**Redux State Management in React**

**Introduction**

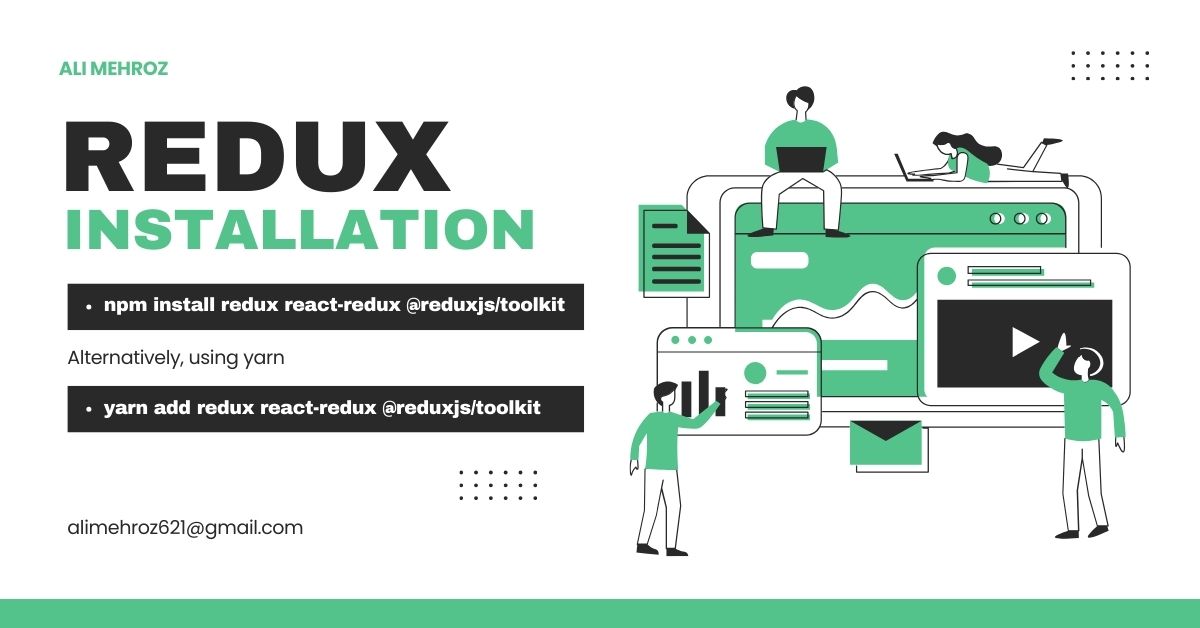
Redux is a state management library for JavaScript applications, commonly used with React.js. It helps manage and centralize application state, making it predictable and easier to debug.

**Why Use Redux?**

* Centralized state management
* Predictable state updates
* Debugging tools like Redux DevTools
* Easy state sharing across components
* Improves performance by minimizing unnecessary re-renders

