Experiment 3

Aim: Manage complex state with Redux or Context API

Theory:

When building applications, especially ones with multiple components that need to share and update data, **state management** becomes crucial. Simple local state (`useState`) works for small apps, but as the app grows, managing state across many components can become **messy and error-prone**. This is where **Redux** and **Context API** come in.

---

1. Redux

Redux is a **predictable state container** for JavaScript apps.

- **Centralized State:** All your application's state is stored in a single **store**. This makes it easier to manage, debug, and trace changes.
- **Unidirectional Data Flow:** Components dispatch **actions** → the **reducers** update the state → the UI reflects the new state. This flow is predictable and avoids unexpected state changes.
- **Middleware Support:** Redux supports middlewares like **Thunk** or **Saga**, which help manage asynchronous operations like API calls.
- **Scalability:** Redux shines in large applications where many components need access to shared data, or complex updates are frequent.

Example use cases: E-commerce carts, user authentication, dashboards with dynamic data.

---

2. Context API

React's Context API provides a way to **share state between components without prop drilling**.

- **Scoped State Sharing:** Unlike Redux, Context is part of React itself and doesn't require extra libraries.
- **Simpler for Small/Medium Apps:** If your app has a few global states (like theme, user info), Context is easier and lighter than Redux.
- **Provider & Consumer:** State is provided by a `<Context.Provider>` and consumed via `useContext`. It works well for static or semi-dynamic data.
- **Limitations:** Context isn't ideal for frequent updates or highly dynamic data, because every change re-renders all consuming components, which can hurt performance.

Example use cases: Theme toggling, user login status, language preference.

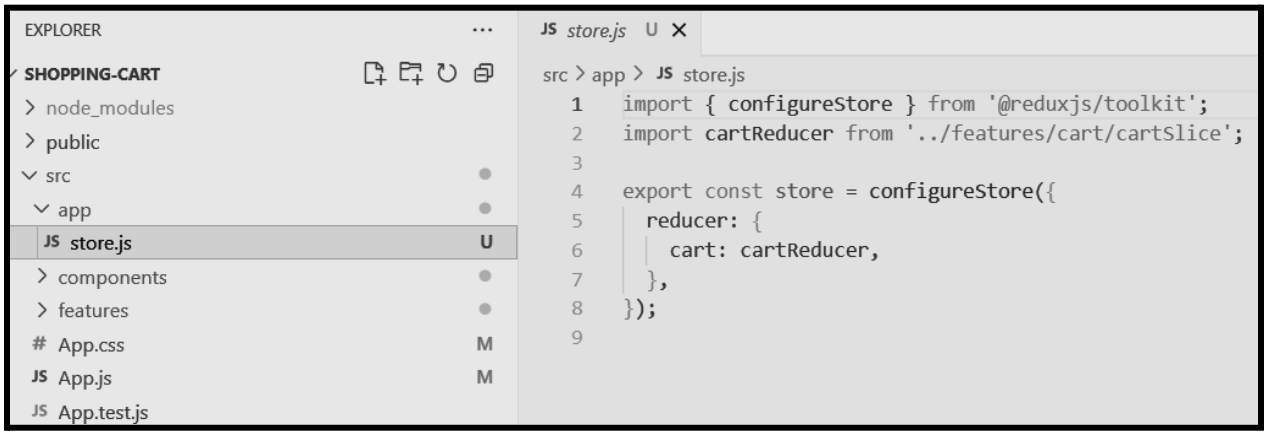
Comparison and Practical Tips

- **Use Redux** when:
    - Your app is large or complex.
    - Many components need to read/write the same state.
    - You need advanced tools like devtools, logging, or middleware.
- **Use Context API** when:
    - The app is medium or small.
    - The state is relatively static or changes infrequently.
    - You want a simple, built-in solution without installing libraries.
- **Combining:** Some apps use both—Redux for core app state and Context for UI-level settings like theme or language.

Extra:

The **search bar** allows users to filter products dynamically based on their input, improving usability and helping users find items quickly. The search functionality updates the displayed products in real-time without reloading the page, demonstrating **reactive state management**. The **popup/toast notification** shows a brief message whenever a product is added to the cart, giving immediate feedback to the user. Both features utilize React hooks (`useState` and `useEffect`) to manage state and side effects, illustrating how **UI and application state can be handled efficiently** in a React app.

