Facultad de Informática
Programación Web Avanzada

Trabajo Práctico: REACT

Desarrollar la siguiente práctica utilizando la librería react.dev

Gestor de Películas y Series por Ver y Vistas

Construir en React una aplicación que funcione como un gestor personal de películas y series.
La aplicación debe permitir al usuario agregar nuevas películas o series, marcarlas como vistas, editarlas y eliminarlas (con confirmación).
Cada ítem debe tener título, director, año, género (usar un select para este campo), rating y tipo (película o serie).
La aplicación debe mostrar dos listas: una con contenido por ver y otra con contenido visto.

Debe haber:

- Un contador que muestre la cantidad total por lista y por género.
- Un input de búsqueda para filtrar por título o director.
- Filtros para género y tipo.
- Opciones de ordenamiento por año y rating, ascendente y descendente.
- Si no hay ítems en una lista, debe mostrarse un mensaje indicando que está vacía.
- Si no hay resultados que coincidan con los filtros, también debe mostrarse un mensaje adecuado.

Todos los datos deben guardarse en el navegador usando localStorage, para que no se pierdan al actualizar la página.

Recuerden que el desarrollo de software es un ciclo iterativo, no es necesario definir todas los componentes e implementar todas las funcionalidades en la primera iteración. Es una práctica normal agregar más componentes y refactorizar en la mitad del trabajo.

No hay restricciones ni pautas en cuanto a cómo se debe ver, sean creativos y apliquen lo aprendido en css para llevar adelante sus ideas.

1) Crear un proyecto react y agregarlo a un repositorio de Github o crear un repositorio en github, clonarlo y luego crear el proyecto.
    a) Compartir el link del repositorio con los profesores.

2) Armar un archivo readme.md en el cual se detallen los miembros del grupo y funcione como carátula del repositorio.
    a) En el archivo readme.md:
        i) Explicar la función de los siguientes archivos iniciales: index.js, App.js, index.css y package-json.js.

3) Crear una carpeta pages en la raíz del proyecto y dentro de ella otra carpeta Home. Dentro de esta última, un archivo Home.js y Home.module.css de ser necesario estilar. Renderizar home dentro del componente App.js.

4) Crear una carpeta Components en la raíz del proyecto. Dados los requerimientos de la aplicación. ¿Qué componentes reutilizables cree que debería definir?
    a) Para cada componente crear una carpeta con el nombre del componente.

5) Crear un componente Titulo dentro de la carpeta components, el cual es un <h1> con un estilo particular y que recibe por props un texto a renderizar. Utilizarlo desde Home.js como título para nuestra aplicación.

6) Completar los requerimientos de la aplicación. Tener en cuenta:
    a) Utilizar componentes reutilizables. Evitar tener código duplicado.
    b) Utilizar el hook useState para poder almacenar el estado interno de las listas de peliculas/series. Qué otros elementos de nuestra aplicación necesitan un useState? Utilizar todos los estados que crea necesarios para cumplir los requerimientos.
    c) Utilizar el renderizado de listas con la función .map() de JavaScript.
    d) Utilizar el renderizado condicional. Si no hay peliculas/series por ver hay que mostrar un mensaje, caso contrario mostrar la lista de peliculas/series.
    e) Utilizar botones para poder cambiar el estado de las peliculas/series o eliminarlas.

7) Actualizar el archivo readme.md para tener una documentacion adecuada:
    a) Armar una carátula con los datos de los miembros del grupo.
    b) Incluir una descripción básica de la aplicación.
    c) Incluir una guía e instrucciones de instalación paso a paso (clonar el repositorio - correr el comando `npm i`…)
    d) Agregar capturas de pantalla.
    e) Cualquier otra información que crean relevante.

Facultad de Informática
Programación Web Avanzada

Estructura de archivos y carpetas recomendada:

```
-Pages
        —Home
                —Home.js
                —Styles.css
-Components
        —Button
                —Button.js
                —Styles.css
        —Title
                —Title.js
                —Styles.css
```

Componentes Reutilizables:

En el desarrollo de software, la reutilización de código es una práctica fundamental que nos permite escribir menos código duplicado y mantener una base de código más limpia y modular. Los componentes reutilizables son una estrategia de aplicar este principio en el desarrollo de aplicaciones web con React.

Un componente reutilizable es un bloque de código independiente que encapsula una funcionalidad específica y puede ser utilizado en múltiples partes de una aplicación. Al crear componentes reutilizables, estamos construyendo piezas modulares de nuestra aplicación que pueden ser fácilmente integradas y adaptadas según las necesidades.

Algunas de las ventajas de utilizar componentes reutilizables incluyen:

- 1. **Modularidad:** Los componentes reutilizables nos permiten dividir nuestra aplicación en partes más pequeñas y manejables, lo que facilita la comprensión y el mantenimiento del código.
- 2. **Facilidad de mantenimiento:** Al tener componentes independientes y bien definidos, es más fácil realizar cambios y actualizaciones en la aplicación sin afectar otras partes del código.
- 3. **Mejora de la productividad:** La reutilización de componentes reduce la cantidad de código que necesitamos escribir, lo que nos permite desarrollar más rápidamente y con menos errores.
- 4. **Consistencia**: Utilizar los mismos componentes en diferentes partes de la aplicación garantiza una experiencia de usuario consistente y coherente.

Para identificar qué partes de nuestra aplicación pueden ser convertidas en componentes reutilizables, es útil buscar patrones comunes o funcionalidades que se repiten en varios lugares. Por ejemplo, en nuestra aplicación de lista de peliculas/series, podríamos identificar el componente de la pelicula/serie individual como un buen candidato para la reutilización, ya que se mostrará en múltiples lugares dentro de la aplicación.

Al diseñar componentes reutilizables, es importante pensar en cómo pueden ser configurados y personalizados mediante props para adaptarse a diferentes contextos de uso. Esto nos permite crear componentes flexibles y versátiles que pueden ser utilizados en una variedad de situaciones.