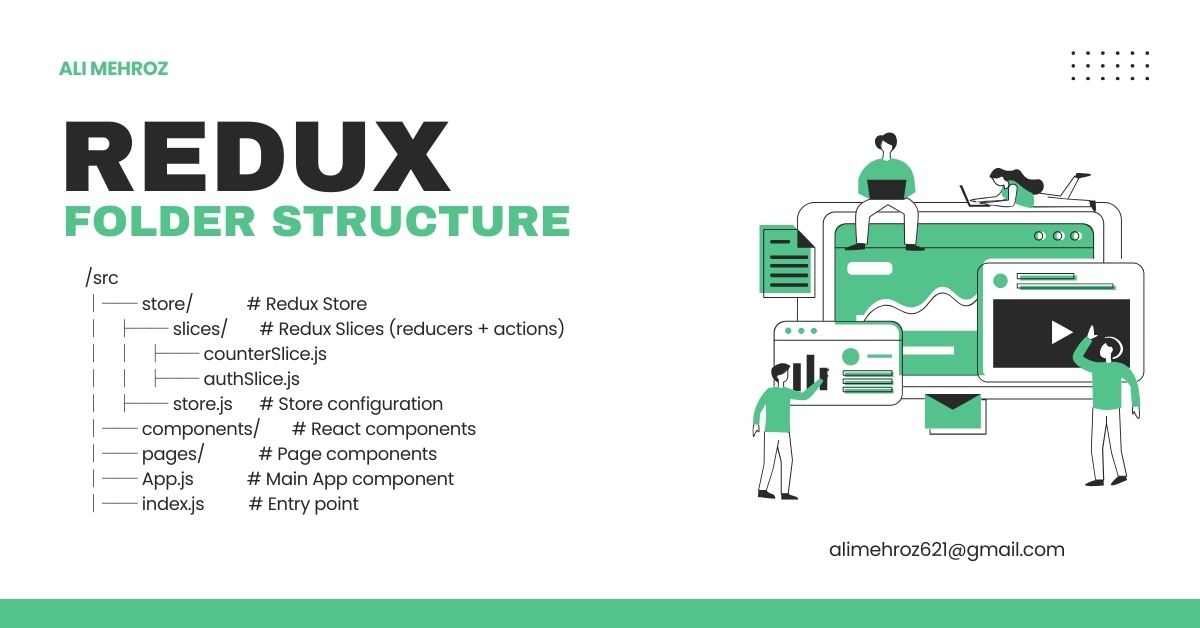
**Redux Installation**

* To get started with Redux in a React application, install the necessary packages



**Redux Folder Structure**

* A well-structured Redux implementation follows this pattern

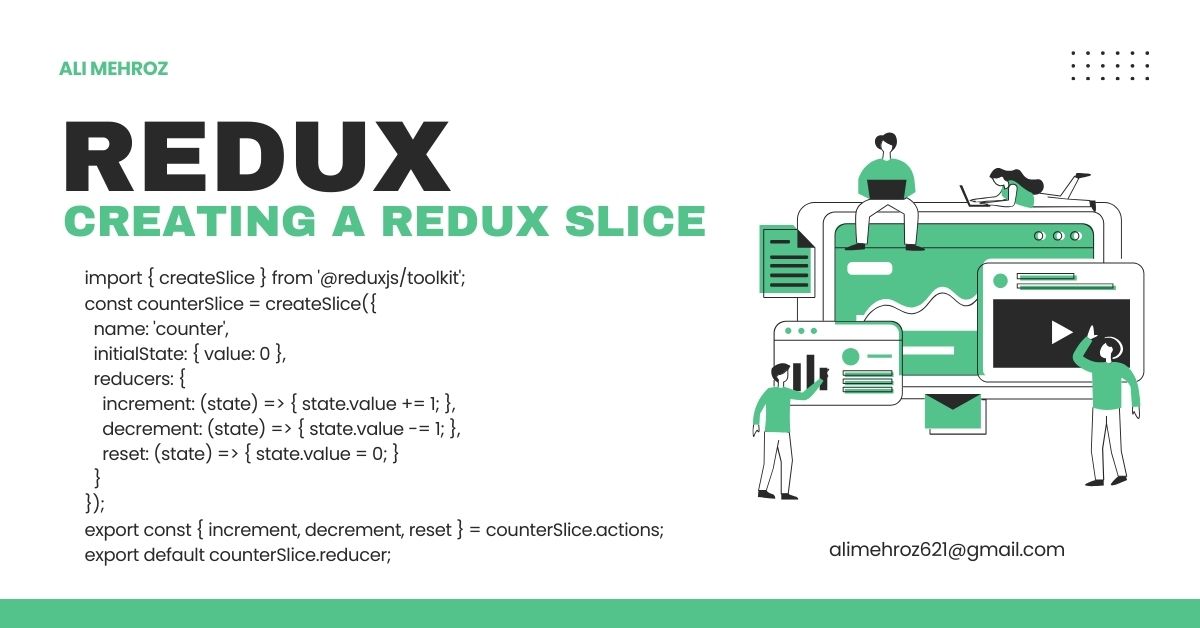


**Setting Up Redux Store**

* Create a **store.js** file inside the **store/** directory

****

* **Creating a Redux Slice** A Redux slice consists of a reducer and corresponding actions.
* Create a file **counterSlice.js** inside the **store/slices/** directory

