

# day49-rn-Context API

태그	
날짜	@2022년 12월 8일

이전엔 컴포넌트가 아니라 전역적으로 상태를 관리하려면 어떻게 해야할까.

Context API는 데이터를 전역적으로 관리하고 사용할 수 있도록 하는 기능이다.

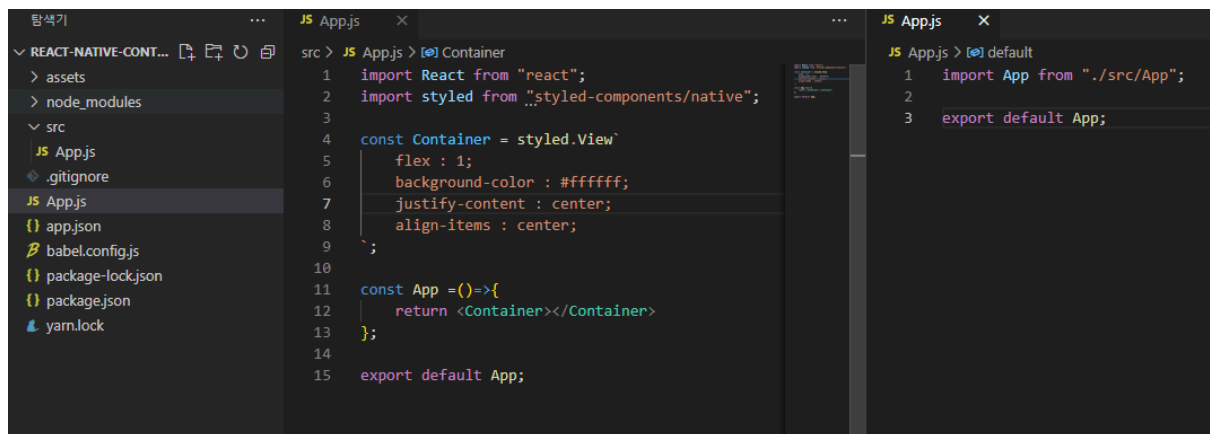
대표적인 예시로는 스타일드 컴포넌트가 Context API를 이용해 theme을 애플리케이션 전체에서 사용할 수 있도록 구현된 것을 들 수 있다.

이번엔 Context API를 이용하여 상태를 전역적으로 관리하는 바업에 대해 알아보자

프로젝트를 생성하자.

- expo init react-native-context
- npm install styled-components

이전과 같이 사전 작업을 하도록 하자.



전역 상태 관리

일반적인 리액트 네이티브 애플리케이션의 경우 데이터는 부모 컴포넌트에서 자식 컴포넌트로 전달된다. 만약 데이터를 사용하는 컴포넌트가 많다면, 최상위 컴포넌트인 App 컴포넌트에서 상태를 관리하여 하위 컴포넌트 어디서 필요로 하든 전달할 수 있게 해야한다.

예를 들어, 어떤 데이터를 App 컴포넌트에서 관리하고 하위 컴포넌트 중 몇 개의 컴포넌트에서 데이터를 사용한다면, App 컴포넌트로부터 데이터를 필요로 하는 컴포넌트까지 props를 통해 값을 전달해서 사용할 수 있다.

그림 p.231

필기 & 그림 p.232

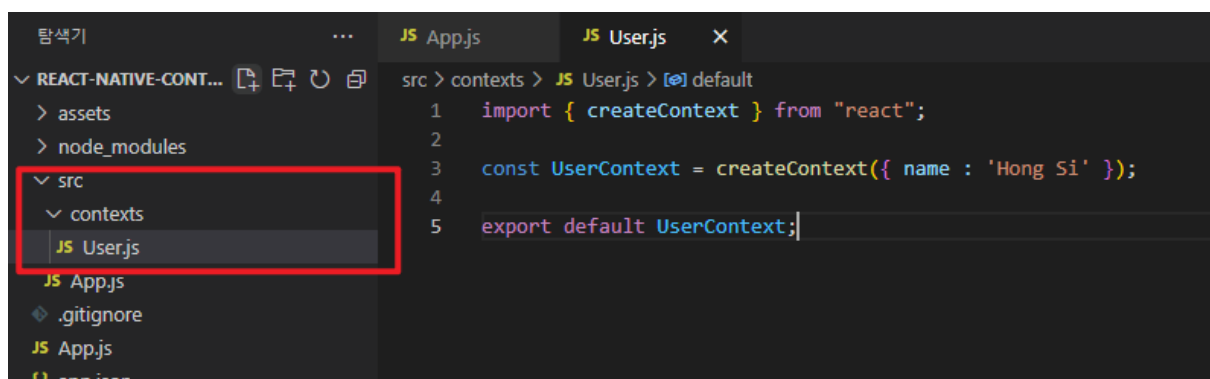
## Context API

이번에는 Context를 생성하는 createContext 함수는 파라미터에 생성되는 Context의 기본 값을 지정할 수 있다.

- 기본형태

```
const Context = createContext(defaultValue);
```

src폴더 밑에 contexts 폴더를 생성하고 폴더 안에 User.js 파일을 만들어서 아래처럼 작성하자.

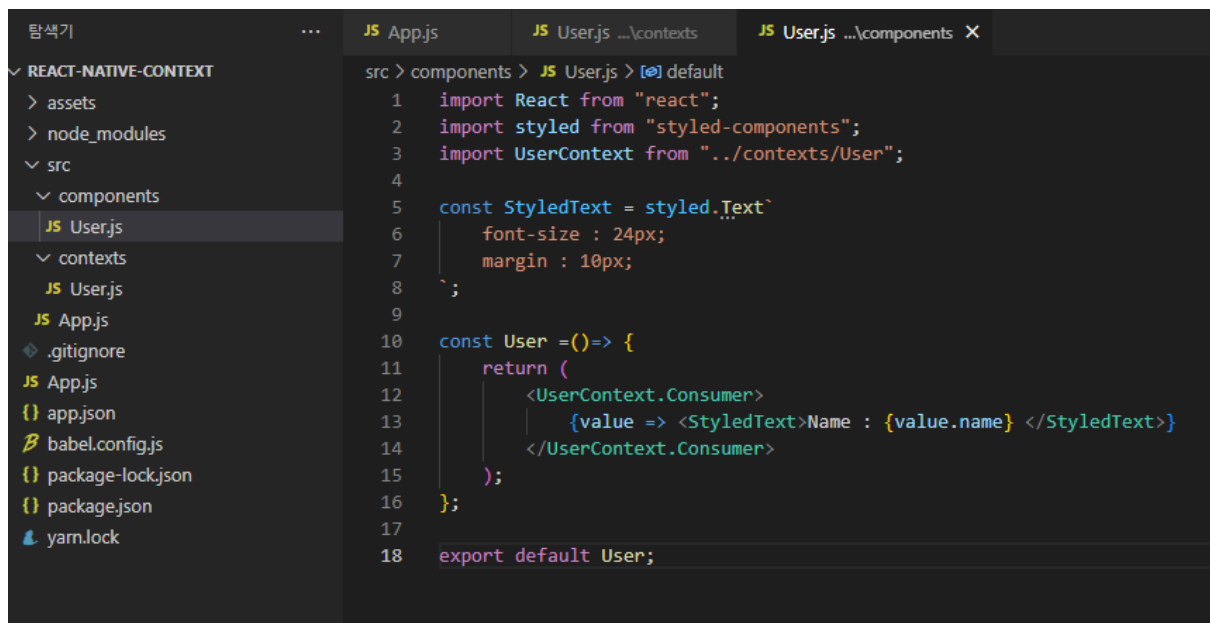


## consumer

생성된 Context 오브젝트는 입력된 시본값 외에도 Consumer 컴포넌트와 Provider 컴포넌트를 갖고 있다. 이번엔 Context의 내용을 읽고 사용하는 Consumer 컴포넌트의 사용 방법에 대해 알아보자

Consumer 컴포넌트는 상위 컴포넌트 중 가장 가까운 곳에 있는 Provider 컴포넌트가 전달하는 데이터를 이용한다. 만약 상위 컴포넌트 중 Provider 컴포넌트가 없다면 createContext 함수의 파라미터로 전달된 기본값을 사용한다.

src폴더 안에 components 폴더를 생성하고 Consumer 컴포넌트를 이용해서 createContext함수의 파라미터로 전달된 기본값을 출력하는 User 컴포넌트를 작성해 보자.



```
src > components > JS User.js > [0] default
1  import React from "react";
2  import styled from "styled-components";
3  import UserContext from "../contexts/User";
4
5  const StyledText = styled.Text`
6    font-size : 24px;
7    margin : 10px;
8  `;
9
10 const User = () => {
11   return (
12     <UserContext.Consumer>
13       {value => <StyledText>Name : {value.name} </StyledText>}
14     </UserContext.Consumer>
15   );
16 };
17
18 export default User;
```

Consumer 컴포넌트의 자식은 반드시 리액트 컴포넌트를 반환하는 함수여야 하고, 이 함수는 Context의 현재값을 파라미터로 전달받아 데이터를 사용할 수 있다. 이제 작성된 User 컴포넌트를 App 컴포넌트에서 사용해보자

```
src > JS App.js > App
1  import React from "react";
2  import styled from "styled-components/native";
3  import User from "../components/User";
4
5  const Container = styled.View`
6    flex : 1;
7    background-color : #ffffff;
8    justify-content : center;
9    align-items : center;
10 `;
11
12 const App = ()=>{
13   return (
14     <Container>
15       <User />
16     </Container>
17   )
18 };
19
20 export default App;
```



