

day40-rn-event

☰ 태그	
📅 날짜	@2022년 12월 2일

p.94

이벤트 버튼

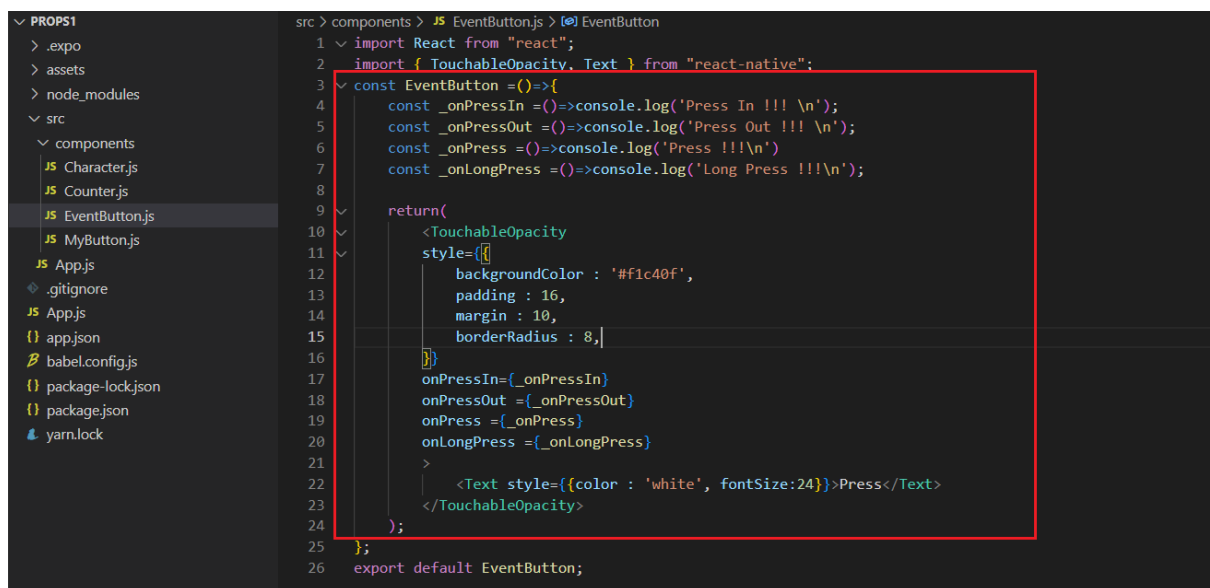
버튼을 만들 때 사용하는 TouchableOpacity 컴포넌트에서 설정할 수 있는 Press 이벤트의 종류는 4가지이다.

