# React

- Context

정수아

# Contents

01 Context란?

02 Context 사용 방법

03 테마 변경

# 01 Context

# Context란?

❖ **Context**

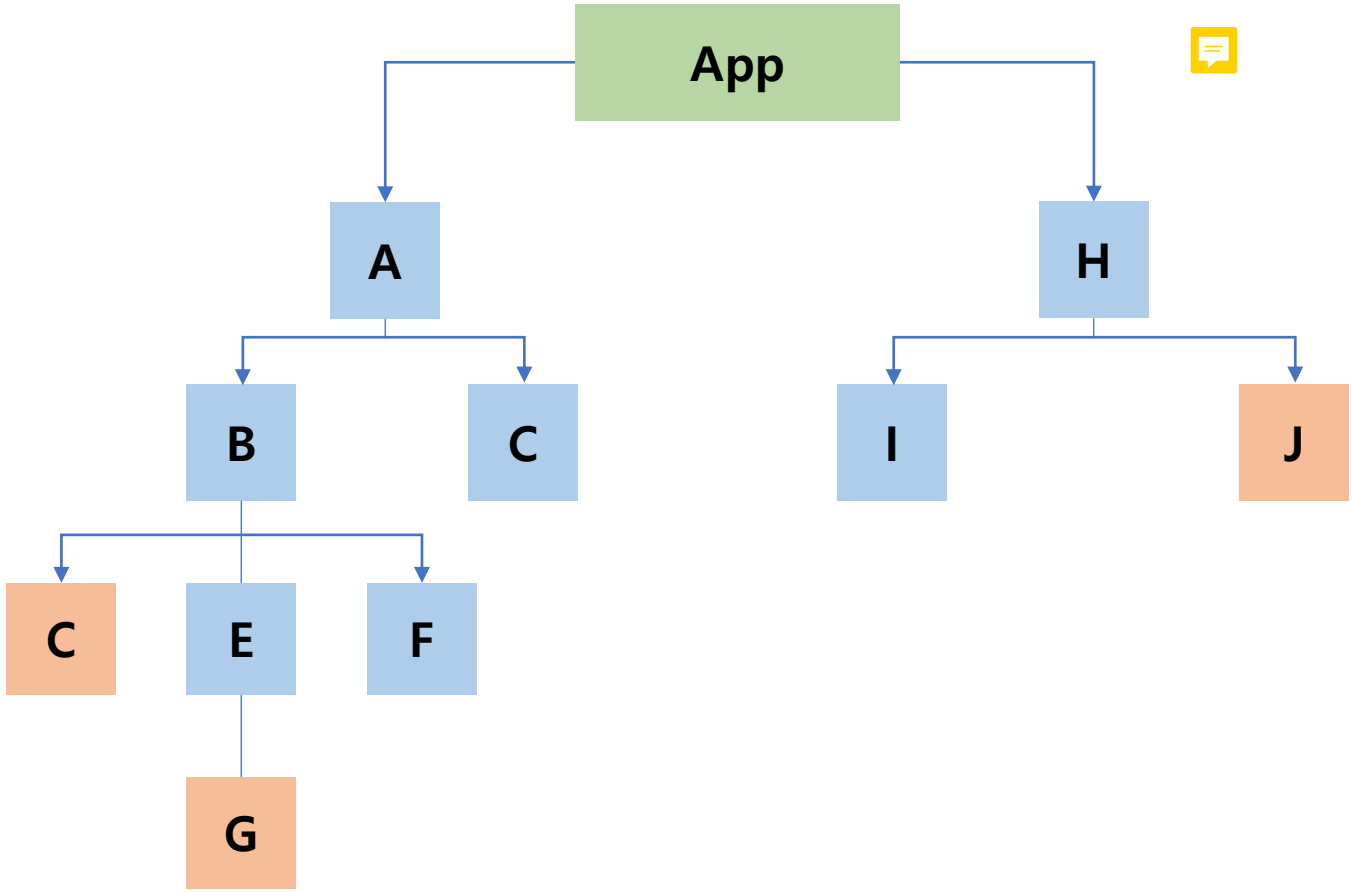
- 어플리케이션에서 전반적으로 사용할 값을 관리
- 예) 사용자의 언어, 로그인 상태, UI 테마 등 환경 설정

- 주의사항
  - Context와 컴포넌트가 연동되면 컴포넌트를 재사용하기 어려움
  - 자주 변경되는 상태인 경우, 사용하지 않는 것이 좋음
    - Context 내부의 값이 변경되면 Context를 사용하는 모든 자식 컴포넌트들이 리렌더링 됨

