Tarea Semana 3

Desarrollo Web con React y Ruby on Rails

Proyecto carrito de Compras

Objetivo

Desarrollar una aplicación de carrito de compras que permita a los usuarios ver productos, añadir/eliminar/actualizar ítems en el carrito, y persistir el estado del carrito utilizando React, TypeScript y los hooks principales.

Requisitos

- useState: Para el estado de la UI
- useEffect: Para efectos secundarios
- useReducer: Para la gestión del estado complejo del carrito.

Fecha de Entrega

Domingo 8 de Junio

Entrega

- Deben entregar un archivo de texto con un enlace a un repositorio de GitHub que tenga el proyecto.
    - Sino han usado github, contactenme para darles guía y apoyo en como utilizarlo
- Un archivo README.md en el repositorio explicando cómo ejecutar la aplicación, cómo se utilizaron los hooks y explicando lo que hace el app y las funcionalidades que tiene. (A continuación dejo guías de uso)
    - https://www.makeareadme.com/
    - https://docs.github.com/es/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Implementación

Paso 1: Configuración del Proyecto y Tipado de Datos

1. **Inicialización del Proyecto:**
    - Crear un nuevo proyecto de React con soporte para TypeScript utilizando tu herramienta `Vite`:
      ```
      npm create vite@latest mi-carrito-ts -- --template react-ts
      ```
    - Seguir los pasos para instalar el proyecto y correrlo
2. **Definición de Tipos (en `src/types.ts` o `src/models/`):**

- Crear una interfaz `Producto` con las siguientes propiedades:
  - `id: string`
  - `nombre: string`
  - `precio: number`
  - `descripcion: string`
  - `imagen?: string` (opcional, para una URL de imagen)
- Crea una interfaz `ItemCarrito` para los elementos dentro del carrito:
  - `producto: Producto` (el objeto producto completo)
  - `cantidad: number`
- Define los tipos para las acciones del `useReducer` del carrito:
  - `TipoAccionCarrito`: Un tipo de unión de cadenas para los tipos de acciones (ej. `'ADD_ITEM'`, `'REMOVE_ITEM'`, `'UPDATE_QUANTITY'`).
  - `AccionCarrito`: Una interfaz de unión que represente cada posible acción con su `type` y `payload` (si es necesario).
    - Ej: `{ type: 'ADD_ITEM'; payload: Producto; }`
    - Ej: `{ type: 'REMOVE_ITEM'; payload: string; }` (el `id` del producto)
    - Ej: `{ type: 'UPDATE_QUANTITY'; payload: { productId: string; quantity: number; }; }`

Paso 2: Gestión del Estado del Carrito con `useReducer`

3. **Configuración del Reducer del Carrito:**

   - Cree un nuevo archivo para el reducer (por ejemplo: `src/reducers/cartReducer.ts`).
   - Define el estado inicial del carrito (un array vacío de `ItemCarrito`).
   - Implementa la función `cartReducer(state: ItemCarrito[], action: AccionCarrito)`:
     - Utilice un `switch` para manejar los diferentes `TipoAccionCarrito`.
     - **`ADD_ITEM`**:
       - Si el producto ya está en el carrito, incrementa su `cantidad`.
       - Si no está, añádelo con `cantidad: 1`.
       - **Importante**: Siempre devuelve un *nuevo* array y nuevos objetos `ItemCarrito` para asegurar la inmutabilidad del estado.
     - **`REMOVE_ITEM`**:
       - Filtra el array para eliminar el `ItemCarrito` con el `id` del producto especificado.
     - **`UPDATE_QUANTITY`**:

        - Mapea el array de `ItemCarrito`. Cuando encuentres el `productId` coincidente, actualiza su `cantidad` al `payload.quantity` provisto.
        - Si la `cantidad` resultante es `0` o menos, elimina el ítem del carrito.
4. **Integración del Reducer en el `App.tsx`:**
    - Importar la función `cartReducer` y el estado inicial.
    - Utiliza el hook `useReducer` para gestionar el estado del carrito:
      TypeScript

```
TypeScript

const [cartState, dispatch] = useReducer(cartReducer,
initialState);
```

    - Cree un array de datos de productos de ejemplo..