onPressIn : 터치가 시작될 때 항상 호출

onPressOut : 터치가 해제될 때 항상 호출

onPress : 터치가 해제될 때 onPressOut 이후 호출

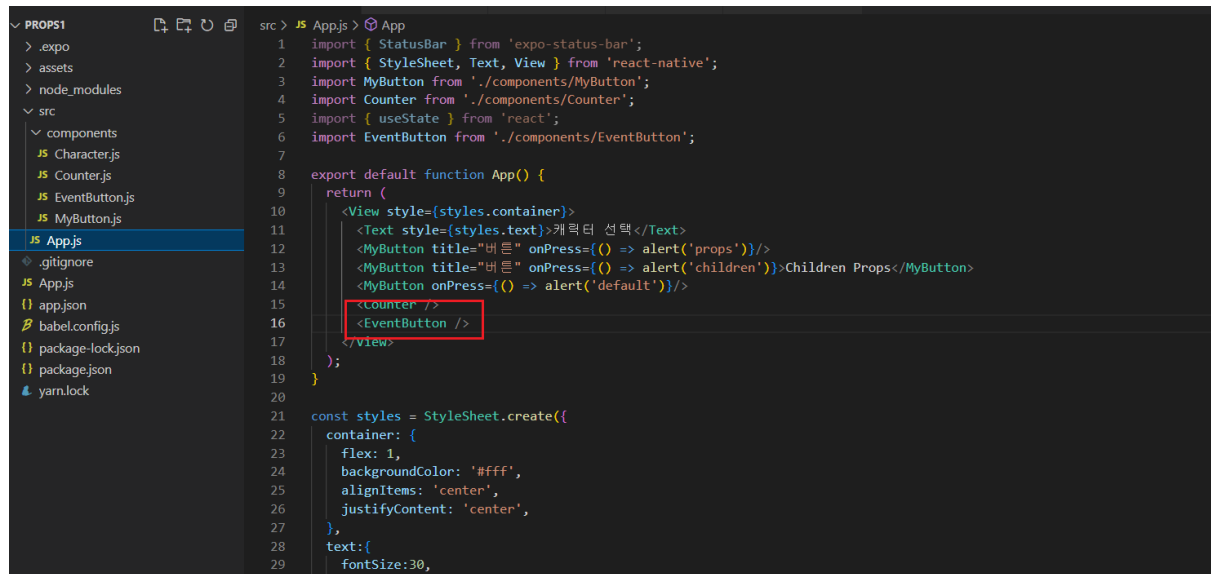
onLongPress : 터치가 일정 시간 이상 지속되면 호출



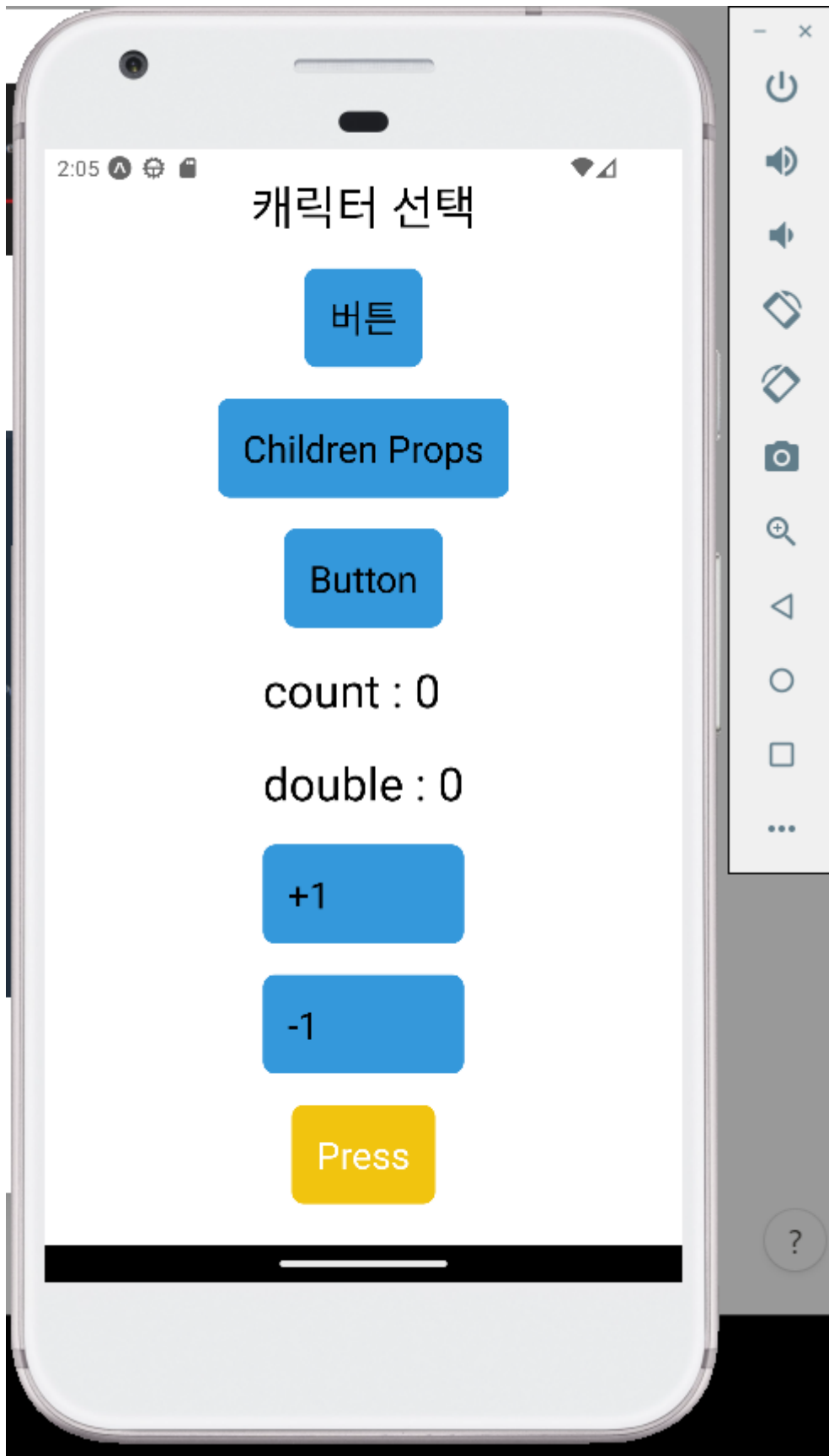
```
src > components > JS EventButton.js > [x] EventButton
1  import React from "react";
2  import { TouchableOpacity, Text } from "react-native";
3  const EventButton = () => {
4    const _onPressIn = () => console.log('Press In !!! \n');
5    const _onPressOut = () => console.log('Press Out !!! \n');
6    const _onPress = () => console.log('Press !!! \n');
7    const _onLongPress = () => console.log('Long Press !!! \n');
8
9    return (
10     <TouchableOpacity
11       style={{
12         backgroundColor : '#f1c40f',
13         padding : 16,
14         margin : 10,
15         borderRadius : 8,
16       }}
17       onPressIn={_onPressIn}
18       onPressOut={_onPressOut}
19       onPress={_onPress}
20       onLongPress={_onLongPress}
21     >
22       <Text style={{color : 'white', fontSize:24}}>Press</Text>
23     </TouchableOpacity>
24   );
25 };
26 export default EventButton;
```

EventButton.js파일을 만들고 코드를 작성하자.

그후 app.js에 추가하여 작동을 확인하자.



```
src > JS App.js > App
1  import { StatusBar } from 'expo-status-bar';
2  import { StyleSheet, Text, View } from 'react-native';
3  import MyButton from './components/MyButton';
4  import Counter from './components/Counter';
5  import { useState } from 'react';
6  import EventButton from './components/EventButton';
7
8  export default function App() {
9    return (
10     <View style={styles.container}>
11       <Text style={styles.text}>캐릭터 선택</Text>
12       <MyButton title="버튼" onPress={() => alert('props')}/>
13       <MyButton title="버튼" onPress={() => alert('children')}>Children Props</MyButton>
14       <MyButton onPress={() => alert('default')}/>
15       <Counter />
16       <EventButton />
17     </View>
18   );
19 }
20
21 const styles = StyleSheet.create({
22   container: {
23     flex: 1,
24     backgroundColor: 'ffff',
25     alignItems: 'center',
26     justifyContent: 'center',
27   },
28   text: {
29     fontSize: 30,
```



노란색 press 버튼이 생겼다.

버튼을 한번 눌렀다 뺐을때의 로그이다.

```
LOG Press In !!!  
LOG Press Out !!!  
LOG Press !!!
```

길게 눌렀다 뺐을때의 로그

```
LOG Press In !!!  
LOG Long Press !!!  
LOG Press Out !!!
```

onPress와 onLongPress는 사용자가 클릭하는 시간에 따라 둘 중 하나만 호출된다.

만약 onLongPress가 호출되는 시간을 조절하고 싶다면, delayLongPress의 값을 조절해서 원하는 시간으로 설정할 수 있다.

```
1 import React from "react";  
2 import { TouchableOpacity, Text } from "react-native";  
3 const EventButton = ()=>{  
4   const _onPressIn = ()=>console.log('Press In !!! \n');  
5   const _onPressOut = ()=>console.log('Press Out !!! \n');  
6   const _onPress = ()=>console.log('Press !!!\n');  
7   const _onLongPress = ()=>console.log('Long Press !!!\n');  
8  
9   return(  
10     <TouchableOpacity  
11       style={{  
12         backgroundColor : '#f1c40f',  
13         padding : 16,  
14         margin : 10,  
15         borderRadius : 8,  
16       }}  
17       onPressIn={_onPressIn}  
18       onPressOut ={_onPressOut}  
19       onPress ={_onPress}  
20       onLongPress ={ onLongPress}  
21       delayLongPress={3000}  
22     >  
23       <Text style={{color : 'white', fontSize:24}}>Press</Text>  
24     </TouchableOpacity>  
25   );  
26 };  
27 export default EventButton;
```

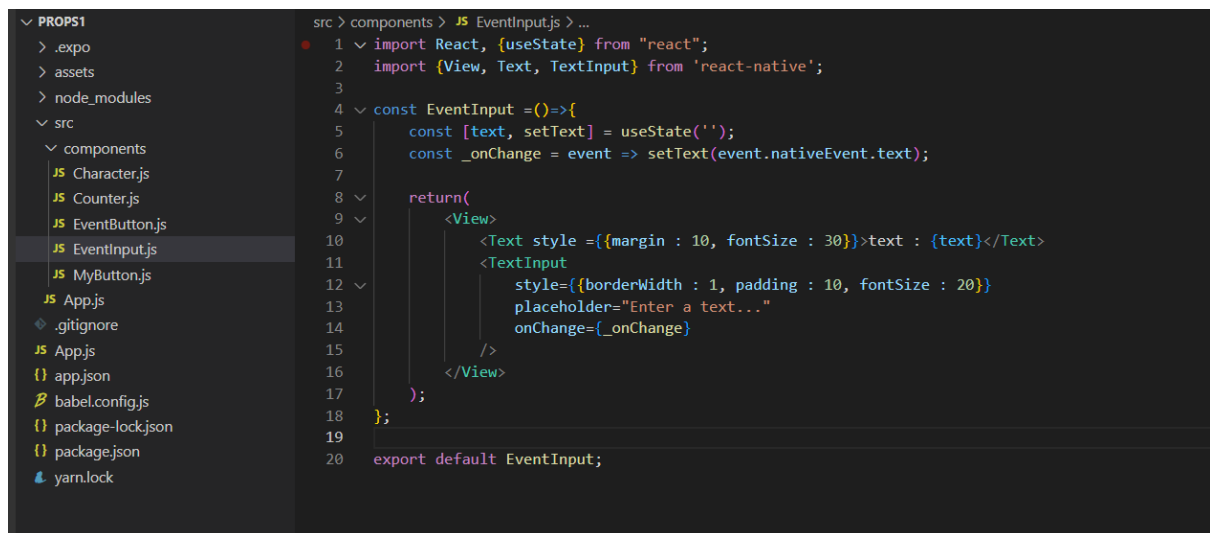
이 코드처럼 delayLongPress 의 값을 3000으로 변경하면 3초 동안 클릭하고 있어야 onLongPress가 호출된다는 것을 확인할 수 있다.

change 이벤트

변화를 감지하는 change 이벤트 값을 입력하는 TextInput 컴포넌트에서 많이 사용된다.

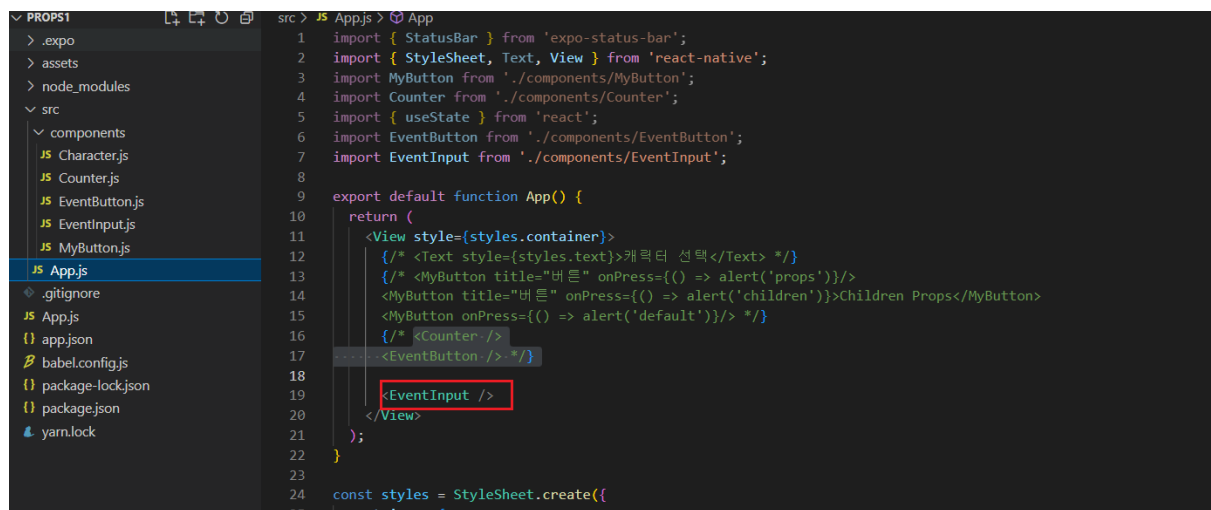
TextInput 컴포넌트를 이용하여 변화하는 텍스트를 출력해보자.

components 폴더 밑에 EventInput.js 파일을 생성하고 코드를 작성하자.