Name : Hong Si



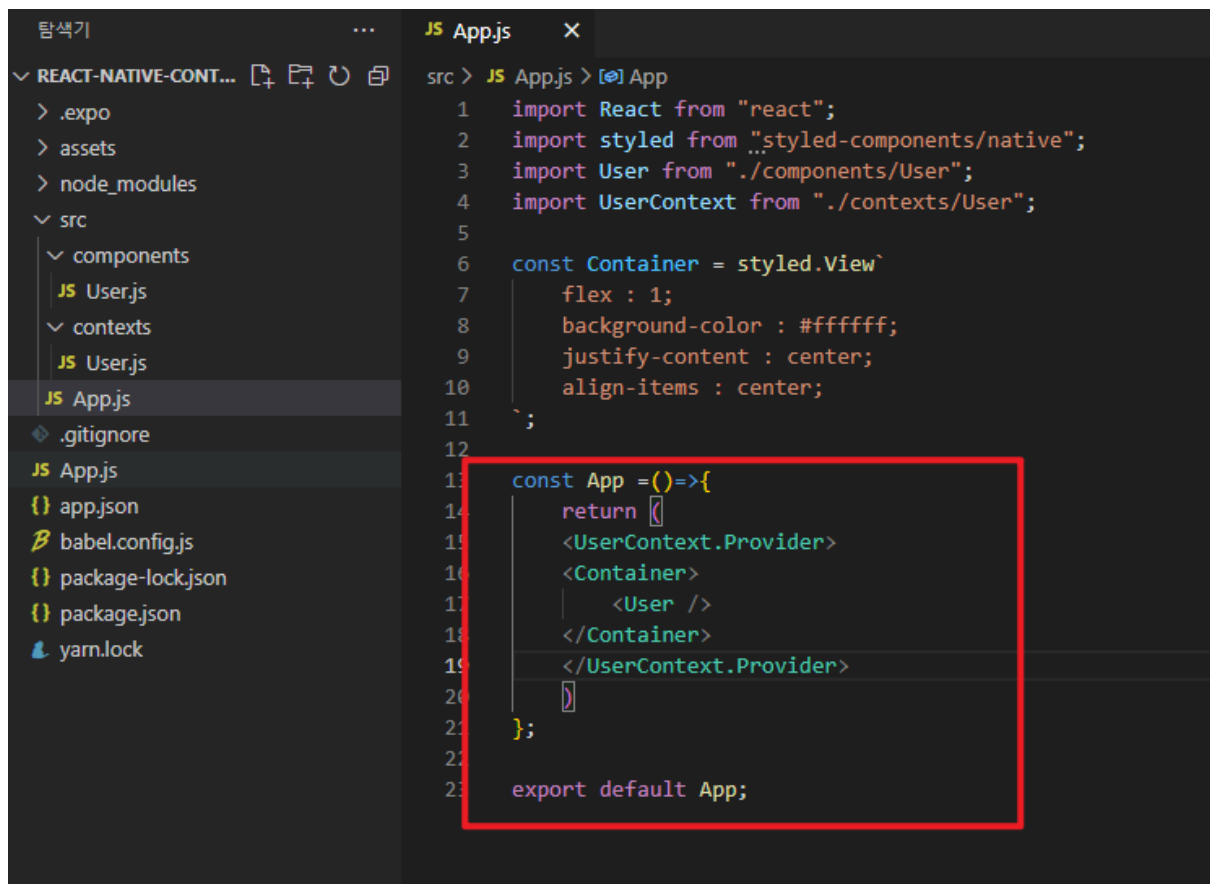
입력한 이름이 잘 나타나는 것을 확인할 수 있다.

## Provider

Context에 있는 Provider 컴포넌트는 하위 컴포넌트에 Context의 변화를 알리는 역할을 한다. Provider 컴포넌트는 value를 받아서 모든 하위 컴포넌트에 전달하고, 하위 컴포넌트는 Provider 컴포넌트의 value가 변경될 때마다 다시 렌더링 된다.

우리가 사용한 스타일드 컴포넌트를 생각하면 쉽게 이해할 수 있다. ThemeProvider 컴포넌트를 최상위 컴포넌트로 이용하여 theme에 우리가 정의한 색을 지정하면 모든 하위 컴포넌트에서 theme을 사용할 수 있었고 theme으로 지정한 색을 변경하면 모든 컴포넌트에 변경된 색이 반영되었다.

이제 App 컴포넌트에서 Provider 컴포넌트를 사용해보자.



