Experiment: Implement Redux Toolkit for State Management in Shopping Cart

Snigdha Ghosh(23ICS10004)

Objective:

Learn how to use Redux Toolkit to manage global application state in a React project, focusing on implementing a shopping cart. This helps understand slices, actions, reducers, and connecting React components to a Redux store efficiently using modern Redux Toolkit practices.

Task Description:

1. Create a React application that displays a list of products.
2. Allow users to add products to a shopping cart.
3. Use Redux Toolkit to manage the cart state globally by creating a slice with actions for adding,removing, and updating quantities.
4. Create a Cart component to display items, name, price, and quantity.
5. Connect components to the Redux store using **useSelector** and **useDispatch** hooks.
6. Test the app by adding, updating, and removing products, ensuring that the cart updates inreal-time.

Redux Slice (cartSlice.js):

```
// cartSlice.js import { createSlice } from
'@reduxjs/toolkit';

const cartSlice = createSlice({ name: 'cart', initialState:
[], reducers: { addItem: (state, action) => { const item =
state.find(i => i.name === action.payload.name); if (item) {
item.quantity += 1; } else { state.push({ ...action.payload,
quantity: 1 }); } }, removeItem: (state, action) => {
return state.filter(item => item.name !== action.payload);
}, updateQuantity: (state, action) => { const { name,
quantity } = action.payload; const item = state.find(i =>
i.name === name); if (item) item.quantity = quantity; }
}
});

export const { addItem, removeItem, updateQuantity } = cartSlice.actions;
export default cartSlice.reducer;
```

Redux Store (store.js):

```
// store.js import { configureStore } from
'@reduxjs/toolkit'; import cartReducer from
'./cartSlice';

export const store = configureStore({
reducer: { cart: cartReducer, },
});
```

Product Component (ProductList.js):

```
// ProductList.js import React from
'react'; import { useDispatch } from
'react-redux'; import { addItem } from
'./cartSlice';

const products = [ { name:
"Laptop", price: 1200 },
{ name: "Mouse", price: 25 },
{ name: "Keyboard", price: 45 }
];

function ProductList() { const

dispatch = useDispatch();

return ( Products

{products.map((p, index) => (

{p.name} ${p.price}

dispatch(addItem(p))}>Add to Cart

))}


);
}

export default ProductList;
```

Cart Component (Cart.js):

```
// Cart.js import React from 'react'; import {
useSelector, useDispatch } from 'react-redux'; import {
removeItem, updateQuantity } from './cartSlice';

function Cart() { const cart =
useSelector(state => state.cart); const
dispatch = useDispatch();

return (

Shopping Cart
{cart.map((item, index) => (

{item.name} (${item.price}) type="number" value={item.quantity}
onChange={(e) => dispatch(updateQuantity({ name: item.name, quantity:
Number(e.target.value) }))} style={{ width: '50px', margin: '0 10px'
}} /> dispatch(removeItem(item.name))}>Remove ))}

);
}

export default Cart;
```

Expected Output:

A React shopping cart application displaying products and a cart section where users can add, remove, and update quantities of items. The cart state is managed globally using Redux Toolkit.

Conclusion:

This experiment demonstrates how Redux Toolkit simplifies state management in React applications. By creating slices and using actions and reducers, developers can efficiently manage and synchronize global states such as shopping carts.