# 일반적인 전역 상태 관리 흐름

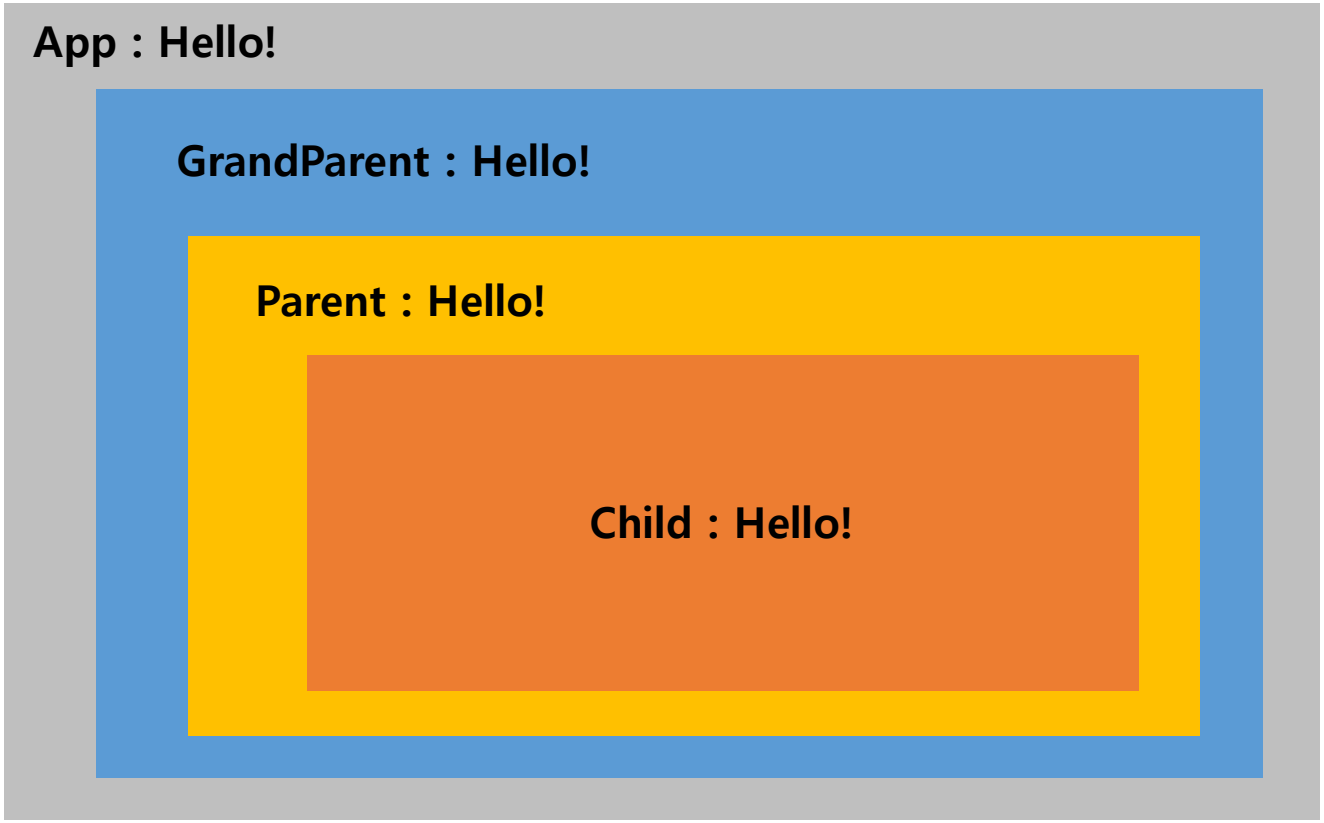
❖ **컴포넌트 여기저기서 필요한 데이터가 있는 경우**

- 주로 최상위 컴포넌트인 App의 state에 넣어서 관리



# Props Drilling

## ❖ Props Drilling



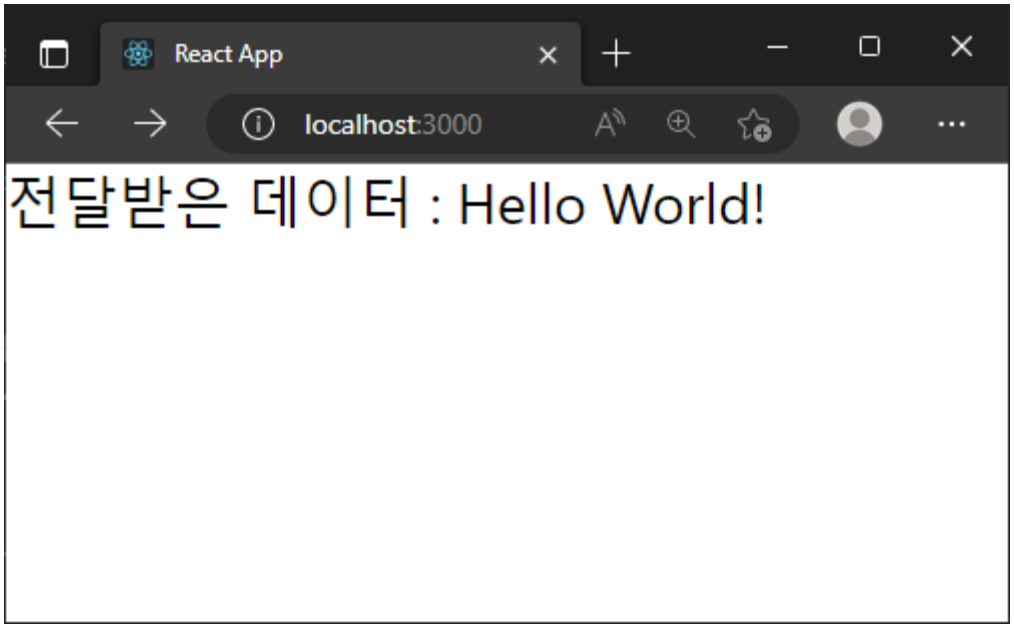
# [실습] Props Drilling

## ❖ App.js

```
function App() {
   return <GrandParent value="Hello World!" />;
}

function GrandParent({ value }) {
   return <Parent value={value} />;
}

function Parent({ value }) {
   return <Child value={value} />;
}

function Child({ value }) {
   return <Message value={value} />;
}

function Message({ value }) {
   return <div>전달받은 데이터 : {value}</div>;
}

export default App;
```

