; // Importa la clave de API desde el otro
hook

/**
 * Hook personalizado para obtener los detalles de una película desde la API de OMDb.
 * @param {string} selectedId - ID único de la película seleccionada.
 * @returns {Object} - Retorna un objeto con:
 *   - movie: Detalles de la película.
 *   - isLoading: Estado de carga de la solicitud.
 *   - error: Mensaje de error en caso de fallo.
 */
export function useFetchMovieDetails(selectedId) {
    // Estado para almacenar los detalles de la película
    const [movie, setMovie] = useState({});

    // Estado para indicar si la solicitud está en curso
    const [isLoading, setIsLoading] = useState(false);

    // Estado para manejar errores
    const [error, setError] = useState("");

    useEffect(() => {
        // Si no hay un ID seleccionado, limpiar el estado
        if (!selectedId) {
            setMovie({});
            setError("");
            return;
        }

        /**
         * Función asíncrona que obtiene los detalles de la película.
         * @param {string} selectedId - ID único de la película a consultar.
         */
        async function fetchMovieDetails(selectedId) {
            try {
                setIsLoading(true); // Activa el estado de carga
                setError(null); // Reinicia errores previos

                // Petición a la API de OMDb con la clave de acceso y el ID de la
                // película
                const response = await

```

```

fetch(`http://www.omdbapi.com/?apikey=${API_KEY}&i=${selectedId}`);

        // Verifica si la respuesta HTTP es correcta
        if (!response.ok)
            throw new Error("Error al cargar los detalles de la película");

        const data = await response.json();

        // Guardar los detalles de la película en el estado
        setMovie(data);
    } catch (err) {
        // Manejo de errores: guardar el mensaje y limpiar el estado
        setError(err.message);
        setMovie({});
    } finally {
        setIsLoading(false); // Finaliza el estado de carga
    }
}

// Llamar a la función para obtener los datos
fetchMovieDetails(selectedId);
}, [selectedId]); // Se ejecuta cada vez que cambia el ID seleccionado
}

// Retornar los valores necesarios para su uso en componentes
return { movie, isLoading, error };
}

```

9. Crea un archivo llamado **Nav.jsx** en la carpeta **components**. Digita el siguiente código fuente dentro del archivo.

```

export const Nav = ({children}) => {
    return (
        <nav className="nav-bar">
            {children}
        </nav>
    )
}

export function Logo(){
    return(
        <div className="logo">
            <span role="img">🍿</span>
            <h1>Palomitas de papel</h1>
        </div>
    )
}

```

```

export function Search({query, setQuery}){
    return(
        <input
            className="search"
            type="text"
            placeholder="Buscar peliculas..."
            value={query}
            onChange={(e) => setQuery(e.target.value)}
        />
    )
}

export function NumResults({movies}){
    return(
        <p className="num-results">
            <strong>{movies.length}</strong> resultados encontrados
        </p>
    )
}

```

10. Crea un archivo llamado **Box.jsx** en la carpeta **components**. Digita el siguiente código fuente dentro del archivo.

```

import { useState } from "react";

export const Box = ({ children }) => {

    const [isOpen, setIsOpen] = useState(true);
    return (
        <div className="box">
            <button
                className="btn-toggle"
                onClick={() => setIsOpen((open) => !open)}>
                {isOpen ? "-" : "+"}
            </button>
            {isOpen && children}
        </div>
    )
}

```

11. Crea un archivo llamado **Movie.jsx** en la carpeta **components**. Digita el siguiente código fuente dentro del archivo.