Paso 3: Persistencia del Carrito con `useEffect`

5. **Guardar el Carrito en `localStorage`:**
    - En `App.tsx`, implementa un `useEffect` para guardar el `cartState` en `localStorage` cada vez que cambie.
    - Serializa el `cartState` usando un JSON (método `JSON.stringify`).
    - Asegúrate de que este `useEffect` tenga `[cartState]` en el array de dependencias.
6. **Cargar el Carrito desde `localStorage`:**
    - Modificar el `initialState` de tu `useReducer` para que intente cargar el carrito desde `localStorage` primero.
    - Si hay datos guardados, parsearlos o convertirlos de JSON a texto o tipo (`JSON.parse`) para usarlos como estado inicial.
    - Si no hay datos o hay un error al parsear, usa un array vacío.
    - ** Pueden investigar y usar el lazy loading para `useReducer` e implementarlo para esto, o manejarlo directamente en la definición del estado inicial.

Paso 4: Mostrar Productos y Añadir al Carrito (`useState`, `props`)

7. **Componente `ListaProductos`:**
    - Crea un componente funcional `ListaProductos.tsx`.
    - Recibe una prop `productos: Producto[]` (tipada).
    - Recibe una prop `onAddToCart: (producto: Producto) => void`
    - Mapea sobre el array `productos` y renderiza un `TarjetaProducto` para cada uno.
8. **Componente `TarjetaProducto`:**
    - Crea un componente funcional `TarjetaProducto.tsx`.
    - Recibe una prop `producto: Producto` (tipada).
    - Recibe una prop `onAddToCart: (producto: Producto) => void`
    - Muestra la información del producto (nombre, precio, descripción).
    - Incluye un botón "Añadir al Carrito" que, al hacer clic, llame a `onAddToCart` pasando el `producto` completo.
9. **Integración en `App.tsx`:**
    - Renderiza `ListaProductos` en `App.tsx`.
    - Pasa el array de `productos` de ejemplo a `ListaProductos`.
    - Pasa una función `handleAddToCart` (definida en `App.tsx`) a `ListaProductos`. Esta función `handleAddToCart` recibirá un `producto` y debe usar `dispatch({ type: 'ADD_ITEM', payload: producto })` para actualizar el estado del carrito.

Paso 5: Mostrar y Gestionar el Carrito (`props`)

10. **Componente `CarritoCompras`:**
    - Crea un componente funcional `CarritoCompras.tsx`.
    - Recibe una prop `items: ItemCarrito[]` (tipada).
    - Recibe una prop `dispatch: React.Dispatch<AccionCarrito>` (tipada, pasada directamente desde `useReducer`).
    - Muestra una tabla o lista de los `items` en el carrito. Para cada `ItemCarrito`:
        - Muestra el nombre del producto, la cantidad y el precio unitario.
        - Calcula y muestra el subtotal del ítem (`cantidad * precio`).
        - Incluye botones o inputs para:

        - Incrementar cantidad (dispara `dispatch({ type: 'UPDATE_QUANTITY', payload: { productId, quantity: currentQuantity + 1 } })`).
        - Decrementar cantidad (dispara `dispatch({ type: 'UPDATE_QUANTITY', payload: { productId, quantity: currentQuantity - 1 } })`).
        - Eliminar ítem (dispara `dispatch({ type: 'REMOVE_ITEM', payload: productId })`).
    - Calcula y muestra el "Total del Carrito" al final de la lista.
11. **Integración en `App.tsx`:**

    - Renderiza `CarritoCompras` en `App.tsx`.
    - Pasa el `cartState` (el estado actual del carrito) como prop `items`.
    - Pasa la función `dispatch` directamente como prop `dispatch`.

Paso 6: Búsqueda de Productos (`useState`)

12. **Componente `BarraBusqueda`:**

    - Crea un componente funcional `BarraBusqueda.tsx`.
    - Incluye un `<input type="text">` para el término de búsqueda.
    - Acepta una prop `onSearchChange: (searchTerm: string) => void` (tipada).
    - Utiliza el `useState` interno para gestionar el valor del input y llama a `onSearchChange` cuando el valor cambie.
13. **Lógica de Búsqueda en `App.tsx`:**

    - En `App.tsx`, añade un nuevo estado `searchTerm: string` utilizando `useState`, inicializado como cadena vacía.
    - Crea una función `handleSearchChange: (term: string) => void` que actualice el estado `searchTerm`.
    - Pasa `BarraBusqueda` y pasa `handleSearchChange` como prop.
    - Modifica el array de `productos` que pasas a `ListaProductos`. Antes de pasarlos, fíltralos basándote en el `searchTerm` actual (ej. `productos.filter(p => p.nombre.toLowerCase().includes(searchTerm.toLowerCase()))`).

Recursos adicionales

https://www.youtube.com/watch?v=-eUYrsToD-E
https://www.freecodecamp.org/espanol/news/react-crud-app-tutorial-como-construir-una-aplicacion-de-administracion-de-libros-en-react-desde-cero/