**Objectives**

* Explain the need and Benefits of React Context API
* Working with createContext()
* List the types of Router Components

In this hands-on lab, you will learn how to:

* Create a context to be used by child components
* Create a provider and consumer of the context

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **30 minutes.**

Developers of Apps Centric Solutions have created an employee management application which supports light and dark themes for the buttons. The current solution uses the react state and props to provide the theme name to be used from App component to Employee List component and from there to Employee Card component. Quality assurance team analyzed the solutions and found the technique being used to be a substandard one. React architect suggested to use the react context API to share the theme name with nested child components instead of passing them down using props from the parent component.

You are assigned the task of converting the application form props only to React Context API.

Application can be downloaded from below



1. Unzip the application and open it using VS Code
2. Go to terminal and execute *npm install* command to restore all the node modules



Figure 1: Restore node modules

1. Run the application once to see the output. Use npm start command.



Figure 2: Starting application

1. Explore the components present in **App.js**, **EmployeesList.js** and **EmployeeCard.js** files.
2. Create a new file with the name as **ThemeContext.js**. Define a new context in the file with the name as ThemeContext and assign it a default value of ‘light’ and export it as default form the module.
3. Open App component present in **App.js** file.
   1. Import the ThemeContext in App component.
   2. Define the theme context provider to be the entire JSX of the App component.
   3. Assign the value for the theme provider from the state of the component.
   4. Modify the call to EmployeeList component so that theme name is no longer passed as props.
4. Go to EmployeeList component present in **EmployeeList.js** file and modify it so that theme name is not passed explicitly to its child component.
5. Go to **EmployeeCard** component inside **EmployeeCard.js** file
   1. Import the ThemeContext into the component file
   2. Retrieve the value of the context with the help of **useContext()** and store it in a variable
   3. Use the variable to pass the className for the buttons.

Output:

