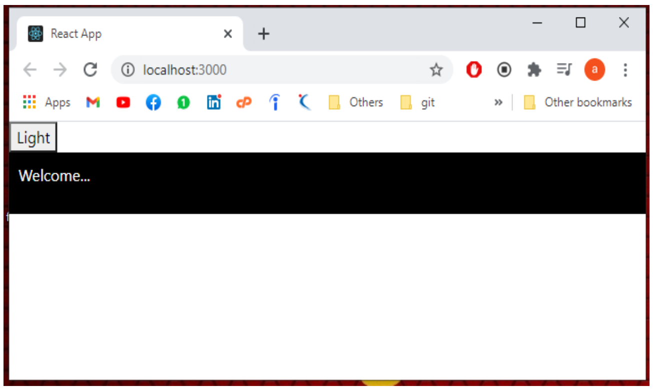
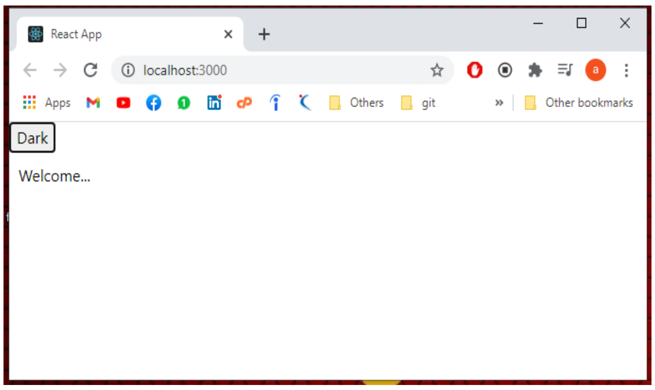
Working with useContext() Hook

- In a typical React application, data is passed parent to child via props, but this can be difficult for certain types of props (e.g. locale preference, UI theme) that are required by many components which are Nested at different levels within an application.
- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- Context provides a way to share values between components without having to explicitly pass a prop through every level of the tree.
- Context is primarily used when some data needs to be accessible by many components at different nesting levels.
- We will understand how to use the context in the case of Function Components.

USE CASE: Dynamic Theme change



Welcome...



Welcome...

hello-app\src\theme-context.js

```
import React from "react";

export const themes = {
  dark: {
    color: "white",
    background: "black",
    padding: "10px",
  },
  light: {
    color: "black",
    background: "white",
    padding: "10px",
  },
};

const ThemeContext = React.createContext(themes.dark);
```

```
export default ThemeContext;
```

hello-app\src\Welcome.js

```
import React, { useContext } from "react";

import ThemeContext from "./theme-context";

const Welcome = () => {
  const theme = useContext(ThemeContext);
  return (
    <div style={theme}>
      <p>Welcome...</p>
    </div>
  );
};
export default Welcome;
```

hello-app\src\index.js

```
import React, { useState } from "react";
import ReactDOM from "react-dom";
import Welcome from "./Welcome";
import ThemeContext, { themes } from "./theme-context";

function App() {
  const [theme, setTheme] = useState(themes.dark);
  const [themeName, setThemeName] = useState("Light");

  const toggleTheme = () => {
    if (theme === themes.dark) {
      setTheme(themes.light);
      setThemeName("Dark");
    } else {
      setTheme(themes.dark);
      setThemeName("Light");
    }
  };

  return (
    <ThemeContext.Provider value={theme}>
      <button onClick={toggleTheme}>{themeName}</button>
      <Welcome />
    </ThemeContext.Provider>
  );
}
ReactDOM.render(<App />, document.getElementById("root"));
```

