

day51-rn-navigation

☰ 태그	
📅 날짜	@2022년 12월 12일

모바일 애플리케이션은 하나의 화면으로 구성되는 경우가 거의 없으며, 일반적으로 다양한 화면이 상황에 맞게 전환되면서 나타난다. 그렇기 때문에 내비게이션은 모바일 애플리케이션에서 가장 중요한 기능 중 하나라고 할 수 있다.

리액트 네이티브에서는 내비게이션 기능을 지원하지 않으므로 외부 라이브러리를 이용해야 한다. 우리는 리액트 네이티브에서 추천하는 리액트 내비게이션 라이브러리를 사용해 내비게이션을 적용하는 방법에 대해 알아보자

- 리액트 내비게이션 : <https://reactnavigation.org/>

아래 명령어로 프로젝트를 생성하고 같이 사용할 스타일드라이브러리도 생성하자

- `expo init react-native-navigation`
- `npm install styled-components`

※ 내비게이션의 화면 전환을 네이티브 방식으로 구현하고 싶으면 Wix에서 개발한 리액트 네이티브 내비게이션을 사용해보자.

리액트 내비게이션

리액트 내비게이션 라이브러리는 리액트 네이티브 애플리케이션의 내비게이션을 쉽고 간단하게 관리할 수 있도록 도와준다.

지원하는 내비게이션의 종류는 스택 내비게이션, 탭 내비게이션, 드로어 내비게이션 세 종류이다.

※공부하는 자료는 리액트 네비게이션 5버전으로 진행했다. 버전이 변경되면 사용법이 조금 달라질 수 있다. 최신 정보는 공식 홈페이지를 참고하자.

네비게이션의 구조

리액트 네비게이션에는 `NavigationContainer` 컴포넌트, `Navigator` 컴포넌트, `Screen` 컴포넌트가 있다.

p.251 그림

`Screen` 컴포넌트는 화면으로 사용되는 컴포넌트로 `name`과 `component` 속성을 지정해야 한다. `name`은 화면 이름으로 `component`에는 화면으로 사용될 컴포넌트를 전달한다. 화면으로 사용되는 컴포넌트에는 항상 `navigation`과 `route`가 `props`로 전달된다는 특징이 있다.

`Navigator` 컴포넌트는 화면을 관리하는 중간 관리자 역할로 네비게이션을 구성하며, 여러개의 `Screen` 컴포넌트를 자식 컴포넌트로 갖고 있다.

`NavigationContainer` 컴포넌트는 네비게이션의 계층 구조와 상태를 관리하는 컨테이너 역할을 하며, 모든 네비게이션 구성 요소를 감싼 최상위 컴포넌트여야 한다.

설정 우선 순위

리액트 네비게이션에서 설정할 수 있는 다양한 속성을 수정하는 방법은 `Navigator` 컴포넌트의 속성으로 수정하는 방법, `Screen` 컴포넌트의 속성을 수정하는 방법, 화면으로 사용되는 컴포넌트의 `props`로 전달되는 `navigation`을 이용해서 수정하는 방법과 같이 크게 세 가지로 나뉜다.

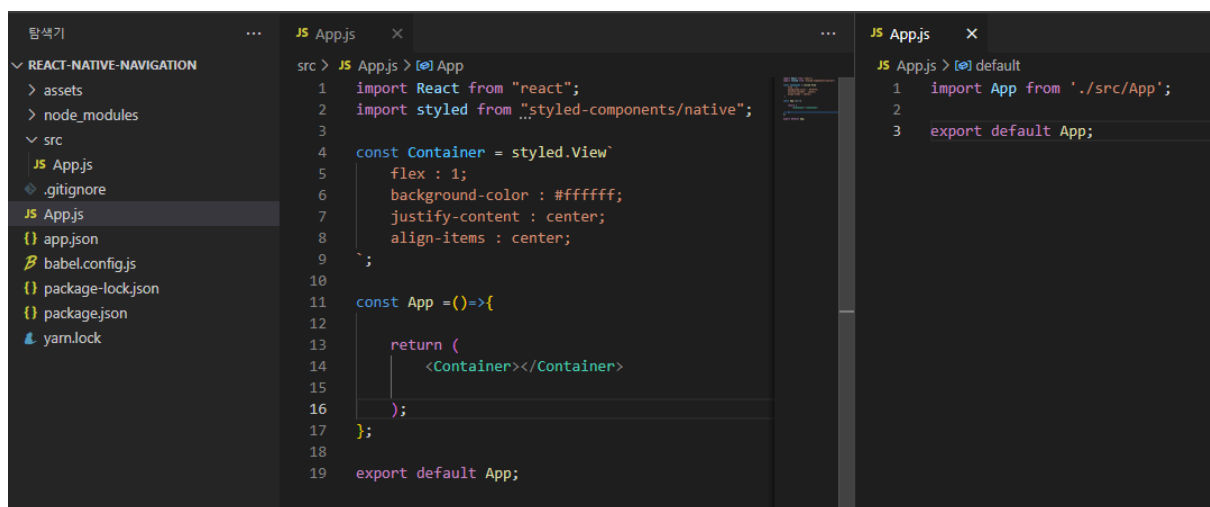
화면으로 사용되는 컴포넌트에서 설정하는 방법과 `Screen` 컴포넌트에서 설정하는 방법은 해당 화면에만 적용되지만, `Navigator` 컴포넌트에서 속성을 지정하는 방법을 사용하면 자식 컴포넌트로 존재하는 모든 컴포넌트에 적용된다는 특징이 있다. 이런 특징을 이용해서 모든 화면에 공통적으로 적용하고 싶은 속성인 경우 `Navigator` 컴포넌트를 이용하고, 개별 화면에만 적용되는 속성인 경우 `Screen` 컴포넌트 혹은 화면으로 사용되는 컴포넌트에서 설정하는 것이 좋다. 작은 범위의 설정일수록 우선순위가 높으므로 만약 `Screen` 컴포넌트와 화면

으로 사용되는 컴포넌트에 동일한 옵션이 적용되었다면 화면 컴포넌트의 설정이 우선한다는 것을 기억해두자.

그림 p252

폴더 구조와 라이브러리 설치

앞에 했던 것들처럼 src폴더 밑에 app.js를 만들고 루트 디렉터리에 app.js를 만들어서 작성하자.

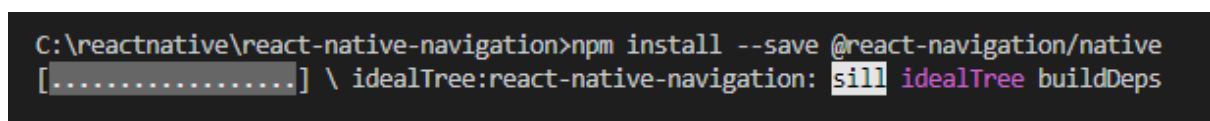


이번에는 src폴더 밑에 화면으로 사용하는 컴포넌트를 관리할 screens 폴더와 작성되는 내비게이션 코드를 관리할 navigations 폴더를 생성하겠다.

그림 p.253

폴더 생성이 완료되면 리액트 내비게이션 실습을 위해 아래 명령어로 리액트 내비게이션 라이브러리를 설치하자.

- `npm install --save @react-navigation/native`



위처럼 했더니 에러가 발생해서 `—force`를 붙여서 설치를 진행 했다.

리액트 내비게이션은 각 기능별로 모듈이 분리되어 있어, 이후에도 사용하는 내비게이션의 종류에 따라 개별적으로 추가 라이브러리를 설치해야 한다. 이번에는 대부분의 내비게이션에서 반드시 필요한 종속성을 설치하자.

