

day46-rn-todolist

☰ 태그	
📅 날짜	@2022년 12월 5일

만들어볼 애플리케이션에는 다음과 같은 기능이 있다

- 등록 : 할일 항목을 추가하는 기능
- 수정 : 완료되지 않은 할일 항목을 수정하는 기능
- 삭제 : 할 일 항목을 삭제하는 기능
- 완료 할 일 항목의 완료 상태를 관리하는 기능

프로젝트 생성

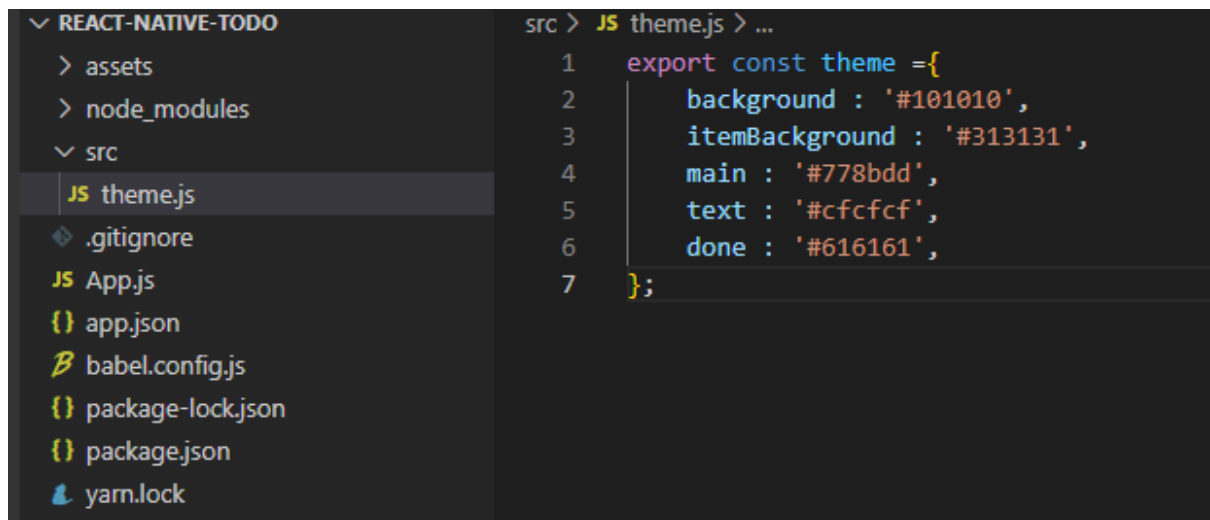
```
expo init react-native-todo
```

프로젝트 생성이 완료되면 스타일드 컴포넌트 라이브러리와 prop-types라이브러리도 설치한다.

```
cd react-native-todo
```

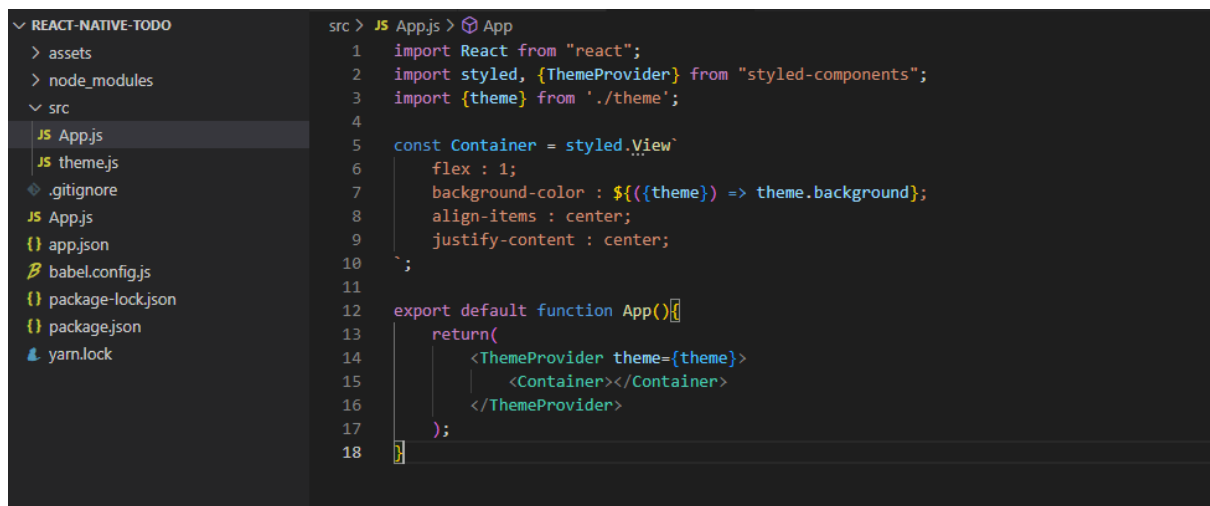
```
npm install styled-components prop-types
```

프로젝트에서 작성하는 코드를 관리할 src폴더를 생성하자. 그 안에 theme.js파일을 생성하고 프로젝트에서 사용할 색을 정의하자



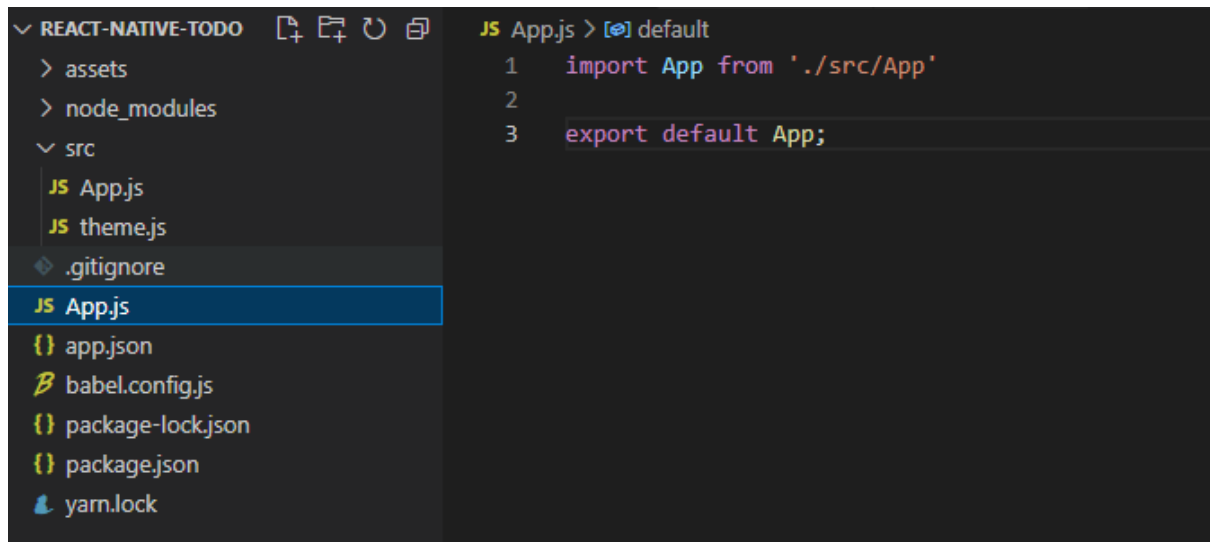
```
src > JS theme.js > ...
1  export const theme = {
2    background : '#101010',
3    itemBackground : '#313131',
4    main : '#778bdd',
5    text : '#cfcfcf',
6    done : '#616161',
7  };
```

이제 src폴더에 App.js를 생성하고 app컴포넌트를 작성하자.



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

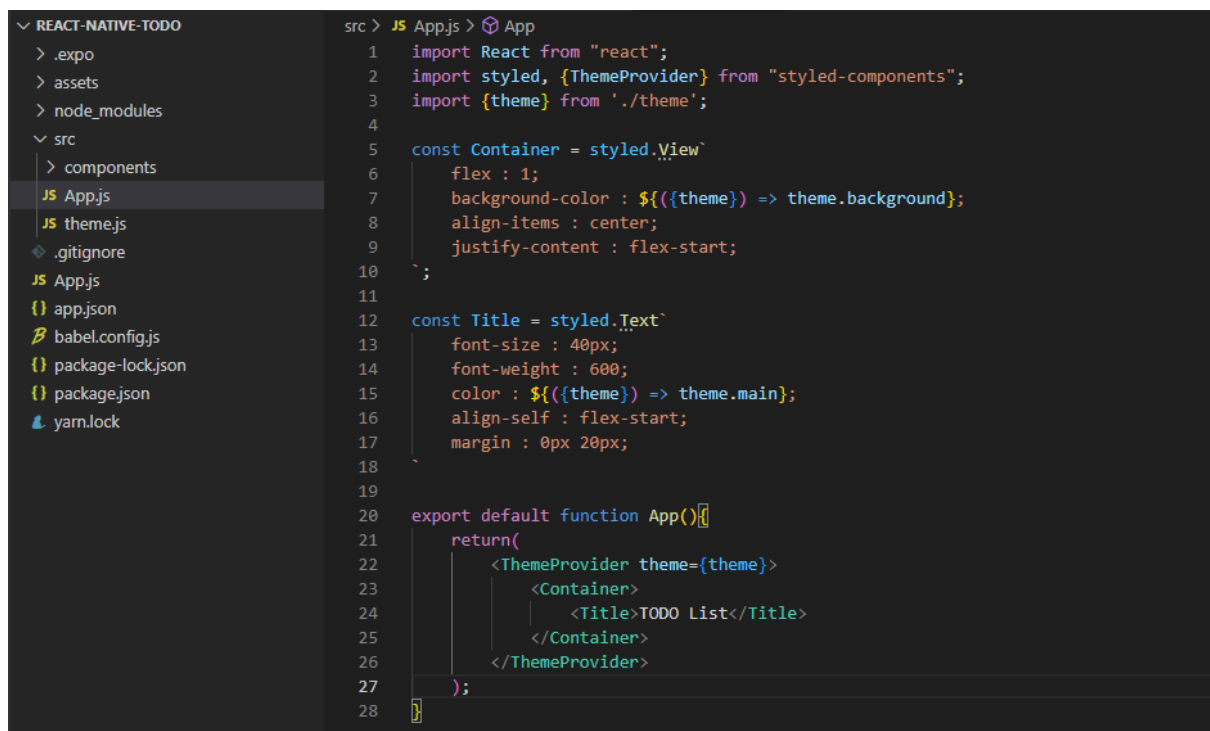
이제 작성된 App 컴포넌트가 프로젝트의 메인 파일이 되도록 프로젝트 루트 디렉터리에 있는 app파일을 변경하자



프로젝트를 진행하면서 만드는 컴포넌트를 관리할 파일을 src폴더 밑에 만들도록 하자

타이틀 만들기

가장 먼저 화면 상단에 TODO LIST 라는 문구가 렌더링 되도록 타이틀을 만들어보자

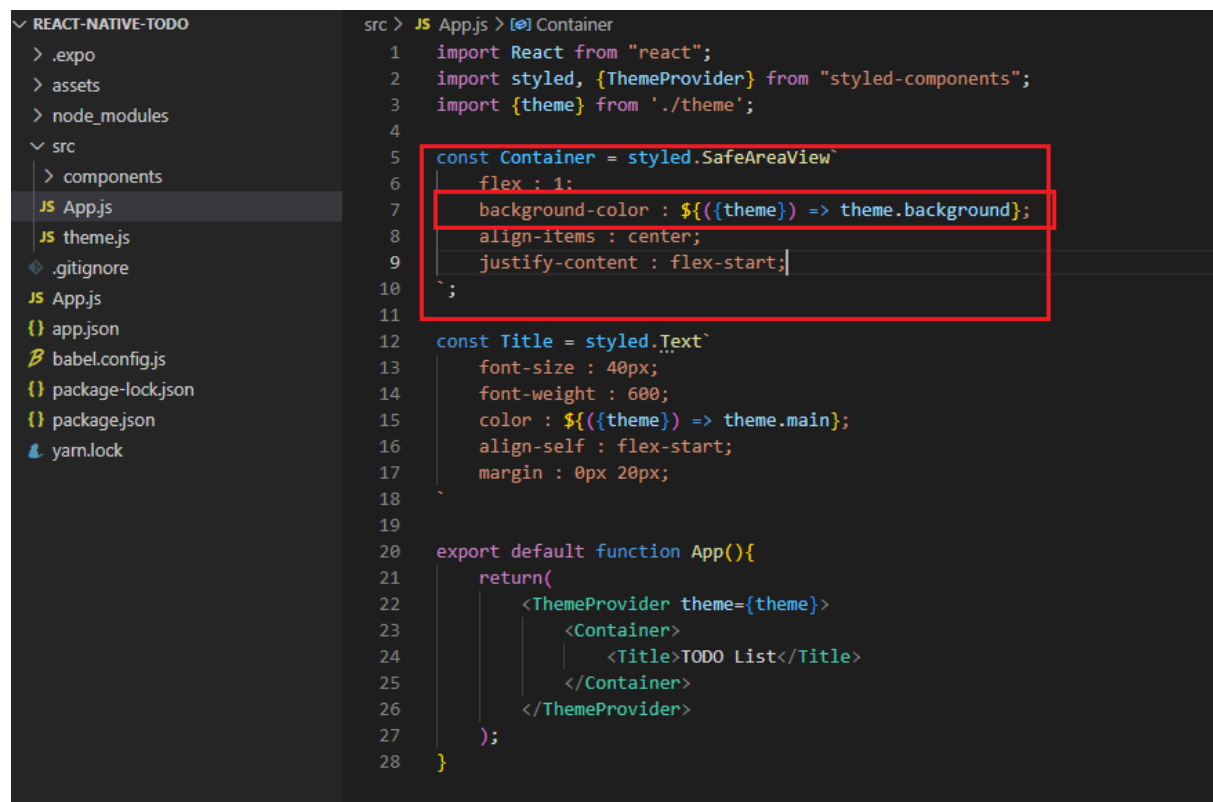




safeAreaView 컴포넌트

title 컴포넌트의 일부가 가려지는 것을 볼 수 있다.

리액트 네이티브에서는 자동으로 padding값이 적용되어 노치 디자인 문제를 해결할 수 있는 safeAreaView 컴포넌트를 제공한다.



```
src > JS App.js > Container
1  import React from "react";
2  import styled, {ThemeProvider} from "styled-components";
3  import {theme} from '../theme';
4
5  const Container = styled.SafeAreaView`
6    flex : 1;
7    background-color : ${({theme}) => theme.background};
8    align-items : center;
9    justify-content : flex-start;
10 `;
11
12 const Title = styled.Text`
13   font-size : 40px;
14   font-weight : 600;
15   color : ${({theme}) => theme.main};
16   align-self : flex-start;
17   margin : 0px 20px;
18 `;
19
20 export default function App(){
21   return(
22     <ThemeProvider theme={theme}>
23       <Container>
24         <Title>TODO List</Title>
25       </Container>
26     </ThemeProvider>
27   );
28 }
```

statusbar 컴포넌트

이번에는 상태바를 변경해 안드로이드에서 title 컴포넌트가 가려지는 문제를 해결하고 어두운 배경에서도 잘 보이도록 스타일을 수정해보자.