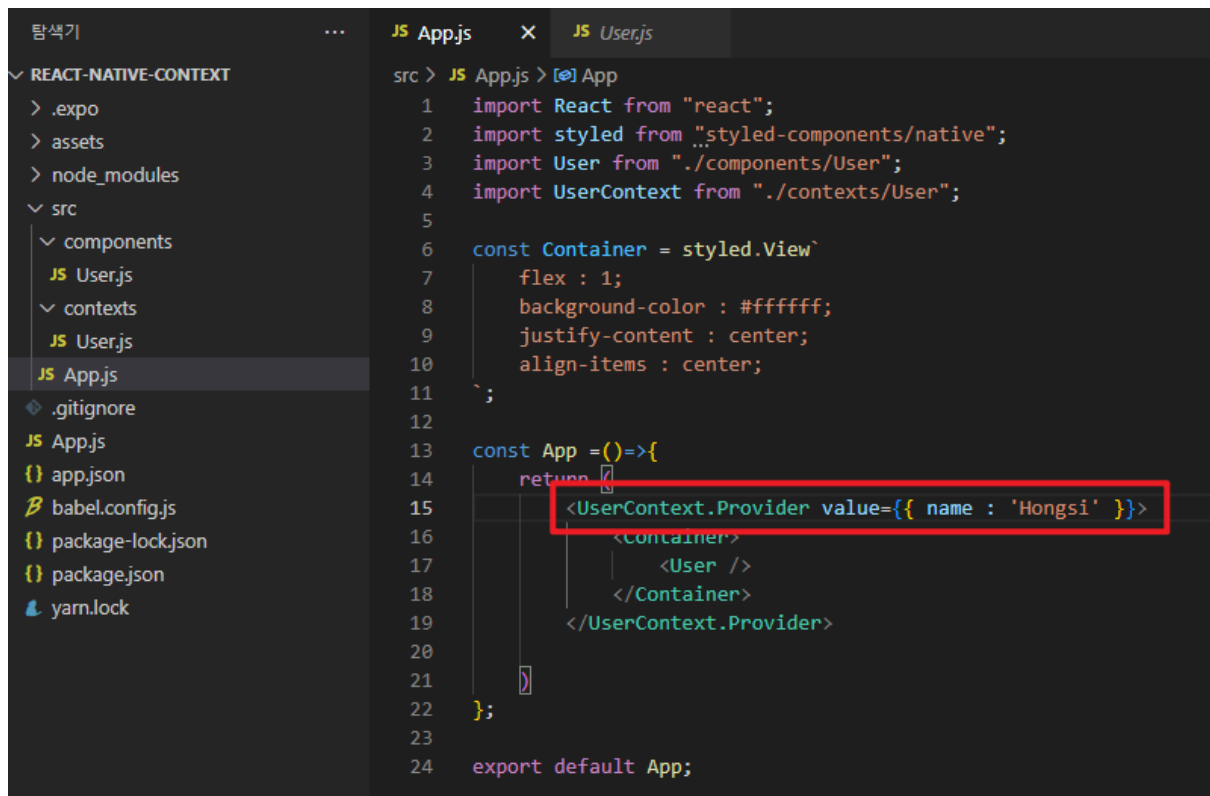
```
src > JS App.js > App
1  import React from "react";
2  import styled from "styled-components/native";
3  import User from "../components/User";
4  import UserContext from "../contexts/User";
5
6  const Container = styled.View`
7    flex : 1;
8    background-color : #ffffff;
9    justify-content : center;
10   align-items : center;
11 `;
12
13 const App = () => {
14   return (
15     <UserContext.Provider>
16       <Container>
17         <User />
18       </Container>
19     </UserContext.Provider>
20   );
21 };
22
23 export default App;
```

결과를 확인해보면 아래 같은 오류 메시지를 볼 수 있다.



TypeError : undefined is not an object (evaluating 'value.name')

App 컴포넌트를 Provider컴포넌트로 감쌌기 때문에 User 컴포넌트에서 사용된 Consumer 컴포넌트는 더 이상 Context의 기본값을 사용하지 않고 상위 컴포넌트인 Provider 컴포넌트가 전달하는 데이터를 사용하도록 변경되었다. 하지만 Provider 컴포넌트에서는 어떤 값도 전달되지 않고 Consumer 컴포넌트의 자식으로 지정된 함수의 파라미터로 undefined가 전달되면서 오류 메시지가 나타나는 것이다. 이 문제는 Provider 컴포넌트의 value에 값을 전달하여 해결할 수 있다.



```
src > JS App.js > App
1  import React from "react";
2  import styled from "styled-components/native";
3  import User from "../components/User";
4  import UserContext from "../contexts/User";
5
6  const Container = styled.View`
7    flex : 1;
8    background-color : #ffffff;
9    justify-content : center;
10   align-items : center;
11 `;
12
13 const App = ()=>{
14   return (
15     <UserContext.Provider value={{ name : 'Hongsi' }}>
16       <Container>
17         <User />
18       </Container>
19     </UserContext.Provider>
20   );
21 }
22
23
24 export default App;
```