- `expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view`

스택 내비게이션

이번에는 가장 기본적인 내비게이션인 스택 내비게이션에 대해 알아보자

아래 명령어로 스택 내비게이션 확용에 필요한 라이브러리를 설치하자.

- `npm install @react-navigation/stack —force`

스택 내비게이션은 일반적으로 가장 많이 사용되는 내비게이션으로, 현재 화면 위에 다른 화면을 쌓으면서 화면을 이동하는 것이 특징이다. 예를 들면, 채팅 애플리케이션에서 채팅방에 입장하는 상황이나 여러 목록 중에서 특정 항목의 상세 화면으로 이동할 때 많이 사용된다.

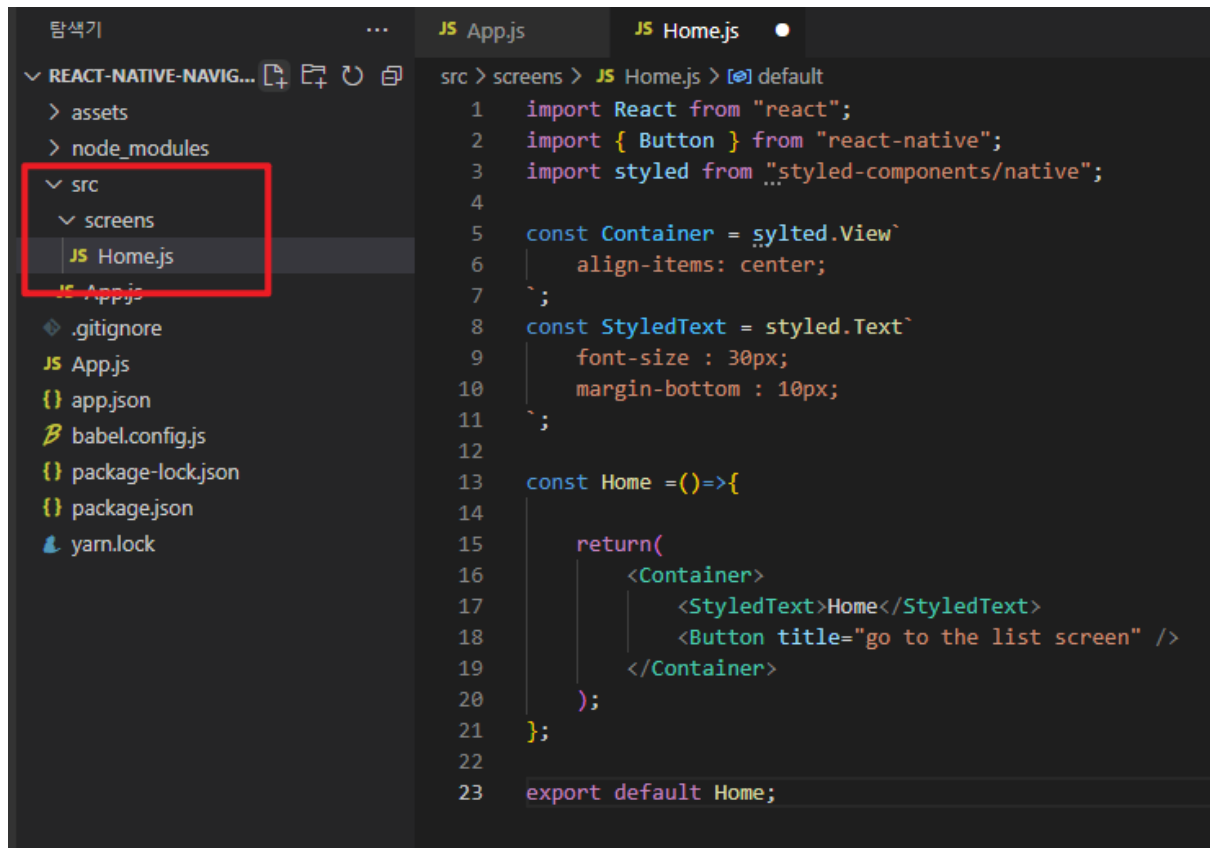
스택 내비게이션은 화면 위에 새로운 화면을 쌓으면서 (push) 이동하기 때문에 이전 화면을 계속 유지한다. 이런 구조 때문에 가장 위에 있는 화면을 들어내면(pop) 이전 화면으로 돌아갈 수 있다는 특징이 있다.

그림 p.255

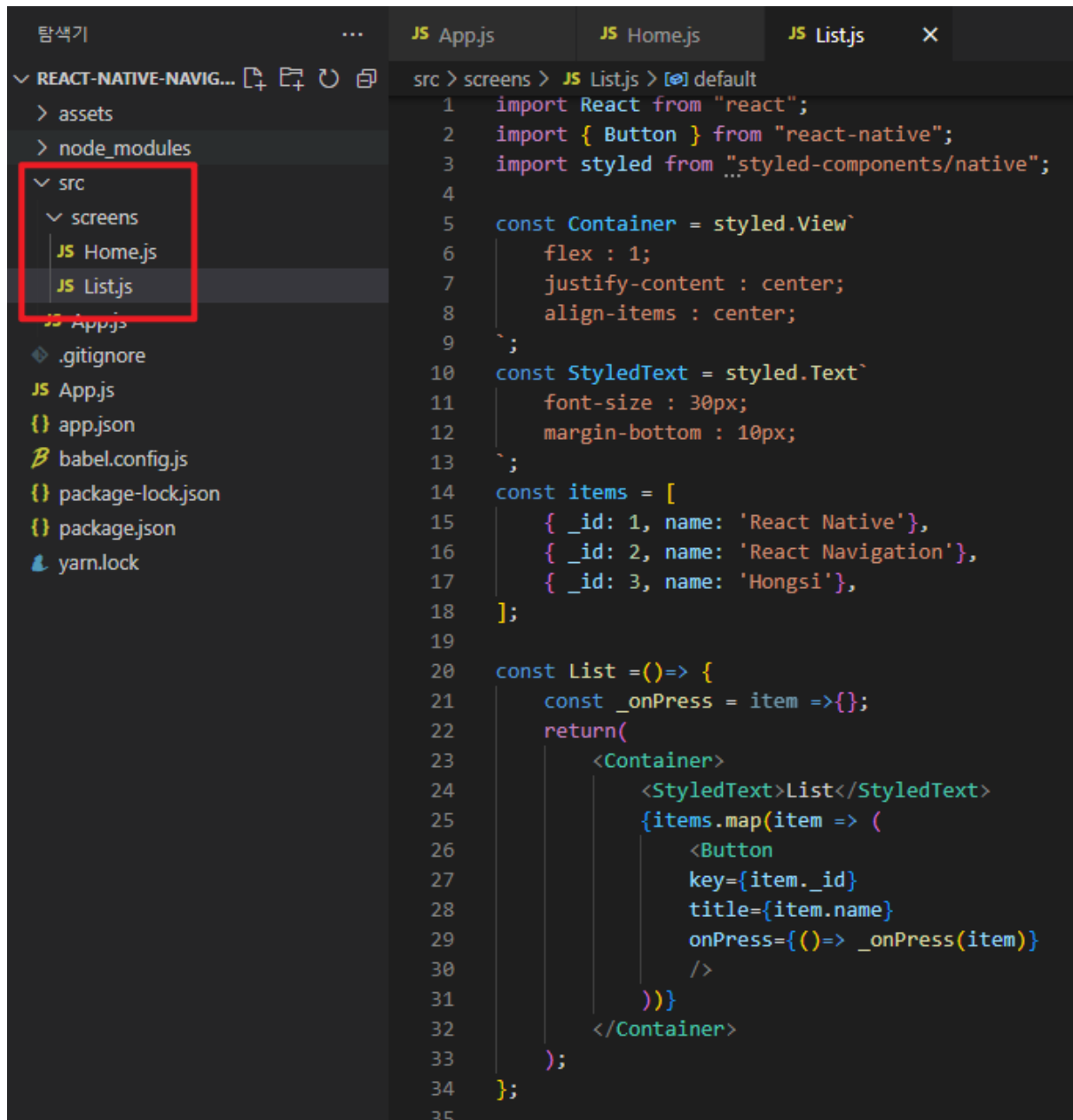
화면 구성

이제 스택 내비게이션을 이용해서 화면을 구성하고 이동하는 방법에 대해 알아보자.

