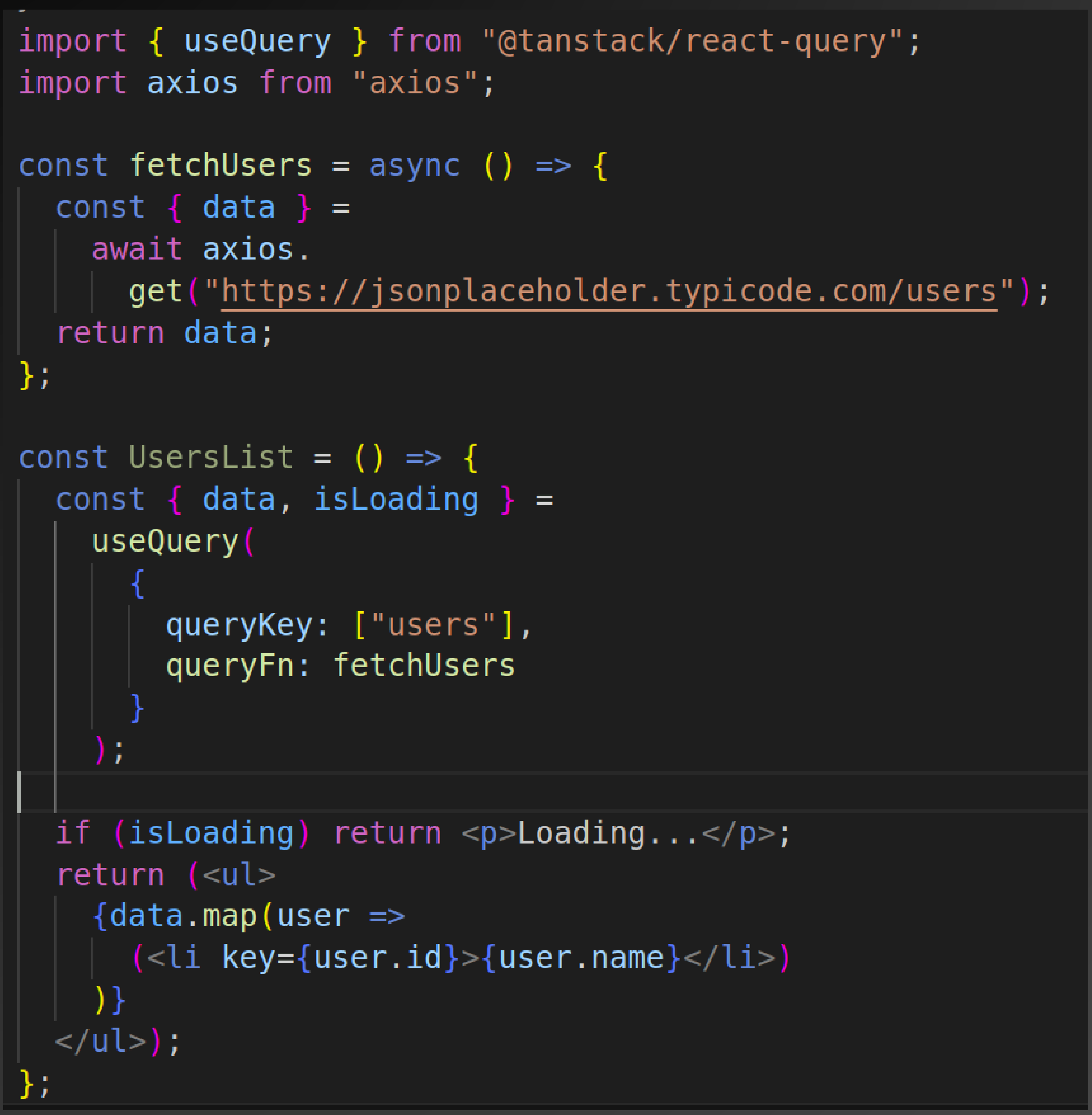
React.js

🔥 React Query vs. Redux Toolkit – Which One Should You Use? 🤔

LocalStorage for Simple Offline Storage

Feature	React Query	Redux Toolkit
Purpose	API fetching & caching	Global client state
Auto Caching	✅ Yes	❌ No (manual caching)
Background Updates	✅ Yes	❌ No
Boilerplate	🚀 Minimal	📜 Requires reducers & actions
Best For	Data fetching, API state	UI state, authentication