```
/***
 * Componente que muestra una lista de películas.
 * @param {Object[]} movies - Lista de películas a renderizar.
 * @param {Function} onSelectMovie - Función que se ejecuta al seleccionar una
película.
 */
export const MovieList = ({ movies, onSelectMovie }) => {
    return (
        <ul className="list list-movies">
            {movies?.map((movie) => (
                <Movie
                    key={movie.imdbID}
                    movie={movie}
                    onSelectMovie={onSelectMovie}
                />
            ))}
        </ul>
    );
};

/***
 * Componente que muestra los detalles básicos de una película.
 * @param {Object} movie - Datos de la película.
 * @param {Function} onSelectMovie - Función que se ejecuta al hacer clic en la
película.
 */
export const Movie = ({ movie, onSelectMovie }) => {
    return (
        <li onClick={() => onSelectMovie(movie.imdbID)}>
            <img src={movie.Poster} alt={`${movie.Title} poster`} />
            <h3>{movie.Title}</h3>
            <div>
                <p>
                    <span>${movie.Rating}</span>
                    <span>{movie.Year}</span>
                </p>
            </div>
        </li>
    );
};
```

12. Crea un archivo llamado **StarRating.jsx** en la carpeta **components**. Digita el siguiente código fuente dentro del archivo.

```
import { useState } from "react";

// Estilos generales del contenedor de estrellas
const containerStyle = {
    display: "flex",
    alignItems: "center",
    gap: "16px"
};

// Estilos del contenedor de estrellas individuales
const starContainerStyle = {
    display: "flex",
    gap: "4px"
};

/***
 * Componente que muestra un sistema de calificación con estrellas interactivas.
 * @param {Object} props
 * @param {number} props.maxRating - Número máximo de estrellas (por defecto 5).
 * @param {string} props.color - Color de las estrellas (por defecto '#fcc419').
 * @param {number} props.size - Tamaño de las estrellas en píxeles (por defecto 30px).
 * @param {number} props.defaultRating - Calificación inicial seleccionada (por defecto 0).
 * @param {Function} props.onSetRating - Función que se ejecuta al seleccionar una calificación.
 */
export default function StarRating({
    maxRating = 5,
    color = '#fcc419',
    size = 30,
    defaultRating = 0,
    onSetRating
}) {
    // Estilos del texto de calificación
    const textStyle = {
        lineHeight: "1",
        margin: "0",
        color,
        fontSize: `${size}px`
    };
}
```

```

// Estado para almacenar la calificación seleccionada
const [rating, setRating] = useState(defaultRating);

// Estado temporal para manejar la calificación al pasar el mouse
const [tempRating, setTempRating] = useState(0);

/**
 * Maneja el evento de selección de una calificación.
 * @param {number} rating - Calificación seleccionada.
 */
function handleRating(rating) {
    setRating(rating);
    onSetRating?.(rating); // Llama a la función de callback si está definida
}

return (
    <div style={containerStyle}>
        {/* Contenedor de estrellas */}
        <div style={starContainerStyle}>
            {Array.from({ length: maxRating }, (_, i) => (
                <Star
                    key={i}
                    full={tempRating ? tempRating >= i + 1 : rating >= i + 1}
                    onRate={() => handleRating(i + 1)}
                    onHoverIn={() => setTempRating(i + 1)}
                    onHoverOut={() => setTempRating(0)}
                    color={color}
                    size={size}
                />
            ))}
        </div>
        {/* Muestra la calificación seleccionada o temporal */}
        <p style={textStyle}>{tempRating || rating || ""}</p>
    </div>
);
}

/**
 * Componente que representa una estrella individual en el sistema de calificación.
 * @param {Object} props
 * @param {boolean} props.full - Indica si la estrella está rellena o vacía.
 * @param {Function} props.onRate - Función que se ejecuta al hacer clic en la estrella.
 * @param {Function} props.onHoverIn - Función que se ejecuta al pasar el mouse sobre la estrella.
 * @param {Function} props.onHoverOut - Función que se ejecuta al quitar el mouse de la estrella.

```