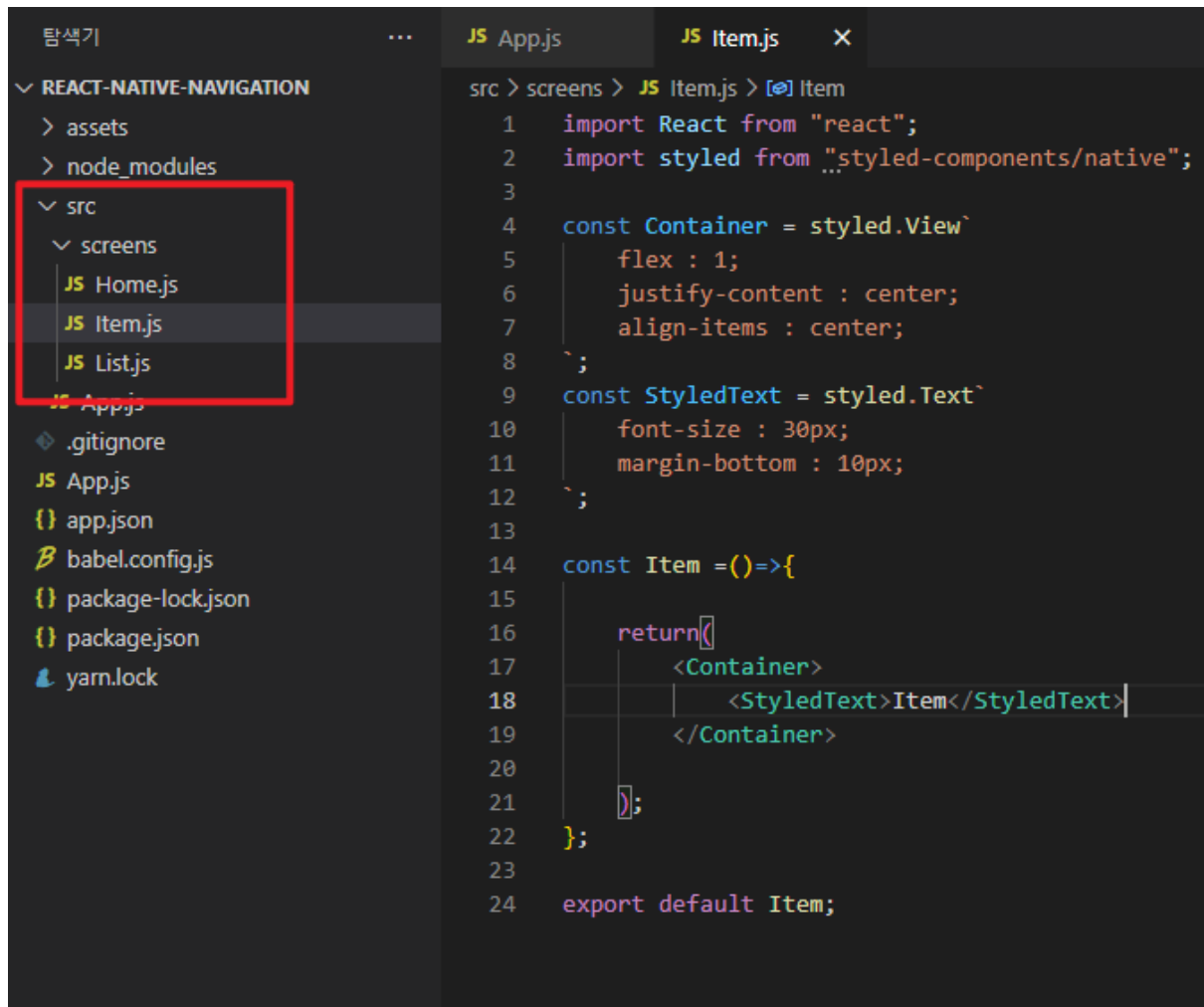
먼저 `screens` 폴더 밑에 스택 내비게이션의 화면으로 사용할 화면을 만든다.



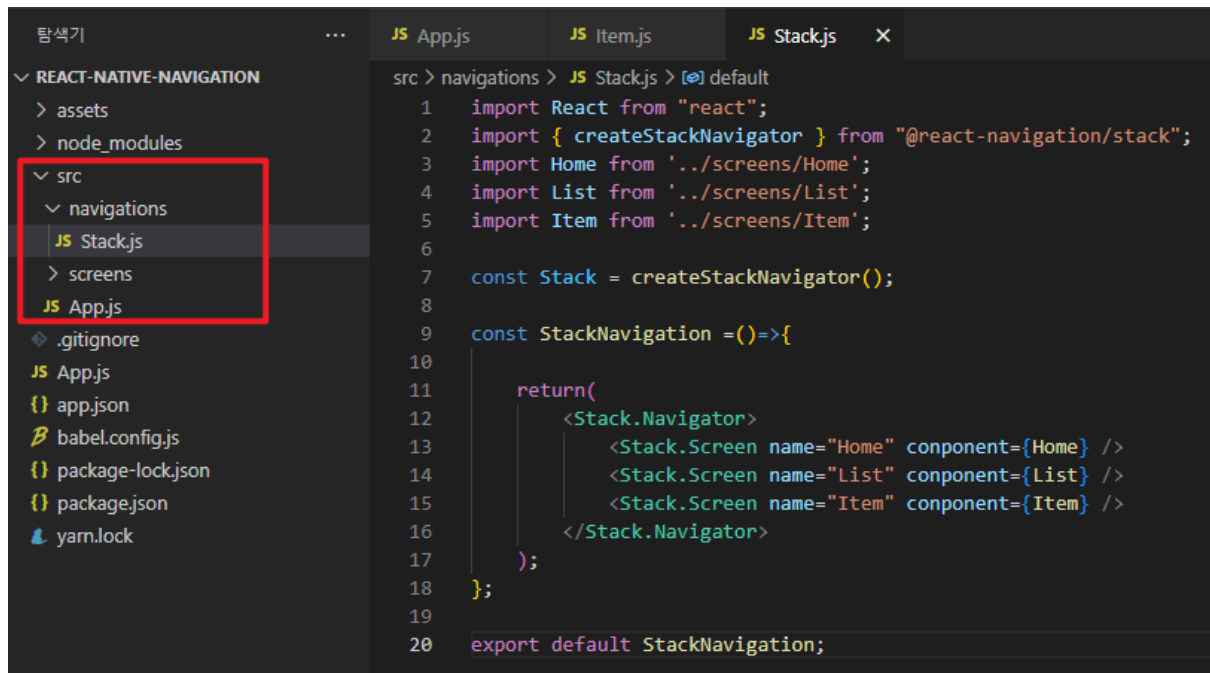
첫 화면으로 사용할 Home 화면을 작성했다. 화면을 확인하기 위한 텍스트와 다음 화면인 List 화면으로 이동하기 위한 버튼으로 화면을 구성했다.