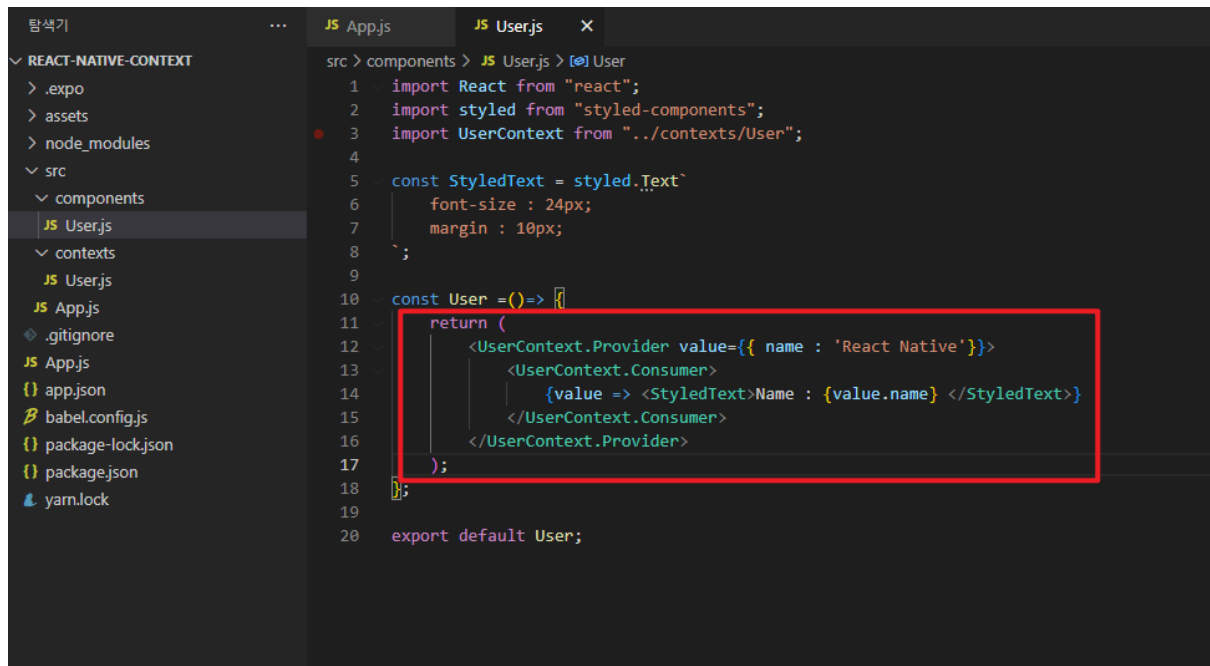


Name : Hongsi



Context를 생성할 때 전달한 기본값이 아닌 Provider컴포넌트의 value로 지정된 값이 잘 나타난다.

Provider 컴포넌트로부터 value를 전달받는 하위 컴포넌트의 수에는 제한이 없다. 하지만 Consumer 컴포넌트는 가장 가까운 Provider 컴포넌트에서 값을 받으므로, 자식 컴포넌트 중 Provider 컴포넌트가 있다면 그 이후의 자식 컴포넌트는 중간에 있는 Provider 컴포넌트가 전달하는 값을 사용한다.



```
src > components > JS User.js > [0] User
1  import React from "react";
2  import styled from "styled-components";
3  import useContext from "../contexts/User";
4
5  const StyledText = styled.Text`
6    font-size : 24px;
7    margin : 10px;
8  `;
9
10 const User = () => {
11   return (
12     <useContext.Provider value={{ name : 'React Native'}}>
13       <useContext.Consumer>
14         {value => <StyledText>Name : {value.name} </StyledText>}
15       </useContext.Consumer>
16     </useContext.Provider>
17   );
18 };
19
20 export default User;
```



Name : React Native

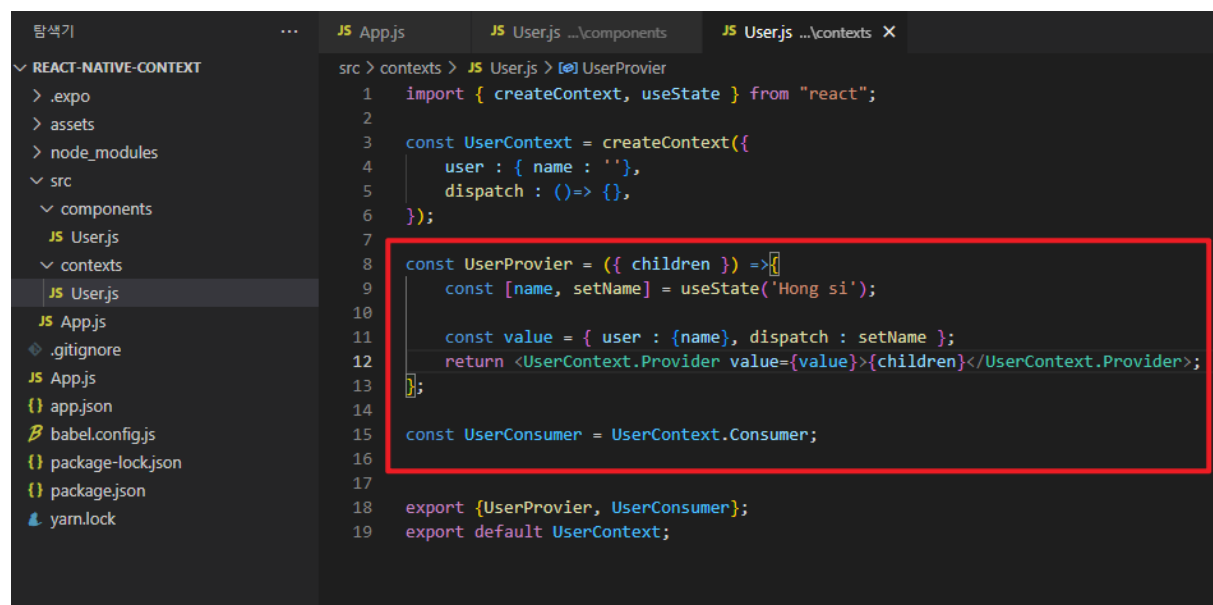


User 컴포넌트에서 사용된 Provider 컴포넌트의 데이터로 나타난다.