React Query for API Fetching



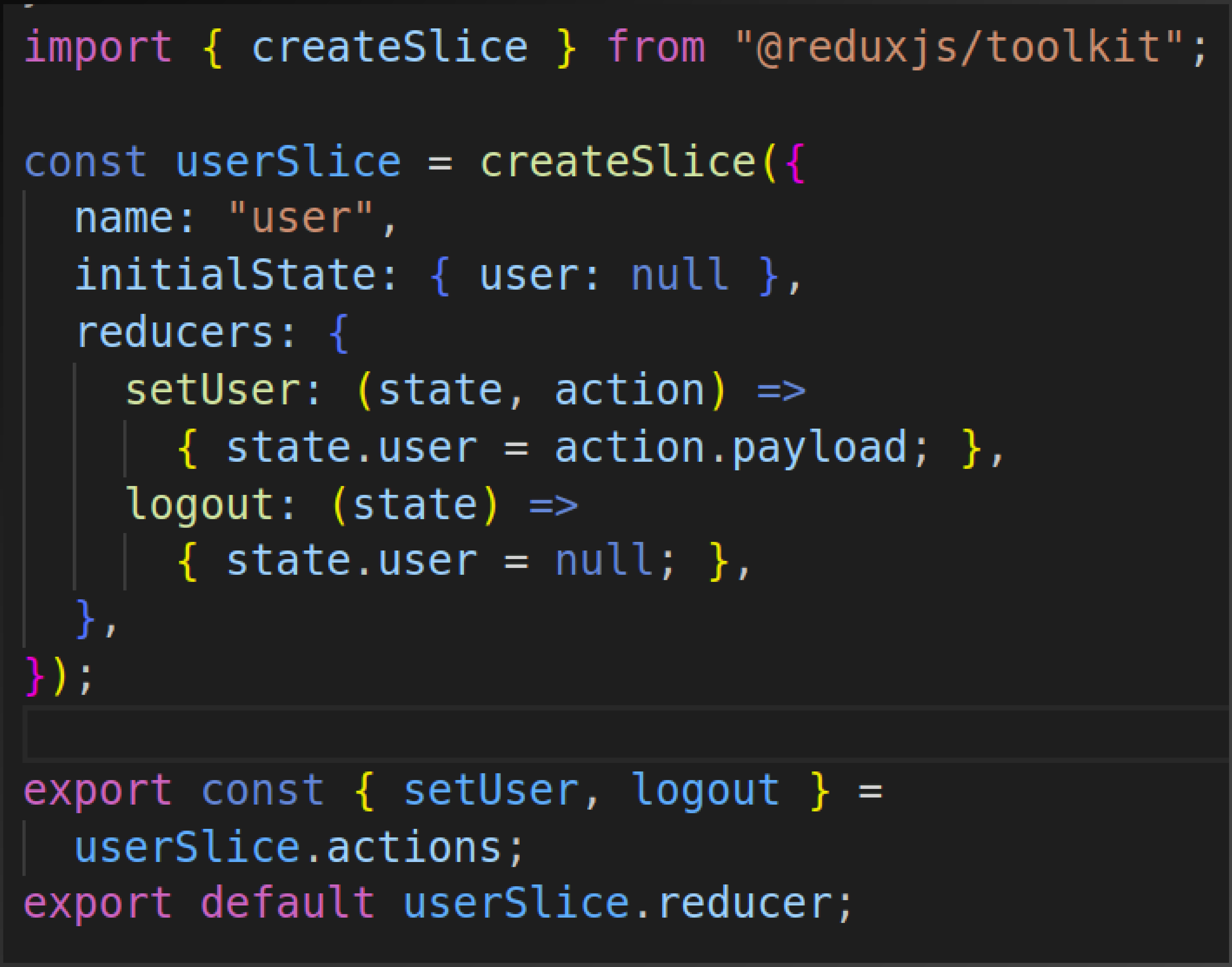
```
import { useQuery } from "@tanstack/react-query";
import axios from "axios";

const fetchUsers = async () => {
  const { data } =
    await axios.
      get("https://jsonplaceholder.typicode.com/users");
  return data;
};

const UsersList = () => {
  const { data, isLoading } =
    useQuery(
      {
        queryKey: ["users"],
        queryFn: fetchUsers
      }
    );

  if (isLoading) return <p>Loading...</p>;
  return (<ul>
    {data.map(user =>
      (<li key={user.id}>{user.name}</li>)
    )}
  </ul>);
};
```

◆ Redux Toolkit for Global State



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

const userSlice = createSlice({
  name: "user",
  initialState: { user: null },
  reducers: {
    setUser: (state, action) =>
      { state.user = action.payload; },
    logout: (state) =>
      { state.user = null; },
  },
});

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

🚀 Pro Tip: Use both together for the best experience!

- Use **React Query** for API state (fetching, caching, background syncing) and **Redux** Toolkit for global UI state (authentication, theme, modals).
- **Avoid storing API data in Redux!** Let React Query handle it to reduce unnecessary re-renders and improve performance.
- Use **React Query's invalidateQueries** to automatically refresh data after a mutation instead of manually updating the Redux state.
- **For large-scale apps**, keep API state in **React Query** and client preferences (e.g., dark mode, sidebar toggle) in **Redux Toolkit**.