리액트 네이티브에서는 상태바를 제어할 수 있는 statusBar 컴포넌트를 제공한다.

statusbar 컴포넌트를 이용하면 상태 바의 스타일을 변경할 수 있고 안드로이드 기기에서 상태바가 컴포넌트를 가리는 문제를 해결할 수 있다.

```

▼ REACT-NATIVE-TODO
  > .expo
  > assets
  > node_modules
  ▼ src
    > components
    JS App.js
    JS theme.js
    .gitignore
    JS App.js
    {} app.json
    {} babel.config.js
    {} package-lock.json
    {} package.json
    {} yarn.lock

src > JS App.js > App
1  import React from "react";
2  import styled, {ThemeProvider} from "styled-components";
3  import {theme} from '../theme';
4  import { StatusBar } from "react-native";
5
6  const Container = styled.SafeAreaView`
7    flex : 1;
8    background-color : ${({theme}) => theme.background};
9    align-items : center;
10   justify-content : flex-start;
11 `;
12
13 const Title = styled.Text`
14   font-size : 40px;
15   font-weight : 600;
16   color : ${({theme}) => theme.main};
17   align-self : flex-start;
18   margin : 0px 20px;
19 `
20
21 export default function App() {
22   return(
23     <ThemeProvider theme={theme}>
24       <Container>
25         <StatusBar barStyle="light-content" backgroundColor={theme.background} />
26         <Title>TODO List</Title>
27       </Container>
28     </ThemeProvider>
29   );
30 }
```



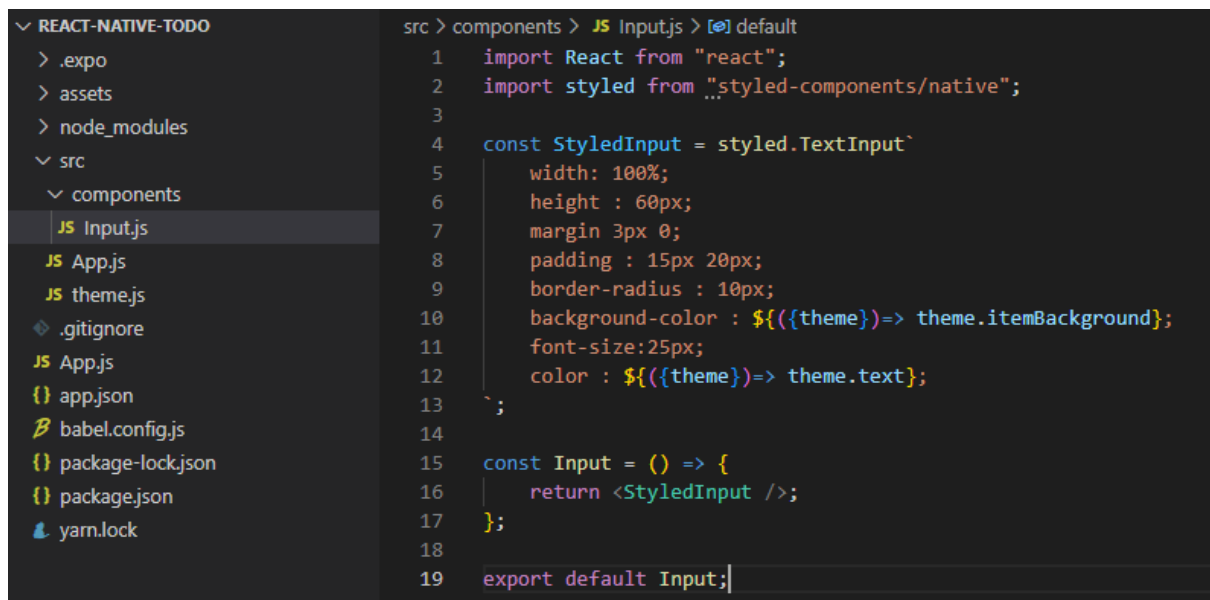
statusbar 컴포넌트를 이용해 상태바의 내용이 흰색으로 나타나도록 수정했다. statusBar 컴포넌트의 backgroundColor 속성은 안드로이드에만 적용되는 속성이며 상태 바의 바탕색을 변경할 수 있다.

input 컴포넌트 만들기

textInput 컴포넌트를 이용해 input 컴포넌트를 만들어 보자

input컴포넌트는 할 일 항목을 추가할 때뿐만 아니라, 등록된 할 일 항목을 수정할 때도 사용할 것이다.

component폴더 밑에 input 파일을 만들고 컴포넌트를 생성하자.



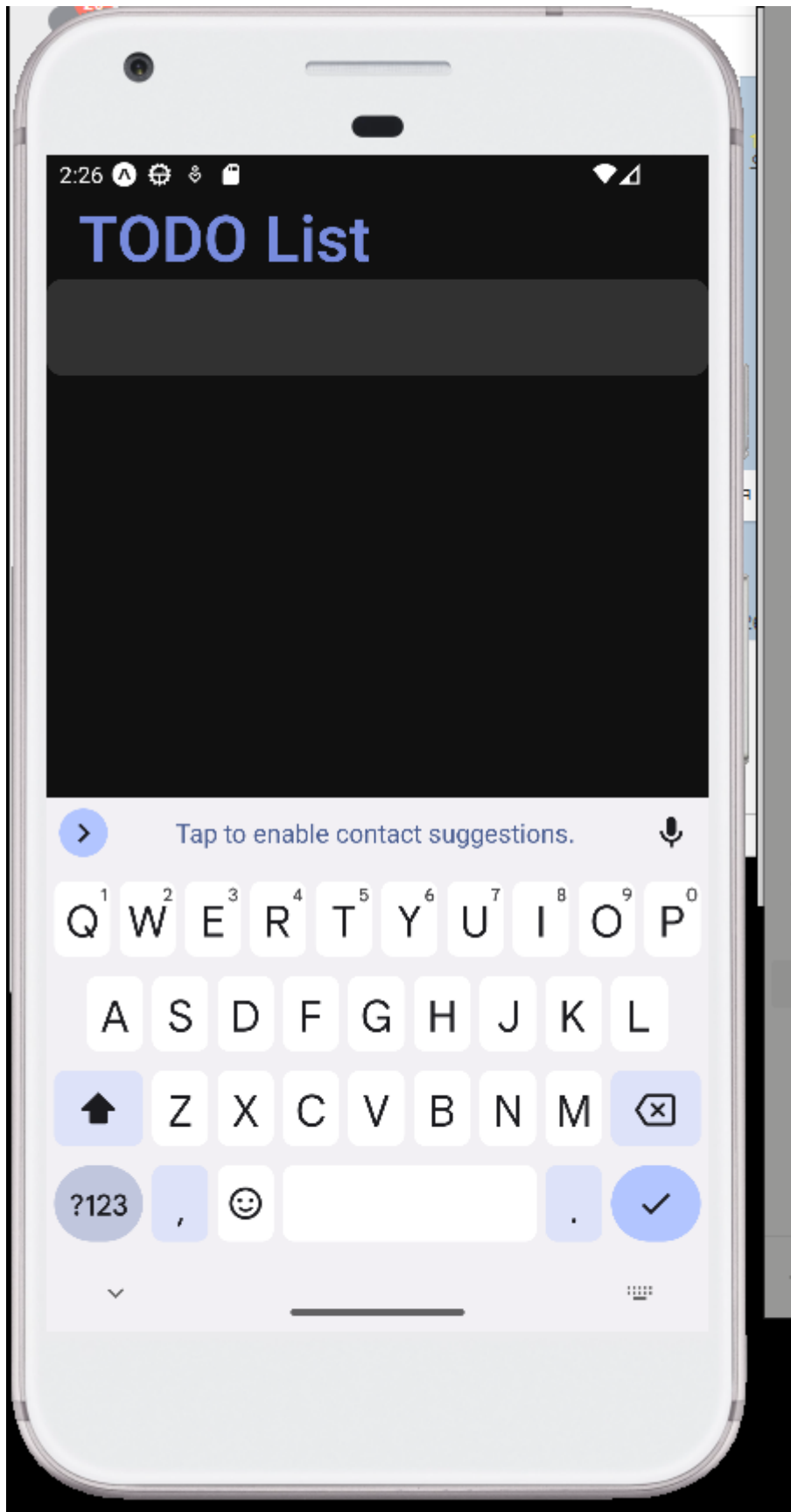
```
src > components > JS Input.js > [🔍] default
1  import React from "react";
2  import styled from "styled-components/native";
3
4  const StyledInput = styled.TextInput`
5    width: 100%;
6    height : 60px;
7    margin 3px 0;
8    padding : 15px 20px;
9    border-radius : 10px;
10   background-color : ${({theme})=> theme.itemBackground};
11   font-size:25px;
12   color : ${({theme})=> theme.text};
13 `;
14
15 const Input = () => {
16   return <StyledInput />;
17 };
18
19 export default Input;
```

app컴포넌트에서 작성된 input컴포넌트를 사용해보자


```

src > JS App.js > [0] Container
1  import React from "react";
2  import styled, {ThemeProvider} from "styled-components";
3  import {theme} from '../theme';
4  import { StatusBar } from "react-native";
5  import Input from "../components/Input";
6
7  const Container = styled.SafeAreaView`
8    flex : 1;
9    background-color : ${({theme}) => theme.background};
10   align-items : center;
11   justify-content : flex-start;
12 `;
13
14  const Title = styled.Text`
15    font-size : 40px;
16    font-weight : 600;
17    color : ${({theme}) => theme.main};
18    align-self : flex-start;
19    margin : 0px 20px;
20 `;
21
22  export default function App(){
23    return(
24      <ThemeProvider theme={theme}>
25        <Container>
26          <StatusBar barStyle="light-content" backgroundColor={theme.background} />
27          <Title>TODO List</Title>
28          <Input />
29        </Container>
30      </ThemeProvider>
31    );
32  }

```



Dimensions

input컴포넌트가 화면에 딱 차있어 답답한 느낌이 든다.

input컴포넌트의 양 옆에 20px씩 공백을 주려면 어떻게 해야할까

리액트 네이티브에서는 크기가 다양한 모바일 기기에 대응하기 위해 현재 화면의 크기를 알 수 있는

Dimensions와 useWindowDimensions를 제공한다.

- Dimensions : <https://reactnative.dev/docs/dimensions>
- useWindowDimensions : <https://reactnative.dev/docs/usewindowdimensions>

두 기능 모두 현재 기기의 화면 크기를 알 수 있고, 이를 이용해 다양한 크기의 기기에 동일한 모습으로 적용될 수 있도록 코드를 작성할 수 있다.

dimensions는 처음 값을 받아왔을 때의 크기로 고정되기 때문에 기기를 회전해서 화면이 전환되면 변화된 화면의 크기와 일치하지 않을 수 있다. 이런 상황을 위해 이벤트 리스너를 등록하여 화면의 크기변화에 대응할 수 있도록 기능을 제공하고 있다.

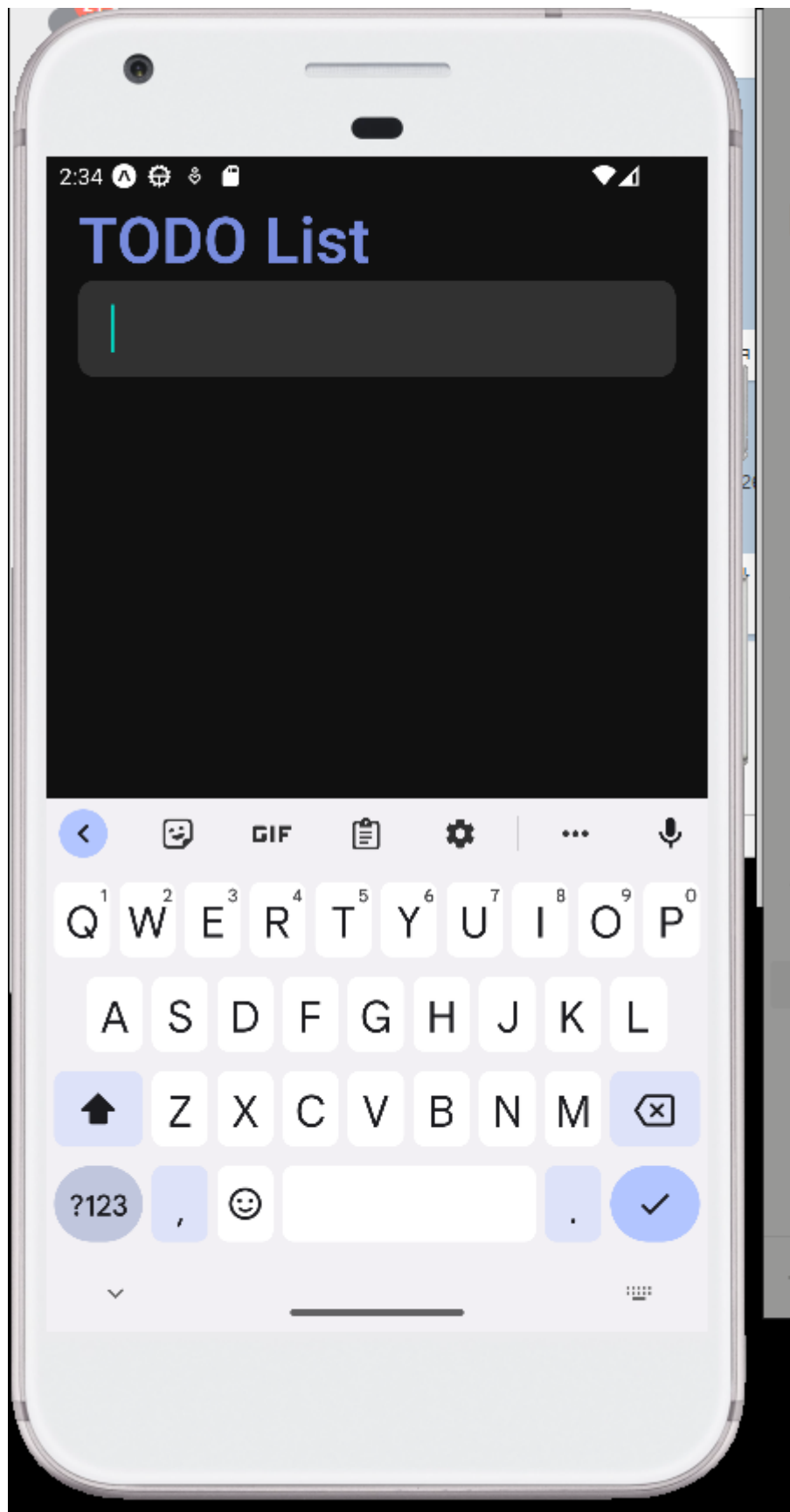
- useWindowDimensions는 리액트 네이티브에서 제공하는 hooks중 하나로, 화면의 크기가 변경되면 화면의 크기, 너비, 높이를 자동으로 업데이트합니다.

dimensions를 사용해 화면의 크기를 확인하고 스타일을 변경해보자.

```

v REACT-NATIVE-TODO
  > .expo
  > assets
  > node_modules
  v src
    v components
      JS Input.js
      JS App.js
      JS theme.js
      .gitignore
      JS App.js
      {} app.json
      Babel.config.js
      {} package-lock.json
      {} package.json
      yarn.lock

src > components > JS Input.js > [x] StyledInput
1  import React from "react";
2  import styled from "styled-components/native";
3  import { Dimensions } from "react-native";
4
5
6  const StyledInput = styled.TextInput`
7    width: ${({width}) => width - 40}px;
8    height : 60px;
9    margin 3px 0;
10   padding : 15px 20px;
11   border-radius : 10px;
12   background-color : ${({theme})=> theme.itemBackground};
13   font-size:25px;
14   color : ${({theme})=> theme.text};
15 `;
16
17 const Input = () => {
18   const width = Dimensions.get('window').width;
19   return <StyledInput width={width} />;
20 };
21
22 export default Input;
```



만약 useWindowDimensions를 사용한다면 다음과 같이 작성한다.

```
src > components > JS Input.js > [x] Input > [x] width
1  import React from "react";
2  import styled from "styled-components/native";
3  import { useWindowDimensions } from "react-native";
4
5
6  const StyledInput = styled.TextInput`
7    width: ${({width}) => width - 40}px;
8    height : 60px;
9    margin 3px 0;
10   padding : 15px 20px;
11   border-radius : 10px;
12   background-color : ${({theme})=> theme.itemBackground};
13   font-size:25px;
14   color : ${({theme})=> theme.text};
15 `;
16
17  const Input = () => {
18    const width = useWindowDimensions().width;
19    return <StyledInput width={width} />;
20  };
21
22  export default Input;
```