Code:


EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js
components
features
App.css
App.js
App.test.js
store.js
src > app > store.js
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from '../features/cart/cartSlice';
export const store = configureStore({
reducer: {
cart: cartReducer,
},
});


EXPLORER
SHOPPING-CART
Cart.js
src > components > Cart.js > Cart
import React from 'react';
import { useDispatch, useSelector } from 'react-redux';
function Cart() {
const { items, total } = useSelector(state => state.cart);
const dispatch = useDispatch();
return (
<h2>Cart</h2>
{items.length === 0 ? No items in cart : (
{item.name} - ₹{item.price} x {item.qty}
<h3>Total: ₹{total}</h3>
export default Cart;


EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js
components
Cart.js
ProductList.js
features
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
OUTLINE
ProductList.js U
src > components > ProductList.js > ...
import React, { useState, useEffect } from "react";
import { useDispatch } from "react-redux";
import { addToCart } from "../features/cart/cartSlice";
const products = [
{ id: 1, name: "Laptop", price: 120000, img: "https://m.media-amazon.com/images/I/510uTHyDqGL.jpg" },
{ id: 2, name: "Headphones", price: 8000, img: "https://avvenice.com/60664-thickbox_default/marshall-major-iii-bluetooth-br
{ id: 3, name: "Phone", price: 15000, img: "https://cdn.mos.cms.futurecdn.net/hf2CQvHr9KNtKuUSDkeQVH.jpg" },
{ id: 4, name: "iPad", price: 11000, img: "https://m.media-amazon.com/images/I/61uA2UVnYWL._UF1000,1000_QL80_.jpg" },
{ id: 5, name: "Camera", price: 10000, img: "https://m.media-amazon.com/images/I/81WtQ64-SOL._UF1000,1000_QL80_.jpg" },
{ id: 6, name: "Earbuds", price: 1050, img: "https://www.boat-lifestyle.com/cdn/shop/files/ACCG6DS7WDJHGWSH_0.png?v=1727669
];
function ProductList() {
const dispatch = useDispatch();
const [search, setSearch] = useState("");
const [toast, setToast] = useState("");
const filteredProducts = products.filter((p) =>
p.name.toLowerCase().includes(search.toLowerCase())
);
useEffect(() => {
if (toast) {
const timer = setTimeout(() => setToast(""), 2000);
return () => clearTimeout(timer);
}
}, [toast]);
const handleAdd = (product) => {
dispatch(addToCart(product));
setToast(`${product.name} added to cart!`);
};
return (


EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js
components
Cart.js
ProductList.js
features
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
ProductList.js U
src > components > ProductList.js > ...
function ProductList() {
return (

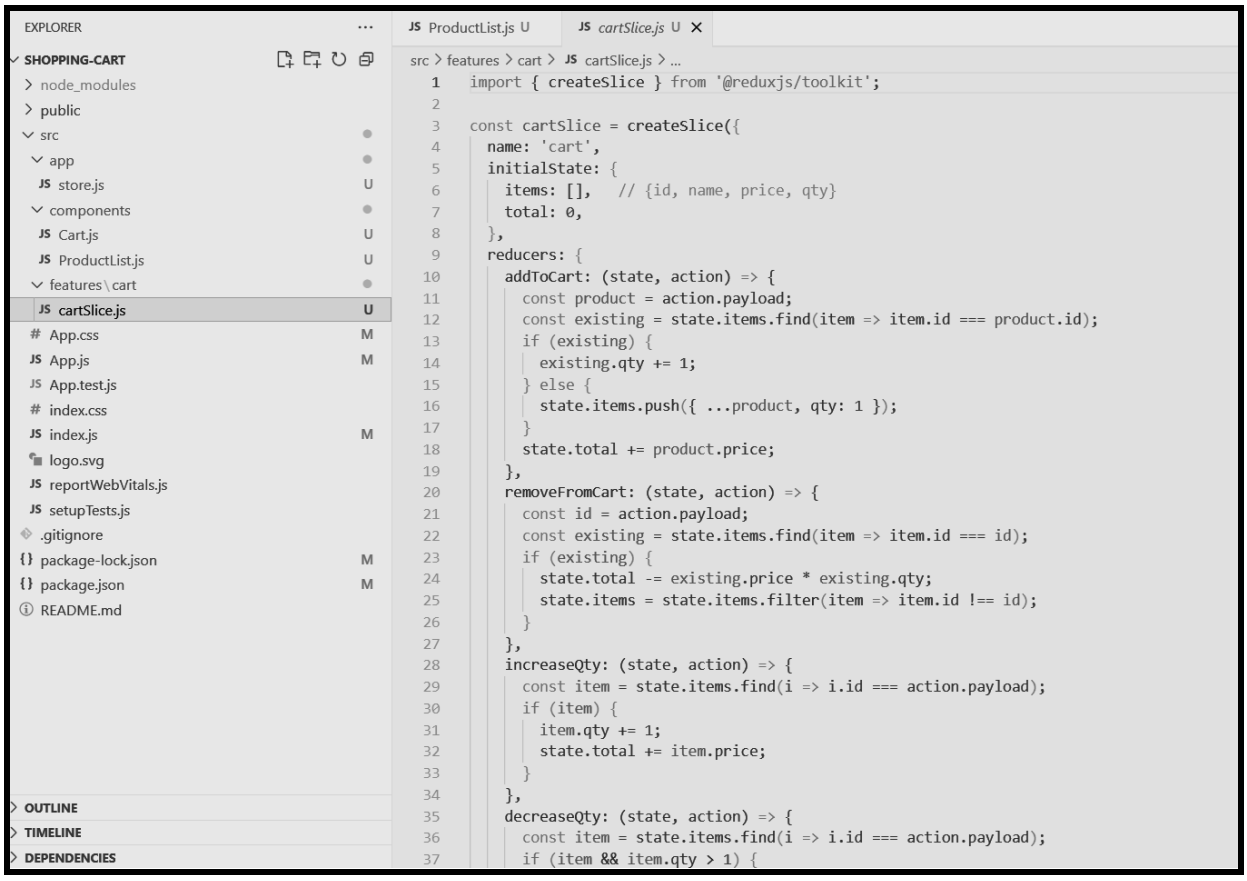
<h1>Our Products</h1>
<input
type="text"
placeholder="Search products..."
value={search}
onChange={(e) => setSearch(e.target.value)}
className="search-bar"
/>
{toast && {toast} }

