

day48-rn-useState

☰ 태그	
📅 날짜	@2022년 12월 7일

우선 환경을 만들자

프로젝트 생성

- expo init react-native-hooks

함께 사용할 스타일드 컴포넌트를 설치하자

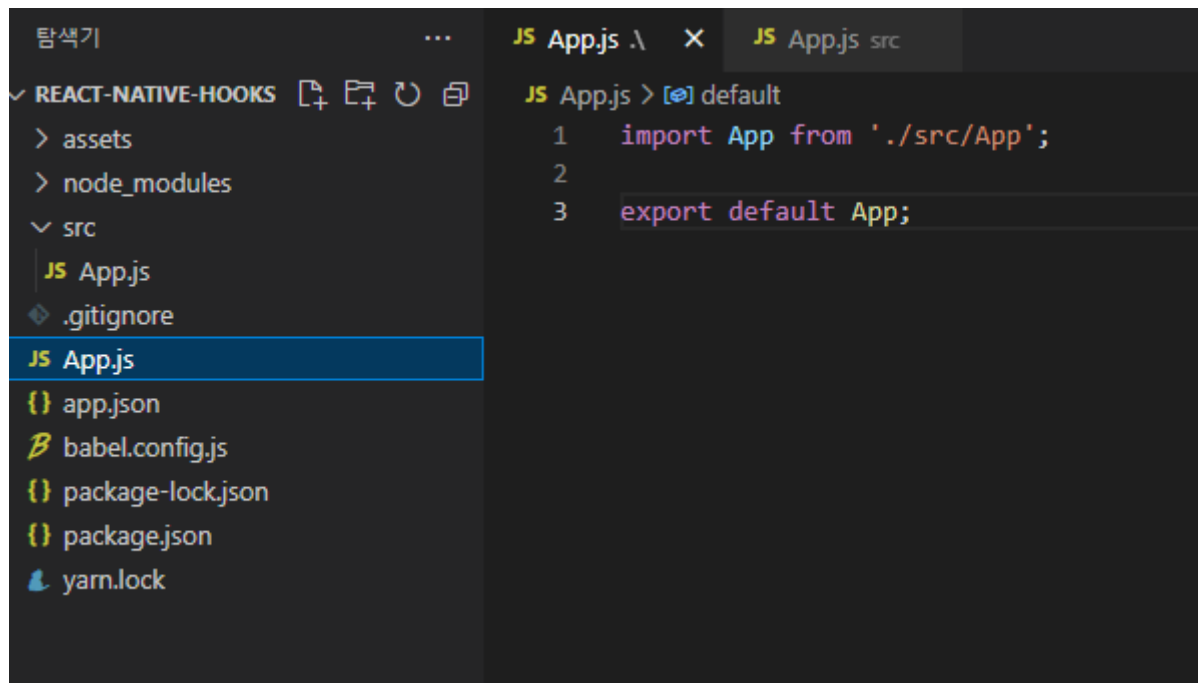
- npm install styled-components

만약 에러가 난다면 뒤에 —force를 붙여서 설치하자

모든 준비가 되었다면 src폴더를 생성하고 App컴포넌트를 아래처럼 작성하자.

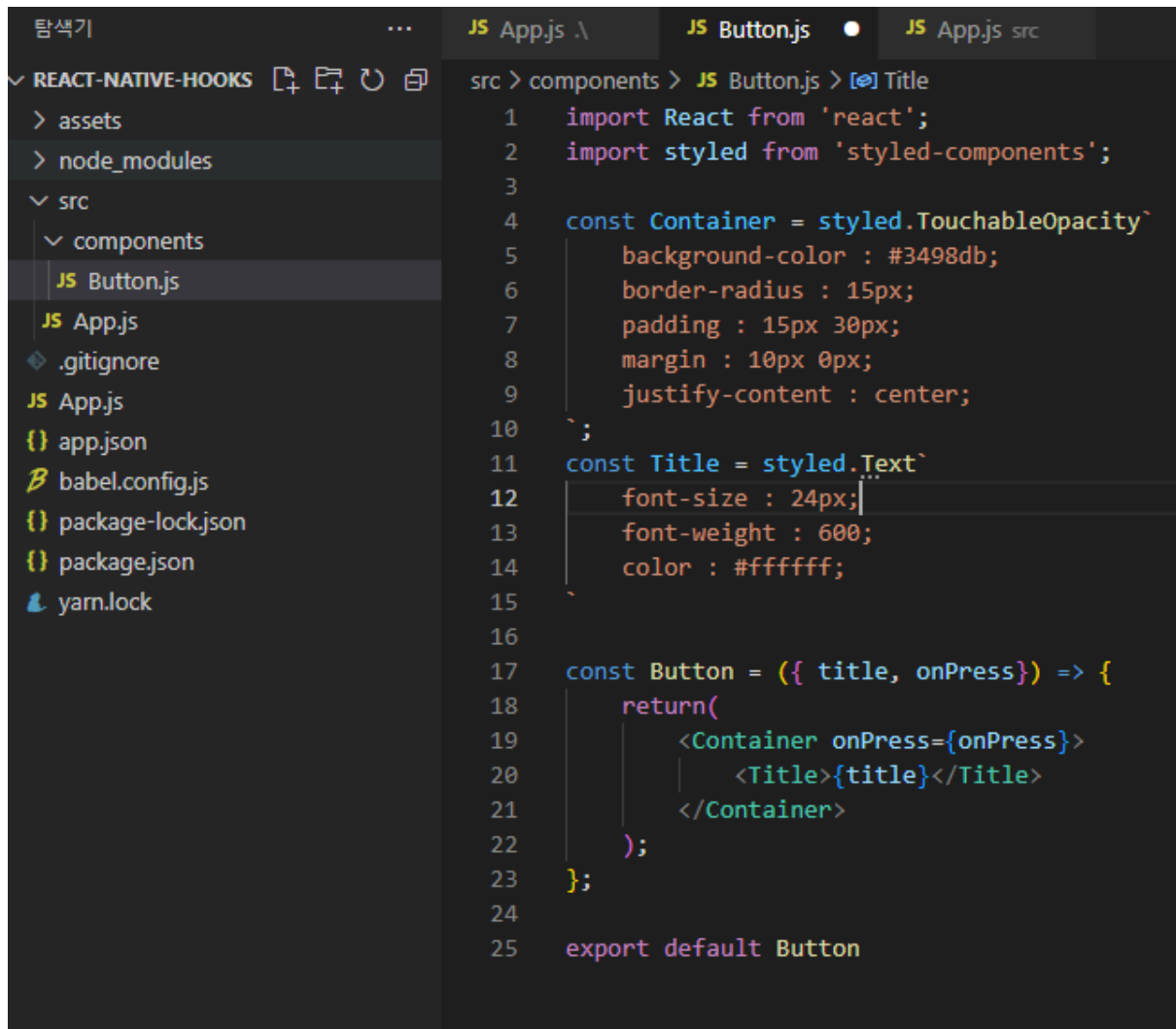
```
src > JS App.js > ...
1  import React from 'react';
2  import styled from 'styled-components';
3
4  const Container = styled.View`
5    flex : 1;
6    background-color : #fff;
7    justify-content : center;
8    align-items : center;
9  `;
10
11
12  const App=()=>{
13    return <Container></Container>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

그 후 루트 디렉터리에 있는 App.js에 아래처럼 작성



```
JS App.js \ X JS App.js src
JS App.js > [e] default
1 import App from './src/App';
2
3 export default App;
```

src폴더 밑에 components폴더를 생성하고 사용할 Button 컴포넌트를 작성하자.