Input 컴포넌트

이번엔 Input 컴포넌트에 다양한 속성을 설정해보자

Placeholder에 적용할 문자열은 props로 받아 설정하고 placeholder의 색은 타이틀과 같은 색상으로 설정한다. 그리고 너무 긴 항목을 입력하지 못하도록 입력 가능한 글자의 수를 50자로 제한한다.

```
REACT-NATIVE-TODO
  > .expo
  > assets
  > node_modules
  > src
    > components
      JS Input.js
      JS App.js
      JS theme.js
      .gitignore
      app.json
      babel.config.js
      package-lock.json
      package.json
      yarn.lock

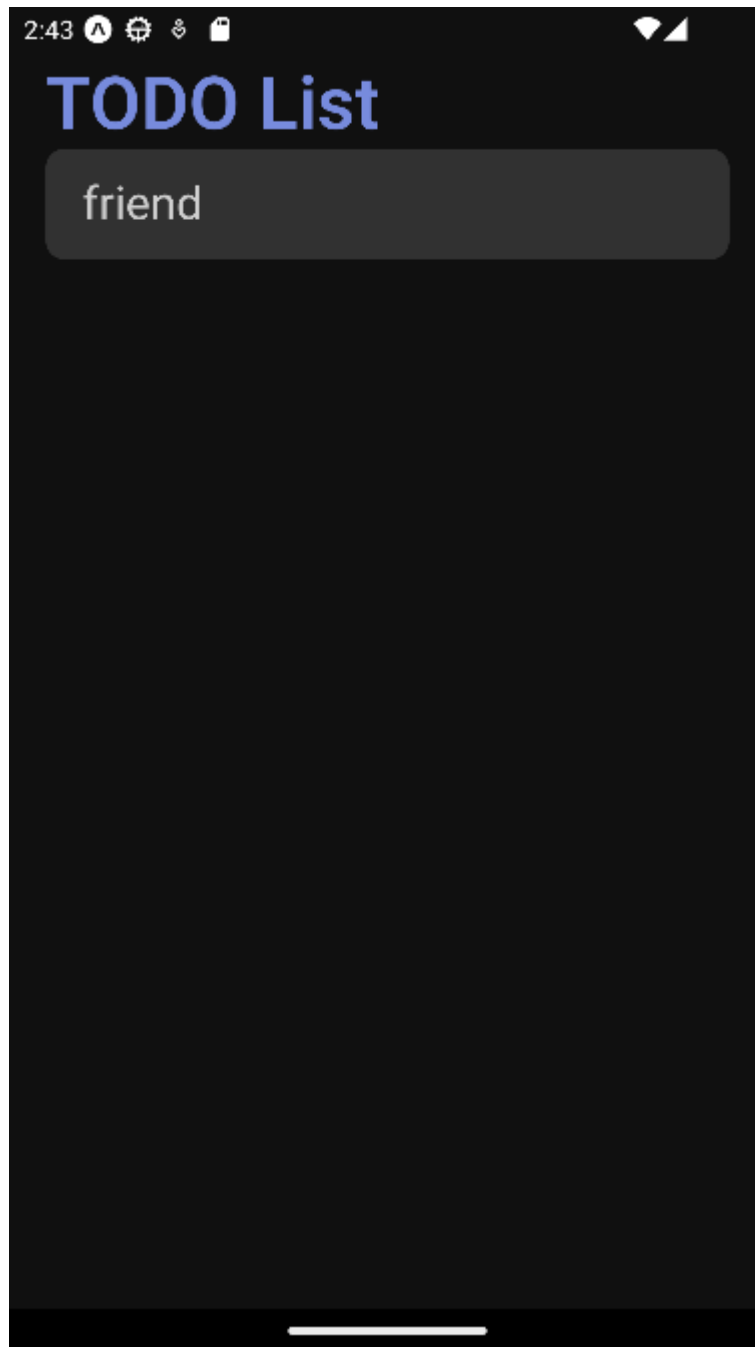
src > JS App.js > App
1  import React from "react";
2  import styled, {ThemeProvider} from "styled-components";
3  import {theme} from './theme';
4  import { StatusBar } from "react-native";
5  import Input from "./components/Input";
6
7  const Container = styled.SafeAreaView`
8    flex : 1;
9    background-color : ${({theme}) => theme.background};
10   align-items : center;
11   justify-content : flex-start;
12 `;
13
14  const Title = styled.Text`
15    font-size : 40px;
16    font-weight : 600;
17    color : ${({theme}) => theme.main};
18    align-self : flex-start;
19    margin : 0px 20px;
20 `;
21
22  export default function App(){
23    return(
24      <ThemeProvider theme={theme}>
25        <Container>
26          <StatusBar barStyle="light-content" backgroundColor={theme.background} />
27          <Title>TODO List</Title>
28          <Input placeholder="+ Add a Task" />
29        </Container>
30      </ThemeProvider>
31    );
32  }
```

app컴포넌트에서 input 컴포넌트에 placeholder를 전달하도록 수정했다.

이제 input 컴포넌트에서 props로 전달된 값을 이용하도록 수정하자

```
REACT-NATIVE-TODO
  > .expo
  > assets
  > node_modules
  > src
    > components
      JS Input.js
      JS App.js
      JS theme.js
      .gitignore
      app.json
      babel.config.js
      package-lock.json
      package.json
      yarn.lock

src > components > JS Input.js > Input > placeholder
1  import React from "react";
2  import styled from "styled-components/native";
3  import { Dimensions } from "react-native";
4
5
6  const StyledInput = styled.TextInput.attrs(({theme})=> ({
7    placeholderTextColor: theme.main,
8  })))`
9
10   width: ${({width}) => width - 40}px;
11   height : 60px;
12   margin 3px 0;
13   padding : 15px 20px;
14   border-radius : 10px;
15   background-color : ${({theme})=> theme.itemBackground};
16   font-size:25px;
17   color : ${({theme})=> theme.text};
18 `;
19
20  const Input = ({placeholder}) => {
21    const width = Dimensions.get('window').width;
22    return <StyledInput width={width} placeholder={placeholder} maxLength={50} />;
23  };
24
25  export default Input;
```



TextInput컴포넌트는 기본적으로 첫 글자가 대문자로 나타나고 오타 입력 시 자동으로 수정하는 기능이 켜져 있다. 그뿐 아니라 ios의 경우 키보드의 완료버튼이 return으로 되어 있다.

※ios에서 키보드가 보이지 않는 경우 단축키 “command + K”를 누르거나 “I/O → Keyboard”로 이동하여 Toggle Software Keyboard 를 클릭하면 키보드가 나타난다.

이번엔 TextInput 컴포넌트에서 제공하는 속성을 이용해 키보드의 설정을 변경해보자


```
src > components > JS Input.js > [🔍] Input
1  import React from "react";
2  import styled from "styled-components/native";
3  import { Dimensions } from "react-native";
4
5
6  const StyledInput = styled.TextInput.attrs(({theme})=> ({
7    placeholderTextColor: theme.main,
8  })))`
9    width: ${({width}) => width - 40}px;
10   height : 60px;
11   margin 3px 0;
12   padding : 15px 20px;
13   border-radius : 10px;
14   background-color : ${({theme})=> theme.itemBackground};
15   font-size:25px;
16   color : ${({theme})=> theme.text};
17 `;
18
19  const Input = ({placeholder}) => {
20    const width = Dimensions.get('window').width;
21    return <StyledInput
22      width={width}
23      placeholder={placeholder}
24      maxLength={50}
25      autoCapitalize="none"
26      autoCorrect={false}
27      returnKeyType="done"/>;
28  };
29
30  export default Input;
```

자동으로 대문자로 전환하는 `autoCapitalize` 속성을 `none`으로 변경해서 동작하지 않도록 하고,

자동 수정 기능도 `autoCorrect` 속성을 이용하여 사용하지 않도록 설정했다. 마지막으로 키보드의 완료 버튼을 설정하는 `returnKeyType`을 `done`으로 변경했다.

특정 플랫폼에만 적용되는 속성 중에는 아이폰의 키보드 색상을 변경하는 `keyboardAppearance`가 있다. 배경색을 어둡게 사용하고 있으므로 아이폰의 키보드 색상을 어둡게 설정해보자

```
const Input = ({placeholder}) => {
  const width = Dimensions.get('window').width;
  return (
    <StyledInput
      width={width}
      placeholder={placeholder}
      maxLength={50}
      autoCapitalize="none"
      autoComplete={false}
      keyboardType="done"
      keyboardAppearance="dark"
    />
  );
};
```

이벤트

입력되는 값을 이용할 수 있도록 input컴포넌트에 이벤트를 등록하자

```
> .expo
> assets
> node_modules
└─ src
   └─ components
      └─ JS Input.js
      └─ JS App.js
      └─ JS theme.js
      └─ .gitignore
      └─ JS App.js
      └─ {} app.json
      └─ {} babel.config.js
      └─ {} package-lock.json
      └─ {} package.json
      └─ {} yarn.lock

17 color : ${({theme}) => theme.main};
18 align-self : flex-start;
19 margin : 0px 20px;
20
21
22 export default function App(){
23   const [newTask, setNewTask] = useState('');
24
25   const _addTask = ()=>{
26     alert(`Add:${newTask}`);
27     setNewTask('');
28   };
29
30   const _handleTextChange = text => {
31     setNewTask(text);
32   };
33
34   return(
35     <ThemeProvider theme={theme}>
36       <Container>
37         <StatusBar barStyle="light-content" backgroundColor={theme.background} />
38         <Title>TODO List</Title>
39         <Input
40           placeholder="+ Add a Task"
41           onChangeText={_handleTextChange}
42           onSubmitEditing={_addTask}
43         />
44       </Container>
45     </ThemeProvider>
46   );
47 }
```

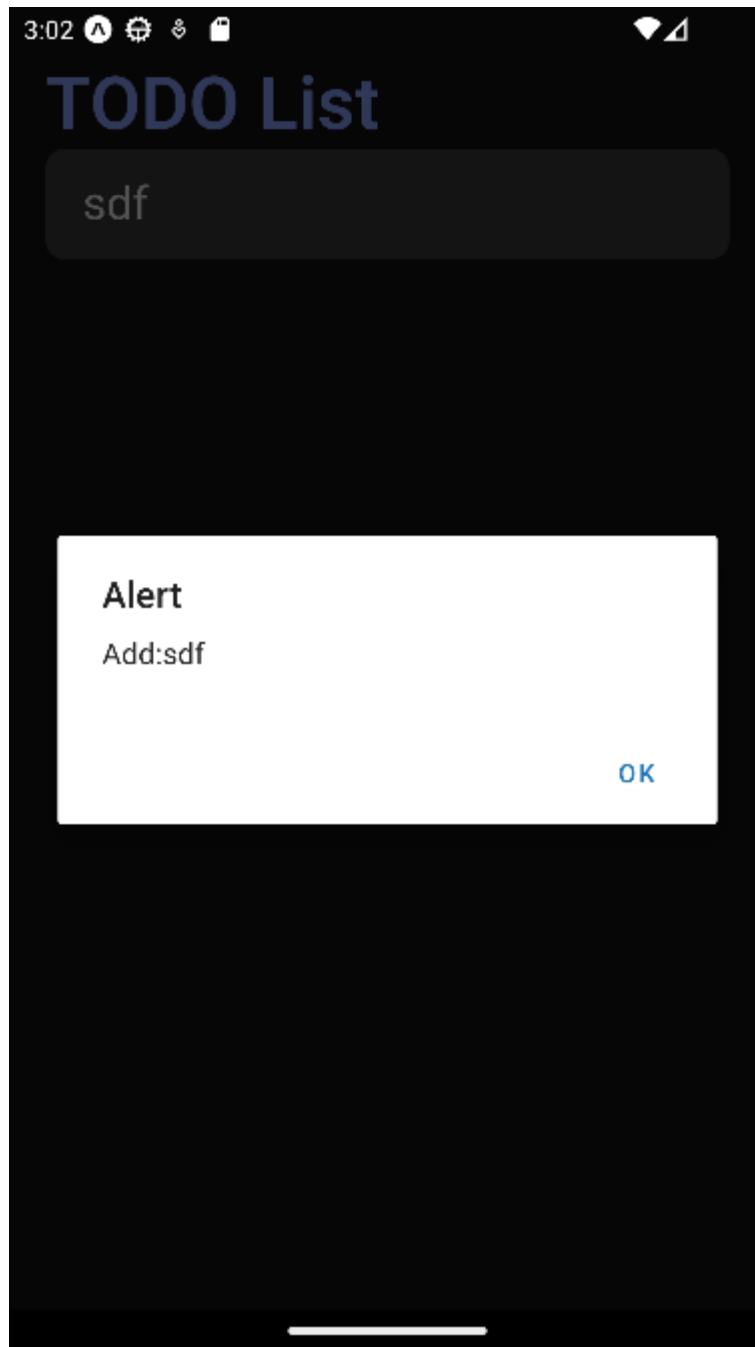
useState를 이용하여 newTask상태 변수와 세터 함수를 생성하고 input컴포넌트에서 값이 변할 때마다 newTask에 저장하도록 작성했다. 완료버튼을 누르면 입력된 내용을 확인하고

input컴포넌트를 초기화하도록 만들었다. 이제 input 컴포넌트에서 props로 전달된 값들을 이용해서 수정해보자.

```
> .expo
> assets
> node_modules
  src
    components
      JS Input.js
      JS App.js
      JS theme.js
      JS ...

1  import React from "react";
2  import styled from "styled-components/native";
3  import { Dimensions, ImageBackgroundComponent } from "react-native";
4  import PropTypes from "prop-types";
5
6
7  const StyledInput = styled.TextInput.attrs(({theme})=> ({
8    placeholderTextColor: theme.main,
9  })))`
10   width: ${({width}) => width - 40}px;
```

```
19
20  const Input = ({placeholder, value, onChangeText, onSubmitEditing}) => {
21    const width = Dimensions.get('window').width;
22    return (
23      <StyledInput
24        width={width}
25        placeholder={placeholder}
26        maxLength={50}
27        autoCapitalize="none"
28        autoCorrect={false}
29        returnKeyType="done"
30        keyboardAppearance="dark"
31        value={value}
32        onChangeText={onChangeText}
33        onSubmitEditing={onSubmitEditing}
34      />
35    );
36  };
37
38  Input.propTypes = {
39    placeholder : PropTypes.string,
40    value : PropTypes.string.isRequired,
41    onChangeText : PropTypes.func.isRequired,
42    onSubmitEditing : PropTypes.func.isRequired,
43  };
44
45  export default Input;
```



할 일 목록 만들기

input컴포넌트를 통해 입력 받은 내용을 목록으로 출력하는 기능을 만들자.

그림 5-13

할 일 목록을 만들기 위해서는 2개의 컴포넌트를 만들어야 한다.

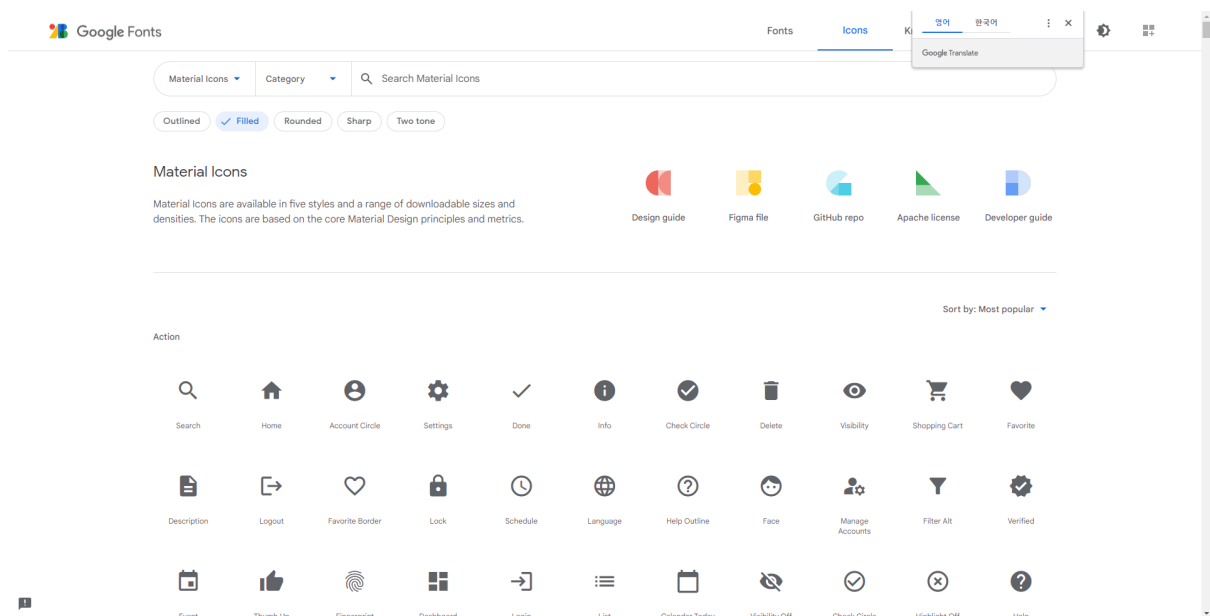
- iconButton 컴포넌트 : 완료, 수정, 삭제 버튼으로 사용할 컴포넌트
- Task 컴포넌트 : 목록의 각 항목으로 사용할 컴포넌트

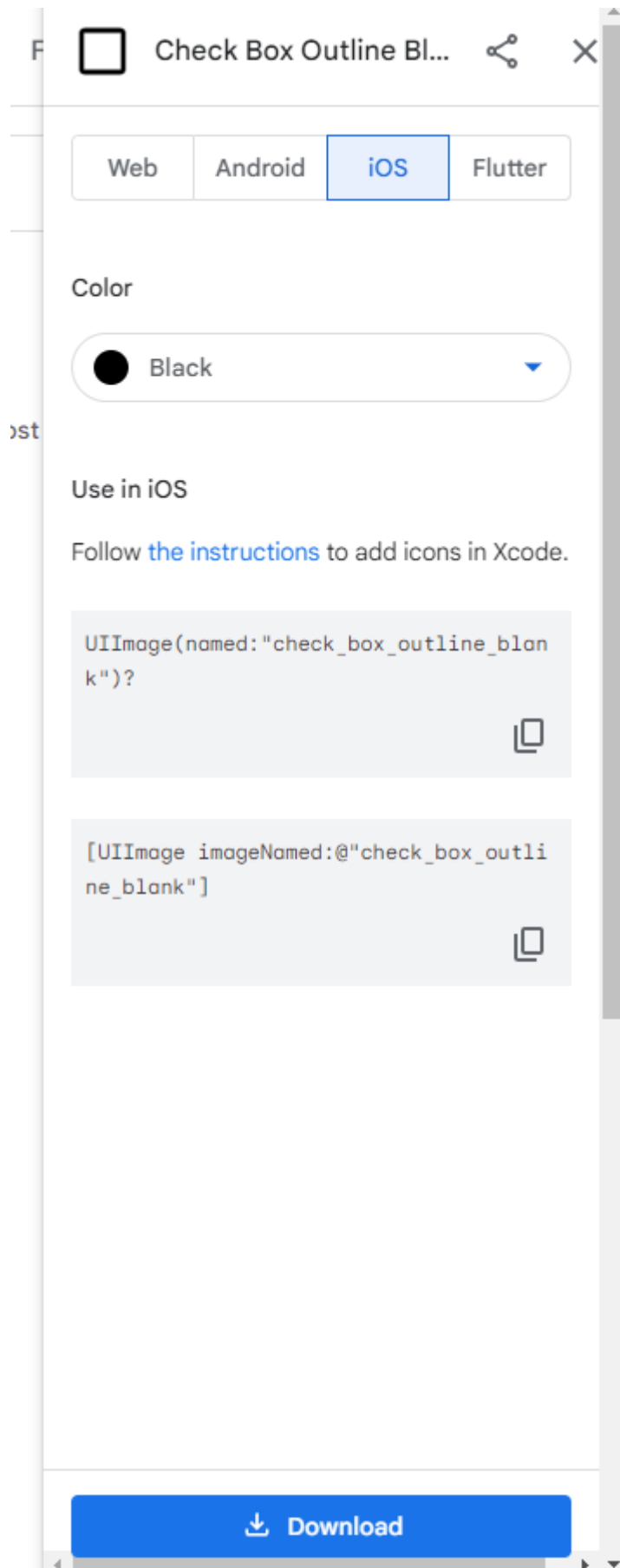
이미지 준비

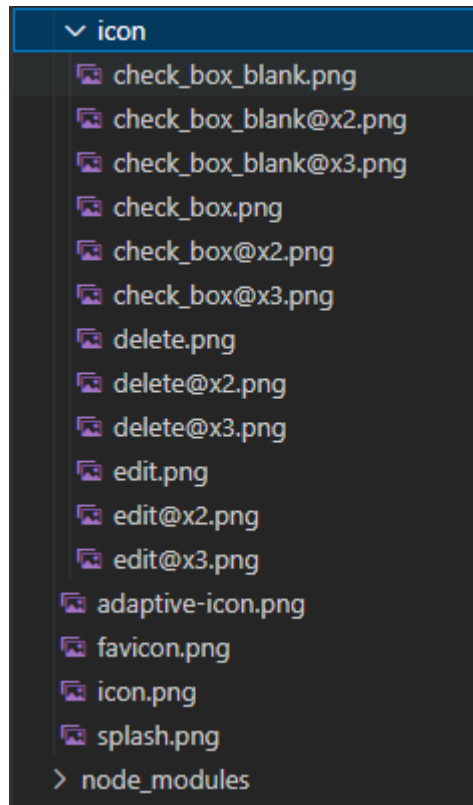
IconButton 컴포넌트를 만들게 전에 프로젝트에서 사용할 아이콘 이미지를 다운받자.

구글 머티리얼 디자인에서 ios용 아이콘을 다운로드 받자.

구글 머티리얼 디자인 아이콘 : <https://material.io/resources/icons/?style=baseline>







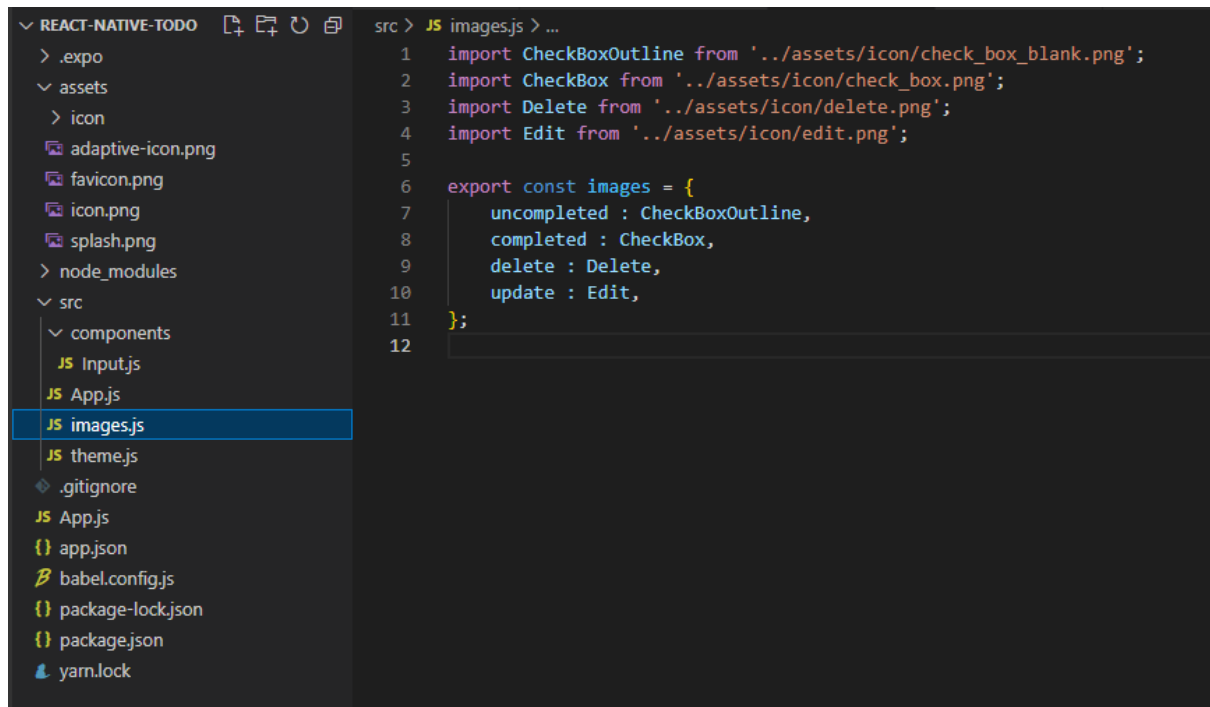
assets폴더 밑에 icons폴더를 생성해서 준비하자.

iconButton 컴포넌트

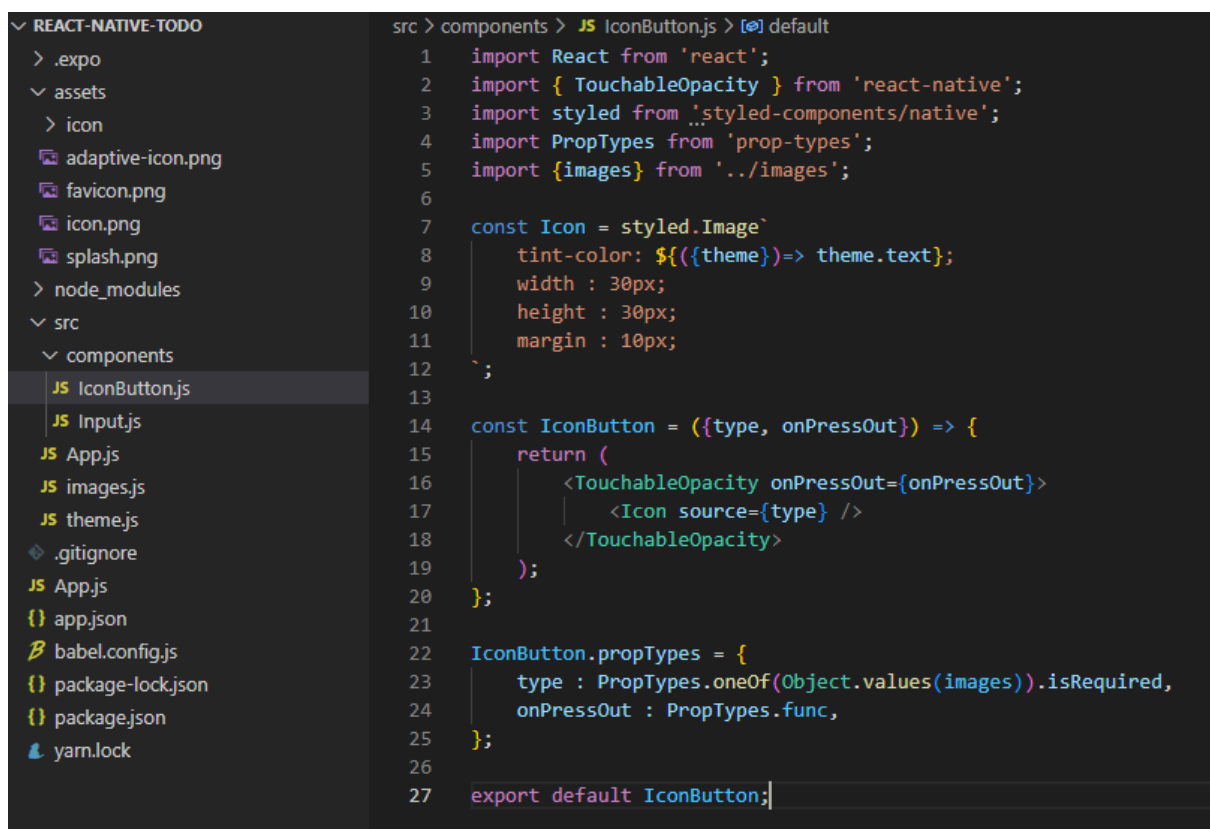
리액트 네이티브에서 제공하는 Image 컴포넌트는 프로젝트에 있는 이미지 파일의 경로나 URL을 이용하여 원격에 있는 이미지를 렌더링할 수 있다.

- Image컴포넌트 : <https://reactnative.dev/docs/image>

앞에서 준비한 아이콘의 경로를 이용해 Image 컴포넌트를 사용한다. 먼저 아이콘 이미지를 관리할 image.js를 src폴더 밑에 생성한다.



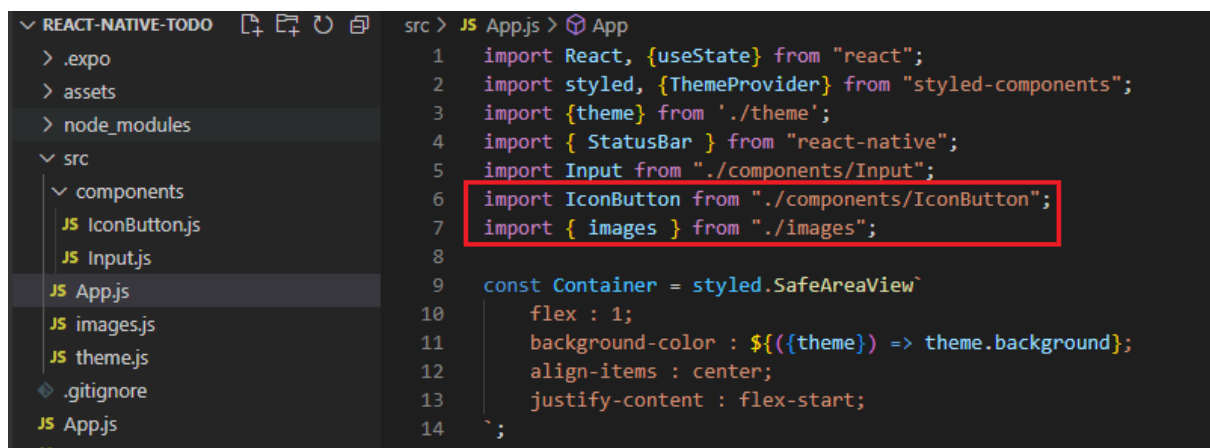
이제 components 폴더 안에 IconButton 컴포넌트를 만들어 보자.



이미지 종류별로 컴포넌트를 만들지 않고 IconButton 컴포넌트를 호출할 때 원하는 이미지의 종류를 props에 type으로 전달하도록 작성했고, 아이콘의 색은 입력되는 텍스트와 동일한 색을 사용하도록 스타일을 적용했다. 사람의 손가락이 버튼을 정확하게 클릭하지 못하는 경우가 많기 때문에, 사용자 편의를 위해 버튼 주변을 클릭해도 정확히 클릭된 것으로 인식하도록 일정 수준의 margin을 주어 여유 공간을 확보했다.

※ Pressable 컴포넌트를 사용할 수 있는 버전으로 진행한다면 HitRect를 설정해서 사용하자

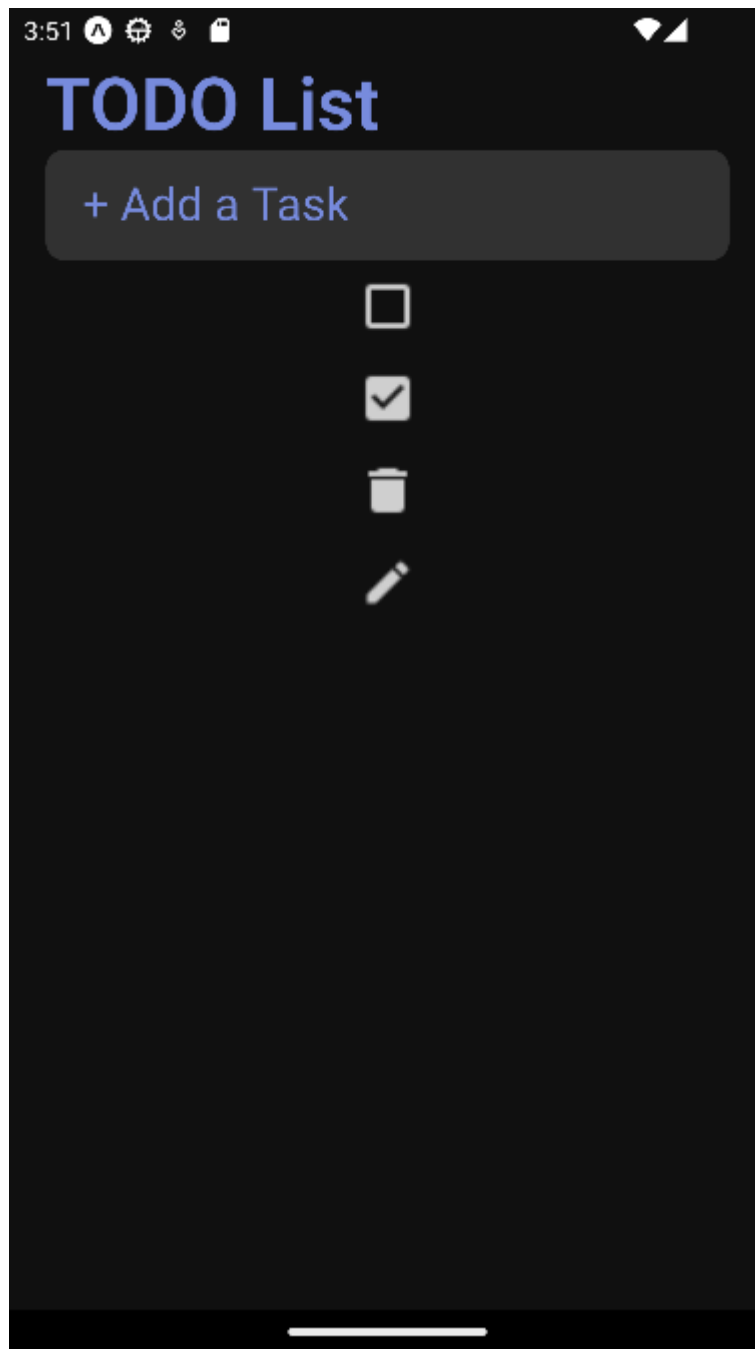
이제 app컴포넌트에 IconButton 컴포넌트를 적용해보자



```
src > JS App.js > App
1  import React, {useState} from "react";
2  import styled, {ThemeProvider} from "styled-components";
3  import {theme} from './theme';
4  import { StatusBar } from "react-native";
5  import Input from "../components/Input";
6  import IconButton from "../components/IconButton";
7  import { images } from "../images";
8
9  const Container = styled.SafeAreaView`
10    flex : 1;
11    background-color : ${({theme}) => theme.background};
12    align-items : center;
13    justify-content : flex-start;
14  `;
```



```
23
24 export default function App(){
25   const [newTask, setNewTask] = useState('');
26
27   const _addTask =()=>{
28     alert(`Add:${newTask}`);
29     setNewTask('');
30   };
31
32   const _handleTextChange = text => {
33     setNewTask(text);
34   };
35
36   return(
37     <ThemeProvider theme={theme}>
38       <Container>
39         <StatusBar barStyle="light-content" backgroundColor={theme.background} />
40         <Title>TODO List</Title>
41         <Input
42           placeholder="+ Add a Task"
43           onChangeText={_handleTextChange}
44           onSubmitEditing={_addTask}
45         />
46         <IconButton type={images.uncompleted} />
47         <IconButton type={images.completed} />
48         <IconButton type={images.delete} />
49         <IconButton type={images.update} />
50       </Container>
51     </ThemeProvider>
52   );
53 }
```



Task 컴포넌트

완성된 IconButton 컴포넌트를 이용해 Task컴포넌트를 만들어보자.