Working with useEffect() hook

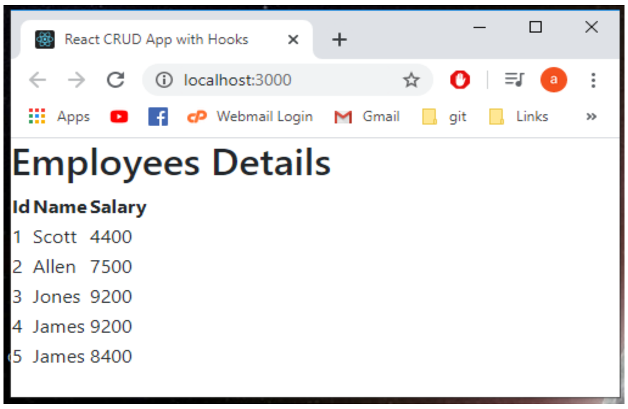
- Many times, we want to run some additional code after React has updated the DOM.
- That code can be getting the Data by Calling a Rest API or Setting up subscriptions or writing the logs after the DOM is ready.
- If we want to write such additional code in Class Components, we have **lifecycle methods** like **componentDidMount, componentDidUpdate** methods.
- What if, if want to write such code in the case of function components.
- We have **useEffect().**

useEffect():

- By using this Hook, you tell React that your component needs to do something after render.
- React will remember the function you passed, and call it later after performing the DOM updates. In this effect, we could also perform data fetching or call some other imperative API.
- useEffect is a function that runs when the component is first rendered, and on every subsequent re-render/update.
- We can think of **useEffect** Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined.

Use Case:

Create Employee Component using which we display the list of Employees.



Employees Details

Id	Name	Salary
1	Scott	4400
2	Allen	7500
3	Jones	9200
4	James	9200
5	James	8400

Example:

hello-app\db.json

```
{
  "employees": [
    {
      "id": 1,
      "name": "Scott",
      "salary": 4000
    },
    {
      "id": 2,
      "name": "Allen",
      "salary": 7500
    }
  ]
}
```

hello-app\src\components\Employee.js

```
import { useEffect, useState } from "react";

function Employee() {
  const [employees, setEmployees] = useState([]);

  const url = "http://localhost:4000/employees/";

  const getEmployees = () => {
    fetch(url)
      .then((res) => res.json())
      .then((result) => {
        setEmployees(result);
        console.log(result)
      });
    console.log("useEffect()");
  };

  useEffect(() => {
    getEmployees();
  }, []);

  return (
    <div>
      <h2>Employees Data</h2>
      <table>
        <thead>
          <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Salary</th>
          </tr>
        </thead>
        <tbody>
          {employees.map((emp) => (
            <tr key={emp.id}>
              <td>{emp.id}</td>
              <td>{emp.name}</td>
              <td>{emp.salary}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}

export default Employee;
```

hello-app\src\App.js

```
import "./App.css";
import Employee from "./components/Employee";

function App() {

  return (
        <Employee></Employee>
    );
}

export default App;
```

hello-app\src\index.js

```
import ReactDOM from "react-dom";
import App from './App';

ReactDOM.render(<App/>, document.getElementById("root"));
```

Run.

hello-app>json-server --watch db.json --port 4000

hello-app>npm start

Note:

- When the DOM is ready, we are sending the API request, getting the employees data and updating our **employees** state variable by calling **setEmployees()** method. But this has a Problem. Remember that when there is any change in the properties data or state data of a component, then that component gets re-rendered.

- Resulting our **useEffect()** function gets called again and again. It will send an API request, get the data and assign it to **employees** state variable. That will make the component gets re-rendered and it will go into infinite loop.

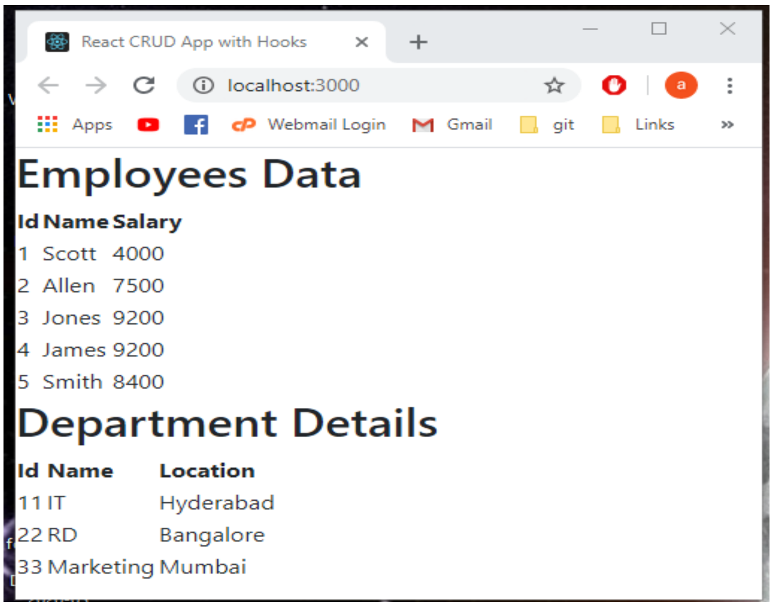
- If you want to run an effect only once (onmount and unmount), you can pass an empty array (**[]**) as a second argument to **useEffect(callback, [])**