# [실습] Props Drilling

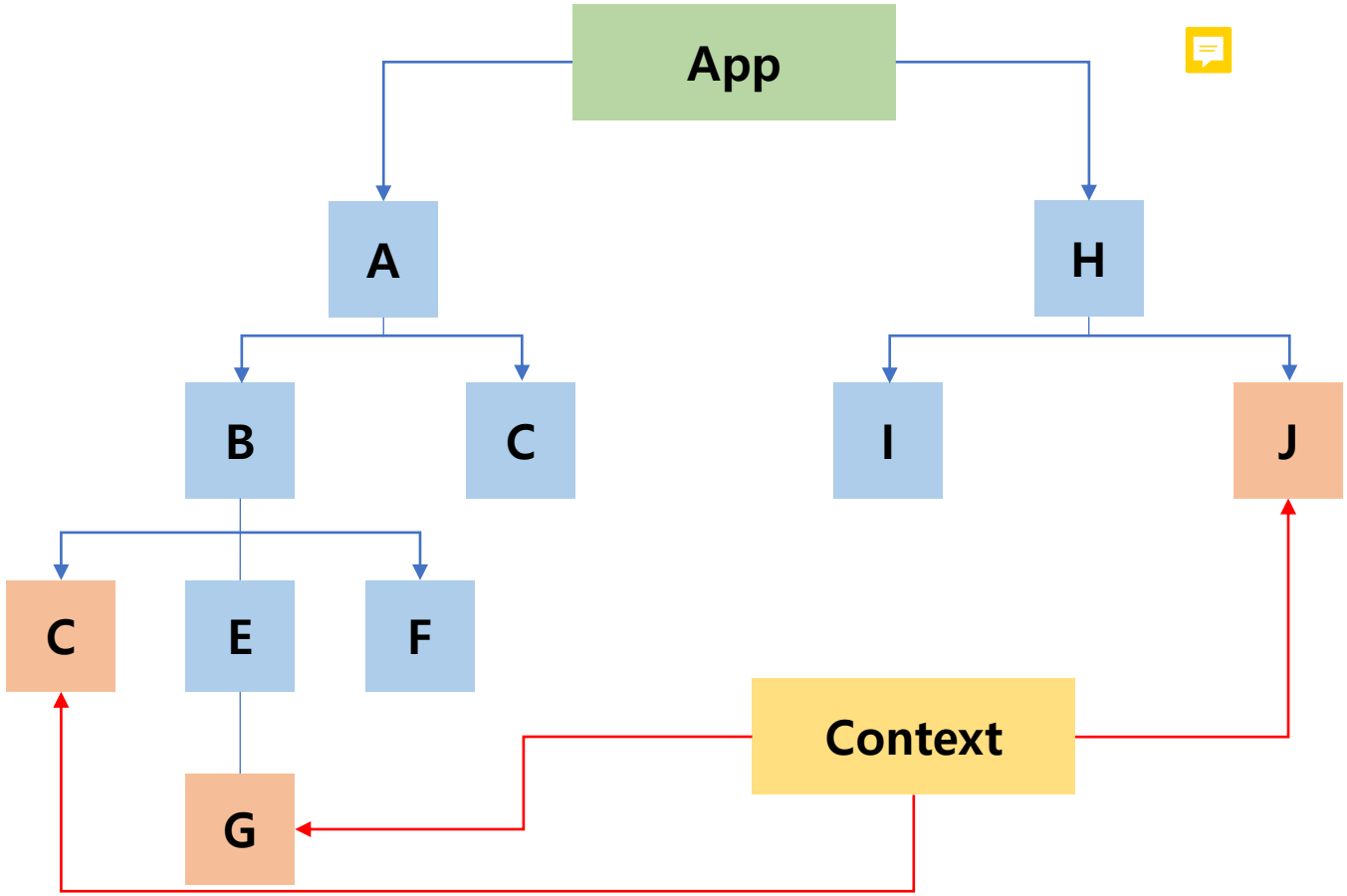
❖ 실행 결과



# Context를 사용한 전역 상태 관리 흐름

❖ **컴포넌트 여기저기서 필요한 데이터가 있는 경우**

- Context를 생성하여 한번에 원하는 값을 전달 받음



# 02 Context 사용 방법

# Context 사용 방법

## ❖ Context 객체 생성

- App.js

```
import { createContext } from 'react';

// MyContext 객체 생성
const MyContext = createContext();
```

## ❖ Context 객체 내 Provider 컴포넌트를 통한 데이터 전달

- Provider 컴포넌트의 하위 컴포넌트는 Context의 데이터에 접근 가능

```
function App() {
  return (
    <MyContext.Provider value="Hello World!">
