Setting Up Protected Routes with Redux Toolkit

This guide explains the step-by-step process for setting up protected routes in a React application using Redux Toolkit. We'll cover everything from configuring the Redux store to implementing the `ProtectedRoute` component.

---

1. Install Required Dependencies

Ensure you have the required dependencies installed:

```
npm install @reduxjs/toolkit react-redux react-router-dom react-loader-spinner
```

---

2. Configure `authSlice`

The `authSlice` manages the authentication state of the application.

Create `authSlice.js`

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

const authSlice = createSlice({
  name: 'auth',
  initialState: {
    user: null,  // Holds authenticated user info
    isAuthenticated: false, // Tracks authentication status
  },
  reducers: {
    setUser: (state, action) => {
      state.user = action.payload;
      state.isAuthenticated = true;
    },
    logout: (state) => {
      state.user = null;
      state.isAuthenticated = false;
    },
  },
});

export const { setUser, logout } = authSlice.actions;
export default authSlice.reducer;
```

---

3. Configure the Redux Store

The Redux store combines all slices of the application. Add the `authSlice` to the store configuration.

Create `store.js`

```
import { configureStore } from '@reduxjs/toolkit';
import authReducer from './authSlice';

export const store = configureStore({
  reducer: {
    auth: authReducer, // Add authSlice reducer
  },
});
```

Provide the Store to Your App

Wrap your application in the `Provider` component from `react-redux` and pass the store as a prop.

Example: `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')
);
```

---

4. Create the `ProtectedRoute` Component

The `ProtectedRoute` component restricts access to certain routes based on the user's authentication status.

Code for `ProtectedRoute.js`

```
import React from 'react';
import { useSelector } from 'react-redux';
import { RotatingLines } from 'react-loader-spinner';
import { Navigate, Outlet } from 'react-router-dom';

const ProtectedRoute = () => {
  const isAuthenticated = useSelector((state) => state.auth.isAuthenticated); //
Access auth state
```

```
  const Loading = false; // Replace with actual loading logic if required

  // Render a loading spinner while authentication is in progress
  if (Loading) {
    return (
      <div className="loading">
        <RotatingLines
          visible={true}
          height="96"
          width="96"
          color="grey"
          strokeWidth="5"
          animationDuration="0.75"
          ariaLabel="rotating-lines-loading"
        />
      </div>
    );
  }

  // Check if the user is authenticated
  return (
    <div>
      {isAuthenticated ? (
        <Outlet /> // Render protected content
      ) : (
        <Navigate to="/login" /> // Redirect unauthenticated users
      )}
    </div>
  );
};

export default ProtectedRoute;
```

---

5. Use `ProtectedRoute` in Your Application

Define routes in your application and use `ProtectedRoute` to restrict access to certain routes.

Example: `App.js`

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import ProtectedRoute from './ProtectedRoute';
import HomePage from './HomePage';
import LoginPage from './LoginPage';

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/login" element={<LoginPage />} />
```

```
          <Route path="/" element={<ProtectedRoute />}>
            <Route path="/" element={<HomePage />} />
          </Route>
        </Routes>
      </Router>
    );
};

export default App;
```

---

6. Testing Authentication

To Authenticate a User:

Dispatch the `setUser` action when a user logs in successfully. Example:

```
import { setUser } from './authSlice';
import { useDispatch } from 'react-redux';

const LoginPage = () => {
  const dispatch = useDispatch();

  const handleLogin = () => {
    const userData = { id: 1, name: 'John Doe' }; // Replace with actual user data
    dispatch(setUser(userData));
  };

  return <button onClick={handleLogin}>Login</button>;
};

export default LoginPage;
```

To Logout a User:

Dispatch the `logout` action when a user logs out. Example:

```
import { logout } from './authSlice';
import { useDispatch } from 'react-redux';

const LogoutButton = () => {
  const dispatch = useDispatch();

  const handleLogout = () => {
    dispatch(logout());
  };

  return <button onClick={handleLogout}>Logout</button>;
};
```

```
export default LogoutButton;
```

Summary

This setup ensures that:

1. Authentication status (`isAuthenticated`) is managed via Redux Toolkit.
2. Protected routes use `ProtectedRoute` to restrict access.
3. Users are redirected to the login page if unauthenticated.

By following these steps, you can effectively secure parts of your web application using Redux Toolkit and React Router.