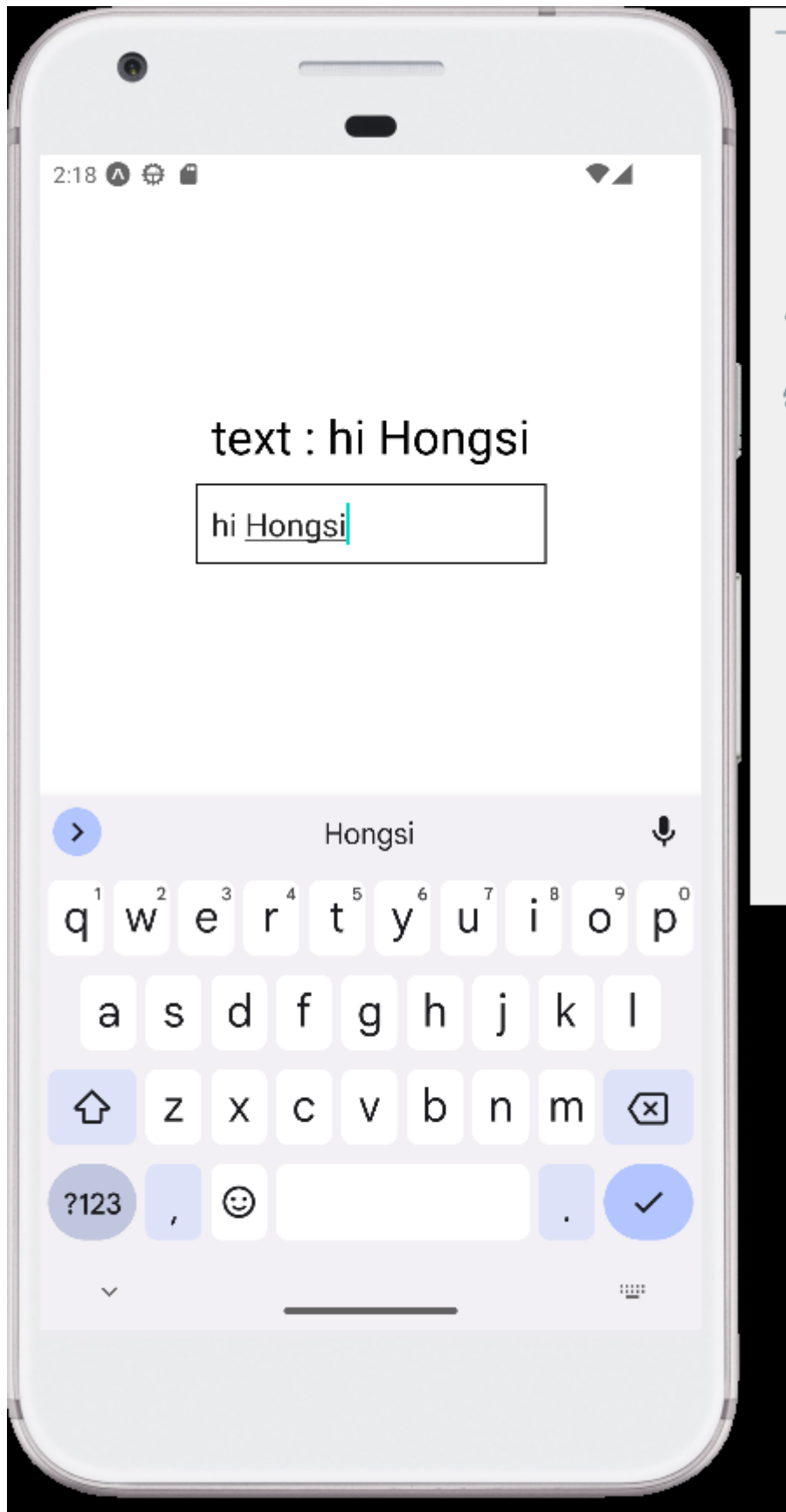


```
src > components > JS EventInput.js > ...
1  import React, {useState} from "react";
2  import {View, Text, TextInput} from 'react-native';
3
4  const EventInput = () => {
5    const [text, setText] = useState('');
6    const _onChange = event => setText(event.nativeEvent.text);
7
8    return(
9      <View>
10        <Text style={{margin : 10, fontSize : 30}}>text : {text}</Text>
11        <TextInput
12          style={{borderWidth : 1, padding : 10, fontSize : 20}}
13          placeholder="Enter a text..."
14          onChange={_onChange}
15        />
16      </View>
17    );
18  };
19
20  export default EventInput;
```

