**Assignment 9: Context API**

**Implement a simple global state using React Context API.**

**Share state between components without prop drilling.**

The React Context API is a great way to manage global state in your application and avoid prop drilling. Here's a simple example of implementing a global state using the Context API:

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,**  **};**  **// 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 };**  **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 Counter from './Counter';  const App = () => {  return (  <AppProvider>  <div>  <h1>Global State Example</h1>  <Counter />  </div>  </AppProvider>  );  };  export default App; |
| --- |

Step 3: Create a Component that Uses the Global State

Now, create a component, say Counter.js, that uses the global state:

| // Counter.js  import React from 'react';  import { useAppContext } from './AppContext';  const Counter = () => {  // Access the global state and dispatch function using the custom hook  const { state, dispatch } = useAppContext();  return (  <div>  <p>Count: {state.count}</p>  <button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>  <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>  </div>  );  };  export default Counter; |
| --- |

Now, any component wrapped with the AppProvider can access the global state using the useAppContext hook, making it easy to share state without prop drilling.

Remember to adjust the example based on your specific use case and requirements. This is a basic example to demonstrate the concept of global state management with the React Context API.