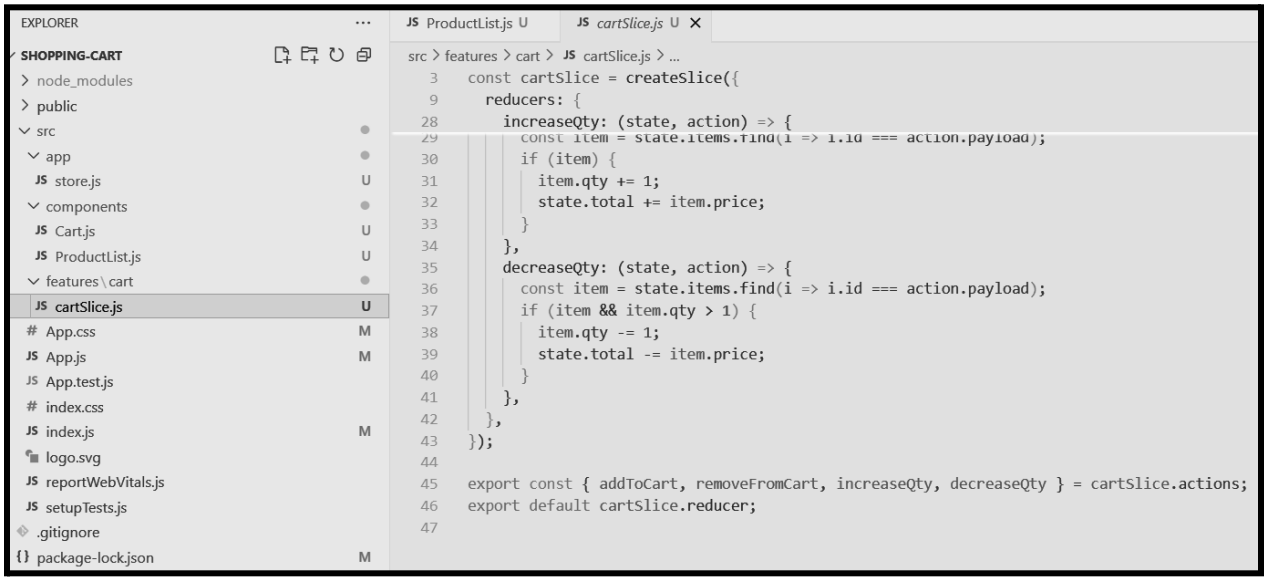
{filteredProducts.map((p) => (

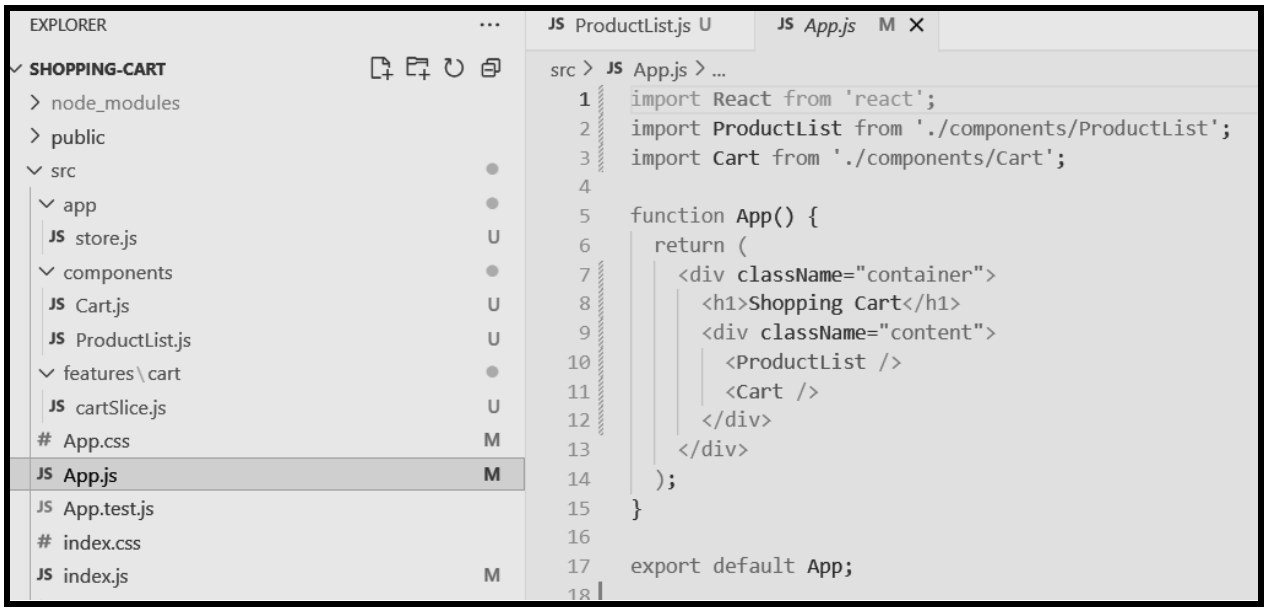
<h3>{p.name}</h3>
₹{p.price}
<button onClick={() => handleAdd(p)}>Add to Cart</button>

))}

);
}
export default ProductList;


EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js U
components
Cart.js U
ProductList.js U
features\cart
cartSlice.js U
App.css M
App.js M
App.test.js
index.css
index.js M
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json M
package.json M
README.md
OUTLINE
TIMELINE
DEPENDENCIES
ProductList.js U
cartSlice.js U
src > features > cart > cartSlice.js > ...
import { createSlice } from '@reduxjs/toolkit';
const cartSlice = createSlice({
name: 'cart',
initialState: {
items: [], // {id, name, price, qty}
total: 0,
},
reducers: {
addToCart: (state, action) => {
const product = action.payload;
const existing = state.items.find(item => item.id === product.id);
if (existing) {
existing.qty += 1;
} else {
state.items.push({ ...product, qty: 1 });
}
state.total += product.price;
},
removeFromCart: (state, action) => {
const id = action.payload;
const existing = state.items.find(item => item.id === id);
if (existing) {
state.total -= existing.price * existing.qty;
state.items = state.items.filter(item => item.id !== id);
}
},
increaseQty: (state, action) => {
const item = state.items.find(i => i.id === action.payload);
if (item) {
item.qty += 1;
state.total += item.price;
}
},
decreaseQty: (state, action) => {
const item = state.items.find(i => i.id === action.payload);
if (item && item.qty > 1) {

