Perfect! Since you're using **React with Vite**, the setup is slightly different from Create React App, but Redux works the same way. I'll guide you **step by step** specifically for Vite.

Learn Redux Basics

Redux adds structure for bigger apps:

1. **Understand Core Concepts**
   - **Store** → central state container
   - **Actions** → "events" that describe changes
   - **Reducers** → functions that decide how state changes
   - **Dispatch** → trigger actions
   - **Selectors** → read state
2. **Start with Redux Toolkit**
   - Reduces boilerplate
   - Provides `createSlice`, `configureStore`
3. **Practice Steps**
   - Create a slice (`createSlice`)
   - Configure store (`configureStore`)
   - Wrap app in `Provider`
   - Use `useSelector` to read, `useDispatch` to update

Step 1: Install Redux and React-Redux

In your Vite project root, run:

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

```
as shown below :-
```

C:\WINDOWS\system32\cmd.exe

```
4:33:09 PM [vite] (client) hmr update /src/App.jsx (x2)
Terminate batch job (Y/N)? y

D:\react-warm-up\my-app>npm install @reduxjs/toolkit react-redux

added 10 packages, and audited 190 packages in 22s

44 packages are looking for funding
  run `npm fund` for details

2 vulnerabilities (1 moderate, 1 high)

To address all issues, run:
  npm audit fix
```

or with yarn:

```
yarn add @reduxjs/toolkit react-redux
```

`@reduxjs/toolkit` is the modern, recommended way to use Redux—less boilerplate.

Step 2: Create the Redux Slice

Create a folder `src/store/`

Inside, create `userSlice.js`:

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

const initialState = {
  username: "Alice",
  email: "alice@example.com",
  age: 25,
};

export const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    updateUser: (state, action) => {
      Object.assign(state, action.payload);
    },
  },
});

export const { updateUser } = userSlice.actions;

export default userSlice.reducer;
```

Step 3: Configure the Store

Create `store.js` in the same folder:

```
import { configureStore } from '@reduxjs/toolkit';
import userReducer from './userSlice';

export const store = configureStore({
  reducer: {
    user: userReducer,
  },
});
```

Step 4: Wrap Your App with Redux Provider

In `src/main.jsx` (Vite entry point):

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

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

This is slightly different from CRA; Vite uses `main.jsx`.

Step 5: Create Components

Profile.jsx

```
import { useSelector, useDispatch } from "react-redux";
import { updateUser } from "./store/userSlice";

const Profile = () => {
  const user = useSelector((state) => state.user);
  const dispatch = useDispatch();

  return (
    <div>
      <h2>Profile Component</h2>
      <p>Name: {user.username}</p>

      <button
        onClick={() => dispatch(updateUser({ username: "Bob" }))}
      >
        Change Name
      </button>
    </div>
  );
};

export default Profile;
```

Dashboard.jsx

```
import { useSelector, useDispatch } from "react-redux";
import { updateUser } from "./store/userSlice";

const Dashboard = () => {
  const user = useSelector((state) => state.user);
  const dispatch = useDispatch();

  return (
    <div>
      <h2>Dashboard Component</h2>
      <p>Email: {user.email}</p>
      <p>Age: {user.age}</p>

      <button
        onClick={() => dispatch(updateUser({ age: user.age + 1 }))}
```

```
      >
        Increase Age
      </button>
    </div>
  );
};

export default Dashboard;
```

Step 6: Use Components in App.jsx

```
import Profile from "./Profile";
import Dashboard from "./Dashboard";

function App() {
  return (
    <div>
      <Profile />
      <Dashboard />
    </div>
  );
}

export default App;
```

□ Key Vite Tips

1. File extensions matter: use `.jsx` for React components.

Output:-

ChatGPT | context-api-simple-2026.pdf | my-app | Geocities.ws File Manager

localhost:5173

Profile Component

Name: Bob

Change Name

Dashboard Component

Email: alice@example.com

Age: 30

Increase Age

Activate Windows
Go to Settings to activate Windows.

Type here to search

31°C Sunny 5:01 PM 4/1/2026

Let's make a **simple beginner-friendly Redux example** using **React + Vite**. We'll do a **counter app**, step by step. This is perfect for practicing `store`, `slice`, `dispatch`, and `useSelector`.

Step 1: Create a Vite React Project

If you haven't already:

```
npm create vite@latest my-redux-app
cd my-redux-app
npm install
npm install @reduxjs/toolkit react-redux
npm run dev
```

Step 2: Create a Redux Slice

Inside `src/store/`, create `counterSlice.js`:

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

// 1. Initial state
const initialState = {
  value: 0,
};

// 2. Create slice
export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
    reset: (state) => {
      state.value = 0;
    },
  },
});

// 3. Export actions
export const { increment, decrement, reset } = counterSlice.actions;

// 4. Export reducer
export default counterSlice.reducer;
```

Step 3: Configure the Redux Store

Create `src/store/store.js`:

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

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});
```

Step 4: Wrap Your App with Redux Provider

In `src/main.jsx`:

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

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

Step 5: Create a Counter Component

Create `src/Counter.jsx`:

```
import { useSelector, useDispatch } from "react-redux";
import { increment, decrement, reset } from "./store/counterSlice";

const Counter = () => {
  // Read state from store
  const count = useSelector((state) => state.counter.value);
  // Dispatch actions
  const dispatch = useDispatch();

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Counter: {count}</h1>
      <button onClick={() => dispatch(increment())}>Increment</button>
      <button onClick={() => dispatch(decrement())}>Decrement</button>
      <button onClick={() => dispatch(reset())}>Reset</button>
    </div>
  );
};

export default Counter;
```

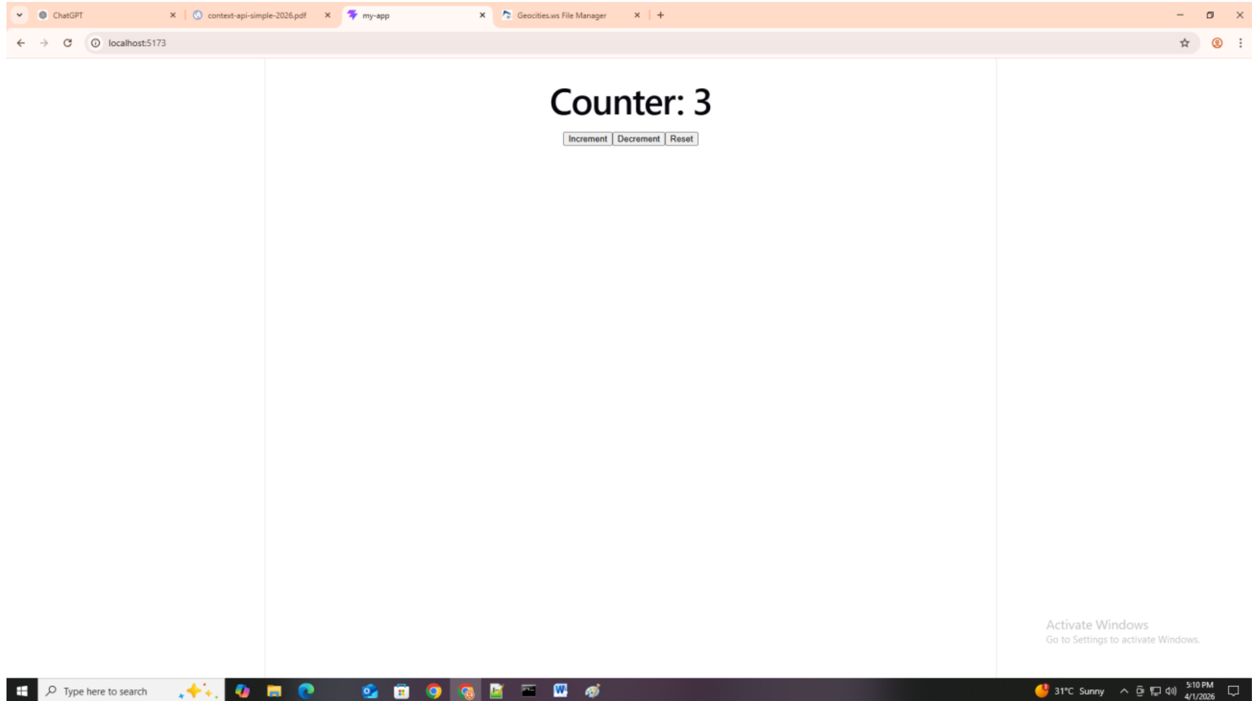
Step 6: Use the Counter in App.jsx

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

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

export default App;
```

