NAME: JANHVI SINGH

REGISTRATION NUMBER: 22BCE8544

Assignment 14

ThemeContext.jsx

// src/ThemeContext.js

import React, { createContext, useState, useContext } from 'react';

// Create the ThemeContext

const ThemeContext = createContext();

// Custom hook to provide theme context

export const useTheme = () => {

  return useContext(ThemeContext);

};

// ThemeProvider component to wrap around components that need access to the theme

export const ThemeProvider = ({ children }) => {

  const [theme, setTheme] = useState('light'); // Default theme

  const toggleTheme = () => {

    setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));

  };

  return (

    <ThemeContext.Provider value={{ theme, toggleTheme }}>

      {children}

    </ThemeContext.Provider>

  );

};

styles.css

/\* src/styles.css \*/

body {

  margin: 0;

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

    sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  transition: background-color 0.3s ease, color 0.3s ease;

}

.light {

  background-color: #f0f2f5;

  color: #333;

}

.dark {

  background-color: #282c34;

  color: #f0f0f0;

}

.container {

  padding: 20px;

  max-width: 800px;

  margin: 20px auto;

  border-radius: 8px;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

  transition: background-color 0.3s ease, box-shadow 0.3s ease;

}

.light .container {

  background-color: #ffffff;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

.dark .container {

  background-color: #3a3f4a;

  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);

}

h1, h2 {

  text-align: center;

  margin-bottom: 20px;

}

button {

  padding: 10px 15px;

  border: none;

  border-radius: 5px;

  cursor: pointer;

  font-size: 16px;

  transition: background-color 0.3s ease, color 0.3s ease;

  margin-right: 10px;

}

.light button {

  background-color: #007bff;

  color: white;

}

.light button:hover {

  background-color: #0056b3;

}

.dark button {

  background-color: #61dafb;

  color: #282c34;

}

.dark button:hover {

  background-color: #21a1f1;

}

input[type="text"] {

  width: 100%;

  padding: 10px;

  margin-bottom: 20px;

  border: 1px solid #ccc;

  border-radius: 5px;

  box-sizing: border-box;

  font-size: 16px;

  transition: border-color 0.3s ease, background-color 0.3s ease, color 0.3s ease;

}

.light input[type="text"] {

  background-color: white;

  color: #333;

  border-color: #ccc;

}

.dark input[type="text"] {

  background-color: #4a4f59;

  color: #f0f0f0;

  border-color: #555;

}

ul {

  list-style: none;

  padding: 0;

}

li {

  padding: 10px 0;

  border-bottom: 1px solid #eee;

}

.light li {

  border-bottom-color: #eee;

}

.dark li {

  border-bottom-color: #555;

}

li:last-child {

  border-bottom: none;

}

.user-item {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 8px 0;

}

.user-item button {

  margin-left: 10px;

  padding: 6px 10px;

  font-size: 14px;

}

.sidebar {

  width: 200px;

  padding: 20px;

  border-right: 1px solid #ddd;

  min-height: calc(100vh - 40px);

  transition: background-color 0.3s ease, border-color 0.3s ease;

}

.light .sidebar {

  background-color: #f8f9fa;

  border-right-color: #ddd;

}

.dark .sidebar {

  background-color: #33373e;

  border-right-color: #444;

}

.sidebar h3 {

  margin-top: 0;

}

.sidebar-button {

  display: block;

  width: 100%;

  text-align: left;

  margin-bottom: 10px;

}

.main-content {

  flex-grow: 1;

  padding: 20px;

}

.dashboard-layout {

  display: flex;

  min-height: 100vh;

}

UserList.jsx

// src/UserList.jsx

import React, { useState, useMemo, useCallback, memo } from 'react';

import { useTheme } from './ThemeContext';

// Mock user data

const mockUsers = [

  { id: 1, name: 'Alice Smith', email: 'alice@example.com' },

  { id: 2, name: 'Bob Johnson', email: 'bob@example.com' },

  { id: 3, name: 'Charlie Brown', email: 'charlie@example.com' },

  { id: 4, name: 'Diana Prince', email: 'diana@example.com' },

  { id: 5, name: 'Eve Adams', email: 'eve@example.com' },

  { id: 6, name: 'Frank White', email: 'frank@example.com' },

];