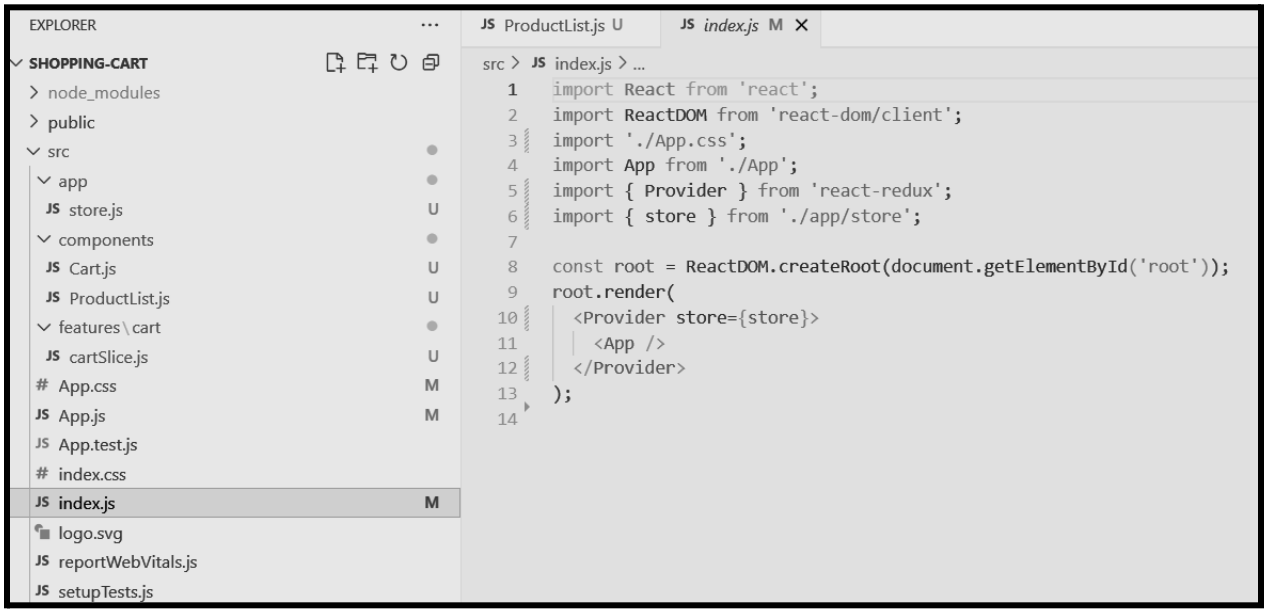

EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js U
components
Cart.js U
ProductList.js U
features\cart
cartSlice.js U
App.css M
App.js M
App.test.js
index.css
index.js M
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json M
ProductList.js U
cartSlice.js U
src > features > cart > cartSlice.js > ...
const cartSlice = createSlice({
reducers: {
increaseQty: (state, action) => {
if (item) {
item.qty += 1;
state.total += item.price;
}
},
decreaseQty: (state, action) => {
const item = state.items.find(i => i.id === action.payload);
if (item && item.qty > 1) {
item.qty -= 1;
state.total -= item.price;
}
},
},
});
export const { addToCart, removeFromCart, increaseQty, decreaseQty } = cartSlice.actions;
export default cartSlice.reducer;


EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js U
components
Cart.js U
ProductList.js U
features\cart
cartSlice.js U
App.css M
App.js M
App.test.js
index.css
index.js M
ProductList.js U
App.js M
src > App.js > ...
import React from 'react';
import ProductList from './components/ProductList';
import Cart from './components/Cart';
function App() {
return (