```
src > components > JS Button.js > [0] Title
1  import React from 'react';
2  import styled from 'styled-components';
3
4  const Container = styled.TouchableOpacity`
5      background-color : #3498db;
6      border-radius : 15px;
7      padding : 15px 30px;
8      margin : 10px 0px;
9      justify-content : center;
10 `;
11 const Title = styled.Text`
12     font-size : 24px;
13     font-weight : 600;
14     color : #ffffff;
15 `;
16
17 const Button = ({ title, onPress }) => {
18     return(
19         <Container onPress={onPress}>
20             <Title>{title}</Title>
21         </Container>
22     );
23 };
24
25 export default Button
```

usestate

- 기본형태

```
const [state, setState] = useState(initialState);
```

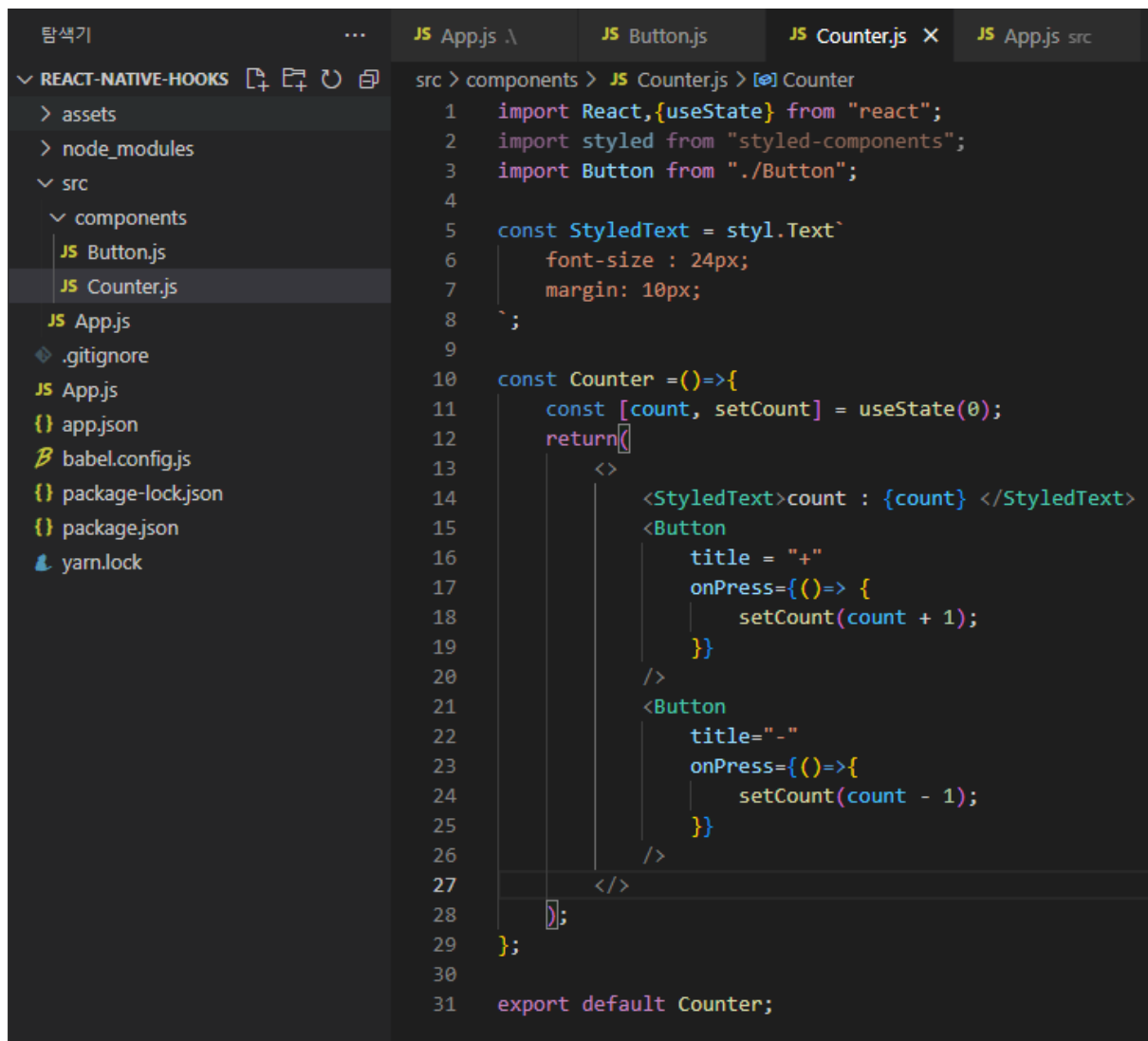
useState 함수를 호출하면 변수와 그 변수를 수정할 수 있는 세터 함수를 배열로 반환한다.

useState 함수를 호출하면 파라미터로 전달한 값을 초깃값으로 갖는 상태 변수와 그 변수를 수정할 수 있는 세터 함수를 배열로 반환한다. useState 함수는 관리해야 하는 상태의 수만

컴 여러 번 사용할 수 있다. 상태를 관리하는 변수는 반드시 세터 함수를 이용해 값을 변경해야 하고, 상태가 변경되면 컴포넌트가 변경된 내용을 반영하여 다시 렌더링 된다.

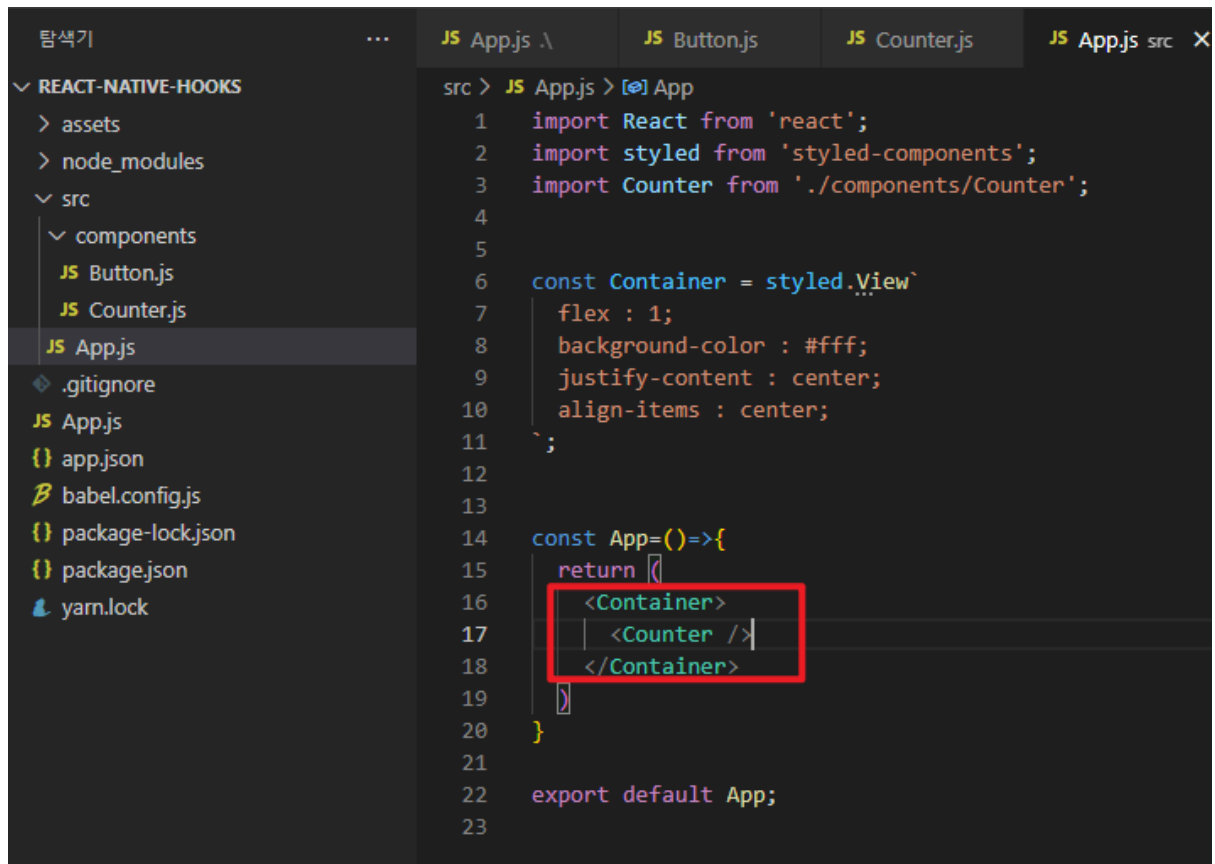
useState 사용

components폴더 안에 Counter.js 파일을 생성하고 useState를 사용해서 아래처럼 작성하자.

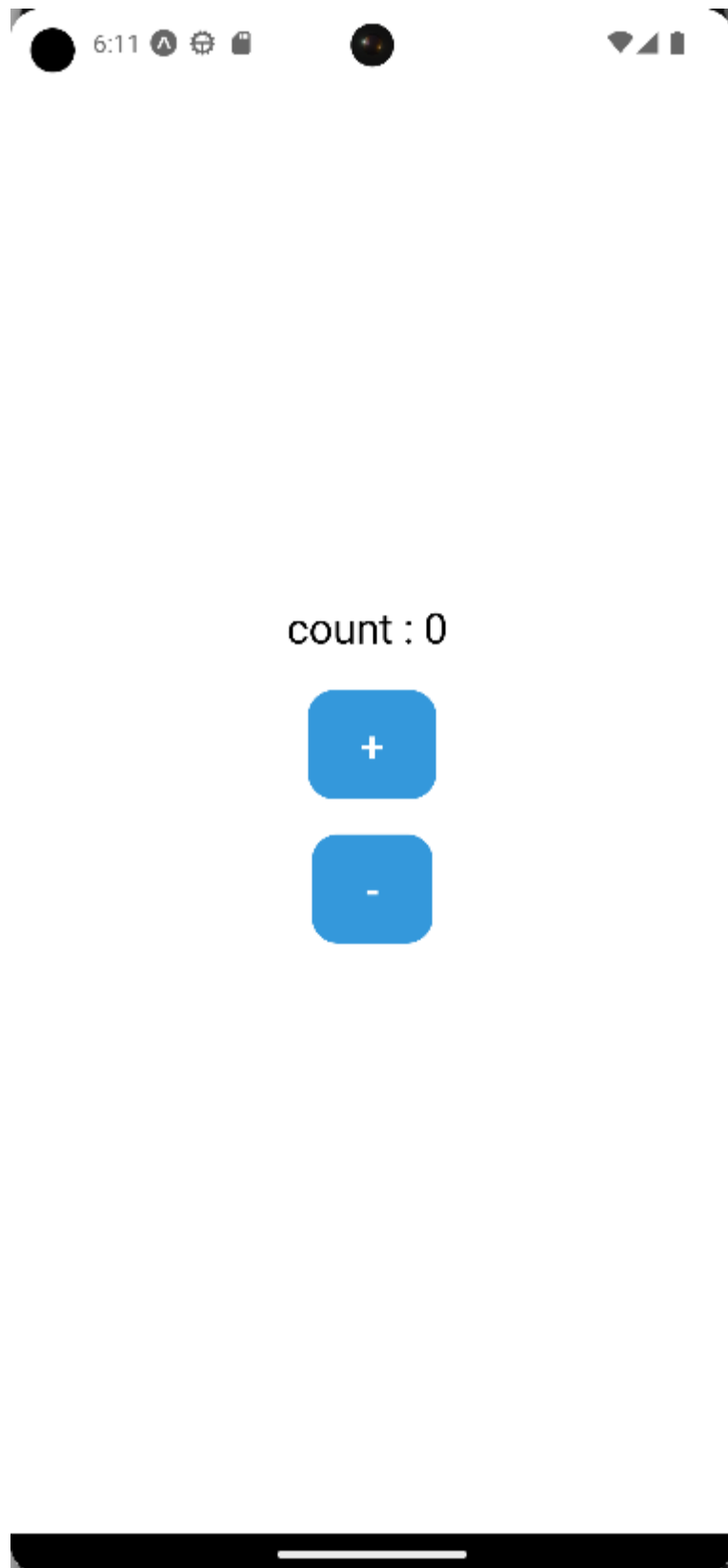