```

 * @param {string} props.color - Color de la estrella.
 * @param {number} props.size - Tamaño de la estrella en píxeles.
 */
function Star({ full, onRate, onHoverIn, onHoverOut, color, size }) {
    // Estilos de la estrella
    const starStyle = {
        width: `${size}px`,
        height: `${size}px`,
        display: "block",
        cursor: "pointer"
    };

    return (
        <span
            role="button"
            style={starStyle}
            onClick={onRate}
            onMouseEnter={onHoverIn}
            onMouseLeave={onHoverOut}
        >
            {full ? (
                // Estrella rellena
                <svg
                    xmlns="http://www.w3.org/2000/svg"
                    viewBox="0 0 20 20"
                    fill={color}
                    stroke={color}
                >
                    <path
                        d="M9.049 2.927c.3-.921 1.603-.921 1.902 0l1.07 3.292a1 1 0
00.95.69h3.462c.969 0 1.371 1.24.588 1.81l-2.8 2.034a1 1 0 00-.364 1.118l1.07
3.292c.3.921-.755 1.688-1.54 1.118l-2.8-2.034a1 1 0 00-1.175 0l-2.8 2.034c-.784.57-
1.838-.197-1.539-1.118l1.07-3.292a1 1 0 00-.364-1.118L2.98 8.72c-.783-.57-.38-
1.81.588-1.81h3.461a1 1 0 00.951-.69l1.07-3.292z"
                    />
                </svg>
            ) : (
                // Estrella vacía
                <svg
                    xmlns="http://www.w3.org/2000/svg"
                    fill="none"
                    viewBox="0 0 24 24"
                    stroke={color}
                >
                    <path
                        strokeLinecap="round"
                        strokeLinejoin="round"

```

```

        strokeWidth="2"
        d="M11.049 2.927c.3-.921 1.603-.921 1.902 0l1.519 4.674a1 1 0
00.95.69h4.915c.969 0 1.371 1.24.588 1.81l-3.976 2.888a1 1 0 00-.363 1.118l1.518
4.674c.3.922-.755 1.688-1.538 1.118l-3.976-2.888a1 1 0 00-1.176 0l-3.976 2.888c-
.783.57-1.838-.197-1.538-1.118l1.518-4.674a1 1 0 00-.363-1.118l-3.976-2.888c-.784-.57-
.38-1.81.588-1.81h4.914a1 1 0 00.951-.69l1.519-4.674z"
      />
    </svg>
  )
</span>
);
}

```

13. Crea un archivo llamado **MovieDetails.jsx** en la carpeta **components**. Digita el siguiente código fuente dentro del archivo.

```

import { useEffect, useState } from 'react';
import { useFetchMovieDetails } from '../hooks/useFetchMovieDetails';
import StarRating from './StarRating';

/**
 * Componente que muestra los detalles de una película y permite al usuario
calificarla y agregarla a su lista de vistas.
 * @param {Object} props
 * @param {string} props.selectedId - ID de la película seleccionada.
 * @param {Function} props.onCloseMovie - Función para cerrar los detalles de la
película.
 * @param {Function} props.onAddWatched - Función para agregar la película a la lista
de vistas.
 * @param {Array} props.watched - Lista de películas ya vistas.
 */
export const MovieDetails = ({ selectedId, onCloseMovie, onAddWatched, watched }) => {

  // Hook personalizado para obtener los detalles de la película
  const { movie, error, isLoading } = useFetchMovieDetails(selectedId);

  // Extraemos la información relevante de la película
  const {
    Title: title,
    Year: year,
    Poster: poster,
    Runtime: runtime,
    imdbRating,
    Plot: plot,
    }

```

```

        Released: released,
        Actors: actors,
        Director: director,
        Genre: genre
    } = movie;

    // Estado para la calificación del usuario
    const [userRating, setUserRating] = useState('');

    // Verifica si la película ya está en la lista de vistas
    const isWatched = watched.some(movie => movie.imdbID === selectedId);

    // Obtiene la calificación previa del usuario si ya la ha visto
    const watchedUserRating = watched.find(movie => movie.imdbID ===
selectedId)?.userRating;