Task 컴포넌트는 완료 여부를 확인하는 버튼과 입력된 할 일 내용, 항목 삭제 버튼, 수정 버튼으로 구성된다.

```
src > components > JS Task.js > [⌕] Contents
1  import React from "react";
2  import styled from "styled-components/native";
3  import PropTypes from 'prop-types';
4  import IconButton from "../IconButton";
5  import {images} from '../images';
6
7  const Container = styled.View`
8    flex-direction : row;
9    align-items: center;
10   background-color: ${({theme}) => theme.itemBackground};
11   border-radius : 10px;
12   padding : 5px;
13   margin : 3px 0px;
14 `;
15
16 const Contents = styled.Text`
17   flex : 1;
18   font-size : 24px;
19   color : ${({theme}) => theme.text};
20 `;
21
22 const Task = ({text}) => {
23   return (
24     <Container>
25       <IconButton type={images.uncompleted} />
26       <Contents>{text}</Contents>
27       <IconButton type={images.update} />
28       <IconButton type={images.delete} />
29     </Container>
30   );
31 };
32
33 Task.propTypes = {
34   text : PropTypes.string.isRequired,
35 };
36
37 export default Task;
```

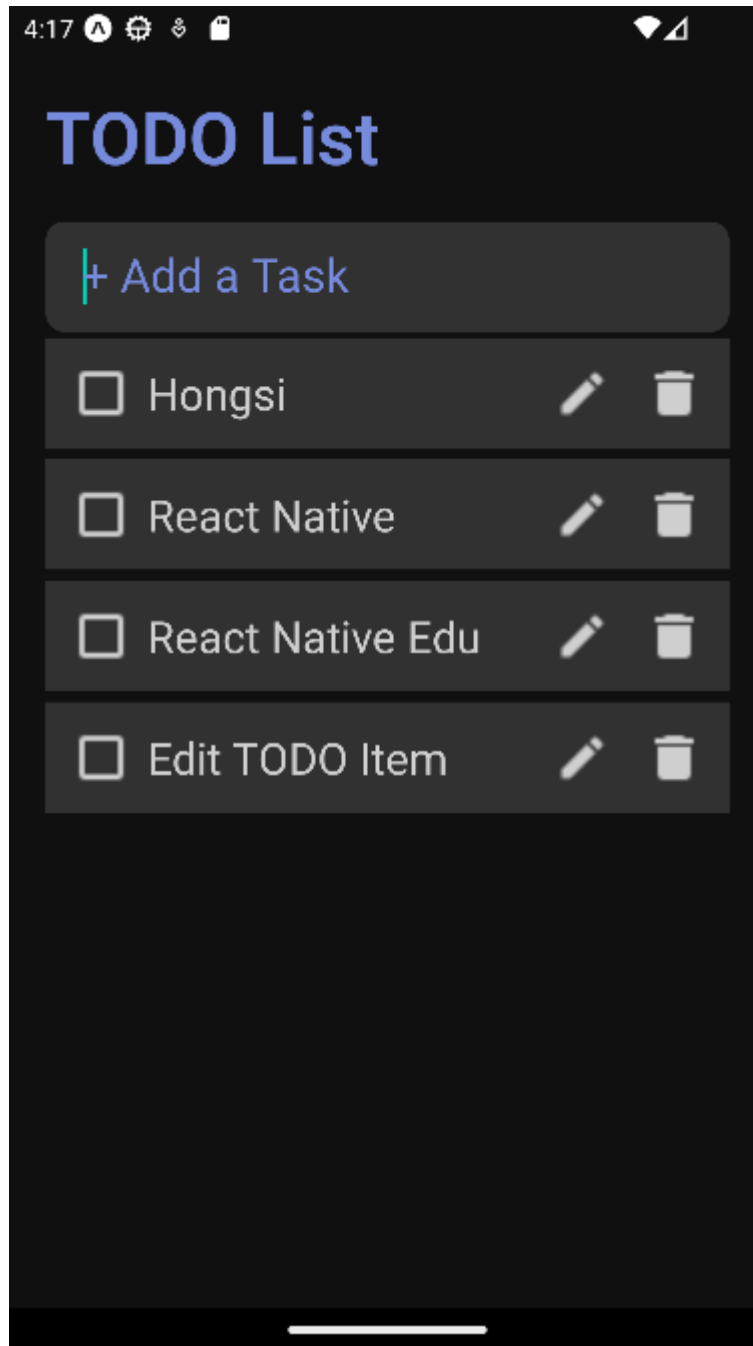
할 일 내용은 props로 전달되어 오는 값을 활용했고, 완료 여부를 나타내는 체크 박스와 수정, 삭제 버튼을 IconButton 컴포넌트를 이용해 만들었다. 이제 app 컴포넌트에서 Task 컴포넌트를 이용해 할 일 목록을 만들어보자

```
src > JS App.js > App
1  import React, {useState} from "react";
2  import styled, {ThemeProvider} from "styled-components";
3  import {theme} from './theme';
4  import { Dimensions, StatusBar } from "react-native";
5  import Input from "../components/Input";
6  import IconButton from "../components/IconButton";
7  import { images } from "../images";
8  import Task from "../components/Task";
9
10 const List = styled.ScrollView`
11   flex : 1;
12   width : ${({width})=> width - 40}px;
13 `
14
15 const Container = styled.SafeAreaView`
16   flex : 1;
17   background-color : ${({theme}) => theme.background};
18   align-items : center;
19   justify-content : flex-start;
20 `;
21
22 const Title = styled.Text`
23   font-size : 40px;
24   font-weight : 600;
25   color : ${({theme}) => theme.main};
26   align-self : flex-start;
27   margin : 20px;
```

```
src > JS App.js > App > _addTask
27   margin : 20px;
28
29
30 export default function App(){
31
32   const width = Dimensions.get('window').width;
33
34   const [newTask, setNewTask] = useState('');
35
36   const _addTask = ()=>{
37     alert(`Add:${newTask}`);
38     setNewTask('');
39   };
40
41   const _handleTextChange = text => {
42     setNewTask(text);
43   };
44
45   return(
46     <ThemeProvider theme={theme}>
47       <Container>
48         <StatusBar barStyle="light-content" backgroundColor={theme.background} />
49         <Title>TODO List</Title>
50         <Input
51           placeholder="+ Add a Task"
52           onChangeText={_handleTextChange}
53           onSubmitEditing={_addTask}
54         />
55         <List width={width}>
56           <Task text="Hongsi" />
57           <Task text="React Native" />
58           <Task text="React Native Edu" />
59           <Task text="Edit TODO Item" />
60         </List>
61       </Container>
62     </ThemeProvider>
63   );
64 }
```

리액트 네이티브에서 제공하는 ScrollView 컴포넌트를 이용해, 할 일 항목의 수가 많아져서 화면을 넘어가도 스크롤을 이용 할 수 있도록 화면을 구성했다.

Input 컴포넌트와 마찬가지로 다양한 크기의 화면에서 양쪽에 동일한 공백을 유지하기 위해 Dimensions로 화면의 너비를 구한 후 스타일에 사용했다.



기능 구현하기

추가, 삭제, 완료, 수정을 구현한다.

추가기능

할 일 항목은 내용, 완료 여부를 갖고 있어야 하고 항목을 구분할 수 있는 고유한 값도 갖고 있어야 하기에 다음과 같이 구성한다.

예시코드

```
{
  "ID1" : { "id" : "ID1", "text" : "text" , "completed" : false},
  "ID2" : { "id" : "ID2", "text" : "text" , "completed" : true},
  ...
}
```

```

export default function App() {
  const width = Dimensions.get('window').width;
  const [newTask, setNewTask] = useState('');
  const [tasks, setTasks] = useState({
    '1': { id: '1', text: 'Hongsi', completed: false },
    '2': { id: '2', text: 'React Native', completed: true },
    '3': { id: '3', text: 'React Native Edu', completed: false },
    '4': { id: '4', text: 'Edit TODO Item', completed: true },
  });

  const _addTask = () => {
    alert(`Add: ${newTask}`);
    setNewTask('');
  };

  const _handleTextChange = text => {
    setNewTask(text);
  };

  return (
    <ThemeProvider theme={theme}>
      <Container>
        <StatusBar barStyle="light-content" backgroundColor={theme.background} />
        <Title>TODO List</Title>
        <Input
          placeholder="+ Add a Task"
          onChangeText={_handleTextChange}
          onSubmitEditing={_addTask}
        />
        <List width={width}>
          {Object.values(tasks)
            .reverse()
            .map(item => {
              <Task text={item.text} />
            })}
        </List>
      </Container>
    </ThemeProvider>
  );
}

```

useState를 이용해 할 일 목록을 저장하고 관리할 tasks 변수를 생성한 후 초기값으로 임의의 내용을 입력했다. 최신 항목이 가장 앞에 보이도록 tasks를 역순으로 렌더링되게 작성했다.

Warning : each child in a list should have a unique "key" prop.

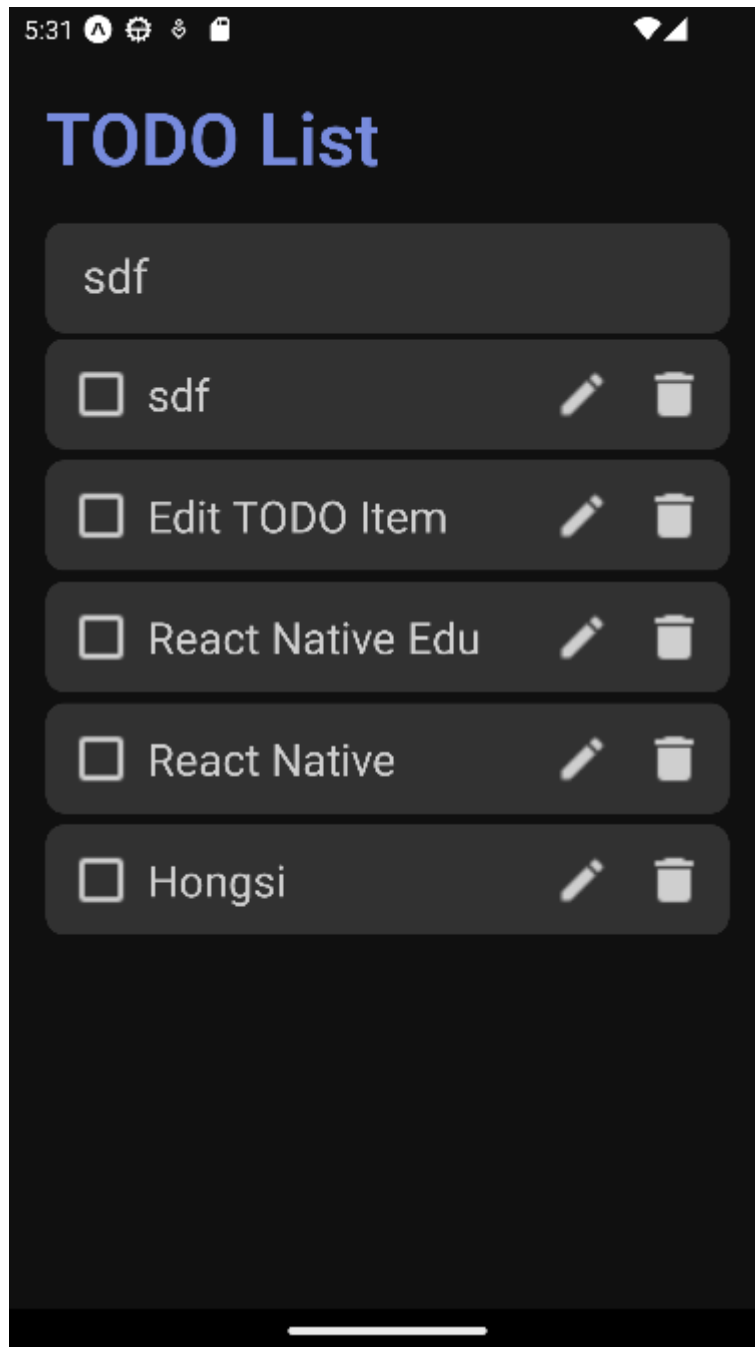
책에서는 출력은 되지만 해당 에러가 발생한다고 하지만, 나는 아무런 에러 없이 잘 작동이 되었다.

Key는 리액트에서 컴포넌트 배열을 렌더링 했을 때 어떤 아이템이 추가, 수정, 삭제되었는지 식별하는 것을 돕는 고유값으로 리액트에서 특별하게 관리되며 자식 컴포넌트의 props로 전

달되지 않는다. 이 경고 메시지는 key를 지정하지 않아서 나타나는 경고 메시지 이다. 우리는 각 항목마다 고유한 id를 갖도록 설계했으므로 id를 key로 지정하겠다.

_addTask 함수가 호출되면 tasks에 새로운 할 일이 추가되도록 수정하자.

```
export default function App(){
  const width = Dimensions.get('window').width;
  const [newTask, setNewTask] = useState('');
  const [tasks, setTasks] = useState({
    '1': { id: '1', text: 'Hongsi', completed: false},
    '2': { id: '2', text: 'React Native', completed: true},
    '3': { id: '3', text: 'React Native Edu', completed: false},
    '4': { id: '4', text: 'Edit TODO Item', completed: true},
  });
  const _addTask = ()=>{
    const ID = Date.now().toString();
    const newTaskObject = {
      [ID] : { id: ID , text: newTask, completed: false},
    };
    setNewTask('');
    setTasks({ ...tasks, ...newTaskObject});
  };
  const _handleTextChange = text => {
    setNewTask(text);
  };
  return(
    <ThemeProvider theme={theme}>
      <Container>
        <StatusBar barStyle="light-content" backgroundColor={theme.background} />
        <Title>TODO List</Title>
        <Input
          placeholder="+ Add a Task"
          onChangeText={_handleTextChange}
          onSubmitEditing={_addTask}
        />
        <List width={width}>
          {Object.values(tasks)
            .reverse()
            .map(item => <Task key={item.id} text={item.text} />)}
        </List>
      </Container>
    </ThemeProvider>
  );
}
```

id는 할 일 항목이 추가되는 시간의 타임스탬프를 이용하고 내용을 나타내는 text는 input 컴포넌트에 입력된 값을 지정한다. 새로 입력되는 항목이므로 완료 여부를 나타내는 completed는 항상 false가 된다. 마지막으로 newTask의 값을 빈 문자열로 지정해서 input 컴포넌트를 초기화하고, 기존의 목록을 유지한 상태에서 새로운 항목이 추가되도록 구성했다.