List 화면에 사용될 컴포넌트를 작성했다. 화면에서 사용할 임시 목록을 만들고 항목 수 만큼 버튼을 생성하도록 만들었다.



마지막으로 목록의 상세 정보를 보여주는 컴포넌트를 작성했다. 이제 navigations 폴더 안에 Stack.js 파일을 생성하고 생성된 컴포넌트를 이용해 스택 네비게이션을 구성해보자.

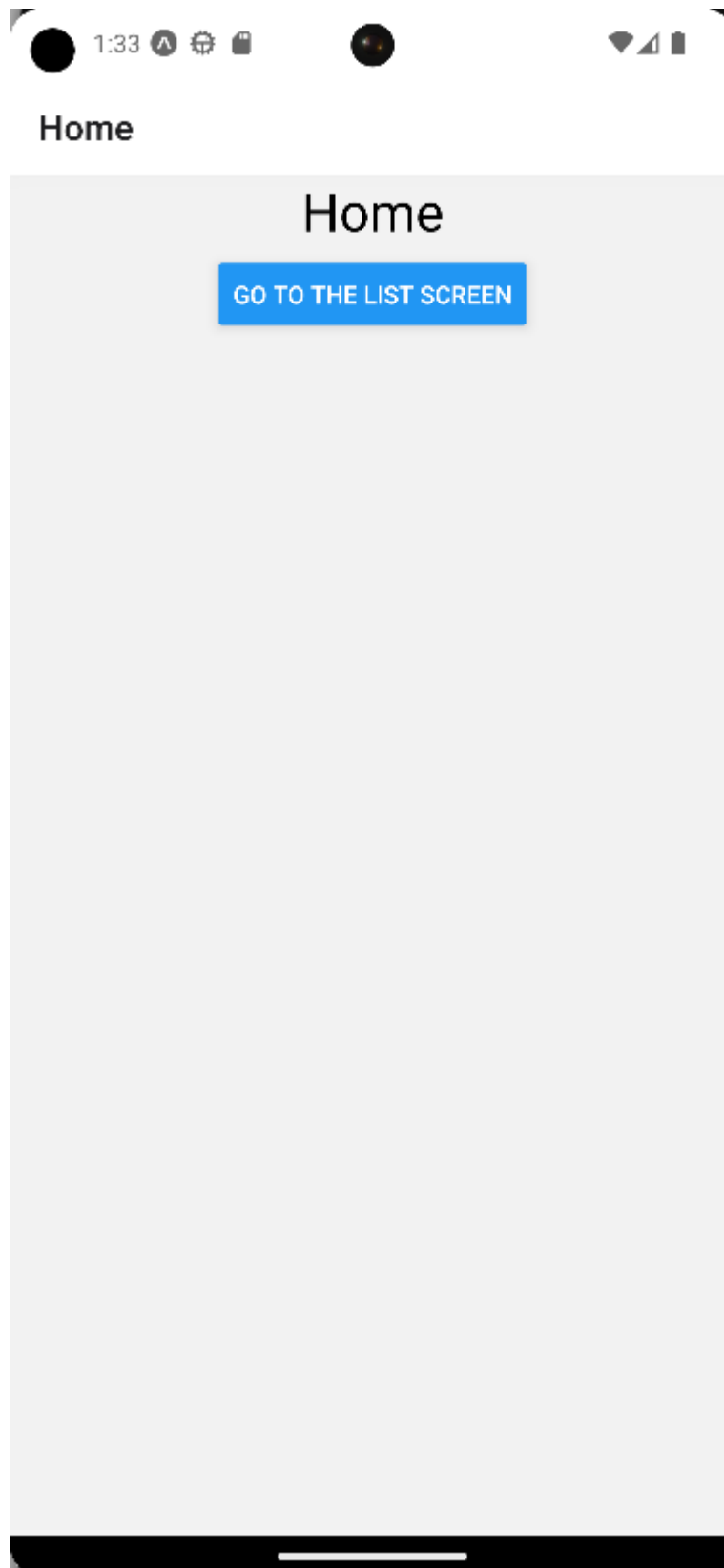


가장 먼저 `createStackNavigator` 함수를 이용해서 스택 내비게이션을 생성했다. 생성된 스택 내비게이션에는 화면을 구성하는 `Screen` 컴포넌트와 `Screen` 컴포넌트를 관리하는 `Navigator` 컴포넌트가 있다.

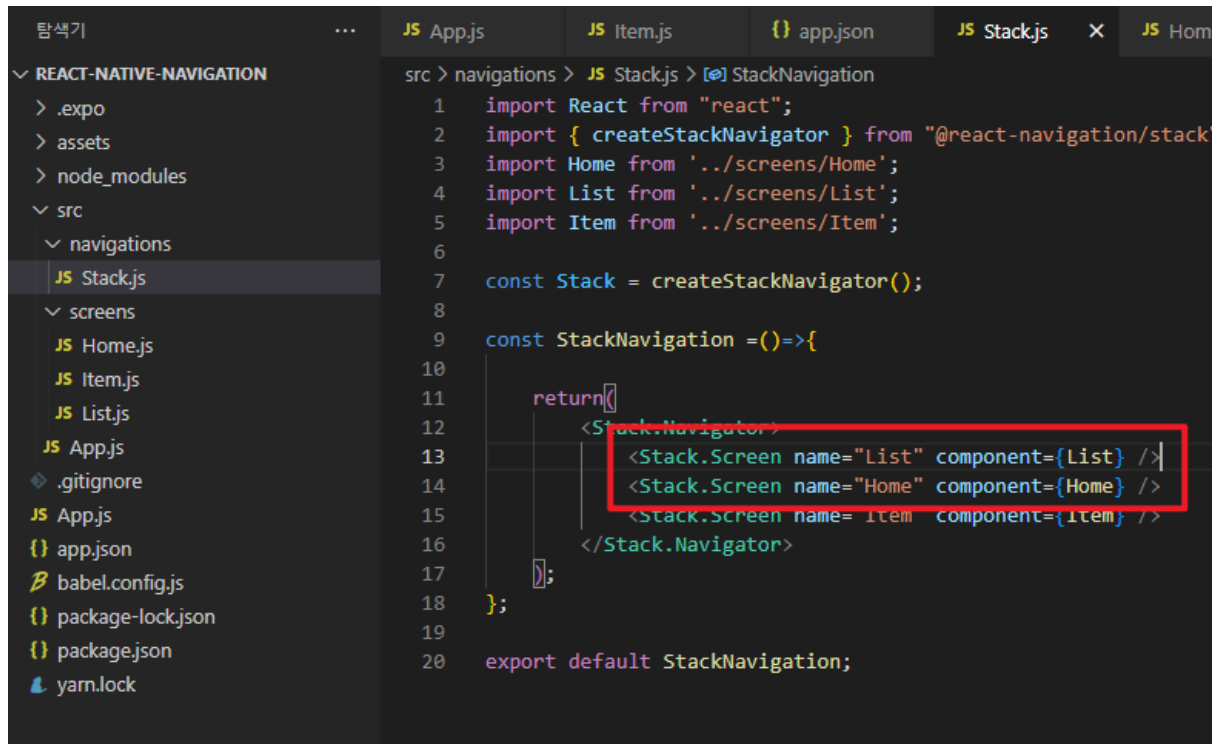
`Navigator` 컴포넌트 안에 `Screen` 컴포넌트를 자식 컴포넌트로 작성하고, 앞에서 만든 컴포넌트를 `Screen` 컴포넌트의 `component`로 지정했다. `name`에는 화면의 이름을 작성하는데, `Screen` 컴포넌트의 `name`은 반드시 서로 다른 값을 가져야 한다는 점에 유의하자.