Provider 컴포넌트를 사용할 때 반드시 value를 지정해야 한다는 점과 Consumer 컴포넌트는 가장 가까운 Provider 컴포넌트가 전달하는 값을 이용한다는 점을 기억하자.

## Context 수정하기

지금까지 Provider 컴포넌트를 이용해 데이터를 전달하고 Consumer 컴포넌트를 이용해 내용을 출력하는 방법에 대해 알아왔다. 이번에는 Context의 값을 수정해서 Context를 사용하는 컴포넌트에 변경된 내용을 반영하는 방법을 알아보자.



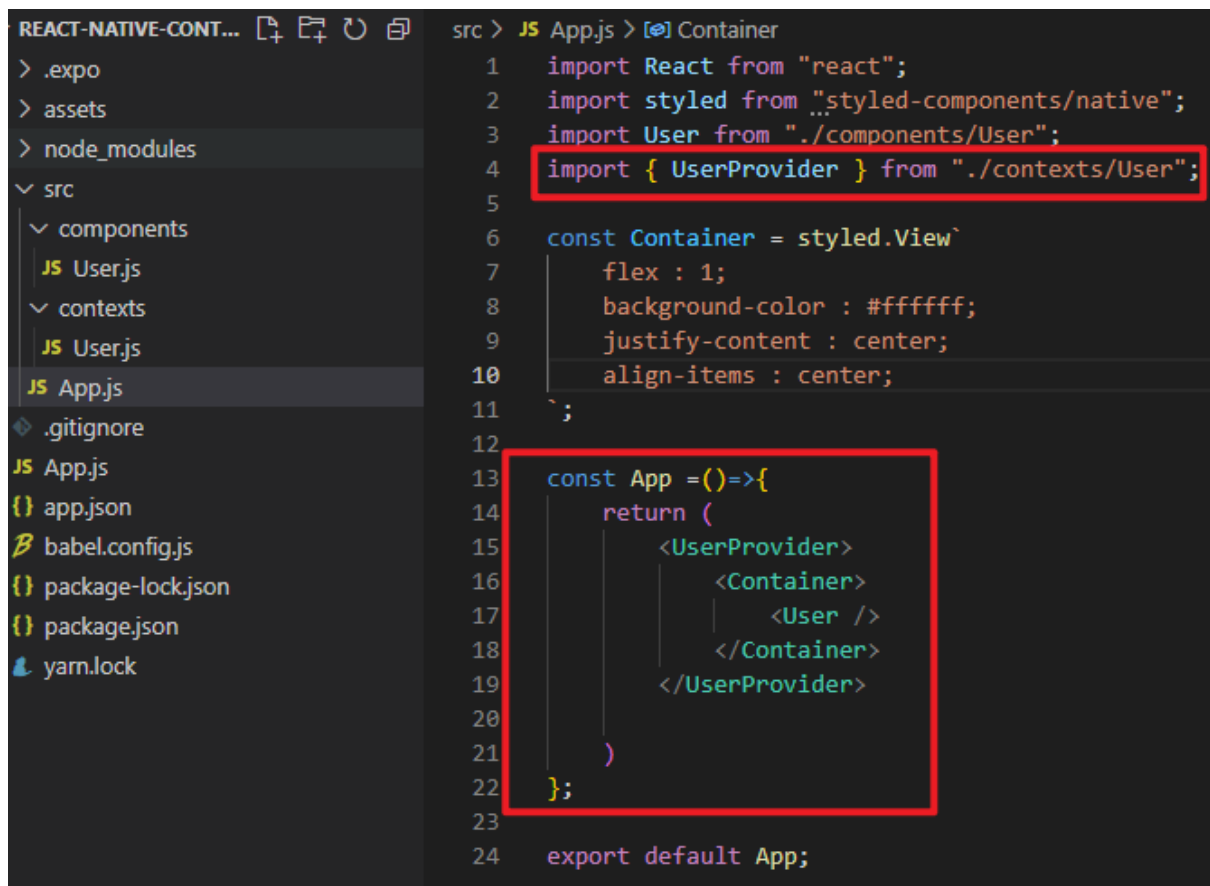
```
src > contexts > JS User.js > [0] UserProvier
1  import { createContext, useState } from "react";
2
3  const UserContext = createContext({
4    user : { name : '' },
5    dispatch : ()=> {},
6  });
7
8  const UserProvider = ({ children }) => {
9    const [name, setName] = useState('Hong si');
10
11    const value = { user : {name}, dispatch : setName };
12    return <UserContext.Provider value={value}>{children}</UserContext.Provider>;
13  };
14
15  const UserConsumer = UserContext.Consumer;
16
17
18  export {UserProvier, UserConsumer};
19  export default UserContext;
```

Provider 컴포넌트의 value에 전역적으로 관리할 상태 변수와 상태를 변경하는 함수를 함께 전달하는 UserProvider 컴포넌트를 생성했다. UserProvider 컴포넌트는 기존의 Provider 컴포넌트와 사용법이 동일하지만 하위에 있는 Consumer 컴포넌트의 자식 함수의 파라미터로 데이터 뿐만 아니라 데이터를 변경할 수 있는 함수도 함께 전달한다.

createContext의 기본값도 UserProvider 컴포넌트의 value로 전달하는 형태와 동일한 형태를 갖도록 작성했다. Consumer 컴포넌트의 상위 컴포넌트에 Provider컴포넌트가 없더라도 동작에 문제가 생기지 않도록 형태를 동일하게 맞추는 것이 좋다.