Step 7: Run the App

```
npm run dev
```

- You should see a counter on the page.
- Clicking **Increment / Decrement / Reset** updates the global state via Redux.
- Open **Redux DevTools** in your browser to see state changes in real time.



▯ Key Concepts Practiced

1. **Store** → central place for state (`store.js`)
2. **Slice** → reducer + actions in one place (`counterSlice.js`)
3. **Provider** → makes store available (`main.jsx`)
4. **useSelector** → read state in component
5. **useDispatch** → send actions to update state

Combine counter + toggle theme in the same app using multiple slices

Perfect! Let's create a **React + Vite app with Redux** that has **two slices**: one for a **counter** and one for **theme toggling**. This is a great beginner exercise to see **multiple slices working together in Redux**.

Step 1: Project Setup (Vite + Redux)

If you don't have a project yet:

```
npm create vite@latest redux-multi-slice-app
cd redux-multi-slice-app
npm install
npm install @reduxjs/toolkit react-redux
npm run dev
```

Step 2: Create the Counter Slice

Create `src/store/counterSlice.js`:

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

const initialState = { value: 0 };

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => { state.value += 1; },
    decrement: (state) => { state.value -= 1; },
    reset: (state) => { state.value = 0; },
  },
});

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

Step 3: Create the Theme Slice

Create `src/store/themeSlice.js`:

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

const initialState = { darkMode: false };

export const themeSlice = createSlice({
  name: 'theme',
  initialState,
  reducers: {
    toggleTheme: (state) => { state.darkMode = !state.darkMode; },
  },
});

export const { toggleTheme } = themeSlice.actions;
export default themeSlice.reducer;
```

Step 4: Configure the Store with Multiple Slices

Create `src/store/store.js`:

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './counterSlice';
import themeReducer from './themeSlice';

export const store = configureStore({
  reducer: {
    counter: counterReducer,
    theme: themeReducer,
  },
});
```

Step 5: Wrap App with Provider

In `src/main.jsx`:

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

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

```
    </Provider>
  </React.StrictMode>
);
```

Step 6: Create the App Component

In `src/App.jsx`:

```
import Counter from "./Counter";
import ThemeToggle from "./ThemeToggle";
import { useSelector } from "react-redux";

function App() {
  const darkMode = useSelector((state) => state.theme.darkMode);

  return (
    <div style={{
      textAlign: "center",
      padding: "50px",
      backgroundColor: darkMode ? "#333" : "#f5f5f5",
      color: darkMode ? "#fff" : "#000",
      minHeight: "100vh"
    }}>
      <h1>Redux Multi-Slice Example</h1>
      <ThemeToggle />
      <Counter />
    </div>
  );
}

export default App;
```

Step 7: Create the Counter Component

`src/Counter.jsx`:

```
import { useSelector, useDispatch } from "react-redux";
import { increment, decrement, reset } from "./store/counterSlice";

const Counter = () => {
  const count = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div style={{ marginTop: "30px" }}>
      <h2>Counter: {count}</h2>
      <button onClick={() => dispatch(increment())}>Increment</button>
      <button onClick={() => dispatch(decrement())}>Decrement</button>
```

```
      <button onClick={() => dispatch(reset())}>Reset</button>
    </div>
  );
};

export default Counter;
```