이제 `App` 컴포넌트에서 완성된 스택 내비게이션을 사용하자.

`NavigationContainer` 컴포넌트를 이용해서 작성된 스택 내비게이션을 감싸도록 `App` 컴포넌트가 수정되었다.



스택 네비게이션에서 첫 번째 화면으로 나오는 화면은 Navigator 컴포넌트의 첫 번째 자식 screen 컴포넌트이다. 만약 다음과 같이 순서를 변경하면 List 화면이 첫 화면으로 나타난다.

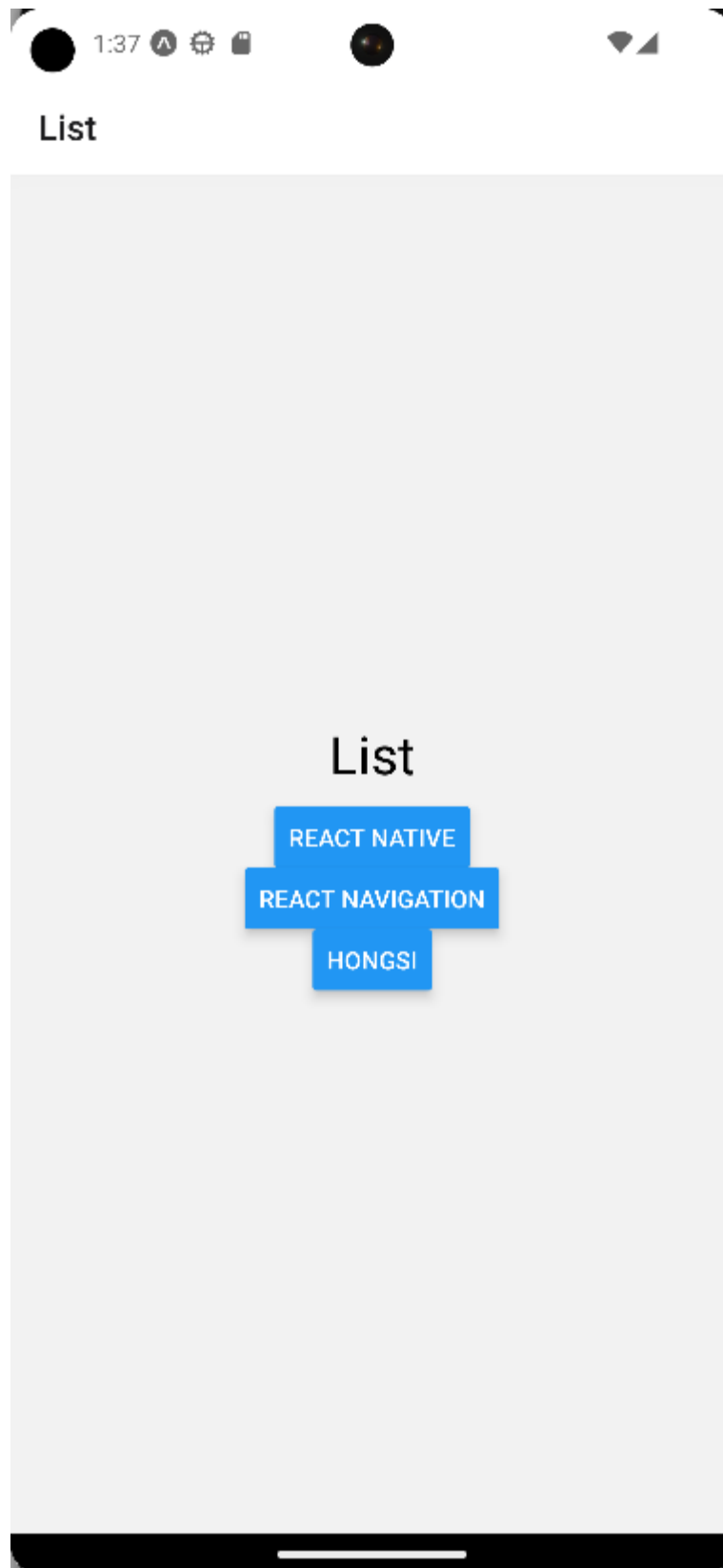


```
src > navigations > JS Stack.js > StackNavigation
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = () => {
10
11    return (
12      <Stack.Navigator>
13        <Stack.Screen name="List" component={List} />
14        <Stack.Screen name="Home" component={Home} />
15        <Stack.Screen name="Item" component={Item} />
16      </Stack.Navigator>
17    );
18  };
19
20  export default StackNavigation;
```

컴포넌트 순서를 변경하는 방법 외에도 `initialRouteName` 속성을 이용해 첫 번째 화면을 지정하는 방법이 있다.

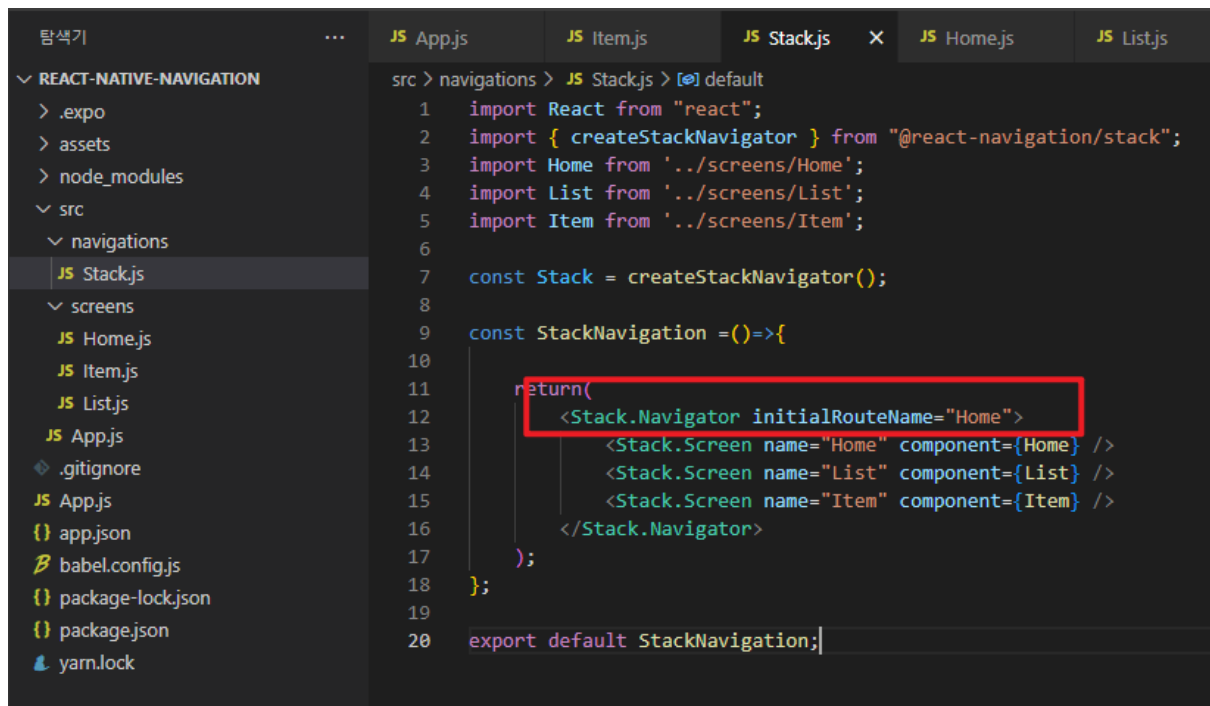
The image shows a code editor with a sidebar on the left displaying a file tree for a React Native project. The main editor area shows the code for `Stack.js`. The code imports `React` and `createStackNavigator` from `@react-navigation/stack`, and imports `Home`, `List`, and `Item` from their respective screen files. It then creates a `Stack` instance and defines a `StackNavigator` function that returns a `Stack.Navigator` with three routes: `Home`, `List`, and `Item`. The `List` route is highlighted with a red box.

```
src > navigations > JS Stack.js > [🔍] default
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = ()=>{
10
11    return(
12      <Stack.Navigator initialRouteName="List">
13        <Stack.Screen name="Home" component={Home} />
14        <Stack.Screen name="List" component={List} />
15        <Stack.Screen name="Item" component={Item} />
16      </Stack.Navigator>
17    );
18  };
19
20  export default StackNavigation;
```