****

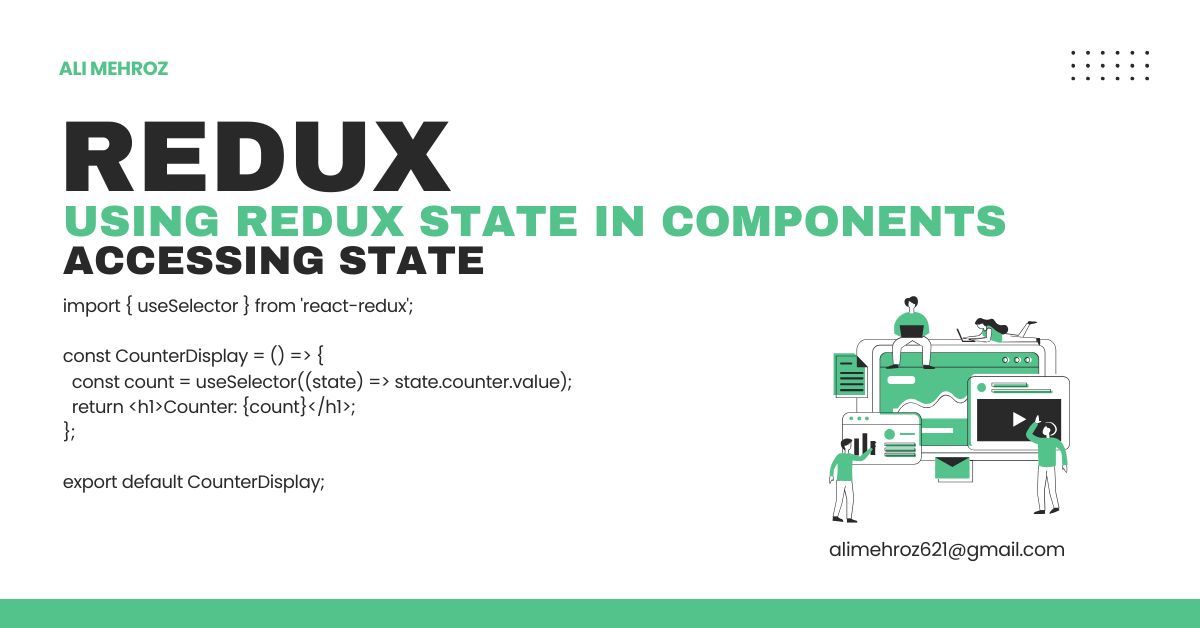
**Providing Store to the Application**

* Wrap your app with the **Provider** component in **index.js** directory

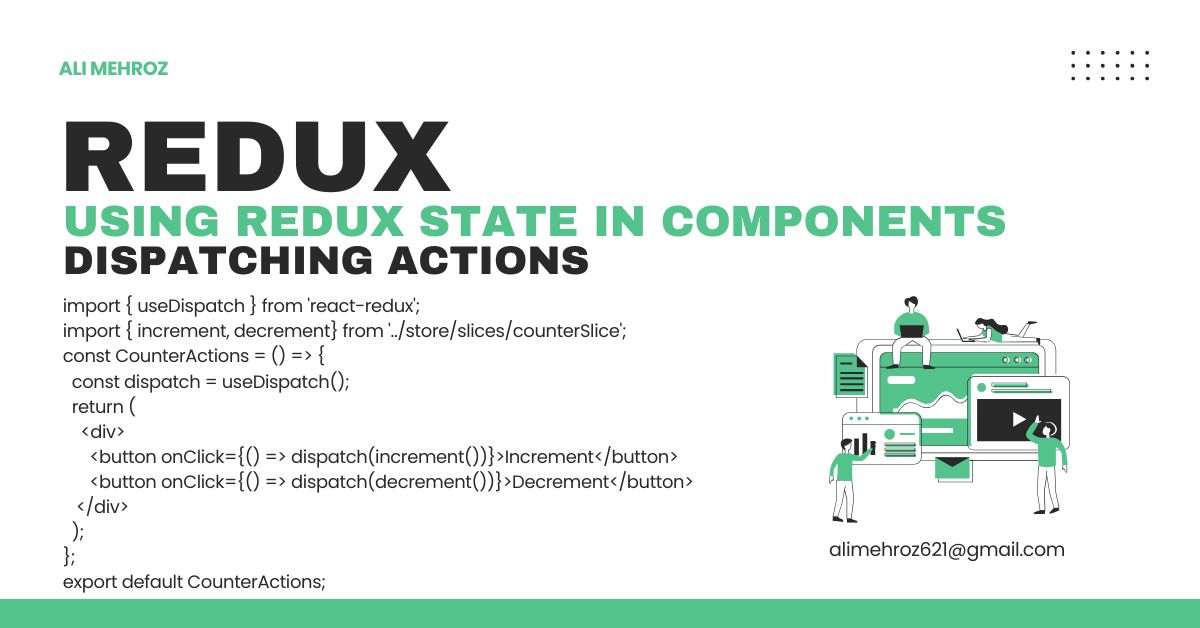
****

**Providing Store to the Application**

* Use the **useSelector** hook to access the Redux state inside a component

****

* Use the useDispatch hook to dispatch actions

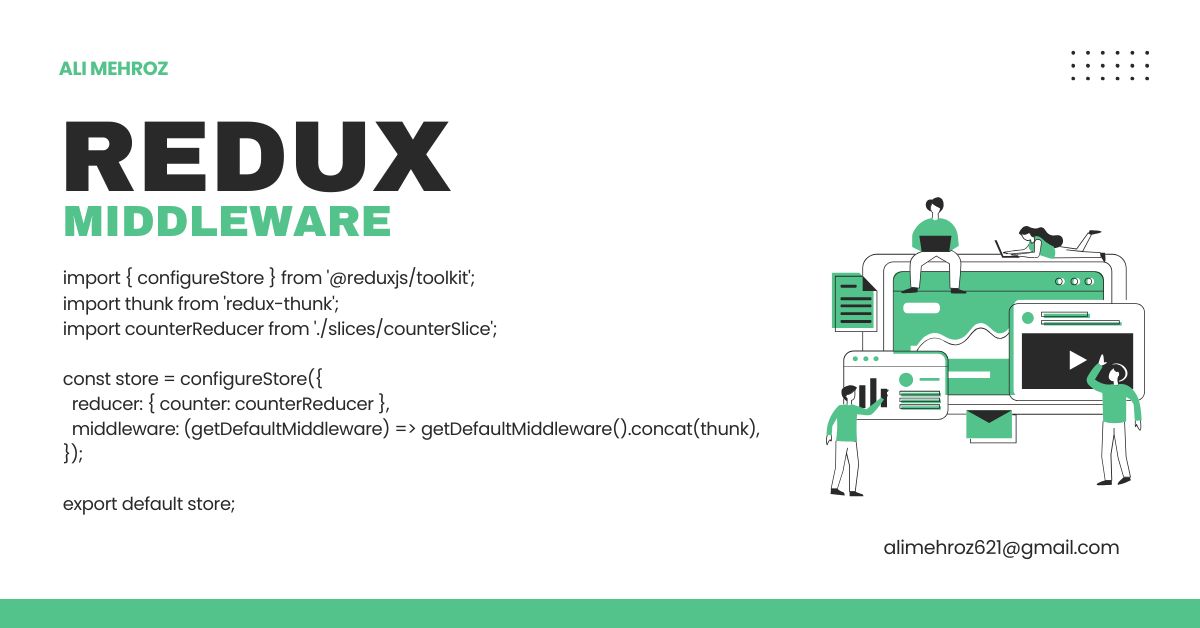
****

**Middleware in Redux**

* Middleware like Redux Thunk allows handling asynchronous logic in Redux.
* Install Redux Thunk



* Modify **store.js** to include middleware

****

**Best Practices for using Redux**

* Use Redux only when state needs to be shared across multiple components.
* Keep the Redux store minimal and avoid storing UI state.
* Use Redux Toolkit to simplify the Redux setup.
* Use **createAsyncThunk** for API calls instead of handling them inside components.

**Conclusion**

Redux is a powerful state management tool for React applications. With Redux Toolkit, managing global state becomes more efficient and easier to implement.