    /**
     * Maneja la adición de una película a la lista de vistas.
     */
    function handleAdd() {
        const newMovie = {
            imdbID: selectedId,
            title,
            year,
            poster,
            imdbRating: Number(imdbRating),
            runtime: Number(runtime.split(" ")[0]), // Extrae solo el número de mins
            userRating
        };

        onAddWatched(newMovie);
        onCloseMovie(); // Cierra los detalles después de agregar
    }

    return (
        <div className="details">
            {isLoading ? (
                <p className="loader">Cargando...</p>
            ) : (
                <>
                    <header>
                        <button className="btn-back" onClick={onCloseMovie}>
                            &larr;
                        </button>
                        <img src={poster} alt={`Poster de ${title}`} />
                        <div className="details-overview">
                            <h2>{title}</h2>

```

```

        <p>{released} &bull; {runtime}</p>
        <p>{genre}</p>
        <p><span>★</span>{imdbRating} IMDB rating</p>
    </div>
</header>
<section>
    <div className="rating">
        {!isWatched ? (
            <>
                {/* Calificación con estrellas */}
                <StarRating maxRating={10} size={18}
onSetRating={setUserRating} />
                {userRating > 0 && (
                    <button className="btn-add"
onClick={handleAdd}>
                        + Agregar a la lista
                    </button>
                )}
            </>
        ) : (
            <p>Has calificado esta película con
{watchedUserRating} ★</p>
        )}
    </div>
    <p><em>{plot}</em></p>
    <p><b>Elenco:</b> {actors}</p>
    <p><b>Director:</b> {director}</p>
</section>
</>
    )
</div>
);
};

}
};
```

14. Crea un archivo llamado **WatchedMovie.jsx** en la carpeta **components**. Digita el siguiente código fuente dentro del archivo.

```

export function WatchedMoviesContainer({ children }) {
    return <>{children}</>;
}

export function WatchedMoviesList({ watched }) {
    return (
        <ul className="list">
            {watched.map((movie) => (
                <WatchedMovie movie={movie} key={movie.imdbID} />
            ))
        </ul>
    );
}
```

```

        ))}
    </ul>
);
}

export function WatchedMovie({ movie }) {
    return (
        <li>
            <img src={movie.poster} alt={`${movie.title} poster`} />
            <h3>{movie.title}</h3>
            <div>
                <p>
                    <span>★</span>
                    <span>{movie.imdbRating}</span>
                </p>
                <p>
                    <span>✿</span>
                    <span>{movie.userRating}</span>
                </p>
                <p>
                    <span>⌚ </span>
                    <span>{movie.runtime} min</span>
                </p>
                <button className="btn-delete">X</button>
            </div>
        </li>
    );
}

/**
 * Calcula el promedio de un arreglo de números.
 * @param {number[]} arr - Arreglo de valores numéricos.
 * @returns {number} Promedio de los valores.
 */
const calculateAverage = (arr) =>
    arr.length ? arr.reduce((acc, cur) => acc + cur, 0) / arr.length : 0;

export function WatchedSummary({ watched }) {
    const avgImdbRating = calculateAverage(watched.map((movie) => movie.imdbRating));
    const avgUserRating = calculateAverage(watched.map((movie) => movie.userRating));
    const avgRuntime = calculateAverage(watched.map((movie) => movie.runtime));

    return (
        <div className="summary">
            <h2>Películas que has visto</h2>
            <div>

```

```

        <p>
          <span>${watched.length} películas</span>
        </p>
        <p>
          <span>⭐</span>
          <span>{avgImdbRating.toFixed(2)}</span>
        </p>
        <p>
          <span>🌟</span>
          <span>{avgUserRating.toFixed(2)}</span>
        </p>
        <p>
          <span>⏳</span>
          <span>{avgRuntime.toFixed(2)} min</span>
        </p>
      </div>
    </div>
  );
}
}

```

15. Crea un archivo llamado **App.jsx** en la carpeta **src** del proyecto. Digita el siguiente código fuente dentro del archivo.