마지막으로 Consumer 컴포넌트와 동일한 UserConsumer 컴포넌트를 생성했다.

수정이 완료되면 App컴포넌트에서 Provider 컴포넌트 대신 UserProvider 컴포넌트를 사용하도록 수정하자.

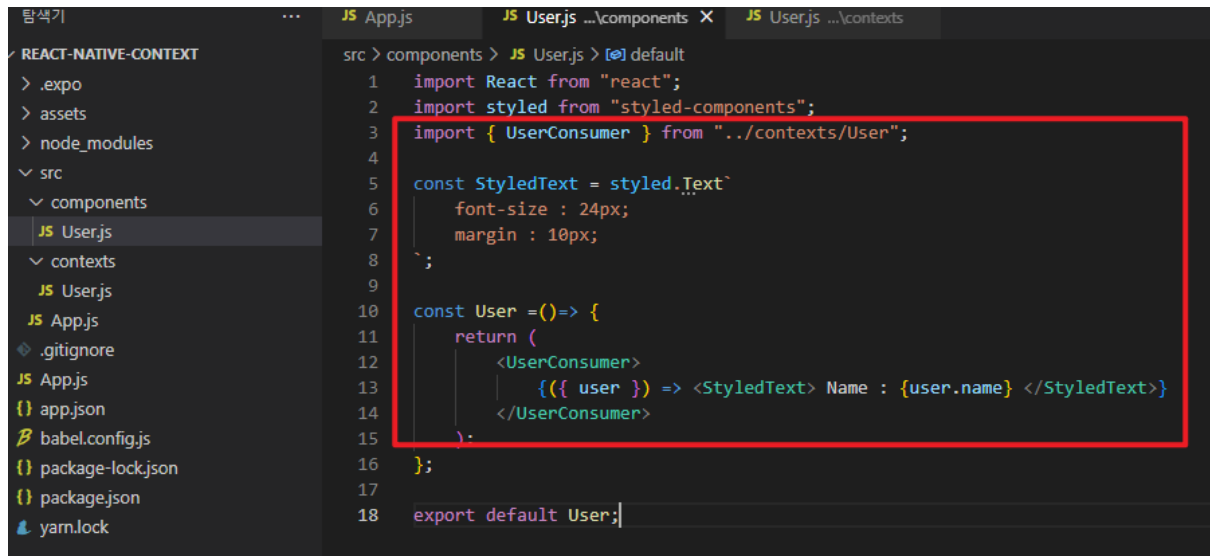


The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the project structure with files like .expo, assets, node\_modules, src, components, contexts, App.js, .gitignore, app.json, babel.config.js, package-lock.json, package.json, and yarn.lock. The code editor shows the content of App.js, which is a React Native application. The code is as follows:

```
src > JS App.js > [e] Container
1  import React from "react";
2  import styled from "styled-components/native";
3  import User from "../components/User";
4  import { UserProvider } from "../contexts/User";
5
6  const Container = styled.View`
7    flex : 1;
8    background-color : #ffffff;
9    justify-content : center;
10   align-items : center;
11 `;
12
13 const App = ()=>{
14   return (
15     <UserProvider>
16       <Container>
17         <User />
18       </Container>
19     </UserProvider>
20   )
21 }
22
23
24 export default App;
```

The code is written in a dark-themed editor. The file explorer on the left shows the project structure. The code editor on the right shows the content of App.js. The code is a React Native application. The code is as follows:

UserPrvider 컴포넌트는 내부에서 Provider 컴포넌트를 이용해 value를 전달하므로 따로 value를 전달하지 않아도 괜찮다. 이제 User 컴포넌트에서 전달된 새로운 Value를 사용하도록 수정하자.



```
src > components > JS User.js > default
1  import React from "react";
2  import styled from "styled-components";
3  import { UserConsumer } from "../contexts/User";
4
5  const StyledText = styled.Text`
6    font-size : 24px;
7    margin : 10px;
8  `;
9
10 const User = () => {
11   return (
12     <UserConsumer>
13       {({ user }) => <StyledText> Name : {user.name} </StyledText>}
14     </UserConsumer>
15   );
16 };
17
18 export default User;
```

Consumer 컴포넌트 대신 UserConsumer 컴포넌트를 사용했다. Provider 컴포넌트에서 전달되는 value의 형태가 변경되었기 때문에 함수 형태도 전달되는 값에 맞게 수정된다.

