HyperionDev

Full-Stack Web Development

HyperionDev

Managing State in React

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Objectives

- Explain the need for global state management in React.
- Use Context API to create and manage global state.
- Consume global state in React components using `useContext`.
- Understand the use cases and limitations of Context API for state management.

Introduction to State Management in React

- In React, state refers to the data that determines how your application behaves. As applications grow, managing state across multiple components becomes more complex. While `useState` works well for local state management, it becomes cumbersome when state needs to be shared among many components.
- To address this issue, React provides the Context API, which enables the management of global state without needing to pass props manually through every level of your component tree.
- Key Concepts:
  - **Local State**: Managed within a single component using hooks like `useState`.
  - **Global State**: Shared among multiple components and needs to be accessed/updated from different parts of your application.

Global State Management in React: The Context API

- The Context API provides a way to share values (global state) between components without passing props explicitly. It includes:
  - **Creating Context** - The context object is created using React.createContext().
  - **Provider Component** - A component that "provides" the global state to the entire application or specific part of the component tree.
  - **Consumer Component/`useContext` Hook** - Components that "consume" the global state provided by the context.

Managing Global State with Context API and useContext

- **Step 1: Setting Up the Project**
  - Create a new React project(feel free to skip this step if you already have a project set up):

```
➜  ~ npx create-react-app react-context-demo
cd react-context-demo
```

  - Start the development server:

```
➜  ~ npm start
```

Step 2: Creating and Providing the Context

- In this example, we will manage a global state for user authentication.
  - Create a `context` folder in your `src` directory:

```
✗ mkdir src/context
✗ touch src/context/AuthContext.js
```

  - Define the Context in `AuthContext.js`:

src > context > JS AuthContext.js > ...

```jsx
import React, { createContext, useState } from 'react';

// Create the context
export const AuthContext = createContext();

// Create the provider component
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);

  const login = (username) => {
    setUser({ name: username });
  };

  const logout = () => {
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

- AuthContext: The context object created using `createContext()`.
- AuthProvider: A component that wraps around the entire app (or specific parts) to provide global state.
- State and Methods: user, login, and logout functions are provided as global state and actions.

Wrap the App with the Provider in index.js

src > JS index.js > ...

```
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3   import './index.css';
4   import App from './App';
5   import reportWebVitals from './reportWebVitals';
6   import { AuthProvider } from './context/AuthContext';
7
8
9   const root = ReactDOM.createRoot(document.getElementById('root'));
10  root.render(
11    <React.StrictMode>
12      <AuthProvider>
13        <App />
14      </AuthProvider>
15    </React.StrictMode>
16  );
```

- Now, all components within the app can access and update the global state.

Step 3: Consuming the Context in Components using useContext

- Let’s create components that consume this global state.
  - Create a Navbar component that displays user login status:

src > components > JS Navbar.js > [@] default

```jsx
import React, { useContext } from 'react';
import { Link } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';

const Navbar = () => {
  const { user, logout } = useContext(AuthContext);

  return (
    <nav className="navbar">
      <h1>My React App</h1>
      <div className="links">
        <Link to="/">Home</Link>
        <Link to="/profile">Profile</Link>
        {user ? (
          <>
            <span>Hello, {user.name}</span>
            <button onClick={logout}>Logout</button>
          </>
        ) : (
          <Link to="/login">Login</Link>
        )}
      </div>
    </nav>
  );
};

export default Navbar;
```

- The useContext(AuthContext) hook allows you to consume the user state and logout function in this component.
- The Navbar conditionally renders content based on whether the user is logged in or not.
- If the user is logged in, it shows a welcome message and a logout button. If not, it shows a login link.

Create a Login component to handle login actions

src > components > JS Login.js > default

```
1  import React, { useState, useContext } from 'react';
2  import { AuthContext } from '../context/AuthContext';
3
4  const Login = () => {
5    const [username, setUsername] = useState('');
6    const { login } = useContext(AuthContext);
7
8    const handleLogin = () => {
9      login(username);
10     setUsername(''); // Clear the input field after login
11   };
12
13   return (
14     <div>
15       <input
16         type="text"
17         placeholder="Enter your username"
18         value={username}
19         onChange={(e) => setUsername(e.target.value)}
20       />
21       <button onClick={handleLogin}>Login</button>
22     </div>
23   );
24 };
25
26 export default Login;
```

- The login function from the context is used to update the global user state when the button is clicked.

Step 4: Creating the Home and Profile Pages

- Create two new files inside the pages/ directory: Home.js and Profile.js.

src > pages > JS Home.js > [] default

```
1   import React from 'react';
2
3   const Home = () => {
4     return (
5       <div className="home">
6         <h2>Welcome to the Home Page</h2>
7         <p>This is a simple React application demonstrating state management using the Context API.
          </p>
8       </div>
9     );
10  };
11
12  export default Home;
```

src > pages > JS Profile.js > default

```jsx
import React, { useContext } from 'react';
import { AuthContext } from '../context/AuthContext';

const Profile = () => {
  const { user } = useContext(AuthContext);

  return (
    <div className="profile">
      <h2>Profile Page</h2>
      {user ? (
        <div>
          <p>Username: {user.name}</p>
          <p>Email: user@example.com (Sample)</p>
        </div>
      ) : (
        <p>Please log in to view your profile information.</p>
      )}
    </div>
  );
};

export default Profile;
```

Add the components to App.js

src > JS App.js > default

```
import './App.css';
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Profile from './pages/Profile';
import Login from './components/Login';

function App() {
  return (
    <Router>
      <div className="App">
        <Navbar />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/profile" element={<Profile />} />
          <Route path="/login" element={<Login />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

- The Router component wraps around the application to enable routing.
- The Routes component defines different routes (e.g., Home, Profile, Login) and maps them to specific components.

Running and Testing the Application

- When you start the application, you'll see:
  - The Navbar initially shows "Please log in".
  - Enter a username in the input field and click "Login".
  - The Navbar updates to show "Welcome, [username]" with a "Logout" button.
  - Clicking "Logout" resets the state to show the initial message.

Hyperiondev

Questions and Answers

HyperionDev

Thank You for attending!