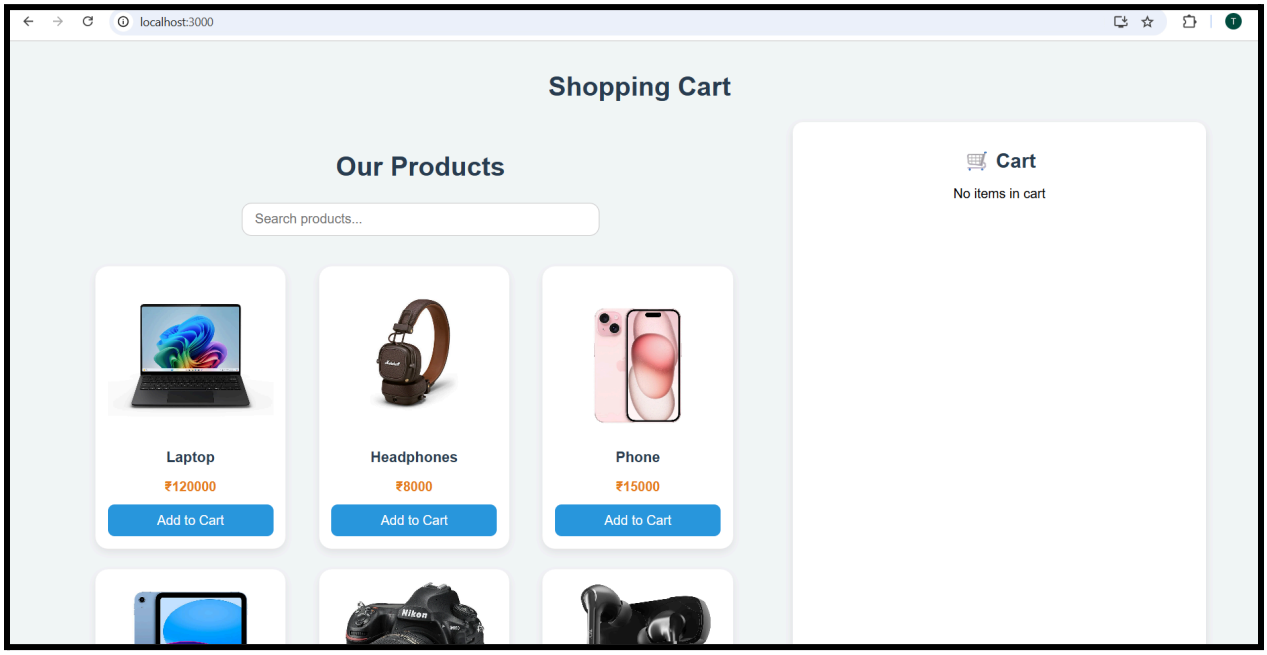
<h1>Shopping Cart</h1>

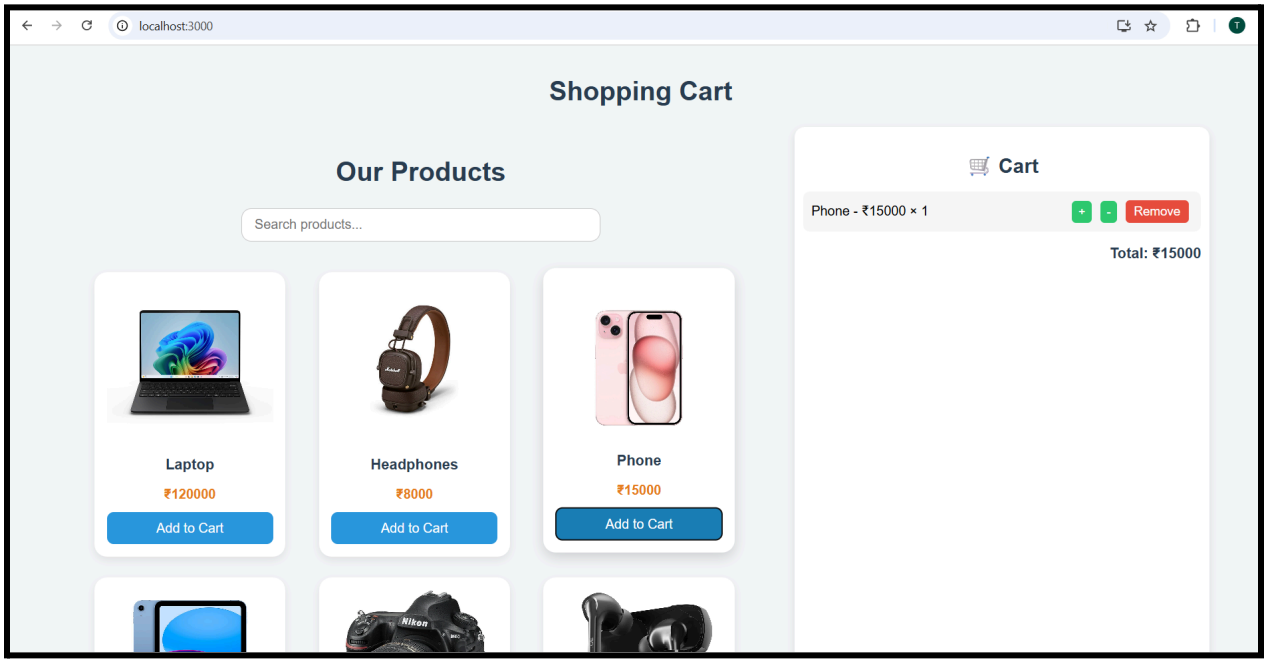
