Monday, August 18, 2025 11:29 PM

🔹 Step 1: Install dependencies

bash

```
npm install @reduxjs/toolkit react-redux
```

🔹 Step 2: Create a Slice

A **slice** is a piece of Redux state (e.g., counter, user, products).

```
import { createSlice } from "@reduxjs/toolkit";
// Initial state
const initialState = {
  value: 0
};
// Create slice
const counterSlice = createSlice({
  name: "counter",    // slice name
  initialState,       // initial state
  reducers: {         // reducers (state changing functions)
    increment: (state) => {
      state.value += 1; // Immer allows us to mutate state
    },
    decrement: (state) => {
      state.value -= 1;
    },
    reset: (state) => {
      state.value = 0;
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload; // payload is passed data
    }
  }
});
// Export actions
```

```
export const { increment, decrement, reset, incrementByAmount } =
counterSlice.actions;
// Export reducer
export default counterSlice.reducer;
```

✨ Explanation:

- `createSlice` auto-creates:
  - **Action types** (e.g., `counter/increment`)
  - **Action creators** (`increment()`)
  - **Reducer function**

This saves you writing lots of boilerplate code.

🔹 Step 3: Configure Store

👉 File: `store.js`

javascript

```
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./counterSlice";

const store = configureStore({
  reducer: {
    counter: counterReducer,  // register slice
  },
});

export default store;
```

✨ Explanation:

- `configureStore` sets up the Redux store with good defaults (like Redux DevTools).
- You combine multiple slices here (`counter`, `auth`, `todos`, etc).

🔹 Step 4: Provide Store to React

👉 File: `index.js`

javascript

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import { Provider } from "react-redux";
import store from "./store";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

✨ Explanation:

- `Provider` makes the store available to all components.

🔹 Step 5: Use Redux in Components

👉 File: `Counter.js`

```
import React, { useState } from "react";
import { useSelector, useDispatch } from "react-redux";
// import { increment } from "./counterSlice";
import { increment, decrement, reset, incrementByAmount } from
"./counterSlice";
export default function Counter() {
  const count = useSelector((state) => state.counter.value); // read
from store
  const dispatch = useDispatch(); // send actions
```

```
  const [amount, setAmount] = useState(0);
  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Redux Toolkit Counter</h1>
      <h2>{count}</h2>
      <button onClick={() => dispatch(increment())}> +1 </button>
      <button onClick={() => dispatch(decrement())}> -1 </button>
      <button onClick={() => dispatch(reset())}> Reset </button>
      <br /><br />
      <input
        type="number"
        value={amount}
        onChange={(e) => setAmount(Number(e.target.value))}
      />
      <button onClick={() => dispatch(incrementByAmount(amount))}>
        Add Amount
      </button>
    </div>
  );
}
```

✨ Explanation:

- `useSelector`: Reads Redux state.
- `useDispatch`: Sends actions.
- We can now increment, decrement, reset, or add a custom value.

🔹 Step 6: Add to App

👉 File: `App.js`

👉 File: `App.js`

javascript

```
import React from "react";
import Counter from "./Counter";

function App() {
  return <Counter />;
}

export default App;
```

🔹 How It All Connects

1. `Counter.js` → dispatches `increment()`.
2. `counterSlice` reducer handles action → updates state.
3. `store.js` holds updated state.
4. `useSelector` re-renders component with new value.