28 – Filtros y Ordenamiento

Introducción

En esta clase aprenderás a implementar tres funcionalidades esenciales en cualquier aplicación de listado de productos: **ordenamiento**, **filtros** y **búsqueda**.
Estas herramientas permiten mejorar la experiencia del usuario, facilitando la exploración de los datos de forma dinámica e intuitiva. Trabajaremos con **Redux Toolkit** para manejar el estado global y con **React** para la interfaz de usuario, aplicando buenas prácticas de modularización y separación de responsabilidades.

1. Ordenamiento de productos

Concepto

El **ordenamiento** permite organizar los productos según un criterio específico (por ejemplo, por precio o nombre). En este caso, aprenderemos a ordenarlos **de menor a mayor** y **de mayor a menor** según el precio.

Implementación en productsSlice.js

Creamos una acción llamada sortProducts dentro del slice de productos. Esta acción recibirá un valor (asc o desc) para definir el tipo de ordenamiento.

```
import { createSlice } from "@reduxjs/toolkit";

const productsSlice = createSlice({
  name: "products",
  initialState: {
    allProducts: [],
    productsCopy: [],
  },
  reducers: {
    setAllProducts: (state, action) => {
      state.allProducts = action.payload;
      state.productsCopy = action.payload;
    },
    updateProductsCopy: (state, action) => {
      state.productsCopy = action.payload;
    },
    sortProducts: (state, action) => {
      const sortedProducts = [...state.productsCopy].sort((a, b) => {
        return action.payload === "asc" ? a.price - b.price : b.price -
a.price;
      });
      state.productsCopy = sortedProducts;
    },
  },
});

export const { setAllProducts, updateProductsCopy, sortProducts } =
productsSlice.actions;
export default productsSlice.reducer;
```

Explicación

- **setAllProducts:** inicializa los estados allProducts y productsCopy.

- **updateProductsCopy:** actualiza la copia de productos mostrada.
- **sortProducts:** ordena los productos en productsCopy según el precio:
  - "asc": orden ascendente (menor a mayor).
  - "desc": orden descendente (mayor a menor).

Aplicación en Sidebar.jsx

Agregamos un select que permita al usuario elegir el tipo de ordenamiento.

```
const Sidebar = () => {
  const dispatch = useDispatch();

  const handleSort = (e) => {
    dispatch(sortProducts(e.target.value));
  };

  return (
    <div className={styles.sidebar}>
      <h3>Ordenar por Precio</h3>
      <select onChange={handleSort}>
        <option value="asc">Menor a mayor</option>
        <option value="desc">Mayor a menor</option>
      </select>
    </div>
  );
};
```

Explicación

- **handleSort:** se ejecuta cuando el usuario selecciona una opción. Envía la acción sortProducts con el valor elegido.

2. Filtro por categoría

Concepto

El **filtro** permite mostrar solo los productos que cumplen con una condición específica, como pertenecer a una determinada categoría. Esto mejora la usabilidad, evitando que el usuario deba recorrer todos los productos.

Implementación en productsSlice.js

Agregamos una acción filterByCategory dentro del slice:

```
filterByCategory: (state, action) => {
  const filteredByCategory = state.allProducts.filter((product) => {
    return product.category.includes(action.payload);
  });
  state.productsCopy = action.payload ? filteredByCategory :
state.allProducts;
},
```

Explicación

- Filtra los productos de allProducts según la categoría seleccionada.

- Si no se elige ninguna categoría, se muestran todos los productos.

Aplicación en Sidebar.jsx

1. Obtenemos las categorías únicas.
2. Las mostramos en un select que el usuario puede usar para filtrar.

```
const categories = Array.from(
  new Set(allProducts.map((product) => product.category))
);

const handleCategoryChange = (e) => {
  dispatch(filterByCategory(e.target.value));
};

return (
  <>
    <h4>Filtrar por Categoría</h4>
    <select onChange={handleCategoryChange}>
      {categories.map((category) => (
        <option key={category} value={category}>
          {category}
        </option>
      ))}
    </select>
  </>
);
```

Explicación

- **categories:** crea un array con las categorías sin repetir.
- **handleCategoryChange:** despacha la acción filterByCategory con la categoría seleccionada.

3. Búsqueda por título

Concepto

La **búsqueda** permite al usuario encontrar rápidamente productos específicos ingresando una palabra clave.
Se implementa mediante un campo de texto controlado que filtra los productos en tiempo real.

Implementación en productsSlice.js

```
searchByTitle: (state, action) => {
  const searchedProducts = state.allProducts.filter((product) =>
    product.title.toLowerCase().includes(action.payload.toLowerCase())
  );
  state.productsCopy =
    searchedProducts.length > 0 ? searchedProducts : state.allProducts;
},
```

Explicación

- Convierte el texto de búsqueda y los títulos a minúsculas para evitar problemas de mayúsculas/minúsculas.
- Si no se encuentran coincidencias, muestra todos los productos.

Aplicación en Sidebar.jsx

```
const [searchTerm, setSearchTerm] = useState("");

const handleSearch = (e) => {
  setSearchTerm(e.target.value);
  dispatch(searchByTitle(e.target.value));
};

return (
  <>
    <h3>Buscar producto</h3>
    <input
      type="text"
      value={searchTerm}
      onChange={handleSearch}
      placeholder="Buscar por título"
    />
  </>
);
```

Explicación

- **searchTerm:** almacena el valor actual del campo de búsqueda.
- **handleSearch:** actualiza el estado local y despacha la acción searchByTitle.

4. Botón "Limpiar filtros"

Agregamos un botón que restaure la lista completa de productos.

```
const allProducts = useSelector((state) => state.products.allProducts);

const handlerClear = () => {
  dispatch(updateProductsCopy(allProducts));
};

<button onClick={handlerClear}>Limpiar</button>;
```

Explicación

- **updateProductsCopy:** vuelve a mostrar todos los productos originales.

Conclusión

En esta clase aprendiste a:

- Ordenar productos por precio.
- Filtrar productos por categoría.
- Buscar productos por título.

- Limpiar los filtros y restaurar la vista completa.

Estas funcionalidades mejoran significativamente la experiencia del usuario en una aplicación de tipo e-commerce y sientan las bases para incorporar sistemas más avanzados de filtrado combinatorio y paginación.