삭제 기능

```
const _addTask = () => {
  const ID = Date.now().toString();
  const newTaskObject = {
    [ID]: { id: ID, text: newTask, completed: false },
  };
  setNewTask('');
  setTasks({ ...tasks, ...newTaskObject });
};

const _deleteTask = id => {
  const currentTasks = Object.assign({}, tasks);
  delete currentTasks[id];
  setTasks(currentTasks);
};

const handleTextChange = text => {
  setNewTask(text);
};

return (
  <ThemeProvider theme={theme}>
    <Container>
      <StatusBar barStyle="light-content" backgroundColor={theme.background} />
      <Title>TODO List</Title>
      <Input
        placeholder="+ Add a Task"
        onChangeText={_handleTextChange}
        onSubmitEditing={_addTask}
      />
      <List width={width}>
        {Object.values(tasks)
          .reverse()
          .map(item => (
            <Task key={item.id} text={item.text} deleteTask={_deleteTask} />
          ))}
      </List>
    </Container>
  </ThemeProvider>
);
}
```

삭제 버튼을 클릭했을 때 항목의 id를 이용하여 tasks에서 해당 항목을 삭제하는 `_deleteTask` 함수를 작성했다. Task 컴포넌트에 생성된 항목 삭제 함수와 함께 항목 내용 전체를 전달해 자식 컴포넌트에서도 항목의 id를 확인할 수 있도록 수정했다. 이제 Task 컴포넌트를 전달받은 내용이 사용되도록 수정하자.

```

const Task = ({item, deleteTask}) => {
  return (
    <Container>
      <IconButton type={images.uncompleted} />
      <Contents>{item.text}</Contents>
      <IconButton type={images.update} />
      <IconButton type={images.delete} id={item.id} onPressOut={deleteTask}/>
    </Container>
  );
};

Task.propTypes = {
  item : PropTypes.object.isRequired,
  deleteTask : PropTypes.func.isRequired,
};

export default Task;

```

PropTypes를 전달되는 props에 맞게 수정했다.

props로 전달된 deleteTask 함수는 삭제 버튼으로 전달했고, 함수에서 필요한 항목의 id도 함께 전달했다. 이제 IconButton 컴포넌트에서 전달된 함수를 이용하도록 수정하자

```

import React from 'react';
import { TouchableOpacity } from 'react-native';
import styled from 'styled-components/native';
import PropTypes from 'prop-types';
import {images} from '../images';

const Icon = styled.Image`
  tint-color: ${({theme})=> theme.text};
  width : 30px;
  height : 30px;
  margin : 10px;
`;

const IconButton = ({type, onPressOut, id}) => {
  const _onPressOut = ()=> {
    onPressOut(id);
  };

  return (
    <TouchableOpacity onPressOut={_onPressOut}>
      <Icon source={type} />
    </TouchableOpacity>
  );
};

IconButton.defaultProps = {
  onPressOut: ()=>{},
};

IconButton.propTypes = {
  type : PropTypes.oneOf(Object.values(images)).isRequired,
  onPressOut : PropTypes.func,
  id: PropTypes.string
};

export default IconButton;

```

props로 onPressOut이 전달되지 않았을 경우에도 문제가 발생하지 않도록 defaultProps를 이용해 onPressOut의 기본값을 지정했다. props로 전달되는 값들의 propTypes를 지정하고 IconButton컴포넌트가 클릭되었을 때 전달된 함수가 호출되도록 작성했다.

완료 기능

항목을 완료 상태로 만들어도 다시 미완료 상태로 돌아올 수 있도록 완료 버튼을 만들어보자

```
const _toggleTask = id => {
  const currentTasks = Object.assign({}, tasks);
  currentTasks[id]['completed'] = !currentTasks[id]['completed'];
  setTasks(currentTasks);
};

const _handleTextChange = text => {
  setNewTask(text);
};

return(
  <ThemeProvider theme={theme}>
    <Container>
      <StatusBar barStyle="light-content" backgroundColor={theme.background} />
      <Title>TODO List</Title>
      <Input
        placeholder="+ Add a Task" |
        onChangeText={_handleTextChange}
        onSubmitEditing={_addTask}
      />
      <List width={width}>
        {Object.values(tasks)
          .reverse()
          .map(item => (
            <Task
              key={item.id}
              item={item}
              deleteTask={_deleteTask}
              toggleTask={_toggleTask}
            />
          ))}
      </List>
    </Container>
  </ThemeProvider>
)
```

함수가 호출될 때마다 완료 여부를 나타내는 completed값이 전환되는 함수를 작성했다. 삭제 기능과 마찬가지로 작성된 함수를 task 컴포넌트로 전달했다. 이제 task 컴포넌트를 수정해서 완료 기능을 완성해보자

```

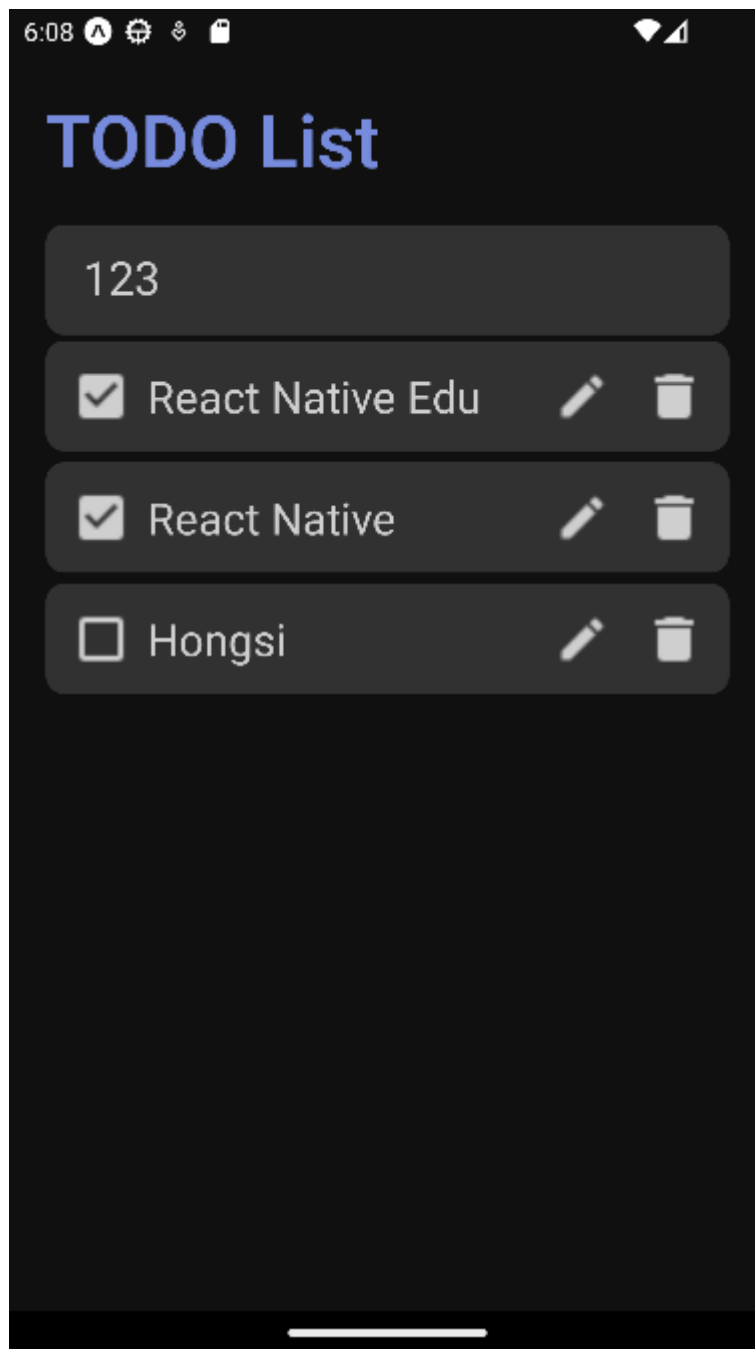
const Task = ({item, deleteTask, toggleTask}) => {
  return (
    <Container>
      <IconButton
        type={item.completed ? images.completed : images.uncompleted}
        id = {item.id}
        onPressOut={toggleTask}
      />
      <Contents>{item.text}</Contents>
      <IconButton type={images.update} />
      <IconButton type={images.delete} id={item.id} onPressOut={deleteTask}/>
    </Container>
  );
};

Task.propTypes = {
  item : PropTypes.object.isRequired,
  deleteTask : PropTypes.func.isRequired,
  toggleTask : PropTypes.func.isRequired,
};

export default Task;

```

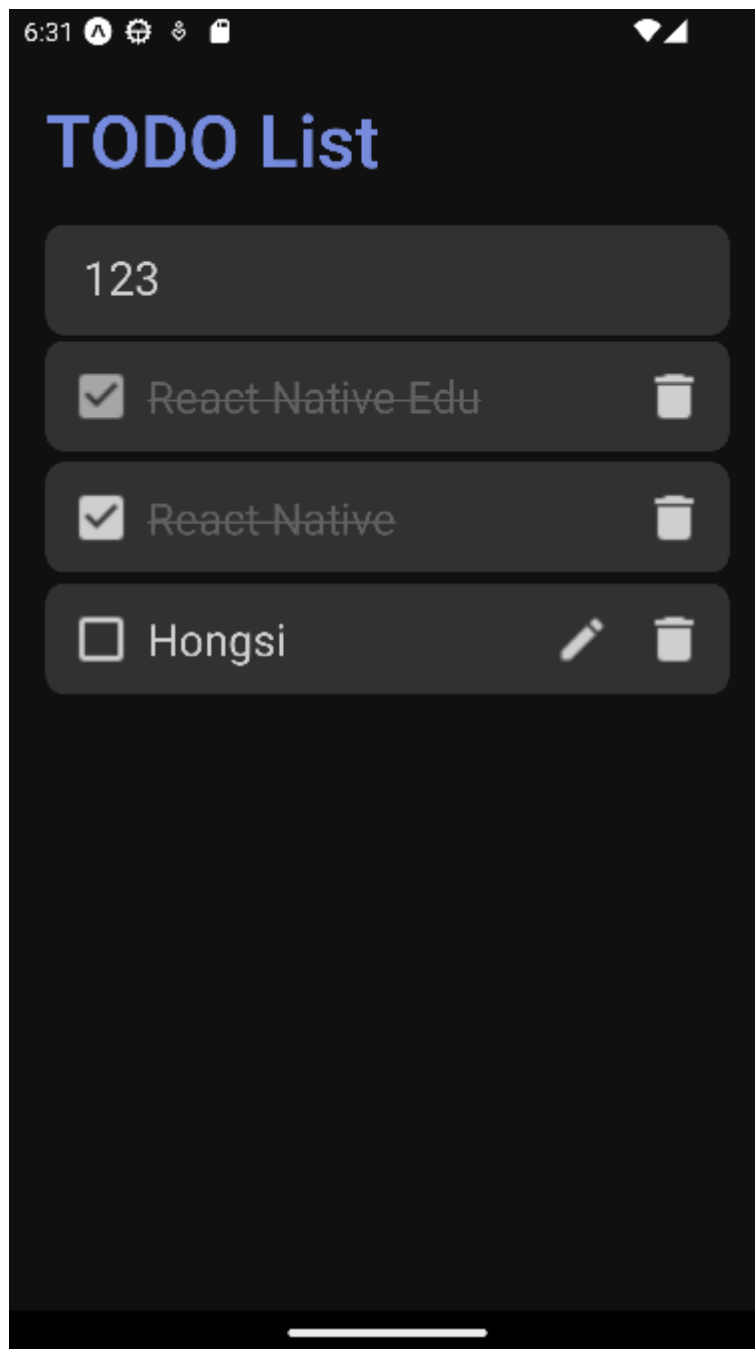
props 로 전달된 toggleTask 함수를 완료 상태를 나타내는 버튼의 onPressOut 으로 설정 하고, 항목 완료 여부에 따라 버튼 이미지가 다르게 나타나도록 수정했다.



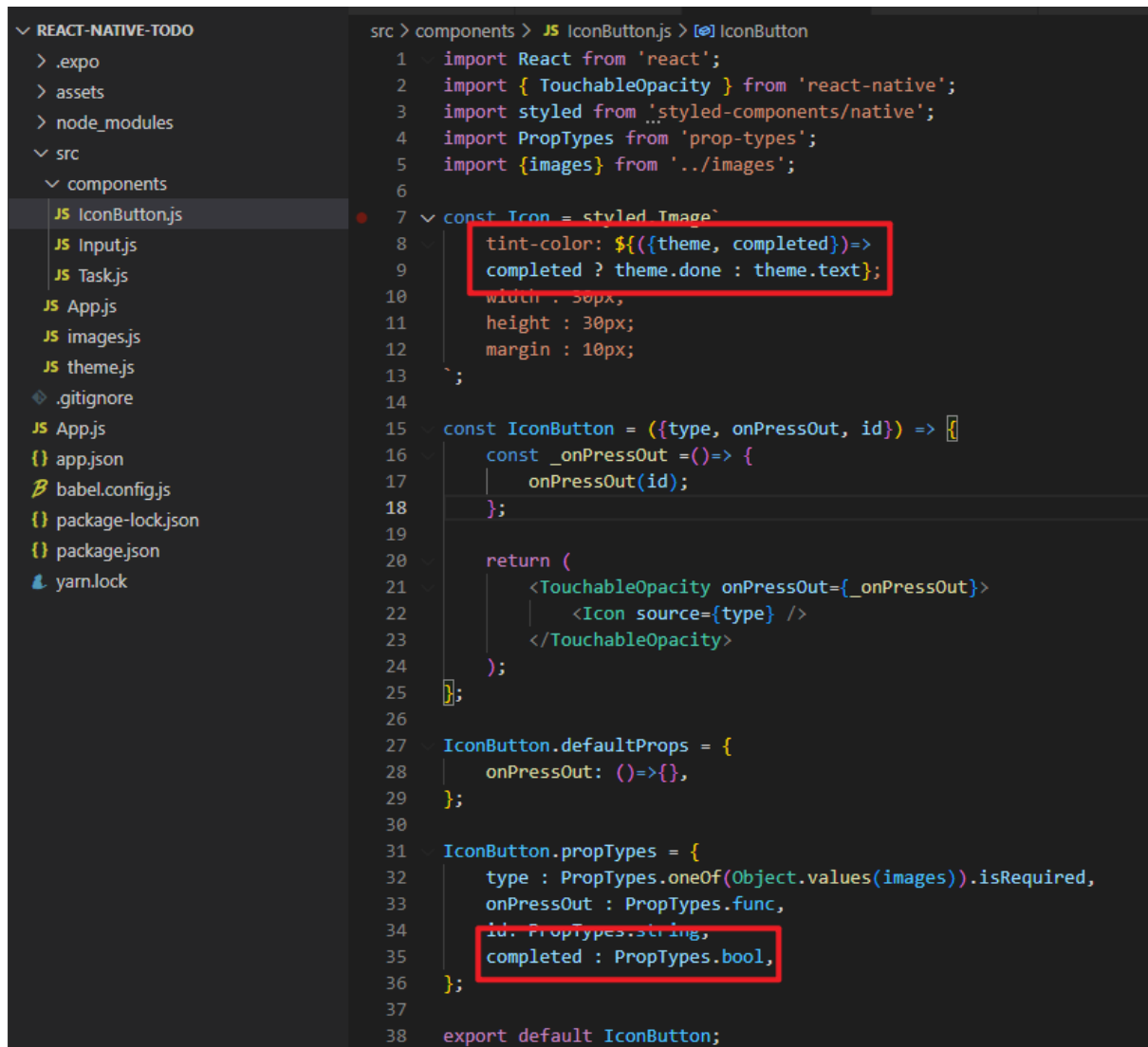
완료된 항목은 아이콘 이미지만 변경되는 것이 아니라, 수정 기능을 사용하지 않도록 수정 버튼이 나타나지 않게 Task 컴포넌트를 수정한다. 추가적으로 미완료 항목과 조금 더 명확하게 구분되도록 디자인도 수정한다.

```
> .expo
> assets
> node_modules
  ✓ src
    ✓ components
      JS IconButton.js
      JS Input.js
      JS Task.js
      JS App.js
      JS images.js
      JS theme.js
    .gitignore
  JS App.js
  {} app.json
  {} babel.config.js
  {} package-lock.json
  {} package.json
  yarn.lock

14 `;
15
16 const Contents = styled.Text`
17   flex : 1;
18   font-size : 24px;
19   color : ${({theme, completed}) => (completed ? theme.done : theme.text)};
20   text-decoration-line : ${({completed})=> completed ? 'line-through' : 'none'};
21 `;
22
23 const Task = ({item, deleteTask, toggleTask}) => {
24   return (
25     <Container>
26       <IconButton
27         type={item.completed ? images.completed : images.uncompleted}
28         id = {item.id}
29         onPressOut={toggleTask}
30         completed={item.completed}
31       />
32       <Contents completed={item.completed}>{item.text}</Contents>
33       {item.completed || <IconButton type={images.update} />}
34       <IconButton
35         type={images.delete}
36         id={item.id}
37         onPressOut={deleteTask}
38         completed={item.completed}/>
39     </Container>
40   );
41 };
42
43 Task.propTypes = {
44   item : PropTypes.object.isRequired,
45   deleteTask : PropTypes.func.isRequired,
46   toggleTask : PropTypes.func.isRequired,
47 };
48
49 export default Task;
```