<ProductList />
<Cart />

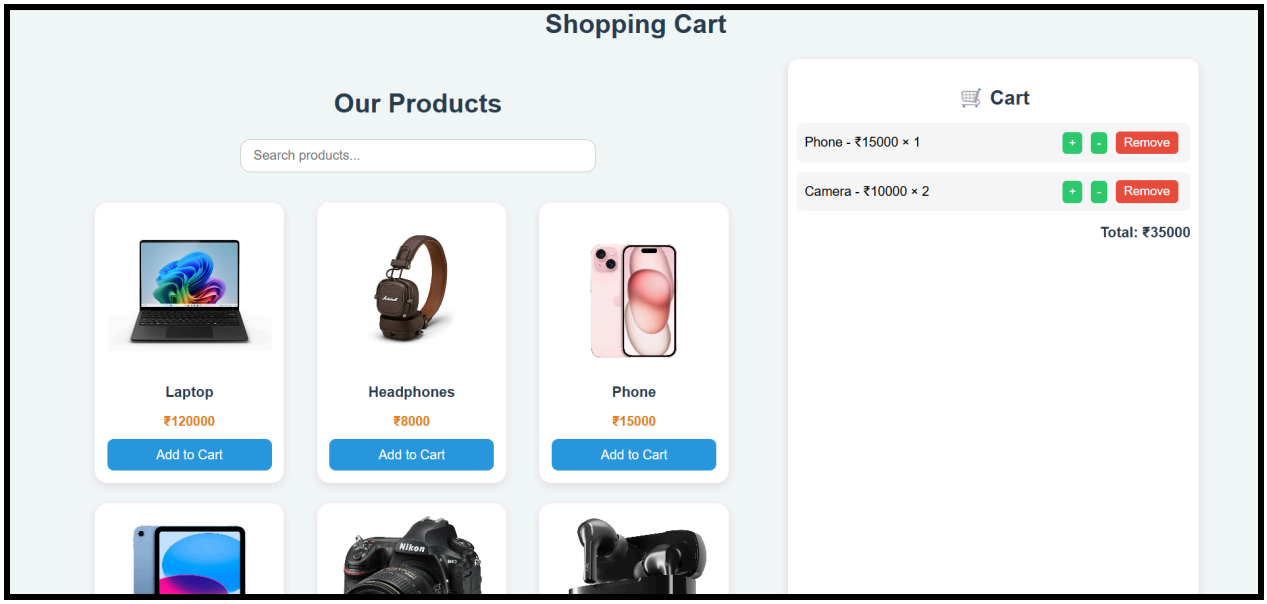
);
}
export default App;