화면에서 보여야 하니 app.js에도 컴포넌트를 추가하자.



```
1 import { StatusBar } from 'expo-status-bar';
2 import { StyleSheet, Text, View } from 'react-native';
3 import MyButton from './components/MyButton';
4 import Counter from './components/Counter';
5 import { useState } from 'react';
6 import EventButton from './components/EventButton';
7 import EventInput from './components/EventInput';
8
9 export default function App() {
10   return (
11     <View style={styles.container}>
12       <Text style={styles.text}><캐릭터 선택</Text> */>
13       <MyButton title="버튼" onPress={() => alert('props')}>
14         <MyButton title="버튼" onPress={() => alert('children')}>Children Props</MyButton>
15       <MyButton onPress={() => alert('default')}> */>
16       <Counter />
17       <EventButton /> */>
18       <EventInput />
19     </View>
20   );
21 }
22
23
24 const styles = StyleSheet.create({
25   container: {
26     flex: 1,
27     alignContent: 'center',
28     justify-content: 'center',
29   },
30   text: {
31     margin: 10,
32   },
33 });
```

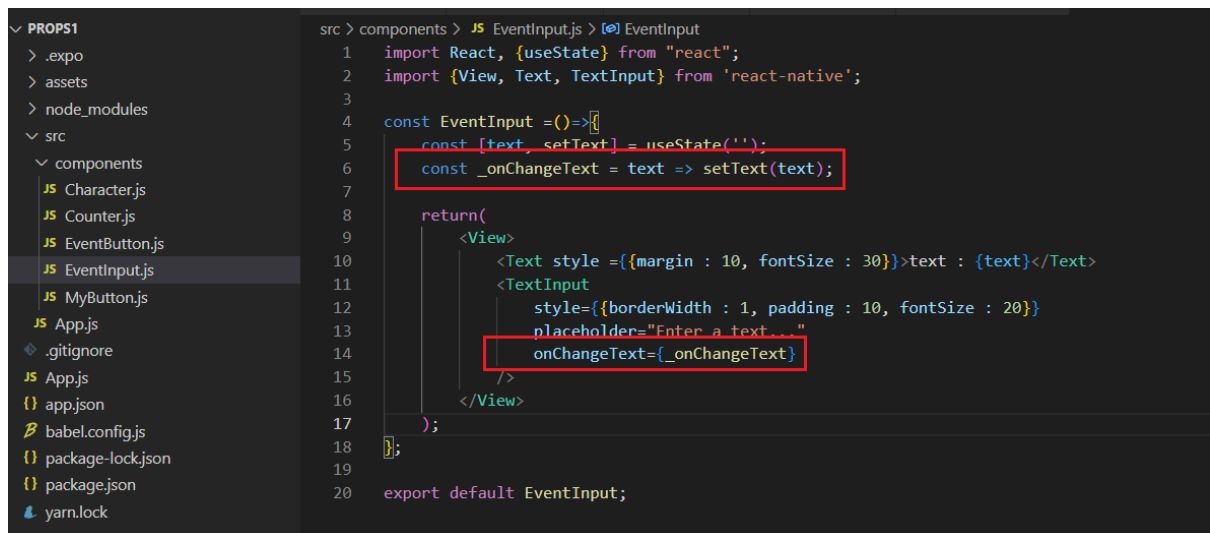


TextInput의 값이 변경될 때마다 화면에 잘나타난다.

onChange를 통해 전달되는 내용 중 필요한 것은 변화된 텍스트 뿐이다.

onChangeText는 이런 상황에서 조금 더 간편하게 사용할 수 있다.

onChangeText는 컴포넌트의 텍스트가 변경되었을 때 변경된 텍스트의 문자열만 인수로 전달하여 호출된다.



```
src > components > JS EventInput.js > [0] EventInput
1  import React, {useState} from "react";
2  import {View, Text, TextInput} from 'react-native';
3
4  const EventInput = ()=>{
5    const [text, setText] = useState('');
6    const _onChangeText = text => setText(text);
7
8    return(
9      <View>
10        <Text style={{margin : 10, fontSize : 30}}>text : {text}</Text>
11        <TextInput
12          style={{borderWidth : 1, padding : 10, fontSize : 20}}
13          placeholder="Enter a text..."
14          onChangeText={_onChangeText}
15        />
16      </View>
17    );
18  };
19
20  export default EventInput;
```

Pressable 컴포넌트

리액트 네이티브 0.63버전 부터 기존 TouchableOpacity 컴포넌트를 대체하는 Pressable 컴포넌트가 추가되었다.