// Memoized child component for displaying a single user

const UserItem = memo(({ user, onUserClick }) => {

  console.log(`Rendering UserItem: ${user.name}`);

  const { theme } = useTheme();

  return (

    <li className="user-item">

      <span>{user.name} - {user.email}</span>

      <button onClick={() => onUserClick(user.id)} className={theme}>

        View Details

      </button>

    </li>

  );

});

const UserList = () => {

  const { theme, toggleTheme } = useTheme();

  const [searchTerm, setSearchTerm] = useState('');

  // Optimize list filtering with useMemo

  const filteredUsers = useMemo(() => {

    console.log('Recalculating filtered users...');

    return mockUsers.filter((user) =>

      user.name.toLowerCase().includes(searchTerm.toLowerCase())

    );

  }, [searchTerm]); // Dependency array: recalculate only when searchTerm changes

  // Memoize event handler with useCallback

  const handleUserClick = useCallback((userId) => {

    console.log(`User with ID ${userId} clicked!`);

    // In a real app, you might navigate or open a modal here

  }, []); // Dependency array is empty as it doesn't depend on any props or state that change

  // Memoize the search input change handler

  const handleSearchChange = useCallback((event) => {

    setSearchTerm(event.target.value);

  }, []); // Empty dependency array as it only sets state

  return (

    <div className={`container ${theme}`}>

      <h1>User Dashboard</h1>

      <div style={{ marginBottom: '20px', textAlign: 'center' }}>

        <button onClick={toggleTheme} className={theme}>

          Toggle Theme ({theme === 'light' ? 'Dark' : 'Light'})

        </button>

      </div>

      <h2>User List</h2>

      <input

        type="text"

        placeholder="Search users by name..."

        value={searchTerm}

        onChange={handleSearchChange}

        className={theme}

      />

      {filteredUsers.length > 0 ? (

        <ul>

          {filteredUsers.map((user) => (

            <UserItem key={user.id} user={user} onUserClick={handleUserClick} />

          ))}

        </ul>

      ) : (

        <p>No users found.</p>

      )}

    </div>

  );

};

export default UserList;

App.jsx

// src/App.jsx

import React, { memo } from 'react';

import UserList from './UserList';

import { ThemeProvider, useTheme } from './ThemeContext';

import './styles.css';

// Bonus: Memoized Sidebar component

const Sidebar = memo(() => {

  const { theme } = useTheme();

  console.log('Rendering Sidebar');

  return (

    <div className={`sidebar ${theme}`}>

      <h3>Dashboard Navigation</h3>

      <button className={`sidebar-button ${theme}`}>Dashboard</button>

      <button className={`sidebar-button ${theme}`}>Settings</button>

      <button className={`sidebar-button ${theme}`}>Reports</button>

    </div>

  );

});

function App() {

  const { theme } = useTheme(); // This is inside App component, not directly rendered by ThemeProvider

  return (

    <div className={theme}> {/\* Apply theme class to the root div \*/}

      <div className="dashboard-layout">

        <Sidebar />

        <div className="main-content">

          <UserList />

        </div>

      </div>

    </div>

  );

}

// Wrap App with ThemeProvider for global theme access

const AppWithTheme = () => (

  <ThemeProvider>

    <App />

  </ThemeProvider>

);

export default AppWithTheme;

README

**# React Theme Dashboard**

This project demonstrates optimizing component performance and state management in React using `useContext`, `useMemo`, and `useCallback`.

**## How to Run**

1.  Clone the repository:

    `git clone <repository-url>`

2.  Navigate to the project directory:

    `cd react-theme-dashboard`

3.  Install dependencies:

    `npm install`

4.  Start the development server:

    `npm run dev`

The application will be accessible at `http://localhost:5173/` (or another port as indicated by Vite).

**## Assignment Tasks Implementation**

**### Theme Toggle with `useContext`**

The application features a global theme context to toggle between light and dark modes.