      <GrandParent value="Hello World!"/>
    </MyContext.Provider>
  );
}
```

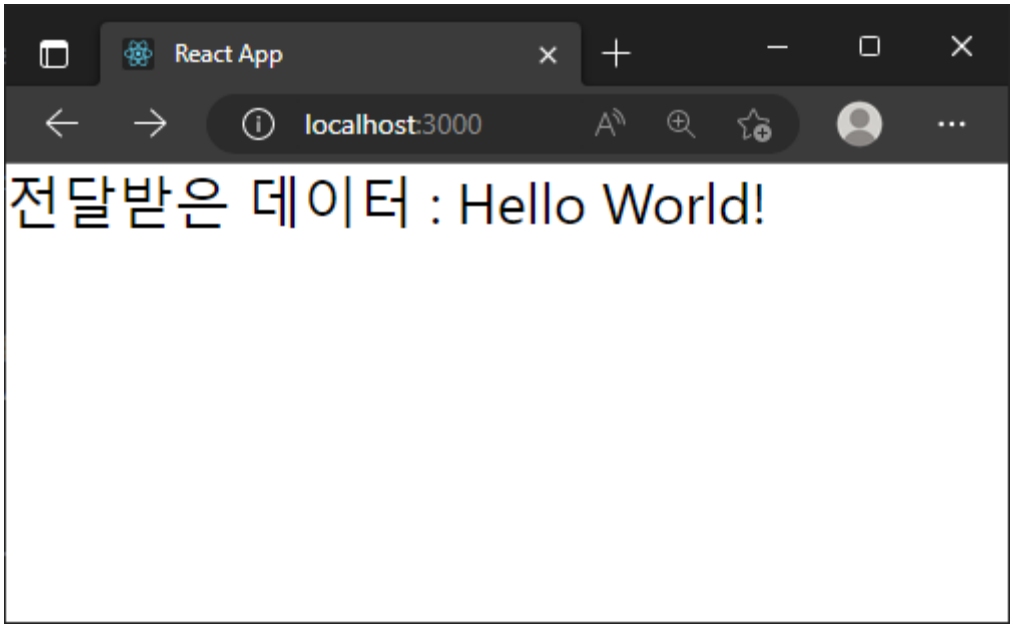
# Context 사용 방법

## ❖ App.js

```
function App() {  // 생략   }

function GrandParent({ value }) {
   return <Parent value={value} />;
}

function Parent({ value }) {
   return <Child value={value} />;
}

function Child({ value }) {
   return <Message value={value} />;
}

function Message({ value }) {
   const value = useContext(MyContext);
   return <div>전달받은 데이터 : {value}</div>;
}

export default App;
```

