## 1. Explain the need and Benefits of React Context API

React Context API is needed when you want to share data across multiple components without having to pass props manually at every level. In deeply nested component trees, "prop drilling" becomes a problem where props have to be passed through several intermediate components just to reach the desired child. This makes the code harder to maintain and scale, especially in large applications.

Using Context API helps in managing global data such as themes, language preferences, user authentication, etc., across your app in a clean and efficient manner. Instead of manually passing data down the component tree, you wrap your components with a Provider that shares the context value with any component that consumes it. This removes repetitive code and improves readability.

One of the biggest benefits of the Context API is that it promotes separation of concerns. Context allows components to subscribe to data only when necessary, making them more focused and modular. Additionally, it improves performance and maintainability by reducing unnecessary re-renders and avoiding tight coupling between unrelated components.

## 2. Working with createContext()

The createContext() function is a built-in method provided by React to create a context object. This context object includes two main components: a Provider, which supplies the data, and a Consumer, which allows components to access that data. The Provider component wraps around the parts of your app that need access to the shared data.

When you call createContext(), you can also specify a default value. This default is useful in case a component consumes the context without being wrapped in a Provider. For example, const ThemeContext = React.createContext('light'); will default to 'light' if no value is provided through a Provider.

To use the context, a component must either use the <ThemeContext.Consumer> component or the useContext() hook if it's a function component. The hook version is more concise and easier to work with in modern React apps. This setup allows for efficient and scalable data sharing without the clutter of prop chains.

## 3. List the types of Router Components

React Router provides a set of components to implement dynamic routing in React applications. These router components enable navigation among views and the rendering of UI based on the URL in the browser. The most common type is <BrowserRouter>, which uses the HTML5 history API to keep your UI in sync with the URL.

Another important component is <HashRouter>, which uses the hash portion of the URL (everything after #) to manage routing. This is often used when deploying apps on static file servers where server-side routing is not supported. It’s useful for simpler use cases or when dealing with older browsers or file-based hosting.

Additionally, React Router includes <Routes> and <Route> components to define path-based rendering logic. Other advanced components like <Navigate>, <Outlet>, and <Link> are used for redirects, nested routing, and navigation links, respectively. These components offer fine-grained control over routing behavior, making React Router a powerful solution for SPA navigation.

## Code for ThemeContext.js

import { createContext } from 'react';  
const ThemeContext = createContext('light');  
export default ThemeContext;

## Code for App.js

import React, { useState } from 'react';  
import EmployeesList from './components/EmployeesList';  
import ThemeContext from './ThemeContext';  
  
function App() {  
 const [theme, setTheme] = useState('light');  
  
 const toggleTheme = () => {  
 setTheme(prev => (prev === 'light' ? 'dark' : 'light'));  
 };  
  
 return (  
 <ThemeContext.Provider value={theme}>  
 <div className="App">  
 <h1>Employee Management</h1>  
 <button onClick={toggleTheme}>  
 Toggle Theme ({theme})  
 </button>  
 <EmployeesList />  
 </div>  
 </ThemeContext.Provider>  
 );  
}  
  
export default App;

## Code for EmployeesList.js

import React from 'react';  
import EmployeeCard from './EmployeeCard';  
  
const employeeData = [  
 { id: 1, name: 'Alice', position: 'Developer' },  
 { id: 2, name: 'Bob', position: 'Designer' },  
 { id: 3, name: 'Charlie', position: 'Manager' },  
];  
  
function EmployeesList() {  
 return (  
 <div>  
 {employeeData.map(emp => (  
 <EmployeeCard key={emp.id} employee={emp} />  
 ))}  
 </div>  
 );  
}  
  
export default EmployeesList;

## Code for EmployeeCard.js

import React, { useContext } from 'react';  
import ThemeContext from '../ThemeContext';  
  
function EmployeeCard({ employee }) {  
 const theme = useContext(ThemeContext);  
  
 const handleViewDetails = () => {  
 alert(`Name: ${employee.name}\nPosition: ${employee.position}`);  
 };  
  
 return (  
 <div className={`card ${theme}`} style={{ margin: '10px', padding: '10px' }}>  
 <h3>{employee.name}</h3>  
 <p>{employee.position}</p>  
 <button className={`btn ${theme}`} onClick={handleViewDetails}>  
 View Details  
 </button>  
 </div>  
 );  
}  
  
export default EmployeeCard;

## Output



