**Assignment 3: Global State Management**

**Implement global state management in React using Context API.**

**Share state between React components that are not directly related.**

The React Context API is a great way to manage global state and share it between React components. Below is an example of how you can implement global state management using the Context API to share state between components that are not directly related.

Step 1: Create the Context

Create a new file, say AppContext.js, to define your context:

| // AppContext.js  import React, { createContext, useReducer, useContext } from 'react';  // Define the initial state  const initialState = {  count: 0,  username: 'Guest',  };  // Create the context  const AppContext = createContext();  // Create a provider component  const AppProvider = ({ children }) => {  // Define a reducer function to handle state changes  const reducer = (state, action) => {  switch (action.type) {  case 'INCREMENT':  return { ...state, count: state.count + 1 };  case 'DECREMENT':  return { ...state, count: state.count - 1 };  case 'SET\_USERNAME':  return { ...state, username: action.payload };  default:  return state;  }  };  // Use the useReducer hook to manage state  const [state, dispatch] = useReducer(reducer, initialState);  return (  <AppContext.Provider value={{ state, dispatch }}>{children}</AppContext.Provider>  );  };  // Create a custom hook to easily access the context in components  const useAppContext = () => {  const context = useContext(AppContext);  if (!context) {  throw new Error('useAppContext must be used within an AppProvider');  }  return context;  };  export { AppProvider, useAppContext }; |
| --- |

Step 2: Use the Context in Components

Now, you can use the AppProvider in your main App.js file and access the global state in any component using the useAppContext hook.

| // App.js  import React from 'react';  import { AppProvider } from './AppContext';  import ComponentA from './ComponentA';  import ComponentB from './ComponentB';  function App() {  return (  <AppProvider>  <div>  <h1>Global State Management Example</h1>  <ComponentA />  <ComponentB />  </div>  </AppProvider>  );  }  export default App; |
| --- |

Step 3: Create Components that Use the Global State

Create components, say ComponentA.js and ComponentB.js, that use the global state:

| // ComponentA.js  import React from 'react';  import { useAppContext } from './AppContext';  function ComponentA() {  const { state, dispatch } = useAppContext();  return (  <div>  <h2>Component A</h2>  <p>Count: {state.count}</p>  <button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>  <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>  </div>  );  }  export default ComponentA; |
| --- |

| // ComponentB.js  import React from 'react';  import { useAppContext } from './AppContext';  function ComponentB() {  const { state, dispatch } = useAppContext();  return (  <div>  <h2>Component B</h2>  <p>Username: {state.username}</p>  <input  type="text"  value={state.username}  onChange={(e) => dispatch({ type: 'SET\_USERNAME', payload: e.target.value })}  />  </div>  );  }  export default ComponentB; |
| --- |

Now, both ComponentA and ComponentB can independently access and update the global state.

Step 4: Run the App

Start your development server:

npm start

Visit http://localhost:3000 in your browser. You should see your React app with global state management, and changes in one component will reflect in the other component as they share the same global state.