```

import { useState } from "react";
import { Logo, Nav, NumResults, Search } from "./components/Nav";
import { Box } from "./components/Box";
import { MovieList } from "./components/Movie";
import { WatchedMoviesContainer, WatchedMoviesList, WatchedSummary } from
"./components/WatchedMovie";
import { useFetchMovies } from "./hooks/useFetchMovies";
import { MovieDetails } from "./components/MovieDetails";

/**
 * Componente principal de la aplicación.
 */
export default function App() {

  // Estado para la búsqueda de películas
  const [query, setQuery] = useState("");

  // Obtiene películas basadas en la consulta
  const { movies, isLoading, error } = useFetchMovies(query);

  // Estado de películas vistas

```

```

const [watched, setWatched] = useState([]);

// Estado para la película seleccionada
const [selectedId, setSelectedId] = useState(null);

/**
 * Maneja la selección de una película.
 * @param {string} id - ID de la película seleccionada.
 */
function handleSelectMovie(id) {
  setSelectedId(id);
}

/**
 * Cierra los detalles de la película.
 */
function handleCloseMovie() {
  setSelectedId(null);
}

/**
 * Agrega una película a la lista de vistas.
 * @param {Object} movie - Película a agregar.
 */
function handleAddWatched(movie) {
  setWatched((watched) => [...watched, movie]);
}

return (
  <>
  <Nav>
    <Logo />
    <Search query={query} setQuery={setQuery} />
    <NumResults movies={movies} />
  </Nav>
  <main className="main">
    <Box>
      {isLoading && <p className="loader">Cargando...</p>}
      {error && <p className="error">⊖ {error}</p>}
      <MovieList movies={movies} onSelectMovie={handleSelectMovie} />
    </Box>

    <Box>
      <WatchedMoviesContainer>
        {selectedId ? (
          <MovieDetails
            selectedId={selectedId}

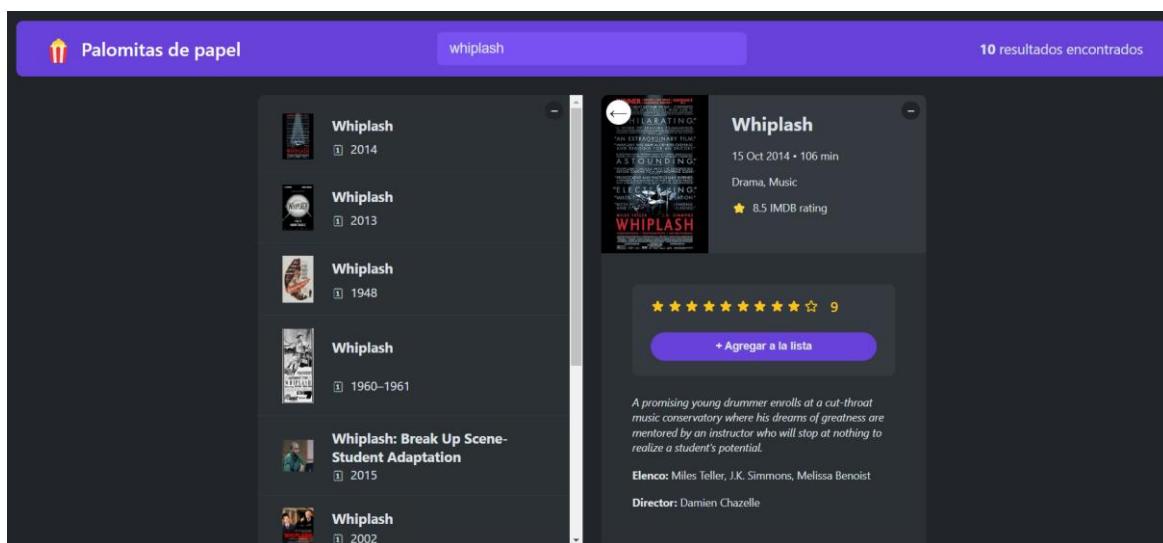
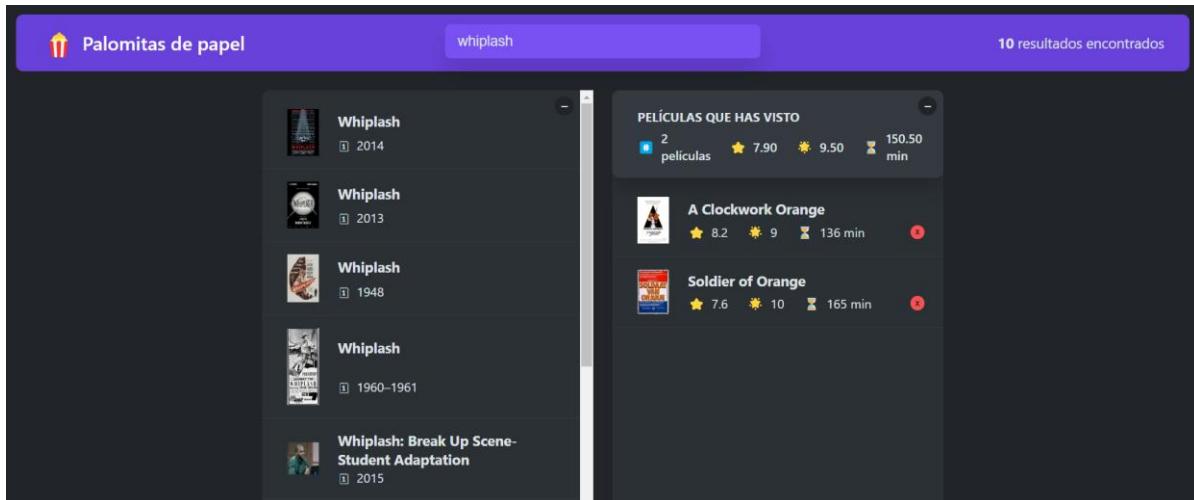
```