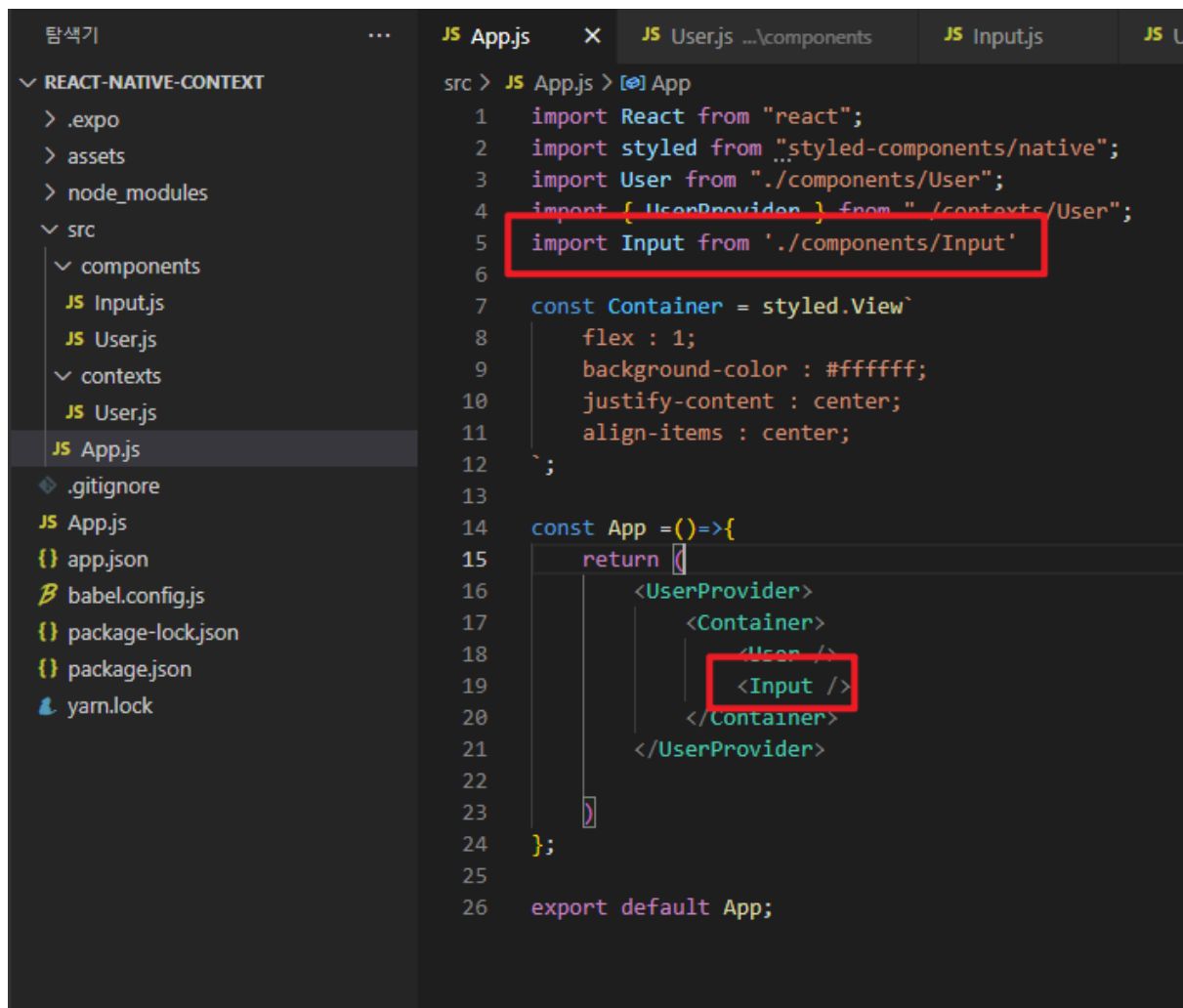


이번엔 UserProvider 컴포넌트의 value로 전달되는 세터 함수를 이용해 입력되는 값으로 Context의 값을 변경하는 Input 컴포넌트를 만들어보자.

```
src > components > JS Input.js > [0] Input
1  import React, {useState} from "react";
2  import styled from "styled-components/native";
3  import { UserConsumer } from "../contexts/User";
4
5  const StyledText = styled.TextInput`
6    border : 1px solid #606060;
7    width : 250px;
8    padding : 10px 15px;
9    margin: 10px;
10   font-size : 24px;
11 `;
12
13 const Input = ()=>{
14   const [name, setName] = useState('');
15
16   return(
17     <UserConsumer>
18       {({ dispatch })=>{
19         return(
20           <StyledInput
21             value={name}
22             onChangeText={text=>setName(text)}
23             onSubmitEditing={()=>{
24               dispatch(name);
25               setName('');
26             }}
27             placeholder="Enter a name..."
28             autoCapitalize="none"
29           />
30         );
31       }}
32     </UserConsumer>
33   );
34 };
```

useState 함수를 이용해서 name 상태 변수를 생성하고 TextInput 컴포넌트에 값이 변경될 때마다 name에 반영되도록 작성했다. UserConsumer 컴포넌트의 자식 함수에 전달되는 value에는 Context의 값을 변경할 수 있는 dispatch 함수가 함께 전달 된다. dispatch 함수를 이용해 키보드의 확인 버튼을 누르면 TextInput 컴포넌트에 입력된 값으로 Context의 값을 변경하도록 작성했다.







입력한 값으로 바뀌는 것을 확인 할 수 있다.