- This tells React that your effect doesn't depend on any values from props or state, so it never needs to re-run.

Custom Hook

- As part programming requirement there might be instances where we have to use the same repetitive and redundant state-full logic (such as setting up a subscription and remembering the current value) inside multiple components.

  For Instance.



Employees Data

Id	Name	Salary
1	Scott	4000
2	Allen	7500
3	Jones	9200
4	James	9200
5	Smith	8400

Department Details

Id	Name	Location
11	IT	Hyderabad
22	RD	Bangalore
33	Marketing	Mumbai

We have two components, one is Employee Component and the other one is Department Component.

Where **Employee** Component connects to a Rest API and fetch the employee data when the Component is mounted and display the data.

Similarly **Department** Component connects to a Rest API and fetch the department data when the Component is mounted and display the data.

In order to avoid duplicate code we can go for custom hook.

Custom Hook:

- A custom Hook is a **JavaScript function** that extract component logic into reusable functions.
- It is a convention that we will start the hook name with **use** else we will be violating the rules of Hooks.
- A custom Hook doesn't need to have a specific signature. We can decide what it takes as arguments, and what, to return.
- One hook can be used by multiple components as we have seen here, and every time we use a custom Hook, all state and effects inside of it are fully independent from one component to the other component.

Example:

hello-app\db.json

```json
{
  "employees": [
    {
      "id": 1,
      "name": "Scott",
      "salary": 4000
    },
    {
      "id": 2,
      "name": "Allen",
      "salary": 7500
    }
  ],
  "departments": [
    {
      "id": 11,
      "name": "IT",
      "location": "Hyderabad"
    },
    {
      "id": 22,
      "name": "RD",
      "location": "Bangalore"
    }
  ]
}
```

hello-app\src\hooks\CustomHook.js

```javascript
import { useEffect, useState } from "react";

export function useList(url) {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch(url)
      .then((response) => response.json())
      .then((result) => {
        setData(result);
      });
  }, []);

  return data;
}
```

hello-app\src\components\Employee.js

```
import React from "react";
import { useList } from "../hooks/CustomHook";

const Employee = () => {
  const url = "http://localhost:4000/employees";
  const employees = useList(url);
  return (
    <div>
      <h2>Employee Details</h2>
      <hr></hr>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>NAME</th>
            <th>SALARY</th>
          </tr>
        </thead>
        <tbody>
          {employees.map((employee) => (
            <tr key={employee.id}>
              <td>{employee.id}</td>
              <td>{employee.name}</td>
              <td>{employee.salary}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default Employee;
```

hello-app\src\components\Dept.js

```
import { useList } from "../hooks/CustomHook";

const Dept = () => {
  const url = "http://localhost:4000/departments";
  const depts = useList(url);

  return (
    <div>
      <h2>Department Details</h2>
      <hr></hr>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>NAME</th>
            <th>LOCATION</th>
          </tr>
        </thead>
        <tbody>
          {depts.map((dept) => (
            <tr key={dept.id}>
              <td>{dept.id}</td>
              <td>{dept.name}</td>
              <td>{dept.location}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default Dept;
```

hello-app\src\App.js

```
import Employee from "./components/Employee";
import Dept from "./components/Dept";

function App() {
  return (
    <div>
      <h2>Custom Hook</h2>
      <Employee></Employee>
      <Dept></Dept>
    </div>
  );
}
export default App;
```

hello-app\src\index.js

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));
```

Execution:

```
hello-app>json-server --watch db.json --port 4000
```

```
hello-app>npm start
```