```
1  import React, {useState} from "react";
2  import styled from "styled-components";
3  import Button from "../Button";
4
5  const StyledText = styl.Text`
6    font-size : 24px;
7    margin: 10px;
8  `;
9
10 const Counter = () => {
11   const [count, setCount] = useState(0);
12   return (
13     <>
14       <StyledText>count : {count} </StyledText>
15       <Button
16         title = "+"
17         onPress={() => {
18           setCount(count + 1);
19         }}
20       />
21       <Button
22         title = "-"
23         onPress={() => {
24           setCount(count - 1);
25         }}
26       />
27     </>
28   );
29 };
30
31 export default Counter;
```

숫자의 상태를 나타내는 count를 생성하고, Button 컴포넌트를 이용하여 클릭될 때마다 세터 함수를 이용해 상태를 변경하는 버튼 두 개를 만들었다. 이제 App컴포넌트에서 Counter 컴포넌트를 사용해보자



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



세터 함수

useState에서 변환된 세터 함수를 사용하는 방법은 두 가지이다.

하나는 우리가 지금까지 사용했던 방법으로 세터 함수에 변경될 상태의 값을 전달하는 방법이다.

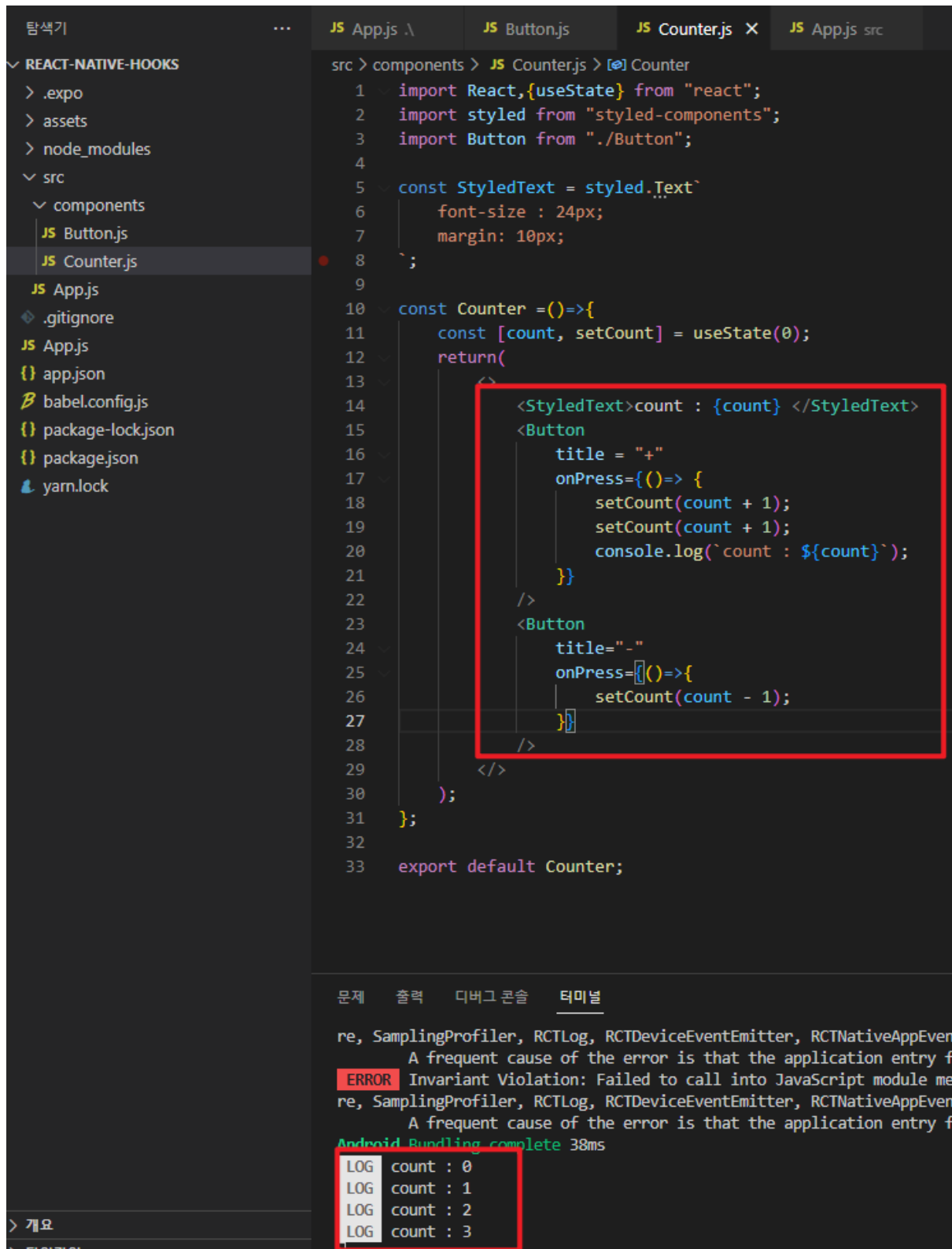
두번째 방법은 세터 함수의 파라미터에 함수를 전달하는 방법이다.

- 기본형태

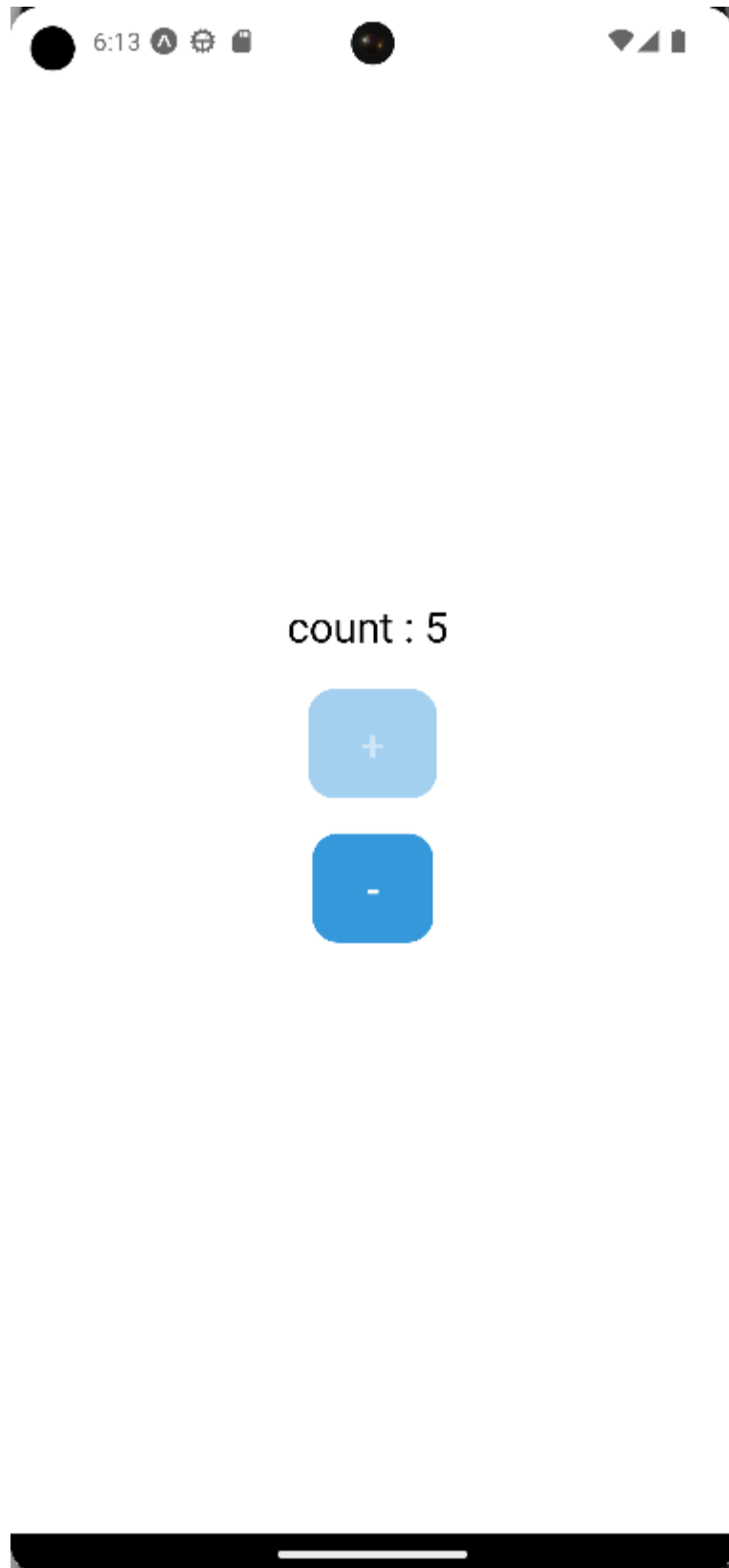
```
setState(prevState => {});
```

전달된 함수에는 변경되기 전의 상태값이 파라미터로 전달되며 이값을 어떻게 수정할지 정의하면 된다.

세터 함수는 비동기로 동작하기 때문에 상태 변경이 여러 번 일어난 경우 상태가 변경되기 전에 또 다시 상태에 대한 업데이트가 실행되는 상황이 발생한다.

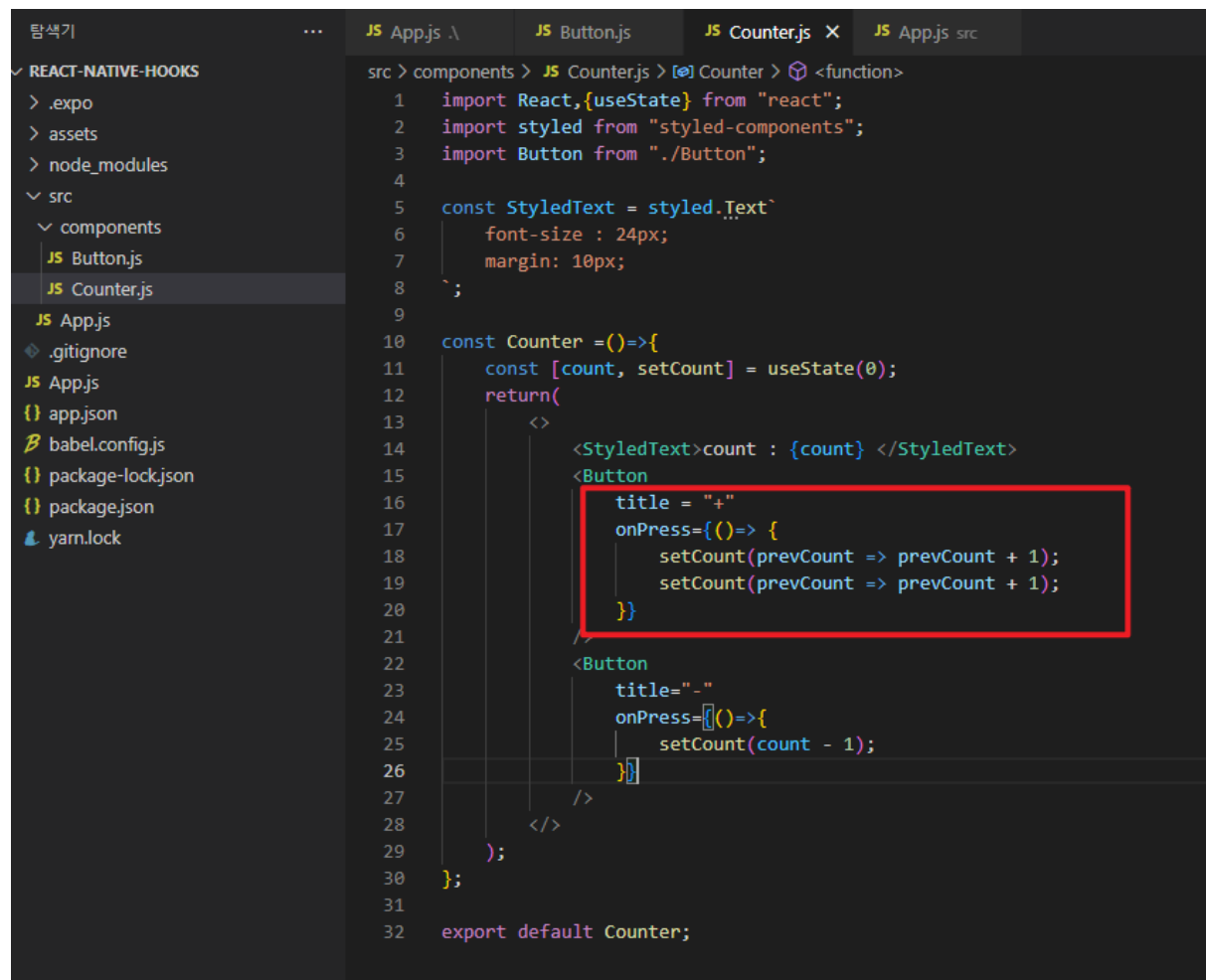


Counter 컴포넌트에서 증가 버튼을 클릭했을 때, 세터 함수를 두 번 호출하도록 수정했다.
그리고 모든 세터 함수를 호출한 후 상태의 값을 확인해보자.



증가 버튼을 클릭해도 1씩 증가하고 로그를 확인하면 증가되기 전인 현재 상태값이 나타나는 것을 확인할 수 있다.

세터 함수가 비동기로 동작하기 때문에 세터 함수를 호출해도 바로 상태가 변경되지 않는다는 점에서 발행하는 문제이다. 이렇게 상태에 대해 여러 업데이트가 함께 발생할 경우, 세터 함수에 함수를 인자로 전달하여 이전 상태값을 이용하면 문제를 해결할 수 있다.