# Context 사용 방법

❖ 실행 결과



# [실습] Context 파일 생성

## ❖ Context.js

```
import { createContext } from "react";

export const MyContext = createContext("");
```

## ❖ ParentComponent.js

```
import ChildComponent from "./ChildComponent";
import { MyContext } from "./Context";

const ParentComponent = () => {
  return (
    <MyContext.Provider value="안녕하세요">
      <ChildComponent />
    </MyContext.Provider>
  );
};

export default ParentComponent;
```

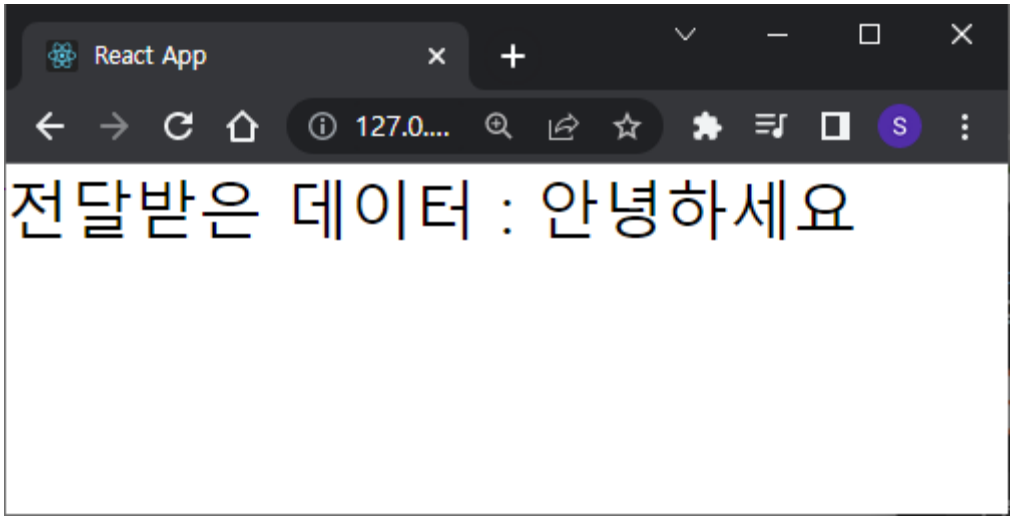
# [실습] Context 파일 생성

## ❖ ChildComponent.js

```
import React, { useContext } from "react";
import { MyContext } from "./Context";

const ChildComponent = () => {
  const value = useContext(MyContext);
  return <div>전달받은 데이터 : {value}</div>;
};

export default ChildComponent;
```

# [실습] Context 파일 생성

❖ 실행 결과



# Context 기본값 지정

❖ **createContext() 함수에 기본값 지정**

- 자식 컴포넌트에서 useContext() 함수를 사용하고 있는데, 부모 컴포넌트가 Provider를 사용하지 않은 경우
- value 값을 지정해주지 않았기 때문에 해당 값이 출력될 자리에 아무것도 나타나지 않음
- 기본값을 설정하면 이러한 문제를 방지할 수 있음