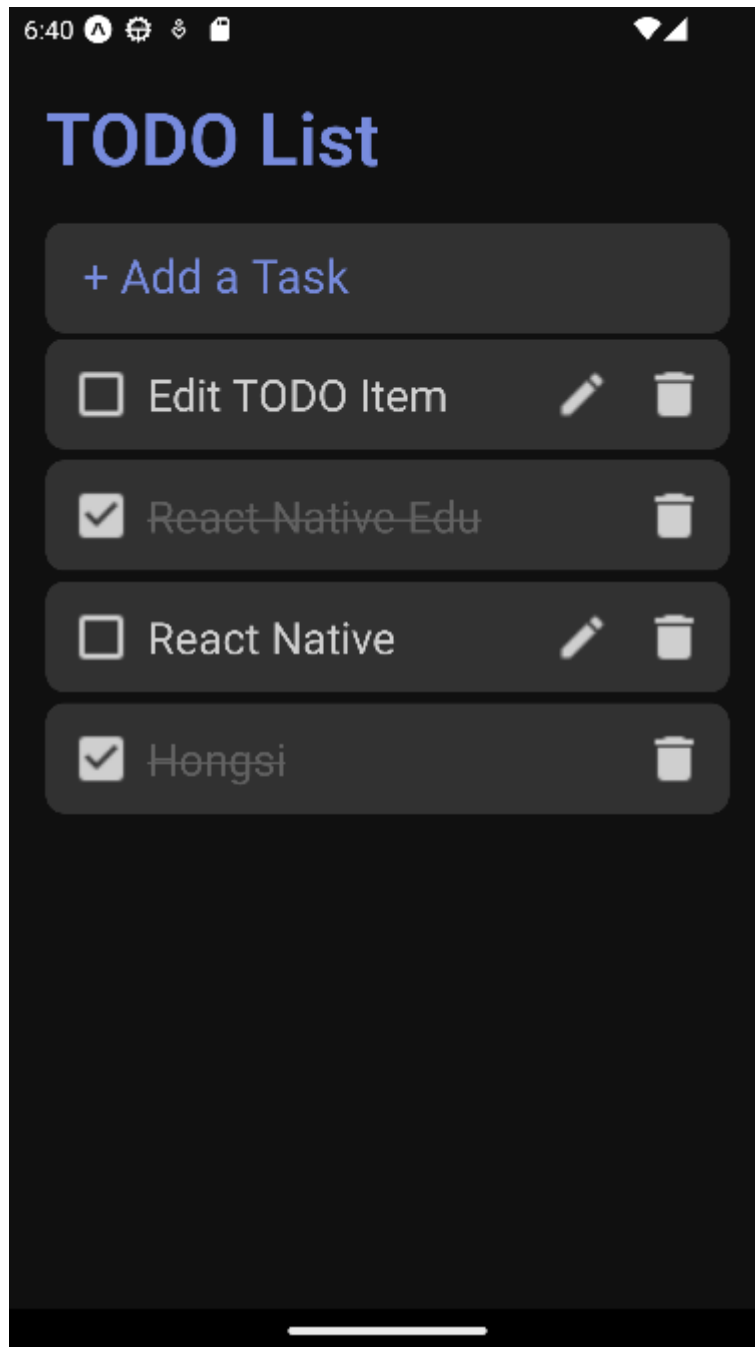



항목의 completed 값에 따라 수정 버튼이 렌더링 되지 않도록 수정했다. Contents 컴포넌트에 completed를 전달한 후 완료 여부에 따라 취소선이 나타나고 글자 색이 변경되도록 스타일이 변경되었고, 아이콘 버튼에도 completed를 전달했다. 이제 IconButton 컴포넌트에서 completed의 값에 따라 다른 스타일이 적용되도록 수정하자.



```
src > components > JS IconButton.js > IconButton
1  import React from 'react';
2  import { TouchableOpacity } from 'react-native';
3  import styled from 'styled-components/native';
4  import PropTypes from 'prop-types';
5  import {images} from '../images';
6
7  const Icon = styled.Image`
8    tint-color: ${({theme, completed})=>
9      completed ? theme.done : theme.text};
10    width : 30px;
11    height : 30px;
12    margin : 10px;
13 `;
14
15  const IconButton = ({type, onPressOut, id}) => {
16    const _onPressOut = ()=> {
17      onPressOut(id);
18    };
19
20    return (
21      <TouchableOpacity onPressOut={_onPressOut}>
22        <Icon source={type} />
23      </TouchableOpacity>
24    );
25  };
26
27  IconButton.defaultProps = {
28    onPressOut: ()=>{},
29  };
30
31  IconButton.propTypes = {
32    type : PropTypes.oneOf(Object.values(images)).isRequired,
33    onPressOut : PropTypes.func,
34    id : PropTypes.string,
35    completed : PropTypes.bool,
36  };
37
38  export default IconButton;
```

IconButton 컴포넌트에서도 항목 완료 여부에 따라 아이콘 이미지의 색이 변경되도록 수정했다.



수정기능

수정 버튼을 클릭하면 해당 항목이 Input 컴포넌트로 변경되면서 내용을 수정할 수 있도록 구현하자.

app컴포넌트에서 수정 완료된 항목이 전달되면 tasks에서 해당 항목을 변경하는 함수를 작성해보자.

```
src > JS App.js > App
29
30
31 export default function App(){
32   const width = Dimensions.get('window').width;
33   const [newTask, setNewTask] = useState('');
34   const [tasks, setTasks] = useState({ ...
35   });
36   const _addTask = ()=>{ ...
37   };
38   const _deleteTask = id => { ...
39   };
40   const _toggleTask = id => { ...
41   };
42   const _updateTask = item => {
43     const currentTasks = Object.assign({}, tasks);
44     currentTasks[item.id] = item;
45     setTasks(currentTasks);
46   };
47   const _handleTextChange = text => { ...
48   };
49   return(
50     <ThemeProvider theme={theme}>
51       <Container>
52         <StatusBar barStyle="light-content" bac
53         <Title>TODO List</Title>
54         <Input
55           placeholder="+ Add a Task"
56           onChangeText={_handleTextChange}
57           onSubmitEditing={_addTask}
58         />
59         <List width={width}>
60           {Object.values(tasks)
61             .reverse()
62             .map(item => (
63               <Task
64                 key={item.id}
65                 item={item}
66                 deleteTask={_deleteTask}
67                 toggleTask={_toggleTask}
68                 update={_updateTask}
69             )
68         />
69       )
70     )
71   )
72 }
```

수정된 항목이 전달되면 할 일 목록에서 해당 항목을 수정하는 `_updateTask` 함수를 작성하고 Task 컴포넌트에서 사용할 수 있도록 함수를 전달했다. 이제 task 컴포넌트에서 수정 버튼을 클릭하면 항목의 현재 내용을 가진 Input 컴포넌트가 렌더링 되어 사용자가 수정할 수 있도록 만들자.

```

import React, {useState} from "react";
import styled from "styled-components/native";
import PropTypes from 'prop-types';
import IconButton from "../IconButton";
import {images} from "../images";
import Input from "../Input";

const Container = styled.View` ...
`;
const Contents = styled.Text` ...
`;

const Task = ({item, deleteTask, toggleTask, updateTask}) => {
  const [isEditing, setIsEditing] = useState(false);
  const [text, setText] = useState(item.text);
  const _handleUpdateButtonPress = () =>{
    setIsEditing(true);
  };
  const _onSubmitEditing =()=> {
    if (isEditing){
      const editedTask = Object.assign({}, item, {text});
      setIsEditing(false);
      updateTask(editedTask);
    }
  };
};

```

```

return isEditing ? (
  <Input
    value={text}
    onChangeText={text=>setText(text)}
    onSubmitEditing={_onSubmitEditing}
  /> ) : (
  <Container>
    <IconButton
      type={item.completed ? images.completed : images.uncompleted}
      id = {item.id}
      onPressOut={toggleTask}
      completed={item.completed}
    />
    <Contents completed={item.completed}>{item.text}</Contents>
    {item.completed || (<IconButton type={images.update} onPressOut={_handleUpdateButtonPress} />)}
    <IconButton
      type={images.delete}
      id={item.id}
      onPressOut={deleteTask}
      completed={item.completed}/>
  </Container>
);
};

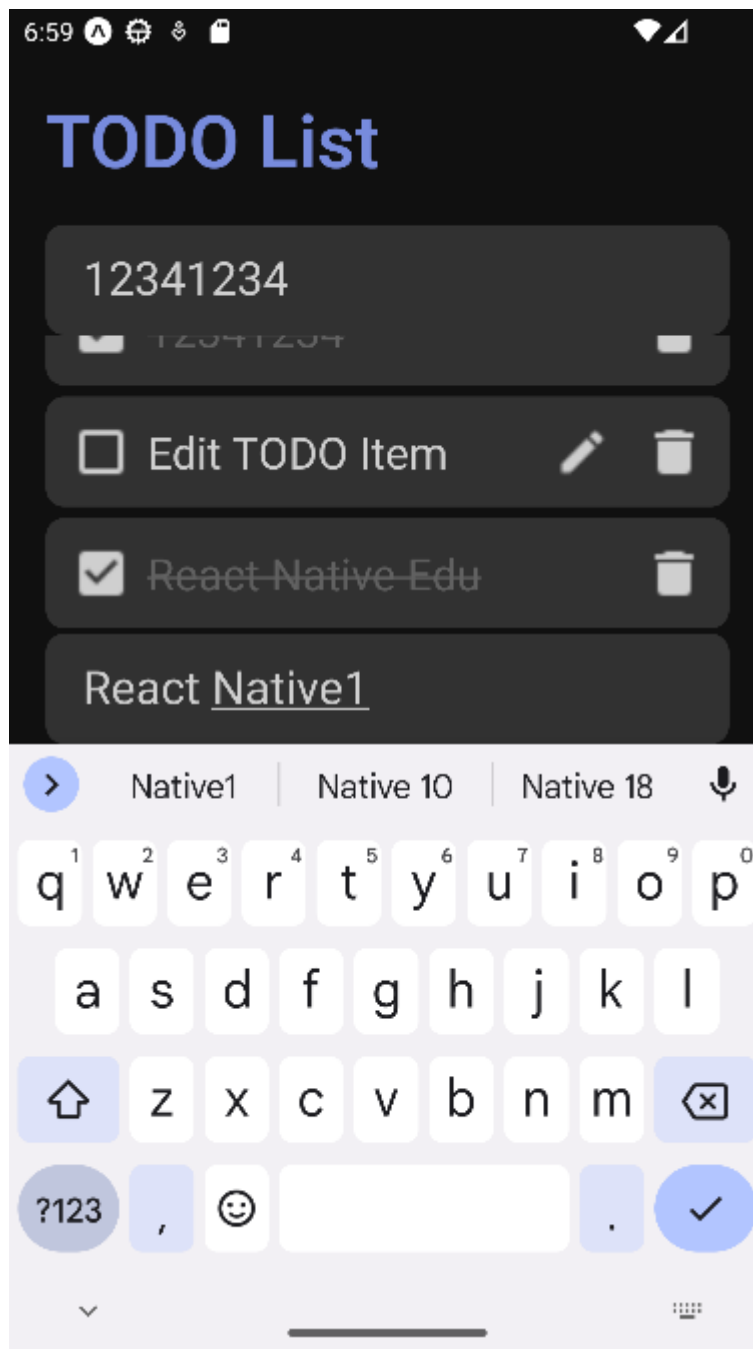
Task.propTypes = {
  item : PropTypes.object.isRequired,
  deleteTask : PropTypes.func.isRequired,
  toggleTask : PropTypes.func.isRequired,
};

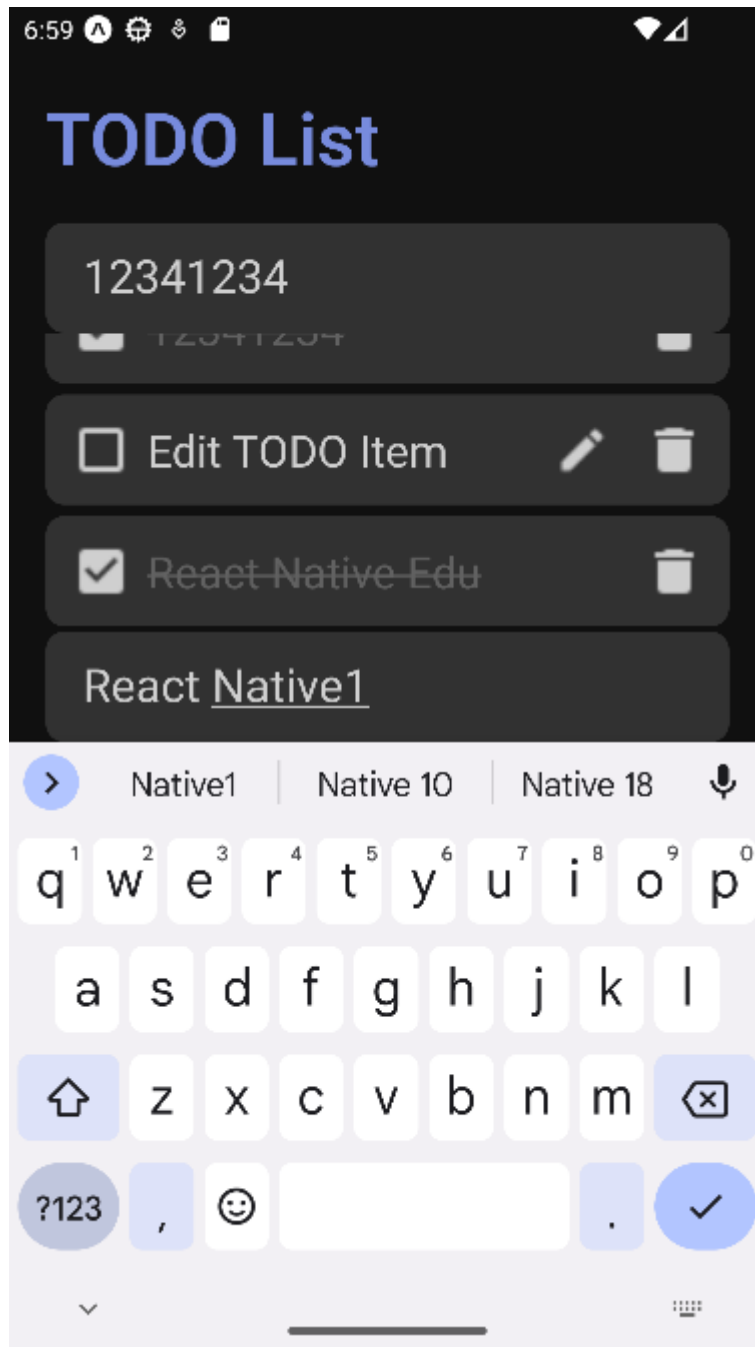
export default Task;

```

수정 상태를 관리하기 위해 isEditing 변수를 생성하고 수정 버튼이 클릭되면 값이 변하도록 작성.

수정되는 내용을 담을 text변수를 생성하고 input 컴포넌트의 값으로 설정.