```
src > components > JS Counter.js > Counter > <function>
1  import React,{useState} from "react";
2  import styled from "styled-components";
3  import Button from "../Button";
4
5  const StyledText = styled.Text`
6    font-size : 24px;
7    margin: 10px;
8  `;
9
10 const Counter = ()=>{
11   const [count, setCount] = useState(0);
12   return(
13     <>
14       <StyledText>count : {count} </StyledText>
15       <Button
16         title = "+"
17         onPress={()=> {
18           setCount(prevCount => prevCount + 1);
19           setCount(prevCount => prevCount + 1);
20         }}
21       />
22       <Button
23         title="-"
24         onPress={()=>{
25           setCount(count - 1);
26         }}
27       />
28     </>
29   );
30 };
31
32 export default Counter;
```

이제 2씩 증가한다.

이렇게 이전 상태의 값에 의존하여 상태를 변경할 경우, 세터 함수에 함수를 인자로 전달하여 이전 값을 이용하도록 작성해야 문제가 생기지 않는다.

useEffect

useEffect는 컴포넌트가 렌더링될 때마다 원하는 작업이 실행되도록 설정할 수 있는 기능이다

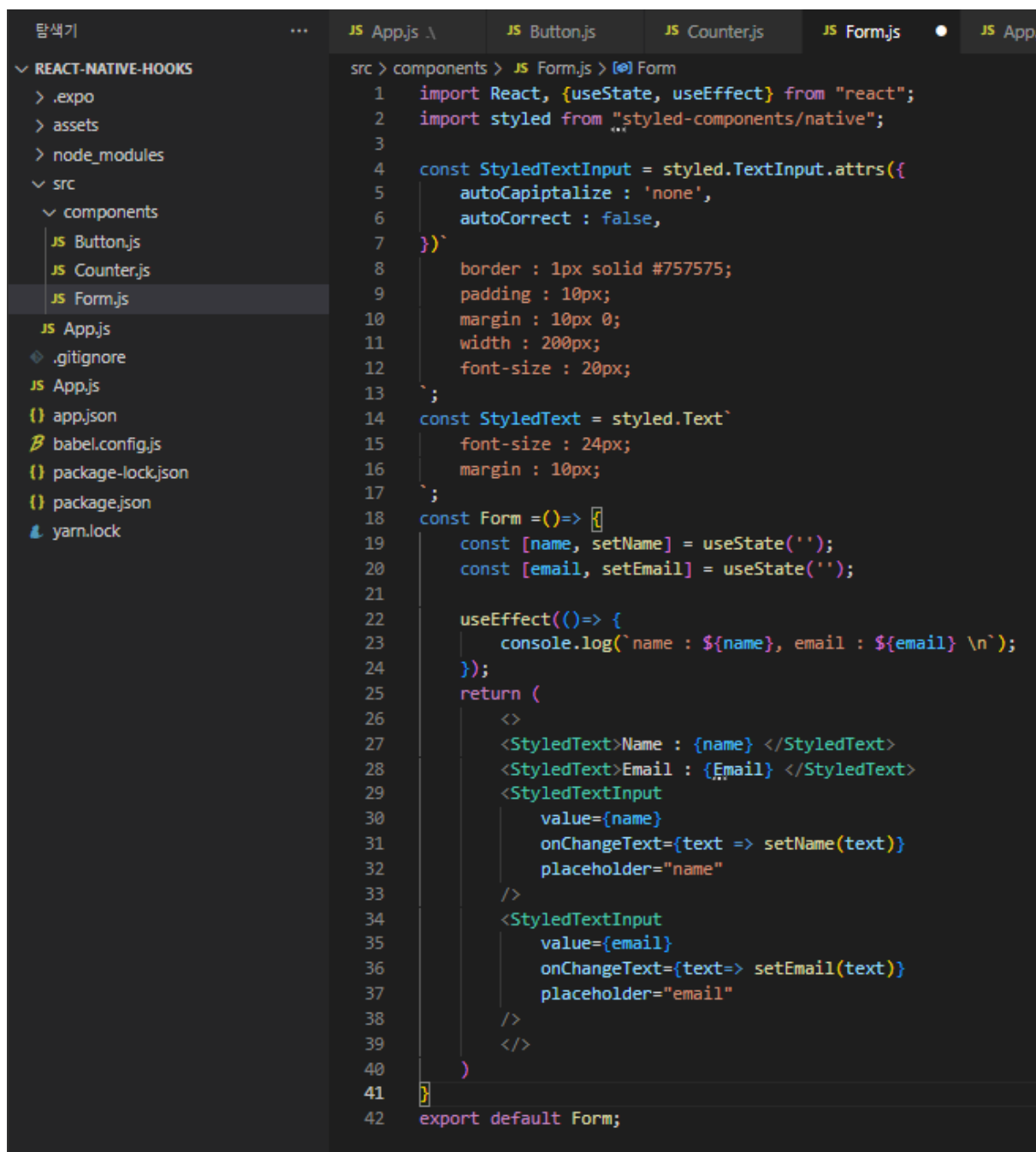
- 기본 형태

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

useEffect의 첫 번째 파라미터로 전달된 함수는 조건을 만족할 때마다 호출되고, 두 번째 파라미터로 전달되는 배열을 이용해 함수가 호출되는 조건을 설정할 수 있다.

useEffect사용하기

useEffect의 두 번째 파라미터에 어떤 값도 전달하지 않으면 useEffect의 첫 번째 파라미터로 전달된 함수는 컴포넌트가 렌더링될 때마다 호출된다. components 폴더 밑에 Form.js 파일을 생성하고 이메일과 이름을 입력받는 컴포넌트를 작성해보자.