화면 이동

이번에는 화면을 이동하는 방법에 대해 알아보자. Screen 컴포넌트의 component로 지정된 컴포넌트는 화면으로 이용되고 navigation이 props로 전달된다. navigation에는 다양한 기능이 있는데 그중 navigate 함수는 원하는 화면으로 이동하는데 사용되는 함수이다. 먼저 Stack.js 파일에서 첫 화면을 다시 Home화면으로 변경하자.

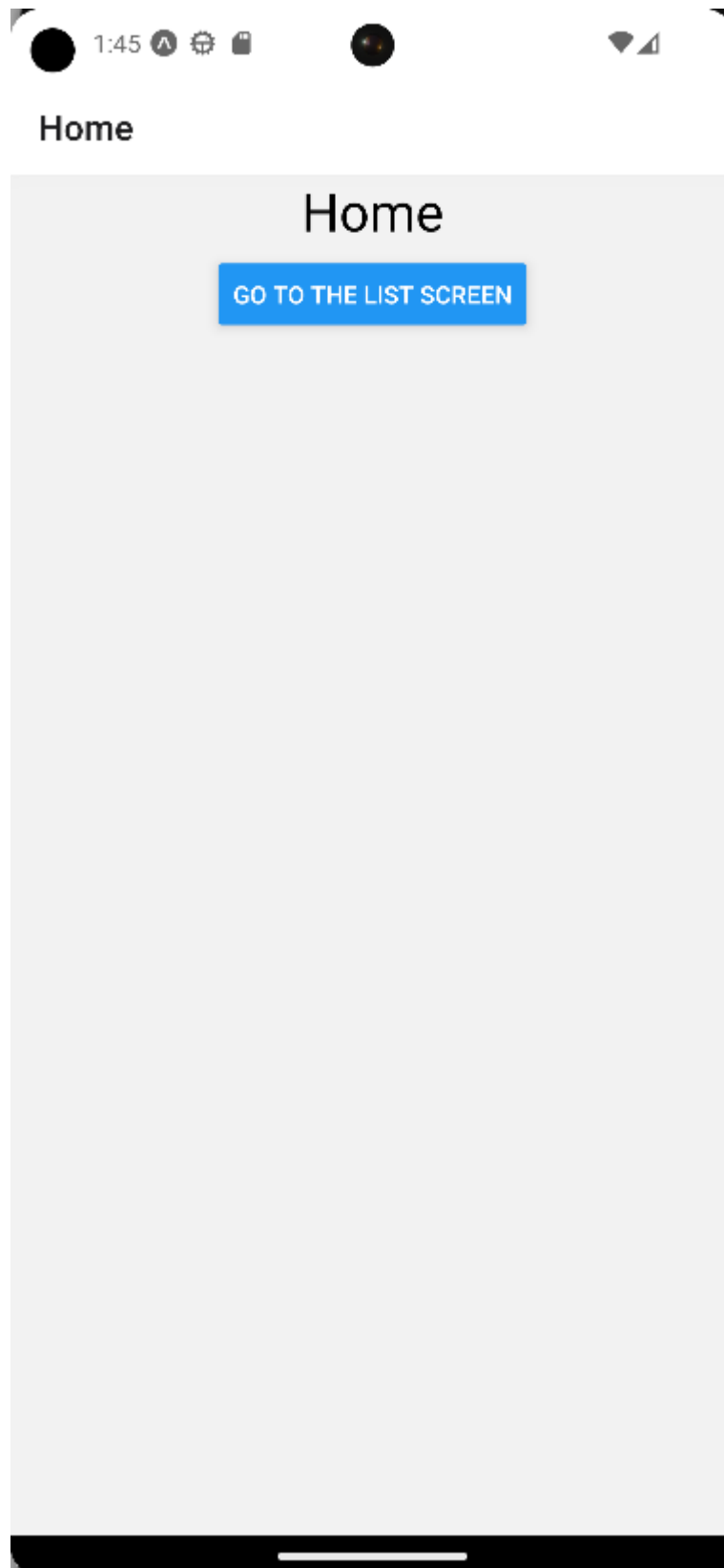


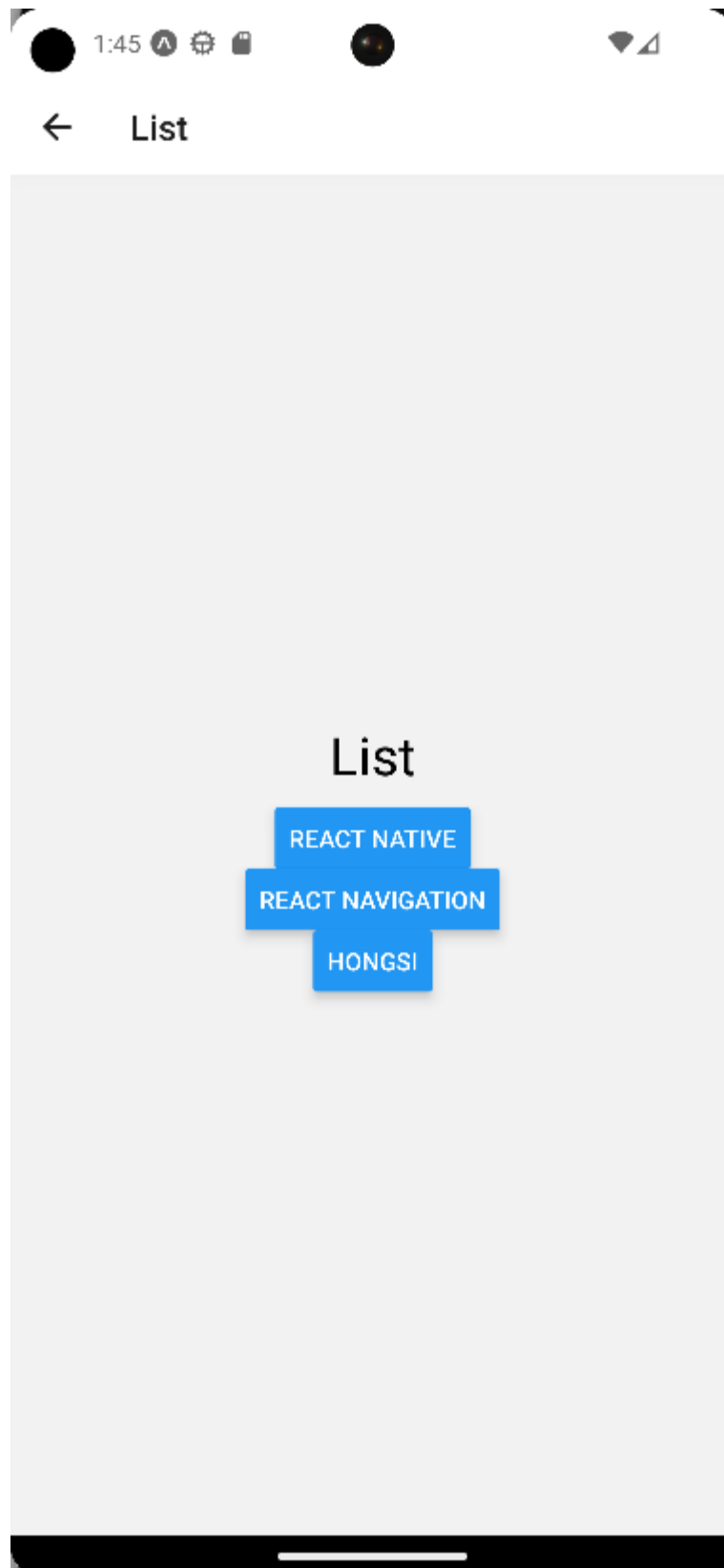
```
src > navigations > JS Stack.js > [🔍] default
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = () => {
10
11    return(
12      <Stack.Navigator initialRouteName="Home">
13        <Stack.Screen name="Home" component={Home} />
14        <Stack.Screen name="List" component={List} />
15        <Stack.Screen name="Item" component={Item} />
16      </Stack.Navigator>
17    );
18  };
19
20  export default StackNavigation;
```