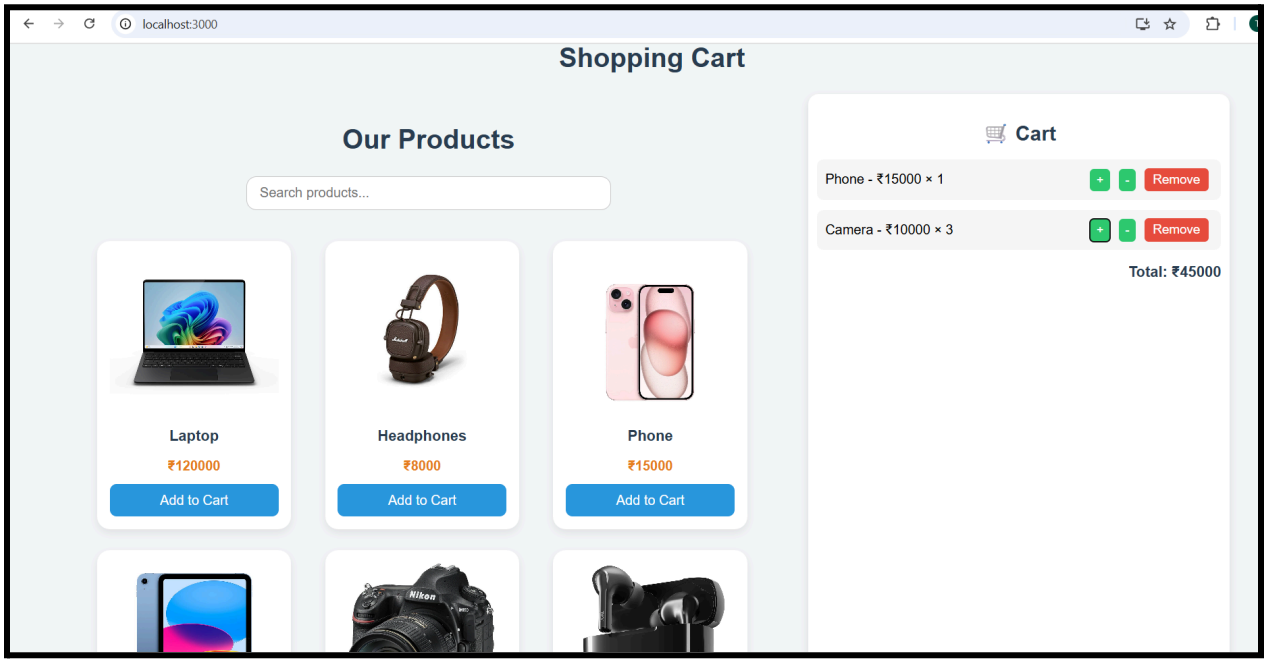

EXPLORER
SHOPPING-CART
node_modules
public
src
app
store.js U
components
Cart.js U
ProductList.js U
features\cart
cartSlice.js U
App.css M
App.js M
App.test.js
index.css
index.js M
logo.svg
reportWebVitals.js
setupTests.js
ProductList.js U
index.js M
src > index.js > ...
import React from 'react';
import ReactDOM from 'react-dom/client';
import './App.css';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './app/store';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
<Provider store={store}>
<App />
</Provider>
);