```
src > components > JS Form.js > Form
1  import React, {useState, useEffect} from "react";
2  import styled from "styled-components/native";
3
4  const StyledTextInput = styled.TextInput.attrs({
5    autoCapitalize : 'none',
6    autoComplete : false,
7  })`
8    border : 1px solid #757575;
9    padding : 10px;
10   margin : 10px 0;
11   width : 200px;
12   font-size : 20px;
13 `;
14 const StyledText = styled.Text`
15   font-size : 24px;
16   margin : 10px;
17 `;
18 const Form = () => {
19   const [name, setName] = useState('');
20   const [email, setEmail] = useState('');
21
22   useEffect(() => {
23     console.log(`name : ${name}, email : ${email} \n`);
24   });
25   return (
26     <>
27     <StyledText>Name : {name} </StyledText>
28     <StyledText>Email : {email} </StyledText>
29     <StyledTextInput
30       value={name}
31       onChangeText={text => setName(text)}
32       placeholder="name"
33     />
34     <StyledTextInput
35       value={email}
36       onChangeText={text=> setEmail(text)}
37       placeholder="email"
38     />
39     </>
40   )
41 }
42 export default Form;
```

TextInput 컴포넌트를 이용해서 이메일과 이름을 입력받는 컴포넌트를 만들고, useEffect를 사용해서 컴포넌트가 다시 렌더링될 때마다 name과 email을 출력하도록 작성했다.

이제 App 컴포넌트에서 Form컴포넌트를 사용해보자

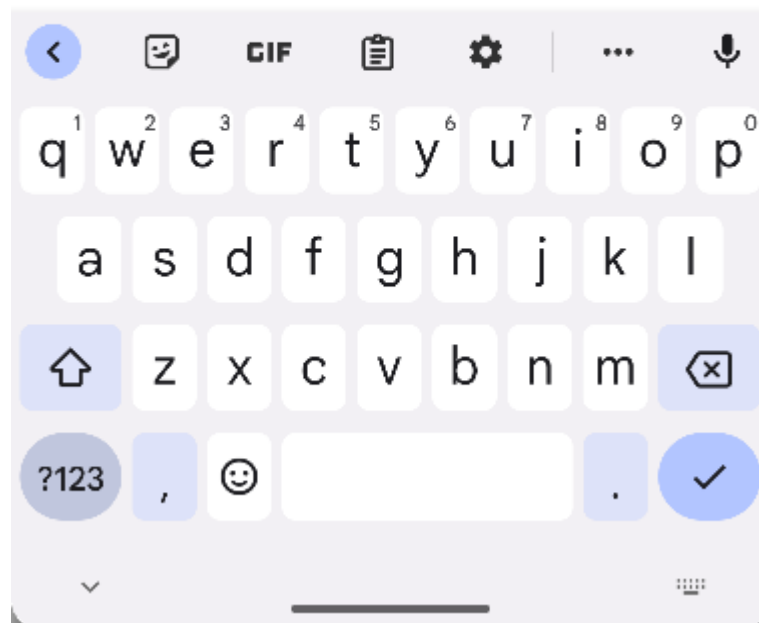


Name : leeminho

Email : krs1994@naver.com

leeminho

krs1994@naver.com




잘 실행되는 것을 확인할 수 있다.

특정 조건에서 실행하기

useEffect에 설정한 함수를 특정 상태가 변경될 때만 호출하고 싶은 경우, useEffect의 두 번째 파라미터에 해당 상태를 관리하는 변수를 배열로 전달하면 된다. 상태의 값을 변경하는 세터 함수가 비동기로 동작하므로 상태의 값이 변경되면 실행할 함수를 useEffect를 이용해서 정의해야 한다.

이번엔 email이 변경될 때만 useEffect가 동작하도록 Form 컴포넌트를 수정해보자