이번엔 Home화면에서 props로 전달되는 navigation을 사용해서 버튼을 클릭하면 List화면으로 이동하도록 만들어보자.

```
src > screens > JS Home.js > [🔍] default
1  import React from "react";
2  import { Button } from "react-native";
3  import styled from "styled-components";
4
5  const Container = styled.View`
6    align-items: center;
7  `;
8  const StyledText = styled.Text`
9    font-size : 30px;
10   margin-bottom : 10px;
11 `;
12
13  const Home = ({ navigation })=>{
14
15    return(
16      <Container>
17        <StyledText>Home</StyledText>
18        <Button
19          title="go to the list screen"
20          onPress={()=> navigation.navigate('List')} />
21        </Container>
22      );
23  };
24
25  export default Home;
```

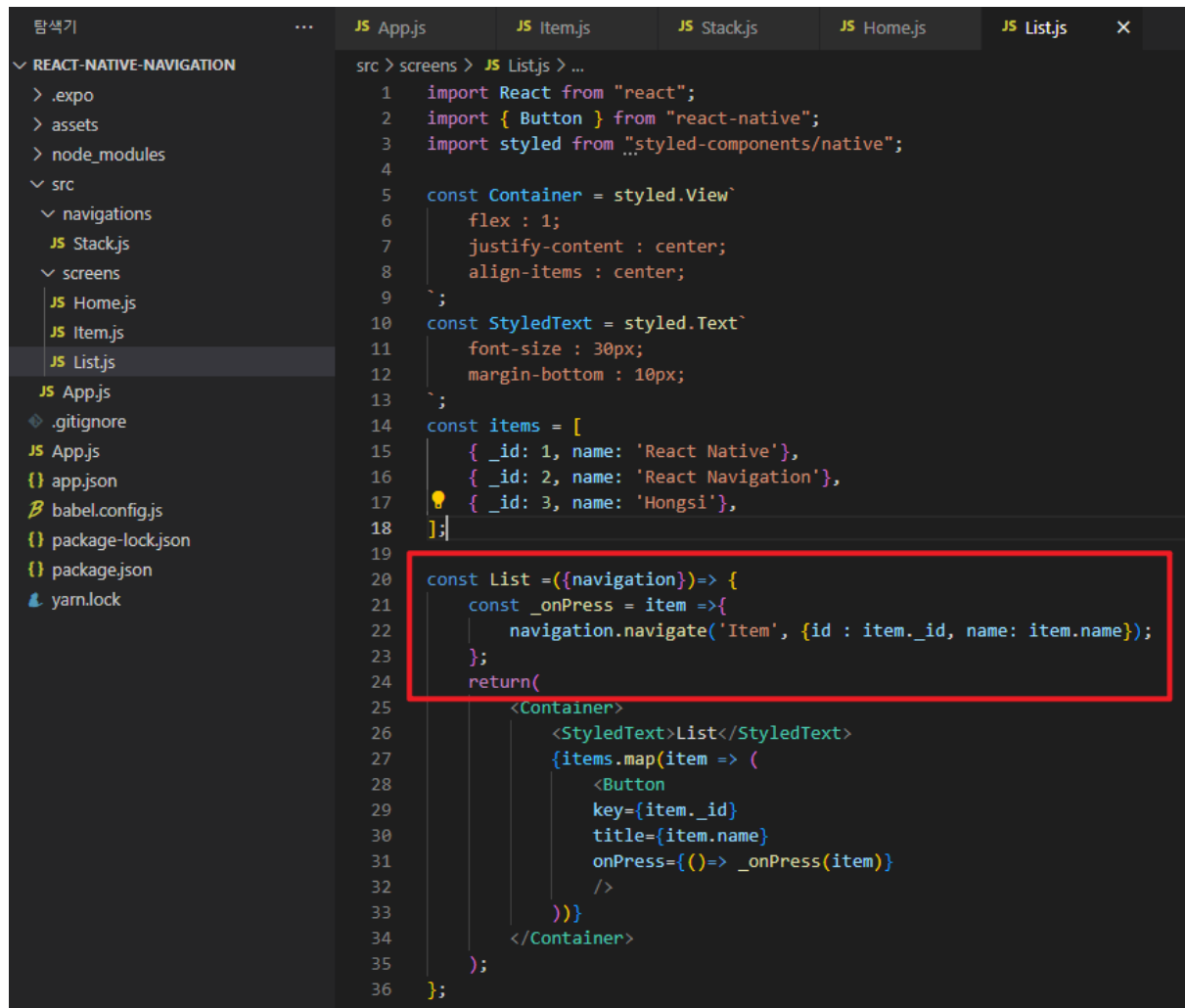
navigation에 있는 navigate 함수를 이용해서 원하는 화면의 이름을 전달하면 해당 화면으로 이동한다. 단, 전달되는 화면의 이름은 Screen 컴포넌트의 name값 중 하나를 입력해야 한다.





만약 이동하는 화면이 이전 화면의 상세 화면이라면, 상세 화면은 어떤 내용을 렌더링해야 하는지 전달받아야 한다. `navigate` 함수를 이용할 때 두 번째 파라미터에 객체를 전달해서 이동하는 화면에 필요한 정보를 함께 전달하는 기능이 있다.

이번에는 navigate 함수를 이용하여, List 화면에서 목록을 클릭하면 해당 항목의 정보와 함께 Item 화면으로 이동하도록 수정해보자.

