## Key Concepts:

1. **Store**: The centralized state container that holds the application’s state.
2. **Actions**: Plain JavaScript objects that describe changes to the state.
3. **Reducers**: Functions that specify how the state changes in response to an action.
4. **Dispatch**: A function used to send actions to the store, triggering state updates.
5. **Selectors**: Functions that extract specific pieces of state from the store.

## Why Use Redux?

* **Predictable State**: Redux centralizes state management, making it easier to debug and maintain.
* **Decoupling State**: It helps in decoupling state logic from UI components, making your application more scalable.
* **DevTools**: Redux comes with great development tools for inspecting and debugging state changes.

## 1. ****Set Up a New React Project with Vite****

1. Open a terminal and run the following command to create a new React project using Vite:

npm create vite@latest

1. Navigate into your project directory:

cd redux-example

1. Install dependencies for Redux and Redux Toolkit:

npm install @reduxjs/toolkit react-redux

1. Start the development server:

npm run dev

## 2. ****Directory Structure****

Organize your project as follows for clarity:

src/

components/

counter/

IncrementCounterApp.jsx

DecrementCounterApp.jsx

counterSlice.js

useCounter.js

store/

store.jsx

App.jsx

main.jsx

index.css

## 3. ****Create the Redux Store****

**File:** src/store/store.jsx

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

// Import the counter reducer

import counterReducer from "../components/counter/counterSlice";

// Create the Redux store

export const store = configureStore({

reducer: {

counter: counterReducer, // Register the counter slice

},

});

## 4. ****Create the Counter Slice****

**File:** src/components/counter/counterSlice.js

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

// Define the slice with initial state and reducers

const counterSlice = createSlice({

name: "counter",

initialState: { value: 0 },

reducers: {

increment: (state) => {

state.value += 1; // Increase count

},

decrement: (state) => {

state.value -= 1; // Decrease count

},

incrementByAmount: (state, action) => {

state.value += action.payload; // Increase count by a specific amount

},

},

});

// Export the actions and reducer

export const { increment, decrement, incrementByAmount } = counterSlice.actions;

export default counterSlice.reducer;

## 5. ****Provide the Store to the App****

**File:** src/main.jsx

import React from "react";

import { createRoot } from "react-dom/client";

import { StrictMode } from "react";

import { Provider } from "react-redux";

import App from "./App";

import { store } from "./store/store";

import "./index.css";

// Provide the Redux store to the React app

createRoot(document.getElementById("root")).render(

<StrictMode>

<Provider store={store}>

<App />

</Provider>

</StrictMode>

);

## 6. ****Create a Custom Hook for Counter State and Actions****

**File:** src/components/counter/useCounter.js

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

import { increment, decrement, incrementByAmount } from "./counterSlice";

// Custom hook for counter state and actions

export const useCounter = () => {

const count = useSelector((state) => state.counter.value); // Access counter state

const dispatch = useDispatch();

// Define action dispatchers

const incrementCount = () => dispatch(increment());

const decrementCount = () => dispatch(decrement());

const incrementByAmountCount = (amount) => dispatch(incrementByAmount(amount));

return { count, incrementCount, decrementCount, incrementByAmountCount };

};

## 7. ****Create Components****

# Increment Counter Component

**File:** src/components/counter/IncrementCounterApp.jsx

import React from "react";

import { useCounter } from "./useCounter";

function IncrementCounterApp() {

const { count, incrementCount } = useCounter(); // Use custom hook

return (

<div>

<h1>Count: {count}</h1>

<button onClick={incrementCount}>Increment</button>

</div>

);

}

export default IncrementCounterApp;

# Decrement Counter Component

**File:** src/components/counter/DecrementCounterApp.jsx

import React from "react";

import { useCounter } from "./useCounter";

function DecrementCounterApp() {

const { count, decrementCount } = useCounter(); // Use custom hook

return (

<div>

<h1>Count: {count}</h1>

<button onClick={decrementCount}>Decrement</button>

</div>

);

}

export default DecrementCounterApp;

## 8. ****Integrate Components into the App****

**File:** src/App.jsx

import React from "react";

import IncrementCounterApp from "./components/counter/IncrementCounterApp";

import DecrementCounterApp from "./components/counter/DecrementCounterApp";

function App() {

return (

<div>

<IncrementCounterApp />

<DecrementCounterApp />

</div>

);

}

export default App;

## 9. ****Run and Test****

1. Start the development server:
2. npm run dev
3. Open [http://localhost:5173](http://localhost:5173/) in your browser.
4. Use the buttons to increment or decrement the counter. Observe the count updating in both components, demonstrating the shared state managed by Redux.

## Explanation of Workflow

1. **Redux Store**: The store is a global state container created using configureStore. It holds the counter slice, allowing components to access and update the counter state.
2. **Counter Slice**: The slice defines the initial state and the actions (increment, decrement, incrementByAmount) to modify it.
3. **Custom Hook**: The useCounter hook simplifies the use of Redux in components by encapsulating useSelector (to read state) and useDispatch (to dispatch actions).
4. **Components**:
   * Each component interacts with the Redux state via the custom hook.
   * Updates to the state trigger re-renders in all components using the counter slice.
5. **Provider**: The Provider ensures the store is accessible to all components within the app.

This structure is scalable and follows best practices for using Redux with React.