Step 8: Create the ThemeToggle Component

`src/ThemeToggle.jsx`:

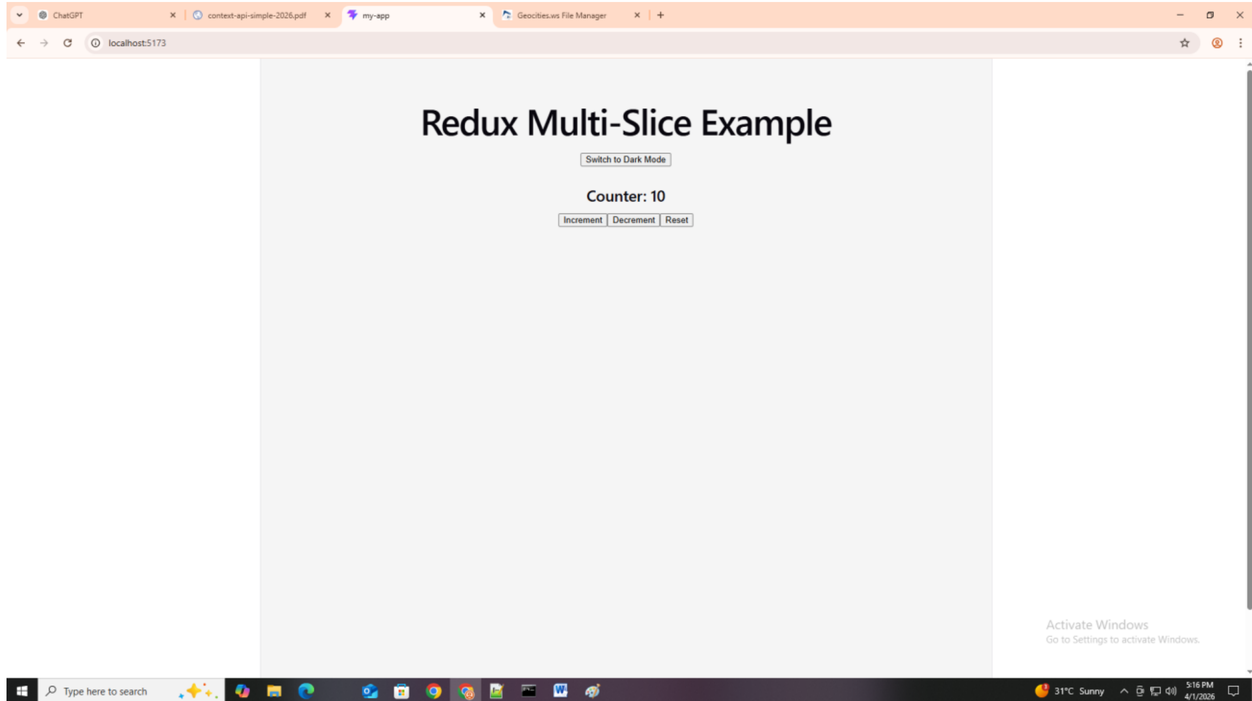
```
import { useDispatch, useSelector } from "react-redux";
import { toggleTheme } from "./store/themeSlice";

const ThemeToggle = () => {
  const darkMode = useSelector((state) => state.theme.darkMode);
  const dispatch = useDispatch();

  return (
    <div style={{ marginBottom: "20px" }}>
      <button onClick={() => dispatch(toggleTheme())}>
        Switch to {darkMode ? "Light" : "Dark"} Mode
      </button>
    </div>
  );
};

export default ThemeToggle;
```

Step 9: Run the App

```
npm run dev
```

You should see:

- **Counter** with Increment / Decrement / Reset buttons
- **Theme toggle button** switching between light and dark mode
- Both slices work independently but are part of the **same Redux store**



☐ Key Learning Points

1. **Multiple slices** can live in the same store.
2. `useSelector` can read **any slice of state**.
3. `useDispatch` triggers **actions** from any slice.
4. Styling or logic can depend on Redux state (like dark mode).
5. Redux Toolkit makes creating slices easier and cleaner than classic Redux.

