What is Redux?

Redux is a state management library that helps you manage and control the state of your entire application in one place. It makes it easier to:

- Share state between different components.
- Manage complex state logic in large applications.
- Debug issues easily using Redux DevTools.

Redux follows the Flux architecture and is based on the concept of a single source of truth, meaning all your app's state is stored in a global store.

How Does Redux Work?

Redux revolves around three main principles:

1. Single Source of Truth

- The entire application state is stored in a single store.

2. State is Read-Only

- You cannot directly modify the state. Instead, you use actions to request changes.

3. Changes are Made Using Pure Functions (Reducers)

- A reducer is a function that takes the current state and an action and returns a new state.

How Redux Connects with React

Normally, React manages state internally using useState or useContext, but in large applications, managing state can become complex.

This is where Redux comes in.

With Redux, React components do not manage state themselves. Instead:

- The state is stored globally in the Redux store.
- Components access the store to read data.
- Components dispatch actions to modify the store.

Redux Key Concepts (with Examples):

Now, let’s go step by step with practical examples to understand each concept.

Step 1: Install Redux and React-Redux

To use Redux in your React project, install the required packages:

```
npm install redux react-redux
```

Step 2: Create a Redux Store

The store holds the global state of the application.

store.js

```
import { createStore } from 'redux';
import counterReducer from './counterReducer';

const store = createStore(counterReducer);

export default store;
```

- Here, we create a Redux store using createStore().

- The counterReducer is responsible for handling state updates.

Step 3: Define Actions

Actions describe what should happen (e.g., "increment counter", "decrement counter").

actions.js

```
export const increment = () => {
   return { type: "INCREMENT" };
};

export const decrement = () => {
   return { type: "DECREMENT" };
};
```

- Each action is just a function that returns an object with a type.

Step 4: Create a Reducer

A reducer updates the state based on the action type.

counterReducer.js

```
const initialState = { count: 0 };

const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
};

export default counterReducer;
```

- The reducer receives the current state and an action, then returns a new state.

Step 5: Provide the Redux Store to React

Use the Provider component to make Redux available to all components.

index.js

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

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

- This wraps the React app with the Provider, giving all components access to Redux.

Step 6: Connect Redux to a React Component

Now, let's create a Counter component that:

- Reads state from Redux.
- Dispatches actions to modify state.

Counter.js

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { increment, decrement } from "./actions";

const Counter = () => {
    const count = useSelector((state) => state.count);  // Get state from Redux
    const dispatch = useDispatch();  // Get dispatch function

    return (
        <div>
            <h2>Counter: {count}</h2>
            <button onClick={() => dispatch(increment())}>+</button>
            <button onClick={() => dispatch(decrement())}>-</button>
        </div>
    );
};

export default Counter;
```

- useSelector() gets the current count value from Redux.
- useDispatch() allows us to dispatch actions (increment and decrement).

Step 7: Use the Counter Component

Finally, use the Counter component inside App.js.

App.js

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

const App = () => {
    return (
        <div>
            <h1>Redux Counter App</h1>
            <Counter />
        </div>
    );
};

export default App;
```

- Now, clicking the "+" or "-" button updates the state, and Redux handles everything globally!

Summary of Redux Flow in React

1. Store – Holds the global state.

2. Actions – Describe what should happen.

3. Reducer – Updates the state based on actions.

4. Provider – Makes Redux available in React.

5. useSelector – Fetches state in components.

6. useDispatch – Dispatches actions to update state.

Extra: Using Redux Toolkit (Modern Redux)

Redux Toolkit (@reduxjs/toolkit) simplifies Redux setup by reducing boilerplate.

Install Redux Toolkit:

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

A Redux Toolkit counter slice:

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

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

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

Replace counterReducer.js with this file, and Redux becomes much cleaner!

Final Thoughts

- Redux is powerful for state management, especially in large apps.

- React and Redux work together through store, actions, reducers, and hooks.

- Redux Toolkit makes Redux much simpler and is recommended for new projects.