```
JS App.js \ JS Button.js JS Counter.js JS Form.js X JS App.js src
src > components > JS Form.js > Form
4  const StyledTextInput = styled.TextInput.attrs({
5    autoCapitalize : 'none',
6    autoComplete : false,
7  })`
8    border : 1px solid #757575;
9    padding : 10px;
10   margin : 10px 0;
11   width : 200px;
12   font-size : 20px;
13 `;
14  const StyledText = styled.Text`
15    font-size : 24px;
16    margin : 10px;
17 `;
18  const Form = () => {
19    const [name, setName] = useState('');
20    const [email, setEmail] = useState('');
21
22    useEffect(() => {
23      console.log(`name : ${name}, email : ${email} \n`);
24    }, [email]);
25    return (
26      <>
27        <StyledText>Name : {name} </StyledText>
28        <StyledText>Email : {email} </StyledText>
29        <StyledTextInput
30          value={name}
31          onChangeText={text => setName(text)}
32          placeholder="name"
33        />
34        <StyledTextInput
35          value={email}
36          onChangeText={text=> setEmail(text)}
37          placeholder="email"
38        />
39      </>
40    )
41  }
42  export default Form;
```

useEffect의 두 번째 파라미터로 email을 지정해 email의 상태가 변경되었을 때만 함수가 실행되도록 수정했다.

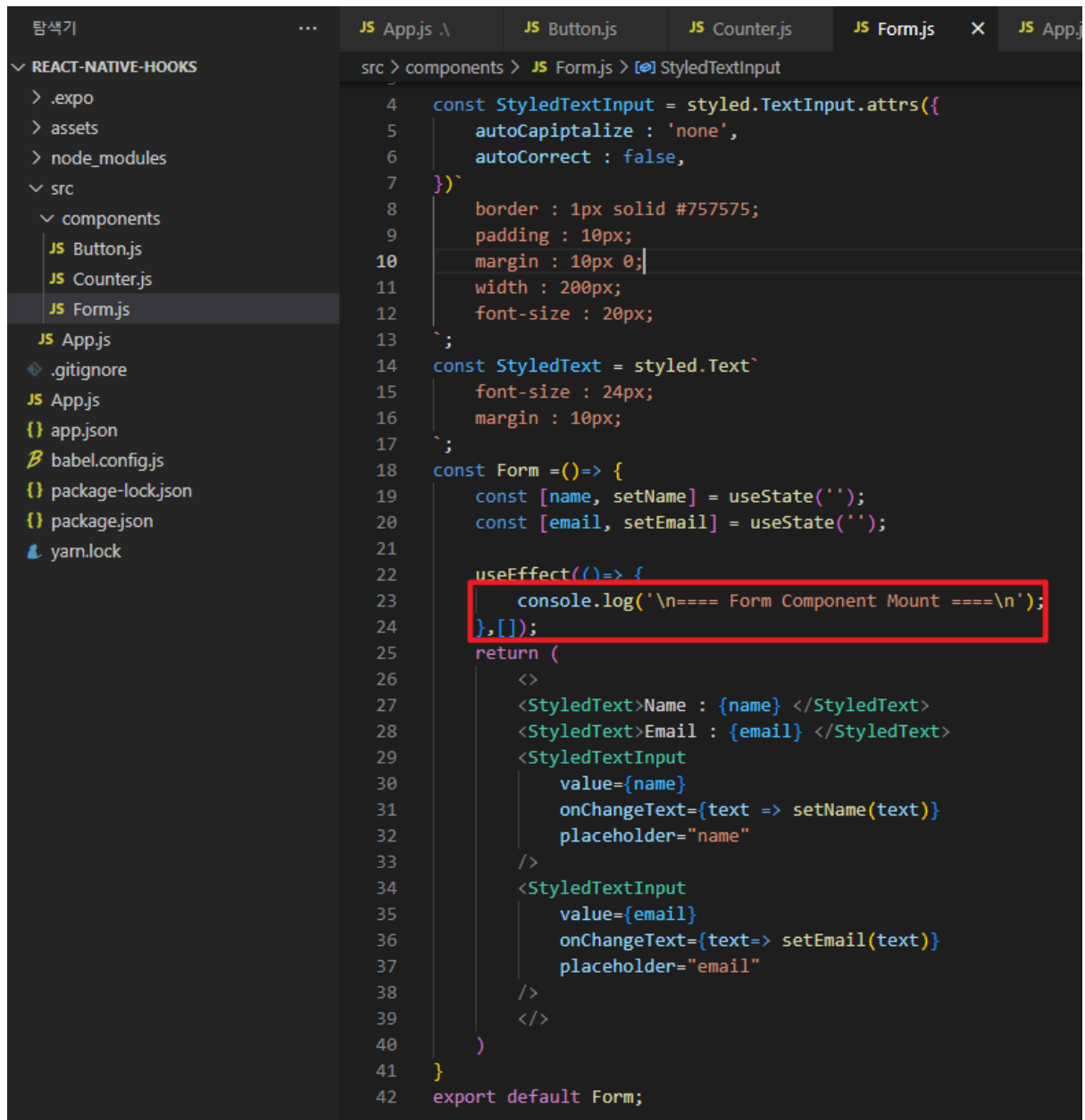
```
LOG name : leehongsi, email : krs1994@naver
LOG name : leehongsi, email : krs1994@naver.
LOG name : leehongsi, email : krs1994@naver.o
LOG name : leehongsi, email : krs1994@naver.om
LOG name : leehongsi, email : krs1994@naver.o
LOG name : leehongsi, email : krs1994@naver.
LOG name : leehongsi, email : krs1994@naver.c
LOG name : leehongsi, email : krs1994@naver.co
LOG name : leehongsi, email : krs1994@naver.com
```

name이 변경될 때는 실행되지 않지만 email이 변경될 때는 함수가 잘 실행되는 것을 확인할 수 있다.

마운트 될때 실행하기

useEffect에 전달된 함수의 실행 조건이 컴포넌트가 마운트 될 때로 설정하려면 어떻게 해야할까.

useEffect의 두 번째 파라미터에 빈 배열을 전달하면 컴포넌트가 처음 렌더링될 때만 함수가 호출되도록 작성할 수 있다.



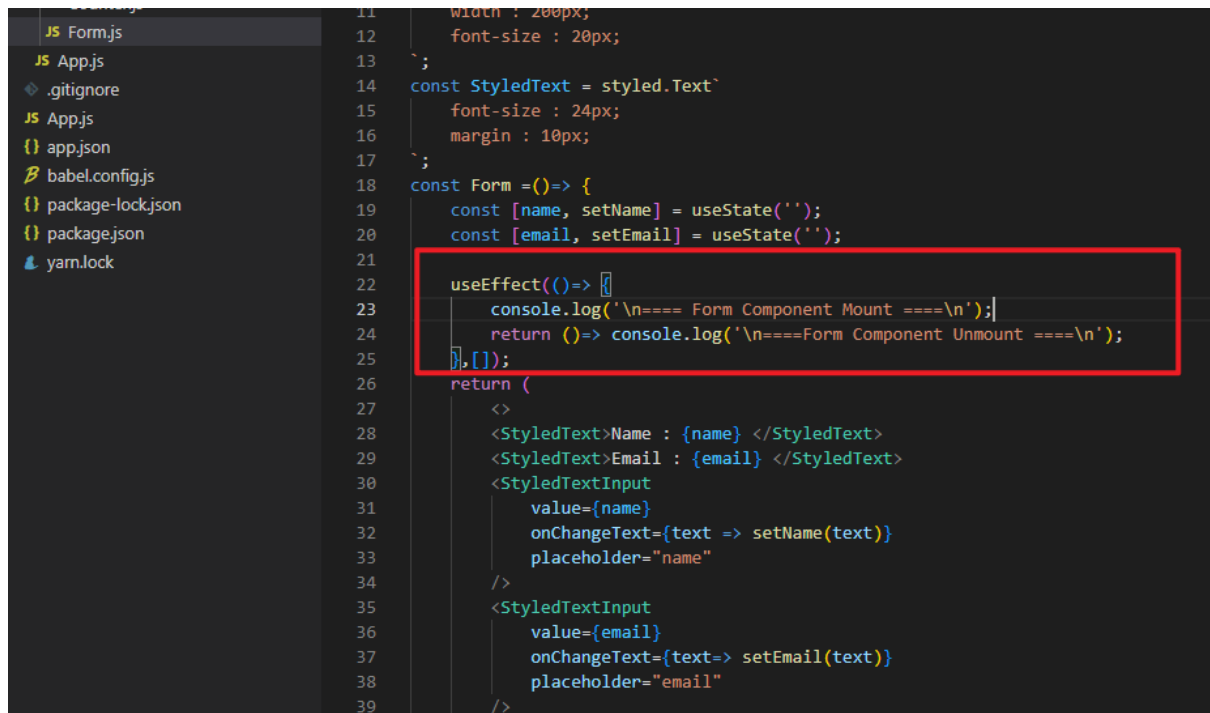
```
4 const StyledTextInput = styled.TextInput.attrs({
5   autoCapitalize : 'none',
6   autoComplete : false,
7 })`
8   border : 1px solid #757575;
9   padding : 10px;
10  margin : 10px 0;
11  width : 200px;
12  font-size : 20px;
13 `;
14 const StyledText = styled.Text`
15   font-size : 24px;
16   margin : 10px;
17 `;
18 const Form = () => {
19   const [name, setName] = useState('');
20   const [email, setEmail] = useState('');
21
22   useEffect(() => {
23     console.log('\n=== Form Component Mount ===\n');
24   }, []);
25   return (
26     <>
27       <StyledText>Name : {name} </StyledText>
28       <StyledText>Email : {email} </StyledText>
29       <StyledTextInput
30         value={name}
31         onChangeText={text => setName(text)}
32         placeholder="name"
33       />
34       <StyledTextInput
35         value={email}
36         onChangeText={text=> setEmail(text)}
37         placeholder="email"
38       />
39     </>
40   )
41 }
42 export default Form;
```

컴포넌트가 처음 렌더링될 때 함수가 호출되고, 이후에는 상태가 변해 컴포넌트가 다시 렌더링 되어도 함수가 실행되지 않는다는 것을 확인할 수 있다.

언마운트될 때 실행하기

만약 컴포넌트가 언마운트(unmount)되기 전에 특정 작업을 해야 한다면 어떻게 해야 할까
useEffect에서는 전달하는 함수에서 반환하는 함수를 정리(cleanup)함수라고 하는데,
useEffect의 실행 조건이 빈 배열인 경우 컴포넌트가 언마운트될 때 정리 함수를 실행시킨

다.