Let's recreate the **same app (Counter + Theme Toggle)** using **React Context API** instead of Redux. This will let you see the differences in **state management style**, **boilerplate**, and **scalability**.

Step 1: Create Contexts

We will create **two contexts**: one for **counter** and one for **theme**.

`src/contexts/CounterContext.jsx`

```
import { createContext, useState } from "react";

export const CounterContext = createContext();

export const CounterProvider = ({ children }) => {
  const [count, setCount] = useState(0);

  const increment = () => setCount(prev => prev + 1);
  const decrement = () => setCount(prev => prev - 1);
  const reset = () => setCount(0);

  return (
    <CounterContext.Provider value={{ count, increment, decrement, reset }}>
      {children}
    </CounterContext.Provider>
  );
};
```

`src/contexts/ThemeContext.jsx`

```
import { createContext, useState } from "react";

export const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [darkMode, setDarkMode] = useState(false);

  const toggleTheme = () => setDarkMode(prev => !prev);

  return (
    <ThemeContext.Provider value={{ darkMode, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

Step 2: Wrap App with Providers

In `src/main.jsx`:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import { CounterProvider } from "./contexts/CounterContext";
import { ThemeProvider } from "./contexts/ThemeContext";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <ThemeProvider>
      <CounterProvider>
        <App />
      </CounterProvider>
    </ThemeProvider>
  </React.StrictMode>
);
```

Notice the **nested providers**: ThemeProvider → CounterProvider → App

---

Step 3: Create the App Component

`src/App.jsx`:

```
import Counter from "./Counter";
import ThemeToggle from "./ThemeToggle";
import { useContext } from "react";
import { ThemeContext } from "./contexts/ThemeContext";

function App() {
  const { darkMode } = useContext(ThemeContext);

  return (
    <div style={{
      textAlign: "center",
      padding: "50px",
      backgroundColor: darkMode ? "#333" : "#f5f5f5",
      color: darkMode ? "#fff" : "#000",
      minHeight: "100vh"
    }}>
      <h1>Context API Multi-Provider Example</h1>
      <ThemeToggle />
      <Counter />
    </div>
  );
}
```

```
export default App;
```

Step 4: Create the Counter Component

src/Counter.jsx:

```
import { useContext } from "react";
import { CounterContext } from "./contexts/CounterContext";

const Counter = () => {
  const { count, increment, decrement, reset } = useContext(CounterContext);

  return (
    <div style={{ marginTop: "30px" }}>
      <h2>Counter: {count}</h2>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
};

export default Counter;
```

Step 5: Create the ThemeToggle Component

src/ThemeToggle.jsx:

```
import { useContext } from "react";
import { ThemeContext } from "./contexts/ThemeContext";

const ThemeToggle = () => {
  const { darkMode, toggleTheme } = useContext(ThemeContext);

  return (
    <div style={{ marginBottom: "20px" }}>
      <button onClick={toggleTheme}>
        Switch to {darkMode ? "Light" : "Dark"} Mode
      </button>
    </div>
  );
};

export default ThemeToggle;
```

Step 6: Run the App

```
npm run dev
```

You should see:

- **Counter** with Increment / Decrement / Reset
- **Theme toggle button** switching between light and dark mode

 Everything works**without Redux**, just Context API.

Step 7: Key Observations (Redux vs Context API)

Feature	Context API	Redux
Boilerplate	Less	Slightly more (store, slices, dispatch)
State Structure	Local state inside provider	Centralized global store
Multiple States	Multiple contexts (nested providers)	Multiple slices in same store
Debugging	React DevTools	Redux DevTools (time travel, actions log)
Async Handling	Manual (`useEffect`)	Middleware (thunk, saga)

Context API is **simpler** for small apps.
Redux scales better for **large apps with multiple features**.