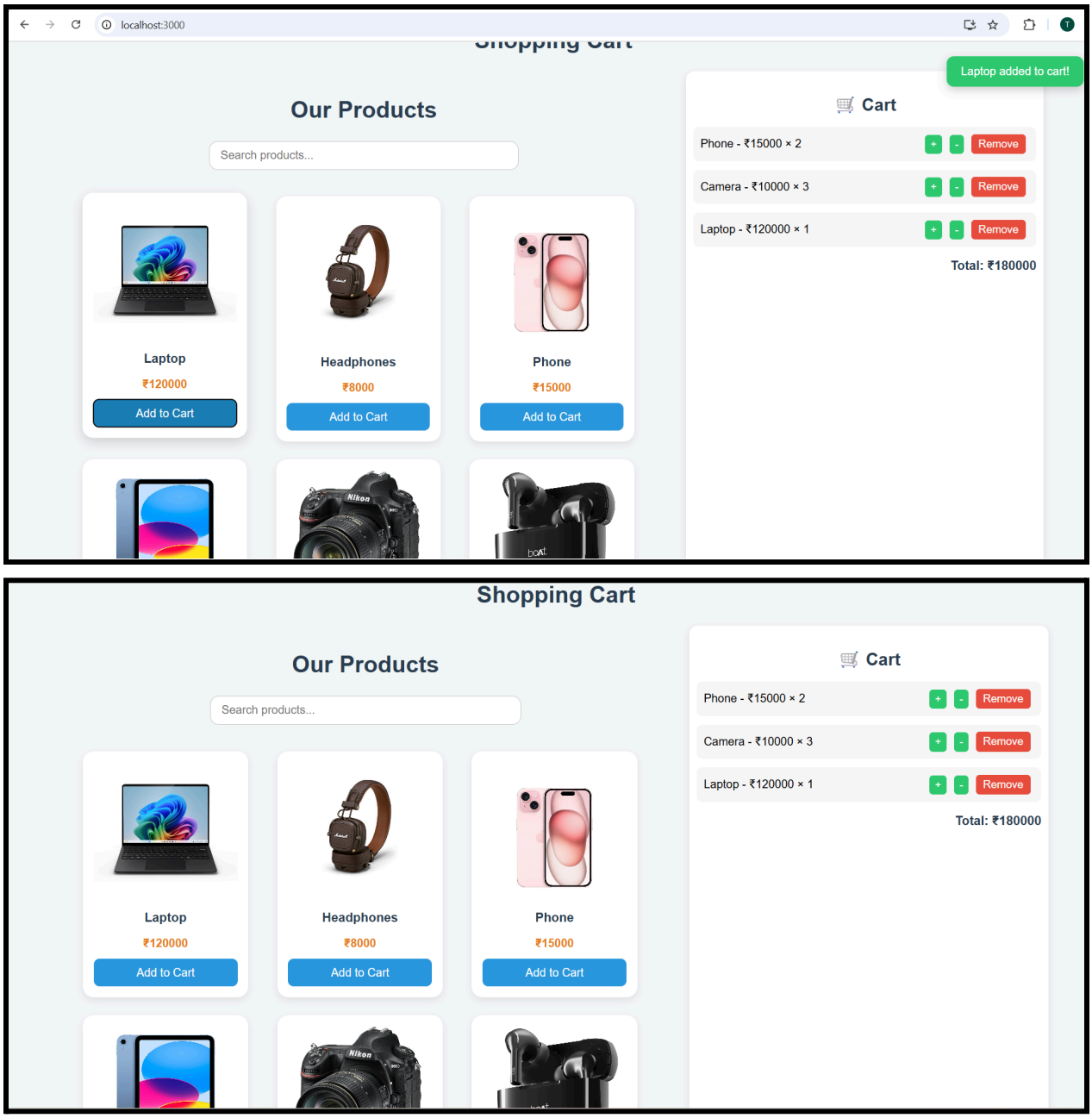
Output:


localhost:3000
Shopping Cart
Our Products
Search products...
Laptop
₹120000
Add to Cart
Headphones
₹8000
Add to Cart
Phone
₹15000
Add to Cart
Cart
No items in cart


localhost:3000
Shopping Cart
Our Products
Search products...
Laptop
₹120000
Add to Cart
Headphones
₹8000
Add to Cart
Phone
₹15000
Add to Cart
Cart
Phone - ₹15000 × 1
+
-
Remove
Total: ₹15000


Shopping Cart
Our Products
Search products...
Laptop
₹120000
Add to Cart
Headphones
₹8000
Add to Cart
Phone
₹15000
Add to Cart
Cart
Phone - ₹15000 × 1
Remove
Camera - ₹10000 × 2
Remove
Total: ₹35000


localhost:3000
Shopping Cart
Our Products
Search products...
Laptop
₹120000
Add to Cart
Headphones
₹8000
Add to Cart
Phone
₹15000
Add to Cart
Cart
Phone - ₹15000 × 1
Remove
Camera - ₹10000 × 3
Remove
Total: ₹45000



Conclusion:

Redux is ideal for managing complex, dynamic app states with predictability and scalability, while Context API is a lightweight solution for simpler global state needs. Choosing between them depends on app complexity and performance requirements.