To impress interviewers and demonstrate your Redux + React skills, you need projects that show:

- State management proficiency
- Async API handling (Redux Thunk / Redux Toolkit Query)
- Scalability and modular structure
- Real-world application of Redux (not just counters)

Here are 10 great projects you can build to showcase your Redux expertise in your portfolio:

1. E-Commerce Store

Key Features:

- Products fetched via API (Redux Thunk / Redux Toolkit Query)
- Shopping Cart (Redux for cart state)
- User Authentication (JWT / Firebase)

- Order Management

What it demonstrates:

- Handling complex state (cart, orders, products)
- Async data fetching
- Redux middleware for authentication

Tech Stack: React, Redux Toolkit, React-Bootstrap, Firebase/Auth

2. Movie Recommendation App

Key Features:

- Fetch movies from TMDB API
- Search, filter, and sort movies
- Save favorite movies using Redux
- Show trending, upcoming, and popular movies

What it demonstrates:

- API calls with Redux Thunk
- Global state management for favorite movies
- Dynamic UI updates using Redux

Tech Stack: React, Redux Toolkit, React-Bootstrap, TMDB API

3. News Aggregator App

Key Features:

- Fetch news from News API
- Category-based news (Tech, Sports, Politics)

- Bookmark articles using Redux
- Dark mode toggle using Redux

What it demonstrates:

- Managing multiple API requests
- Using Redux for UI state (dark mode)
- Data persistence (saved bookmarks)

Tech Stack: React, Redux Toolkit, Material UI, News API

4. Blogging Platform

Key Features:

- User authentication (Login/Signup)
- Create, edit, and delete blogs
- Redux for managing blog posts and users
- Comments & likes system

What it demonstrates:

- CRUD operations with Redux
- User authentication handling
- Real-time updates with WebSockets

Tech Stack: React, Redux Toolkit, Firebase/Auth, Node.js

5. Fitness Tracker App

Key Features:

- Log workouts (Redux for storing data)

- Fetch exercise routines from API
- Progress tracking (charts with Redux state)
- Dark/light mode toggle

What it demonstrates:

- Handling form data with Redux
- Visualization of state data (charts)
- Complex state updates (progress tracking)

Tech Stack: React, Redux Toolkit, Chart.js, Exercise API

6. Task Manager / Kanban Board

Key Features:

- Add, edit, delete tasks
- Drag & drop tasks between columns
- Store tasks globally using Redux
- Filters & search functionality

What it demonstrates:

- Complex state interactions (drag & drop)
- Optimizing UI performance with Redux selectors
- Redux Toolkit for better state structure

Tech Stack: React, Redux Toolkit, React-Beautiful-DnD, Tailwind

7. Expense Tracker App

Key Features:

- Add daily expenses
- Monthly income vs expense chart
- Categories-based filtering
- Persisting data with local storage

What it demonstrates:

- State persistence with Redux
- Working with financial data
- Dynamic UI updates

Tech Stack: React, Redux Toolkit, Chart.js, LocalStorage

8. Music Player App

Key Features:

- Play, pause, and queue songs
- Fetch music from Spotify API
- Save playlists globally using Redux
- Redux for playback state

What it demonstrates:

- Redux for handling complex UI state
- API requests for fetching music
- Managing playback queue

Tech Stack: React, Redux Toolkit, Spotify API, Material UI

9. Ride-Sharing App (Uber Clone)

Key Features:

- User authentication
- Booking rides (Redux for ride state)
- Fetching nearby drivers via API
- Map integration

What it demonstrates:

- Handling real-time state updates
- Asynchronous API integration
- Redux Toolkit for complex UI states

Tech Stack: React, Redux Toolkit, Google Maps API, Firebase

10. Stock Market Tracker

Key Features:

- Fetch stock data from API
- Live updates on stock prices
- Compare multiple stocks
- Save favorite stocks (Redux)

What it demonstrates:

- Handling real-time financial data
- Managing multiple API requests with Redux
- UI updates based on Redux state

Tech Stack: React, Redux Toolkit, Yahoo Finance API, Chart.js

Final Thoughts:

These 10 projects will impress interviewers by demonstrating:

- Strong Redux understanding
- API fetching & middleware usage
- Scalable app structure
- Advanced UI state management

Next Steps:

1. Start with a small project (Expense Tracker or Movie App).
2. Use Redux DevTools to showcase debugging skills.
3. Write clean, scalable Redux code (use Redux Toolkit).
4. Host projects on GitHub & deploy online (Netlify, Vercel).

5 Write a README explaining Redux usage for each project.