기존의 컴포넌트보다 더 다양한 기능을 제공하므로 리액트 네이티브 0.63버전 이상을 사용해서 Pressable 컴포넌트를 사용할 수 있다면 TouchableOpacity 컴포넌트 대신 Pressable 컴포넌트 사용으 권장한다.

리액트 네이티브 CLI를 이용해서 Pressable 컴포넌트를 알아보자

Pressable 컴포넌트는 TouchableOpacity 컴포넌트 처럼 사용자의 터치에 상호작용하는 컴포넌트이다.

press 이벤트도 동일하게 존재하고 동작 방식도 같다. Pressable 컴포넌트에서 지원하는 기능 중 기존의 컴포넌트들과 다른 특징은 HitRect와 PressRect이다.

모바일이라는 작은 화면에서 버튼을 포함하여 다양한 요소들을 보여준다. 화면이 작은 만큼 버튼도 작아지는데, 사람마다 손의 크기나 투께가 모두 다르기 때문에 누군가는 버튼을 정확하게 클릭하는 것이 어려울 수 있다. 이 경우 의도하지 않은 버튼을 클릭하거나 버튼을 클릭하는 것 자체가 어려울 수 있다. 이런 상황을 해결하기 위해 많은 개발자들이 버튼 모양보다 약간 떨어진 부분까지 이벤트가 발생할 수 있도록 조치하고 있다.

Pressable 컴포넌트에서는 HitRect를 이용해 쉽게 설정 할 수 있다.

한번쯤 버튼을 클릭했을 때 해당 버튼이 동작하지 않게 하기 위해 버튼을 누른 상태에서 손가락을 이동시킨 경험이 있다. 그렇다면 얼마나 멀어져야 버튼을 누른 상태에서 벗어났다고 판단할 수 있을까? Pressable 컴포넌트에서는 이 부분도 개발자가 조절하기 편하도록 PressRect기능을 지원한다.

그림 p.101

Pressable컴포넌트를 테스트 하기 위해 아래 명령어를 이용해 새로운 프로젝트를 생성한다.