입력 취소하기

항목을 추가하거나 수정하는 도중에는 입력을 취소할 방법이 없다. 입력 중에 다른 영역을 클릭해서 Input 컴포넌트가 포커스를 잃으면 입력 중인 내용이 사라지고 취소되도록 input 컴포넌트를 수정하자.

```

const Input = ({placeholder, value, onChangeText, onSubmitEditing, onBlur}) => {
  const width = Dimensions.get('window').width;
  return (
    <StyledInput
      width={width}
      placeholder={placeholder}
      maxLength={50}
      autoCapitalize="none"
      autoCorrect={false}
      returnKeyType="done"
      keyboardAppearance="dark"
      value={value}
      onChangeText={onChangeText}
      onSubmitEditing={onSubmitEditing}
      onBlur={onBlur}
    />
  );
};

Input.propTypes = {
  placeholder : PropTypes.string,
  value : PropTypes.string.isRequired,
  onChangeText : PropTypes.func.isRequired,
  onSubmitEditing : PropTypes.func.isRequired,
  onBlur: PropTypes.func.isRequired,
};

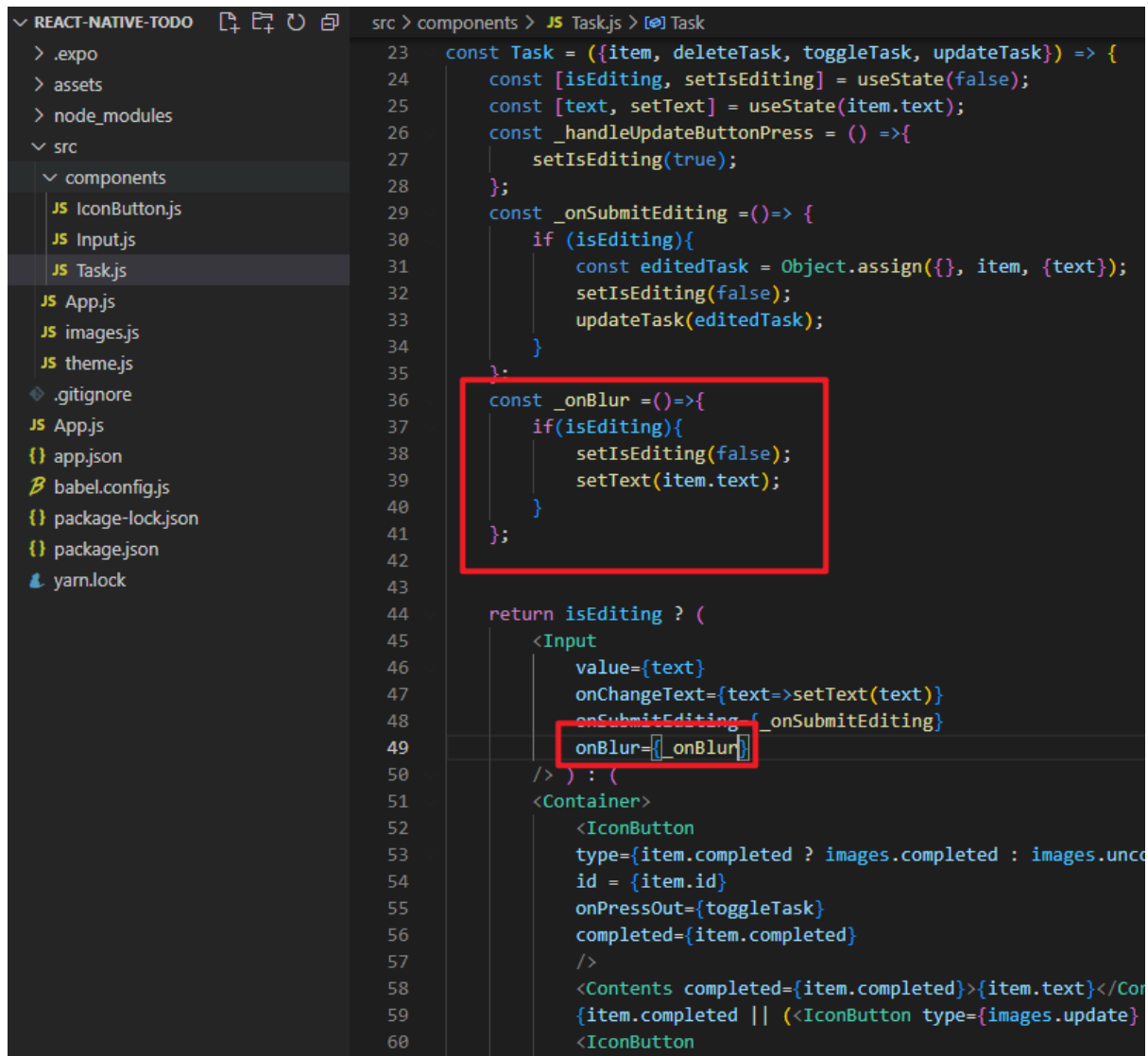
export default Input;

```

input컴포넌트에 onBlur 함수가 반드시 전달되도록 propTypes를 수정하고 전달된 함수를 사용하도록 수정했다. 이제 input 컴포넌트를 이용하는 곳에서 onBlur 함수를 전달하도록 수정.

```
39 >
40 > const _addTask = () => { ...
47 >
48 > const _deleteTask = id => { ...
52 >
53 > const _toggleTask = id => { ...
57 >
58 > const _updateTask = item => { ...
62 >
63 > const _handleTextChange = text => { ...
65 >
66 > const _onBlur = () => {
67 >   setNewTask('');
68 > };
69 > return(
70 >   <ThemeProvider theme={theme}>
71 >     <Container>
72 >       <StatusBar barStyle="light-content" ba
73 >       <Title>TODO List</Title>
74 >       <Input
75 >         placeholder="+ Add a Task"
76 >         onChangeText={_handleTextChange}
77 >         onSubmitEditing={_addTask}
78 >         onBlur={_onBlur}
79 >       />
80 >       <List width={width}>
81 >         {Object.values(tasks)
82 >           .reverse()
83 >           .map(item => (
84 >             <Task
85 >               key={item.id}
86 >               item={item}
87 >               deleteTask={_deleteTask}
88 >               toggleTask={_toggleTask}
89 >               updateTask={_updateTask}
90 >             />
91 >           ))}
92 >       </List>
93 >     </Container>
94 >   </ThemeProvider>
95 > );
```

App 컴포넌트에서 input 컴포넌트가 포커스를 잃으면 추가 중이던 값을 초기화 하는 onBlur 함수를 추가했다. 이제 Task 컴포넌트에서도 수정 중에 포커스를 잃으면 초기화 되도록 수정하자.



```
23 const Task = ({item, deleteTask, toggleTask, updateTask}) => {
24   const [isEditing, setIsEditing] = useState(false);
25   const [text, setText] = useState(item.text);
26   const _handleUpdateButtonPress = () =>{
27     setIsEditing(true);
28   };
29   const _onSubmitEditing =()=> {
30     if (isEditing){
31       const editedTask = Object.assign({}, item, {text});
32       setIsEditing(false);
33       updateTask(editedTask);
34     }
35   };
36   const _onBlur =()=>{
37     if(isEditing){
38       setIsEditing(false);
39       setText(item.text);
40     }
41   };
42
43   return isEditing ? (
44     <Input
45       value={text}
46       onChangeText={text=>setText(text)}
47       onSubmitEditing={_onSubmitEditing}
48       onBlur={_onBlur}
49     /> ) : (
50     <Container>
51       <IconButton
52         type={item.completed ? images.completed : images.unc
53         id = {item.id}
54         onPressOut={toggleTask}
55         completed={item.completed}
56       />
57       <Contents completed={item.completed}>{item.text}</Cor
58       {item.completed || (<IconButton type={images.update}
59       <IconButton
60
```

Task 컴포넌트에서도 수정 상태일 때 Input 컴포넌트의 포커스를 잃으면 수정 중인 내용을 초기화 하고 수정 상태를 종료하는 함수를 추가했다.

부가 기능

현재는 애플리케이션을 재시작할 때마다 내용이 초기화 되고, 로딩될 때 나타나는 화면은 미 완성된 애플리케이션처럼 느껴진다. 이번에는 데이터를 저장하고 불러오는 기능과 로딩 화면을 변경하는 방법을 알아보자

1. 데이터 저장하기

리액트 네이티브에서는 AsyncStorage를 이용해 로컬에 데이터를 저장하고 불러오는 기능을 구현할 수 있다. AsyncStorage는 비동기로 동작하며 문자열로 된 키-값(key-value) 형태의 데이터를 기기에 저장하고 불러올 수 있는 기능을 제공한다.

리액트 네이티브에서 제공하는 AsyncStorage는 아직 사용할 수는 있지만, 공식 문서에는 Deprecated(더 이상 사용되지 않은)이라고 나왔있다고 한다. 문서에도 그렇듯이 AsyncStorage 대신 react-native-community에서 관리하는 async-storage를 설치해서 사용하는 것을 권장한다.

- `async-storage` : <https://github.com/react-native-community/async-storage>

아래 명령어로 `async-storage`를 설치하자

```
expo install @react-native-community/async-storage
```

```
C:\reactnative\react-native-todo>expo install @react-native-community/async-storage
WARNING: expo-cli has not yet been tested against Node.js v18.12.0.
If you encounter any issues, please report them to https://github.com/expo/expo-cli/issues

expo-cli supports following Node.js versions:
* >=12.13.0 <15.0.0 (Maintenance LTS)
* >=16.0.0 <17.0.0 (Active LTS)

This command is being executed with the global Expo CLI. Learn more: https://blog.expo.dev/th
To use the local CLI instead (recommended in SDK 46 and higher), run:
> npx expo install

Installing 1 other package using Yarn.
> yarn add @react-native-community/async-storage
yarn add v1.22.19
warning package-lock.json found. Your project contains lock files generated by tools other than
Yarn. This may lead to inconsistencies caused by unsynchronized lock files. To clear this warning, remove package-lock.json
[1/4] Resolving packages...
warning @react-native-community/async-storage@1.12.1: Async Storage has moved to new organization
warning @react-native-community/async-storage > deep-assign@3.0.0: Check out `lodash.merge` or `deepmerge`
[2/4] Fetching packages...
[ ] 43/892
```

※expo install 은 npm install과 거의 동일한 역할을 한다.

차이점은 사용 중인 expo SDK 버전과 호환되는 버전이 있는지 확인하고, 해당 버전의 라이브러리를 설치하는 과정이 추가된다는 점이다.

```
> .expo
> assets
> node_modules
v src
  v components
    JS IconButton.js
    JS Input.js
    JS Task.js
    JS App.js
    JS images.js
    JS theme.js
  .gitignore
JS App.js
() app.json
B babel.config.js
() package-lock.json
() package.json
yarn.lock

9
10 import AsyncStorage from "@react-native-community/async-storage";
11
12 const List = styled.ScrollView`
13   flex : 1;
14   width : ${({width})=> width - 40}px;
15 `
16
17 const Container = styled.SafeAreaView`
18   flex : 1;
19   background-color : ${({theme}) => theme.background};
20   align-items : center;
21   justify-content : flex-start;
22 `;
23
24 const Title = styled.Text`
25   font-size : 40px;
26   font-weight : 600;
27   color : ${({theme}) => theme.main};
28   align-self : flex-start;
29   margin : 20px;
30 `
31
32 export default function App(){
33   const width = Dimensions.get('window').width;
34   const [newTask, setNewTask] = useState('');
35   const [tasks, setTasks] = useState([]);
36   const _saveTasks = async tasks =>{
37     try{
38       await AsyncStorage.setItem('tasks', JSON.stringify(tasks));
39       setTasks(tasks);
40     } catch (e){
41       console.error(e);
42     }
43   };
44   const _addTask = ()=>{
45     const ID = Date.now().toString();
46     const newTaskObject = { ...
47   };
48   };
49   setNewTask('');
50   _saveTasks({ ...tasks, ...newTaskObject});
51 }
```

```

    };
    const _deleteTask = id => {
      const currentTasks = Object.assign({}, tasks);
      delete currentTasks[id];
      _saveTasks(currentTasks);
    };
    const _toggleTask = id => {
      const currentTasks = Object.assign({}, tasks);
      currentTasks[id]['completed'] = !currentTasks[id]['completed'];
      _saveTasks(currentTasks);
    };
    const _updateTask = item => {
      const currentTasks = Object.assign({}, tasks);
      currentTasks[item.id] = item;
      _saveTasks(currentTasks);
    };
    const _handleTextChange = text => { ...
  };
  const _onBlur = () => { ...
  };

```

테스트를 위해 tasks의 초기값으로 작성했던 내용을 삭제한 후 AsyncStorage를 이용해 tasks라는 문자열을 키로 하여 전달된 항목들을 문자열로 변환해서 저장하는 _saveTasks 함수를 작성했다. tasks의 값이 변경될 때마다 저장해야 하므로 setTasks 세터 함수를 이용하는 곳에서 _saveTasks 함수를 호출하도록 수정했다.

2. 데이터 불러오기

저장된 데이터를 불러오는 함수

```

export default function App() {
  const width = Dimensions.get('window').width;
  const [newTask, setNewTask] = useState('');
  const [tasks, setTasks] = useState({});
  > const _saveTasks = async tasks => { ...
    };
    const _loadTasks = async () => {
      const loadedTasks = await AsyncStorage.getItem('tasks');
      setTasks(JSON.parse(loadedTasks || '{}'));
    };
    const _addTask = () => {
      const ID = Date.now().toString();
      const newTaskObject = {
        [ID]: { id: ID, text: newTask, completed: false },
      };
      setNewTask('');
      _saveTasks({ ...tasks, ...newTaskObject });
    };
  };
}

```

항목을 저장할 때 사용했던 키와 동일한 키로 데이터를 불러오고 객체로 변환하여 tasks에 입력하는 `_loadTasks` 함수를 작성했다. 작성된 `_loadTasks` 함수가 애플리케이션이 로딩 되는 단계에서 실행되고, 첫 화면이 나타나기 전에 완료되어 불러온 항목이 화면에 렌더링 되는 것이 가장 자연스러운 것이다.

expo에서 제공하는 `apploading` 컴포넌트를 이용하면 이런 작업을 쉽게 구현할 수 있다. `apploading` 컴포넌트는 특정 조건에서 로딩 화면이 유지되도록 하는 기능으로, 렌더링 하기 전에 처리해야 하는 작업을 수행하는 데 유용하게 사용된다. `AppLoading` 컴포넌트를 사용 해서 첫 화면이 렌더링 되기 전에 `_loadTask` 함수가 호출되도록 하자.


```
JS App.js
JS images.js
JS theme.js
.gitignore
JS App.js
() app.json
B babel.config.js
() package-lock.json
() package.json
A yarn.lock

10 import AsyncStorage from '@react-native-community/async-storage';
11 import {AppLoading} from 'expo';
12
13
14 const List = styled.ScrollView`
15   flex : 1;
16   width : ${({width})=> width - 40}px;
17 `
18
19 const Container = styled.SafeAreaView`
20   flex : 1;
21   background-color : ${({theme})=> theme.background};
22   align-items : center;
23   justify-content : flex-start;
24 `;
25
26 const Title = styled.Text`
27   font-size : 40px;
28   font-weight : 600;
29   color : ${({theme})=> theme.main};
30   align-self : flex-start;
31   margin : 20px;
32 `
33
34 export default function App(){
35   const width = Dimensions.get('window').width;
36
37   const [isReady, setIsReady] = useState(false);
38   const [newTask, setNewTask] = useState(''),
39   const [tasks, setTasks] = useState({});
40
```

```
Components
JS IconButton.js
JS Input.js
JS Task.js
JS App.js
JS images.js
JS theme.js
.gitignore
JS App.js
() app.json
B babel.config.js
() package-lock.json
() package.json
A yarn.lock

84
85
86 return isReady ? (
87   <ThemeProvider theme={theme}>
88     <Container>
89       <StatusBar barStyle="light-content" backgroundColor={theme.background} />
90       <Title>TODO List</Title>
91       <Input
92         placeholder="+ Add a Task"
93         onChangeText={_handleTextChange}
94         onSubmitEditing={_addTask}
95         onBlur={_onBlur}
96       />
97       <List width={width}>
98         {Object.values(tasks)
99           .reverse()
100           .map(item => (
101             <Task
102               key={item.id}
103               item={item}
104               deleteTask={_deleteTask}
105               toggleTask={_toggleTask}
106               updateTask={_updateTask}
107             />
108           ))}
109       </List>
110     </Container>
111   </ThemeProvider>
112 ) : (
113   <AppLoading
114     startAsync={_loadTasks}
115     onFinish={()=> setIsReady(true)}
116     onError={console.error}
117   />
118 );
119
120
```

useState 를 이용해 화면의 준비 상태를 관리할 isReady를 생성하고 isReady의 값에 따라 AppLoading 컴포넌트를 이용해 로딩 화면이 나타나도록 작성했다.

AppLoading 컴포넌트에 설정된 값들은 각각 다음과 같은 역할을 한다.

- startAsync : AppLoading 컴포넌트가 동작하는 동안 실행될 함수
- onFinish : startAsync가 완료되면 실행할 함수
- onError : startAsync에서 오류가 발생하면 실행할 함수

로딩 화면에서 저장한 데이터를 불러오고, 불러오기 작업이 완료되면 isReady 값을 변경해 화면을 렌더링하도록 작성

Error: [@RNC/AsyncStorage]: NativeModule: AsyncStorage is null.

실행 하려니 이런 에러가 잘생한다.

아래 두개를 설치 하고 하니

```
expo install @react-native-async-storage/async-storage
```

```
expo install expo-app-loading
```

잘 해결 되었다.