```
11 width : 200px;
12 font-size : 20px;
13 `;
14 const StyledText = styled.Text`
15   font-size : 24px;
16   margin : 10px;
17 `;
18 const Form = () => {
19   const [name, setName] = useState('');
20   const [email, setEmail] = useState('');
21
22   useEffect(() => {
23     console.log('\n=== Form Component Mount ===\n');
24     return () => console.log('\n===Form Component Unmount ===\n');
25   }, []);
26   return (
27     <>
28       <StyledText>Name : {name} </StyledText>
29       <StyledText>Email : {email} </StyledText>
30       <StyledTextInput
31         value={name}
32         onChangeText={text => setName(text)}
33         placeholder="name"
34       />
35       <StyledTextInput
36         value={email}
37         onChangeText={text => setEmail(text)}
38         placeholder="email"
39       />
40     </>
41   );
42 }
```

Form컴포넌트가 언마운트되는 상황을 테스트하기 위해 App 컴포넌트를 아래 처럼 수정하자

```

src > JS App.js > [App] App
1  import React, {useState} from 'react';
2  import styled from 'styled-components';
3  import Form from './components/Form';
4  import Button from './components/Button';
5
6
7  const Container = styled.View`
8    flex : 1;
9    background-color : #fff;
10   justify-content : center;
11   align-items : center;
12 `;
13
14
15  const App=()=>{
16     const [isVisible, setIsVisible] = useState(true);
17
18     return (
19       <Container>
20         <Button
21           title={isVisible ? 'Hide' : 'Show'}
22           onPress={() => setIsVisible(prev => !prev)}
23         />
24         {isVisible && <Form />}
25       </Container>
26     )
27   }
28
29   export default App;
30

```

{isVisible && <Form />}

조금 생소한 형식인데 저런식으로 사용하게 되면 isVisible이 true이면 Form이 화면에 보여지고 false이면 보여지지 않는 구문이다.

언마운트 될 때 정리함수가 잘 호출된다.