경로 중요

`npx react-native init RNPressable`

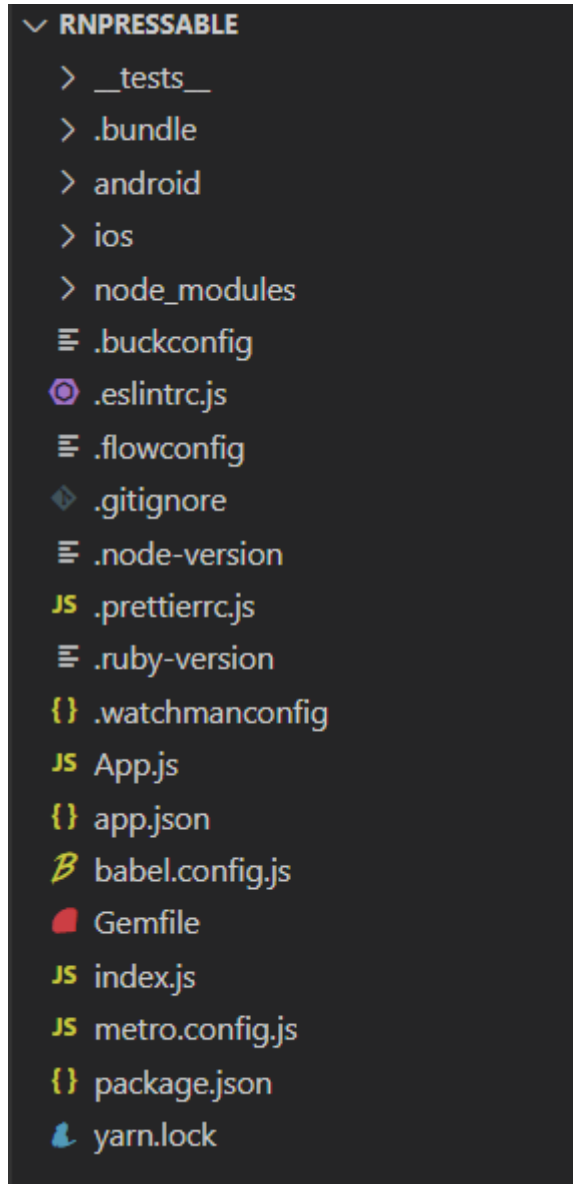
```
Microsoft Windows [Version 10.0.15041.2201]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tjoeun>cd ..

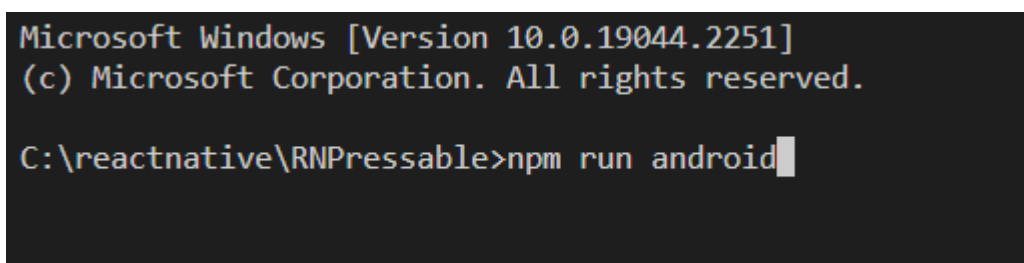
C:\Users>cd ..

C:\>cd reactnative

C:\reactnative>npx react-native init RNPressable
```



이전과 파일이 많이 다르다.



전과는 다르게 위 명령어로 실행 시키게 된다.

프로젝트가 완성되면 App.js파일을 아래처럼 수정하자.

```

9   import React from "react";
10  import {View, Text, Pressable} from 'react-native';
11
12  const Button = (props) => {
13    console.log();
14    return(
15      <Pressable
16        style={{padding : 10, backgroundColor : '#1abc9c'}}
17        onPressIn={() => console.log('Press In')}
18        onPressOut={() => console.log('Press Out')}
19        onPress={() => console.log('Press')}
20        onLongPress={() => console.log('Long Press')}
21        delayLongPress={3000}
22        pressRetentionOffset={{bottom : 50, left:50, right: 50, top: 50}}
23        hitSlop={50}>
24
25        <Text style={{padding : 10, fontSize : 30}}>{props.title}</Text>
26
27      </Pressable>
28    );
29  };
30
31  const App =()=>{
32    return(
33      <View
34        style={{
35          flex : 1,
36          justifyContent : 'center',
37          backgroundColor : '#fff',
38          alignItems : 'center',
39
40        }}>
41        <Button title="Pressable" />
42      </View>
43    );
44  };
45  export default App;

```

버튼에서 조금 떨어져 있어도 클릭되고, 버튼을 누른 상태에서 이동했을 때 항상 같은 위치에서 onPressOut이 호출되는것을 확인 할 수 있다. PressRect의 범위는 HitRect의 범위 끝에서부터 시작되므로 hitSlop의 값에 따라 PressRect의 범위가 달라진다는것을 기억하자.

그냥 클릭했을 때

```
LOG Press In
LOG Press
LOG Press Out
```

길게 클릭했을 때

```
LOG Press In
LOG Long Press
LOG Press Out
```

클릭 후 버튼을 벗어났을 때

```
LOG Press In
LOG Press Out
```

리액트 네이티브는 컴포넌트들의 조합으로 만들어지므로 컴포넌트가 굉장히 중요하다.

리액트 네이티브에서 다양한 내장 컴포넌트를 제공하고는 있지만, 커스텀 컴포넌트를 만들어서 사용하는 경우가 생각보다 많다.

필요한 컴포넌트를 만드는 연습을 많이 하는게 좋다.

컴포넌트들을 조합하다 보면 자연스럽게 부모 컴포넌트의 state를 자식 컴포넌트의 props로 전달하고, 자식 컴포넌트에서 부모 컴포넌트의 state 변경을 요청하게 된다. props와 state를 다루는것은 컴포넌트에서 굉장히 중요한 부분이므로 꼭 기억해두자.