```

        onCloseMovie={handleCloseMovie}
        onAddWatched={handleAddWatched}
        watched={watched}
      />
    ) : (
      <>
        <WatchedSummary watched={watched} />
        <WatchedMoviesList watched={watched} />
      </>
    )
  }
</WatchedMoviesContainer>
</Box>
</main>
</>
);
}

```

16. Pruebe su aplicación la cual debería lucir de la siguiente manera:



The screenshot shows a search interface for the movie 'Whiplash'. The search bar at the top has the text 'whiplash'. Below the search bar, there are two main sections: a list of search results on the left and a 'WatchedMoviesList' summary on the right.

Search Results:

- Whiplash (2014)
- Whiplash (2013)
- Whiplash (1948)
- Whiplash (1960–1961)
- Whiplash: Break Up Scene-Student Adaptation (2015)

WatchedMoviesList Summary:

PELÍCULAS QUE HAS VISTO

- 3 películas ★ 8.10 ★ 9.33 135.67 min
 - A Clockwork Orange (★ 8.2, ★ 9, 136 min)
 - Soldier of Orange (★ 7.6, ★ 10, 165 min)
 - Whiplash (★ 8.5, ★ 9, 106 min)

This screenshot shows the same search interface for 'whiplash', but the right panel is now displaying detailed information for the 2014 movie 'Whiplash'.

Movie Details:

Whiplash
15 Oct 2014 • 106 min
Drama, Music
★ 8.5 IMDB rating

Rating Confirmation:
Has calificado esta película con 9 ★

Plot Summary:
A promising young drummer enrolls at a cut-throat music conservatory where his dreams of greatness are mentored by an instructor who will stop at nothing to realize a student's potential.

Credits:
Elenco: Miles Teller, J.K. Simmons, Melissa Benoist
Director: Damien Chazelle

VI. Ejercicios complementarios

1. Implemente la funcionalidad de quitar una película de la lista WatchedMoviesList al presionar el botón
2. Almacene de forma persistente las películas vistas utilizando localStorage.

Todos los archivos de la guía deben agregarse a un repositorio de GitHub y luego deberá hacerse el deploy a Netlify.

Fecha límite de entrega: martes 10 de febrero (5% de la nota final)