\* **\*\*`src/ThemeContext.js`\*\***:

    \* `ThemeContext` is created using `createContext()`.

    \* `useTheme` is a custom hook that simplifies consuming the context.

    \* `ThemeProvider` component manages the `theme` state (`light` or `dark`) using `useState` and provides it, along with a `toggleTheme` function, to its children via `ThemeContext.Provider`.

\* **\*\*`src/App.jsx`\*\***:

    \* The `App` component is wrapped within `ThemeProvider` in `AppWithTheme`, making the theme context available throughout the application.

    \* The current theme class is applied to the root `div` in `App.jsx` to control global styles.

\* **\*\*`src/UserList.jsx`\*\***:

    \* The `useTheme` hook is used to access the `theme` and `toggleTheme` function.

    \* A button is added to `UserList` that calls `toggleTheme` to switch the application's theme.

    \* Styles in `src/styles.css` apply different `background-color` and `color` based on the `light` or `dark` class applied by the theme context.

**### List Filtering Optimization with `useMemo`**

The user list features a search filter that is optimized using `useMemo`.

\* **\*\*`src/UserList.jsx`\*\***:

    \* A mock `mockUsers` array is provided.

    \* The `searchTerm` state manages the input value for the search.

    \* `filteredUsers` is memoized using `useMemo`. The filtering logic (`mockUsers.filter(...)`) is wrapped in `useMemo`.

    \* The dependency array for `useMemo` is `[searchTerm]`. This ensures that the `filteredUsers` list is only recalculated when the `searchTerm` changes, preventing unnecessary re-calculations on every render of the `UserList` component (e.g., when the theme is toggled or other unrelated state changes). This significantly improves performance for larger datasets.

**### Memoizing Event Handlers with `useCallback`**

Event handlers passed to child components are memoized using `useCallback` to avoid unnecessary re-renders of those children.

\* **\*\*`src/UserList.jsx`\*\***:

    \* The `handleUserClick` function, which is passed as a prop to the `UserItem` child component, is wrapped in `useCallback`.

    \* The dependency array for `handleUserClick` is empty (`[]`) because it doesn't depend on any props or state that would change during the component's lifecycle. This ensures that `handleUserClick` remains the same reference across renders.

    \* The `UserItem` component itself is wrapped in `memo` (`const UserItem = memo(...)`). This higher-order component will only re-render `UserItem` if its props (`user` or `onUserClick`) have shallowly changed. Since `onUserClick` is memoized by `useCallback`, `UserItem` will only re-render if the `user` prop changes, preventing unnecessary re-renders when the `UserList` component re-renders for other reasons (e.g., `searchTerm` changes, but the specific `user` for that item hasn't).

    \* Similarly, `handleSearchChange` is also memoized with `useCallback` to ensure the input's `onChange` handler does not cause unnecessary re-renders of the input itself.

**### Bonus (Optional)**

\* **\*\*Additional UI Elements:\*\*** A `Sidebar` component is added to `src/App.jsx`. It also consumes the `ThemeContext` and applies styles based on the current theme, demonstrating how other UI elements can react to theme changes globally.

\* **\*\*Memoized Child Component:\*\*** The `UserItem` component (`src/UserList.jsx`) is a memoized version using `React.memo`. This ensures that individual `UserItem` components only re-render when their `user` prop (or the memoized `onUserClick` prop) changes, further optimizing the rendering of the list.

**## Learning Outcomes**

By completing this assignment, I have:

\* Gained practical experience using `useContext` to effectively manage global state (theme) in a React application, enabling consistent UI across different components.

\* Learned how to optimize component performance using `useMemo` by preventing redundant calculations of derived state (filtered user list), which is crucial for handling large datasets efficiently.

\* Understood how `useCallback` prevents unnecessary re-renders of child components by memoizing event handlers, thereby maintaining reference equality and leveraging `React.memo` effectively.

\* Developed an understanding of best practices for performance optimization in React, which is essential for building scalable and efficient applications, aligning with LaunchDarkly's focus on high-performance features.

Screenshot