- 기본 값 지정 방법

```
const MyContext = createContext("Default Value");
```

# [실습] Context 기본값 지정

## ❖ Context.js

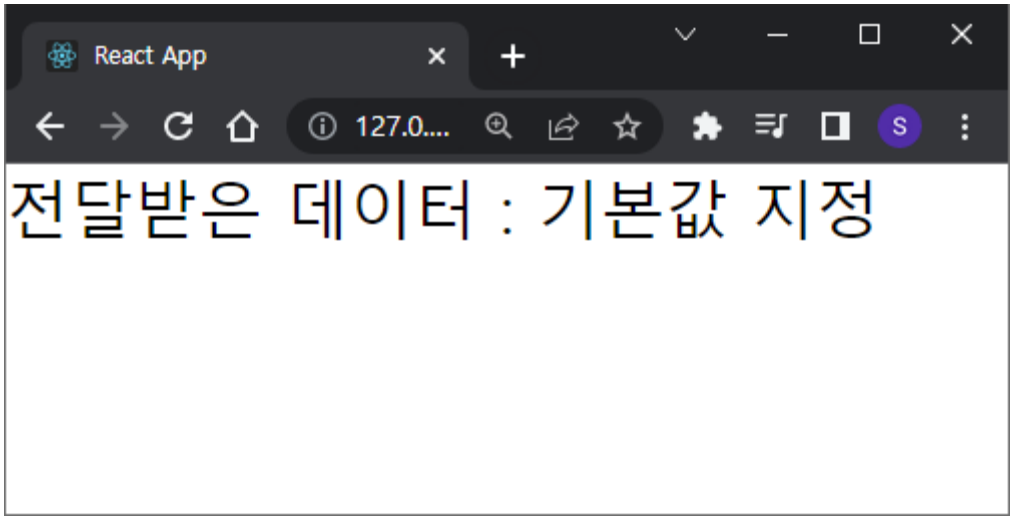
```
import { createContext } from "react";

export const MyContext = createContext("기본값 지정");
```

## ❖ ParentComponent.js

```
import ChildComponent from "./ChildComponent";
import { MyContext } from "./Context";

const ParentComponent = () => {
  return (
    <MyContext.Provider value="안녕하세요">
       <ChildComponent />
    </MyContext.Provider>
  );
};

export default ParentComponent;
```

# [실습] Context 기본값 지정

❖ 실행 결과



# 03

# 테마 변경

# [실습] 어플리케이션 전체 배경색 변경

## ❖ ThemeContext.js

```
import { createContext } from "react";

export const ThemeContext = createContext();
```

## ❖ App.js

```
function App() {
  const [darkMode, setDarkMode] = useState(false);

  return (
    <ThemeContext.Provider value={{ darkMode, setDarkMode }}>
       <HomeComponent />
    </ThemeContext.Provider>
  );
}
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **HomeComponent.js**

```
import "./HomeComponent.scss";

const HomeComponent = () => {
  const data = useContext(ThemeContext);
  console.log(data);

  return (
    <div className="container">
      <HeaderComponent />
      <MainComponent />
      <FooterComponent />
    </div>
  );
};
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **HeaderComponent.js**

```
const HeaderComponent = () => {
    return (
        <div>
            헤더
        </div>
    );
};

export default HeaderComponent;
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **MainComponent.js**

```
const HeaderComponent = () => {
    return (
        <div>
            메인
        </div>
    );
};

export default HeaderComponent;
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **FooterComponent.js**

```
const HeaderComponent = () => {
    return (
        <div>
            푸터
        </div>
    );
};

export default HeaderComponent;
```

