Your current Redux implementation is functional, but there are **some side effects and performance issues** that we can optimize. Below are the **problems** and **best practices to improve performance**:

**🔴 Problems & Side Effects:**

**1️⃣ State Mutation in Redux (Bad Practice)**

* **Problem:** Your reducers modify the state directly (state.cartItems.push(...) and existingItem.quantity++).
* **Side Effect:** Redux Toolkit (RTK) uses Immer.js to handle immutability internally, but directly modifying state is still a bad practice as it can cause unpredictable behavior in some cases.
* **Solution:** Always return a new copy of the state instead of mutating it.

**2️⃣ Performance Issues with find() and filter()**

* **Problem:** state.cartItems.find(item => item.id === newItem.id) runs a **linear search (O(n))** on every addItemToCart() and removeItemFromCart().
* **Side Effect:** If the cart has many items, performance **degrades significantly**.
* **Solution:** Use an **object (hash map) instead of an array** for quick lookups (O(1) complexity).

**3️⃣ Unnecessary Re-Renders in React**

* **Problem:** Every time **any Redux state** changes, components **re-render**, even if they don’t need to.
* **Side Effect:** React may be rendering components that do not need updates.
* **Solution:** Use:
  + useSelector(state => state.cart.cartItems, shallowEqual) to prevent unnecessary re-renders.
  + Memoization (useMemo) and useCallback to optimize function calls.

**✅ Optimized Code for Performance**

**📌 Fix 1: Use an Object for Fast Lookups**

Instead of using an **array**, store cartItems in an **object (dictionary)** using id as the key.

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

const cartSlice = createSlice({

name: "cart",

initialState: {

cartItems: {}, // ✅ Store items as an object instead of an array

totalQuantity: 0,

},

reducers: {

addItemToCart(state, action) {

const newItem = action.payload;

if (!state.cartItems[newItem.id]) {

state.cartItems[newItem.id] = {

id: newItem.id,

price: newItem.price,

quantity: 1,

totalPrice: newItem.price,

name: newItem.title,

};

} else {

state.cartItems[newItem.id].quantity++;

state.cartItems[newItem.id].totalPrice += newItem.price;

}

state.totalQuantity++;

},

removeItemFromCart(state, action) {

const id = action.payload;

if (state.cartItems[id]) {

if (state.cartItems[id].quantity === 1) {

delete state.cartItems[id]; // ✅ Instead of filtering an array, use `delete`

} else {

state.cartItems[id].quantity--;

state.cartItems[id].totalPrice -= state.cartItems[id].price;

}

state.totalQuantity--;

}

},

},

});

export const CartActions = cartSlice.actions;

export default cartSlice.reducer;

✅ **Performance Boost:**

* Instead of iterating through an array (O(n) complexity), accessing an object key is **O(1) (constant time)**.
* Deleting an item is **faster** than filtering an array.

**📌 Fix 2: Optimize useSelector in Cart.jsx**

import React from "react";

import { useSelector, useDispatch, shallowEqual } from "react-redux";

import { CartActions } from "../store/cart-slice";

export default function Cart() {

const cartItems = useSelector((state) => Object.values(state.cart.cartItems), shallowEqual); // ✅ Prevents unnecessary re-renders

const dispatch = useDispatch();

const handleRemoveFromCart = (id) => {

dispatch(CartActions.removeItemFromCart(id));

};

return (

<div className="cart">

{cartItems.length === 0 ? (

<p>Cart is empty</p>

) : (

cartItems.map((p) => (

<div key={p.id} className="cart-item">

<h4>{p.name}</h4>

<p>Price: {p.price}$</p>

<p>Quantity: {p.quantity}</p>

<p>Total: {p.totalPrice}$</p>

<button onClick={() => handleRemoveFromCart(p.id)}>Remove</button>

</div>

))

)}

</div>

);

}

✅ **Performance Boost:**

* **shallowEqual** prevents re-renders if the cartItems object structure **remains unchanged**.
* **Object.values(state.cart.cartItems)** converts the object back into an array for display **only in the UI**, without impacting performance.

**📌 Fix 3: Optimize Header.jsx for Better React Performance**

import React from "react";

import { useDispatch, useSelector } from "react-redux";

import { uiActions } from "../store/ui-slice";

export default function Header() {

const dispatch = useDispatch();

const handleShowCart = () => {

dispatch(uiActions.toggle());

};

// ✅ Prevent unnecessary re-renders by only selecting `totalQuantity`

const quantity = useSelector((state) => state.cart.totalQuantity);

return (

<div className="header">

<button onClick={handleShowCart}>Show Cart</button>

<span>Count: {quantity}</span>

</div>

);

}

✅ **Performance Boost:**

* useSelector(state => state.cart.totalQuantity) ensures that **only totalQuantity updates trigger re-renders**, not the entire cart.

**🚀 Summary of Optimizations**

| **Issue** | **Old Code** | **Optimized Code** |
| --- | --- | --- |
| **State Mutation** | Directly modified state | Used object instead of array |
| **Performance (O(n) Lookups)** | Used .find() and .filter() on an array | Used an object for O(1) lookups |
| **Unnecessary Re-renders** | useSelector(state => state.cart.cartItems) (causing deep comparison) | Used useSelector(state => Object.values(state.cart.cartItems), shallowEqual) |
| **Header Performance** | useSelector(state => state.cart) (caused full re-renders) | useSelector(state => state.cart.totalQuantity) |

**🔥 Now Your Redux Code is More Efficient!**

* **🚀 Faster Add/Remove Operations** (O(1) instead of O(n))
* **📉 Fewer Re-renders** → React renders only the components that need updates
* **🛠️ More Maintainable & Scalable Code**