# [실습] 어플리케이션 전체 배경색 변경

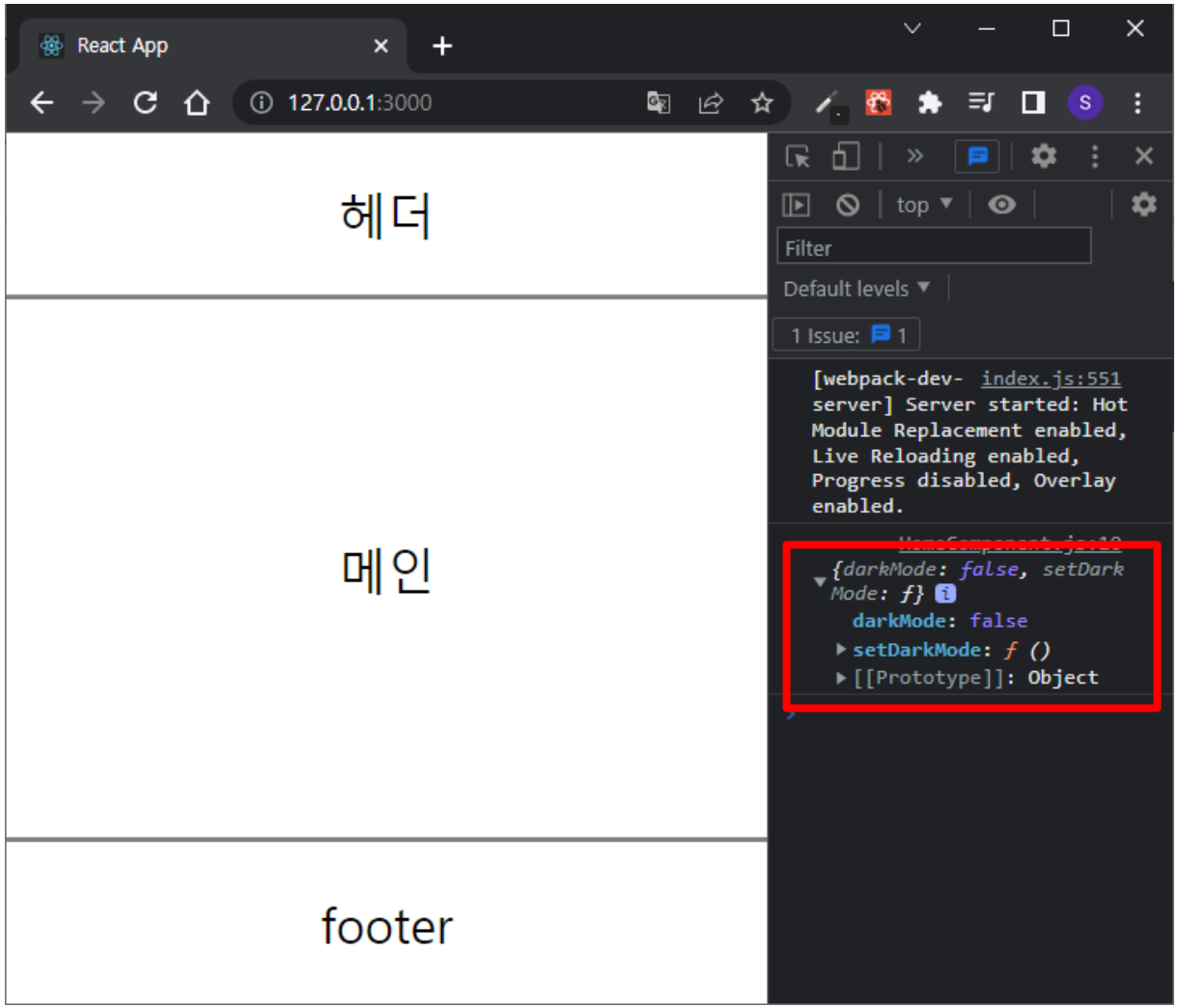
❖ **HomeComponent.scss**

```
* {
    margin: 0;
}

.container {
    width: 100%;
    height: 100vh;
    display: flex;
    flex-direction: column;
}

@mixin flexCenter() {
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 30px;
}
```

```
.header {
    @include flexCenter();
    width: 100%;
    height: 100px;
    border-bottom: 3px solid gray;
}

.main {
    @include flexCenter();
    flex: 1;
}

.footer {
    @include flexCenter();
    width: 100%;
    height: 100px;
    border-top: 3px solid gray;
}
```

# [실습] 어플리케이션 전체 배경색 변경

❖ 실행 결과



# [실습] 어플리케이션 전체 배경색 변경

❖ **HeaderComponent.js**

```
const HeaderComponent = () => {
  const { darkMode, setDarkMode } = useContext(ThemeContext);

  const toggleDarkMode = () => {
     setDarkMode(!darkMode);
  };

  return (
    <div className="header">
      헤더
      {
        darkMode ? (
        <button className="toggleBtn" onClick={toggleDarkMode}>
          😴
        </button> ) : (
        <button className="toggleBtn" onClick={toggleDarkMode}>
          🤩
        </button> )
      }
    </div>
  );
};
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **HeaderComponent.js**

```
const HeaderComponent = () => {
  // 생략

  const theme = {
     backgroundColor: darkMode ? "black" : "white",
     color: darkMode ? "white" : "black",
  };

  return (
    <div className="header" style={theme}>
      헤더

      // 생략

    </div>
  );
};
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **HomeComponent.scss**

```
// 생략

.toggleBtn {
   background-color: transparent;
   border: none;
   font-size: 30px;
}
```

# [실습] 어플리케이션 전체 배경색 변경

❖ **MainComponent.js**

```
const MainComponent = () => {
  const { darkMode } = useContext(ThemeContext);

  const theme = {
    backgroundColor: darkMode ? "black" : "white",
    color: darkMode ? "white" : "black",
  };

  return (
    <div className="main" style={theme}>
      메인
    </div>
  );
};
```

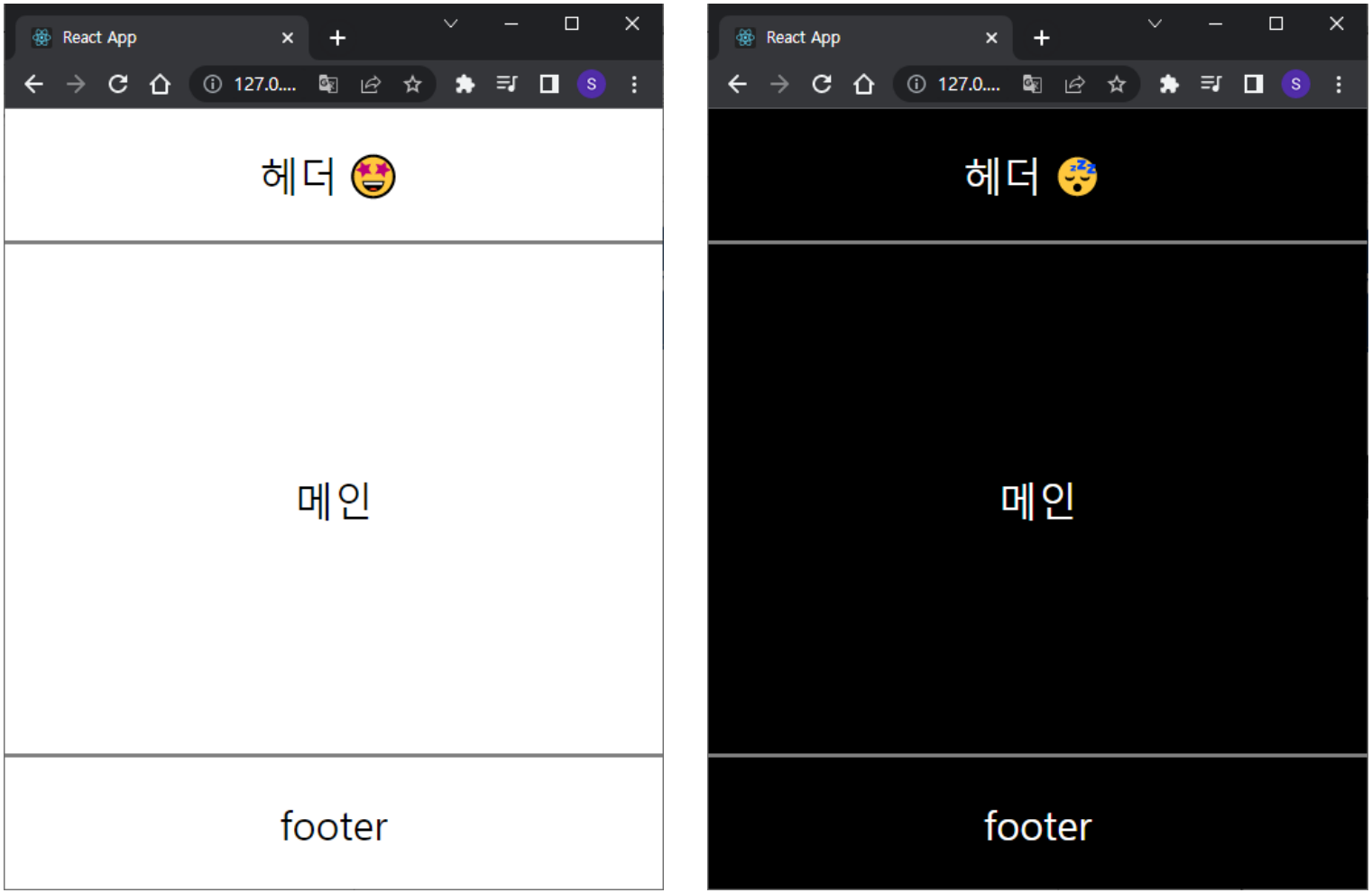
# [실습] 어플리케이션 전체 배경색 변경

❖ **FooterComponent.js**

```
const FooterComponent = () => {
  const { darkMode } = useContext(ThemeContext);

  const theme = {
    backgroundColor: darkMode ? "black" : "white",
    color: darkMode ? "white" : "black",
  };

  return (
    <div className="footer" style={theme}>
      footer
    </div>
  );
};
```

# [실습] 어플리케이션 전체 배경색 변경

❖ 실행 결과




THANK YOU