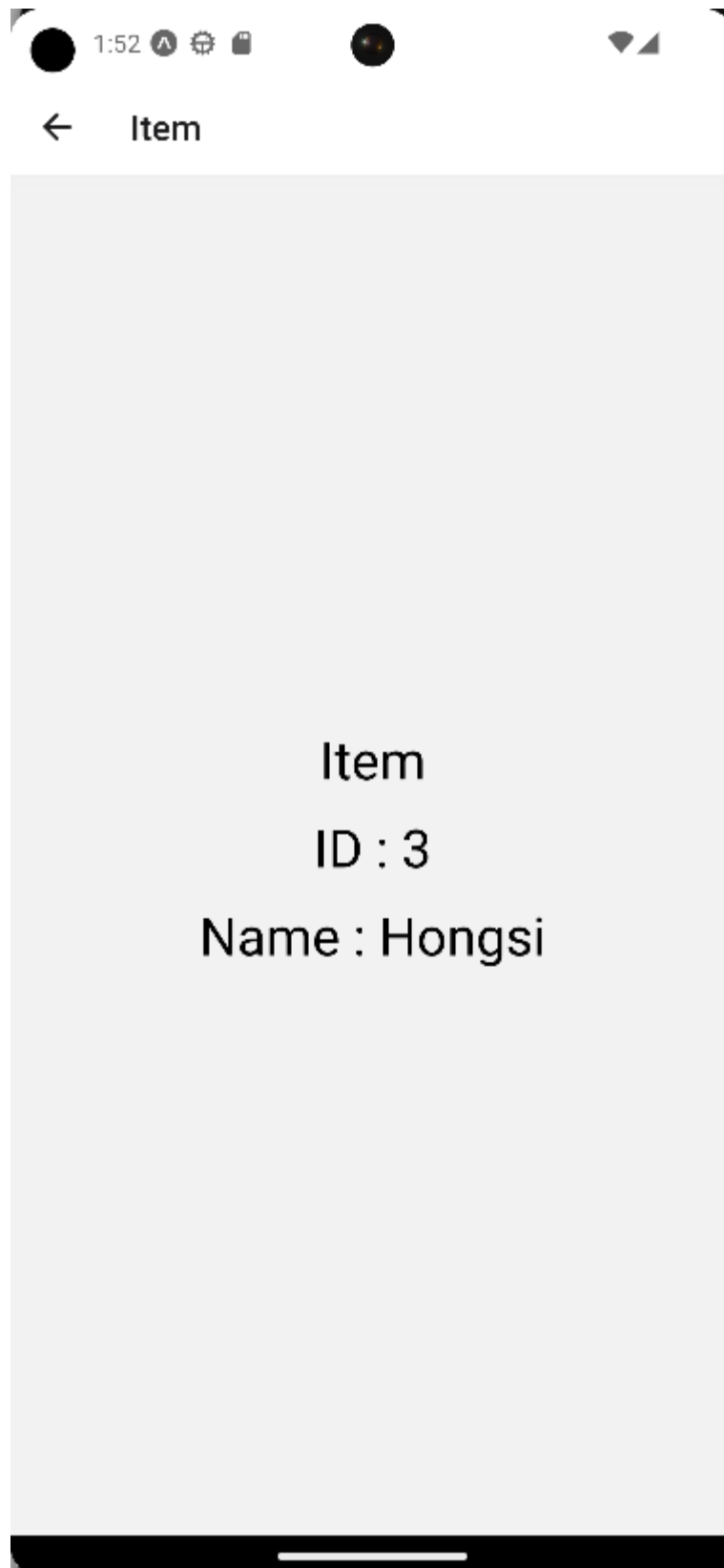


```
src > screens > JS Listjs > ...
1  import React from "react";
2  import { Button } from "react-native";
3  import styled from "styled-components/native";
4
5  const Container = styled.View`
6    flex : 1;
7    justify-content : center;
8    align-items : center;
9  `;
10 const StyledText = styled.Text`
11   font-size : 30px;
12   margin-bottom : 10px;
13 `;
14 const items = [
15   { _id: 1, name: 'React Native'},
16   { _id: 2, name: 'React Navigation'},
17   { _id: 3, name: 'Hongsi'},
18 ];
19
20 const List = ({navigation}) => {
21   const _onPress = item =>{
22     navigation.navigate('Item', {id : item._id, name: item.name});
23   };
24   return(
25     <Container>
26       <StyledText>List</StyledText>
27       {items.map(item => (
28         <Button
29           key={item._id}
30           title={item.name}
31           onPress={() => _onPress(item)}
32         />
33       ))}
34     </Container>
35   );
36 };
37
```

Item 화면으로 이동하면서 항목의 id와 name을 함께 전달하도록 작성했다. 전달된 내용은 컴포넌트의 props로 전달되는 route의 params를 통해 확인할 수 있다.

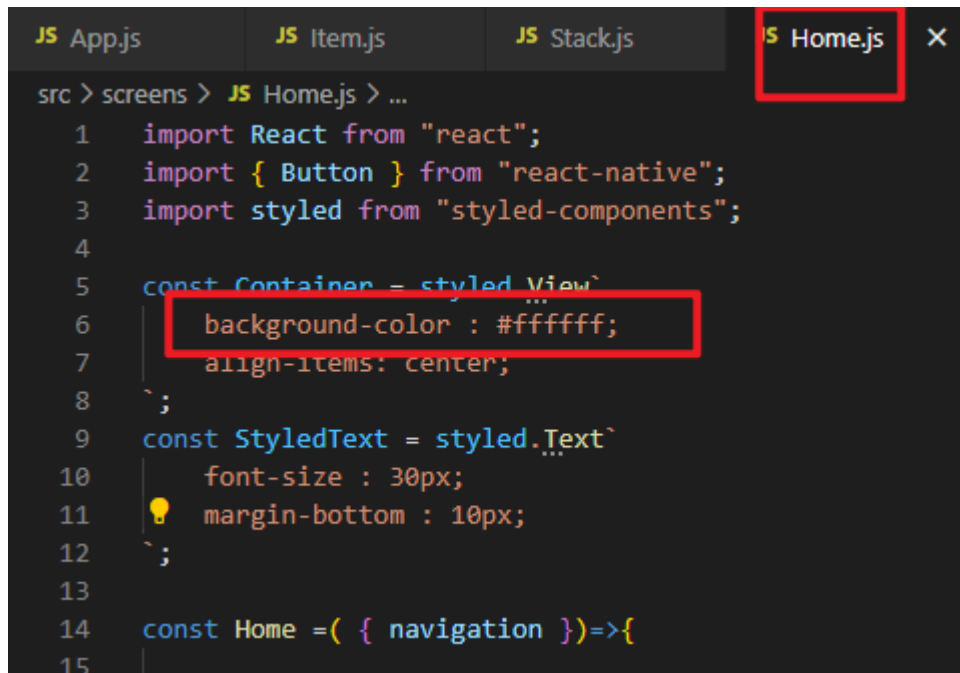
이번에는 Item 화면에서 전달되는 params를 이용하여 화면에 항목의 id와 name 을 출력해보자.

```
src > screens > JS Item.js > [0] Item
1  import React from "react";
2  import styled from "styled-components/native";
3
4  const Container = styled.View`
5    flex : 1;
6    justify-content : center;
7    align-items : center;
8  `;
9  const StyledText = styled.Text`
10   font-size : 30px;
11   margin-bottom : 10px;
12 `;
13
14  const Item = ({ route }) => {
15
16    return(
17      <Container>
18        <StyledText>Item</StyledText>
19        <StyledText>ID : {route.params.id}</StyledText>
20        <StyledText>Name : {route.params.name}</StyledText>
21      </Container>
22    );
23  };
24
25  export default Item;
```

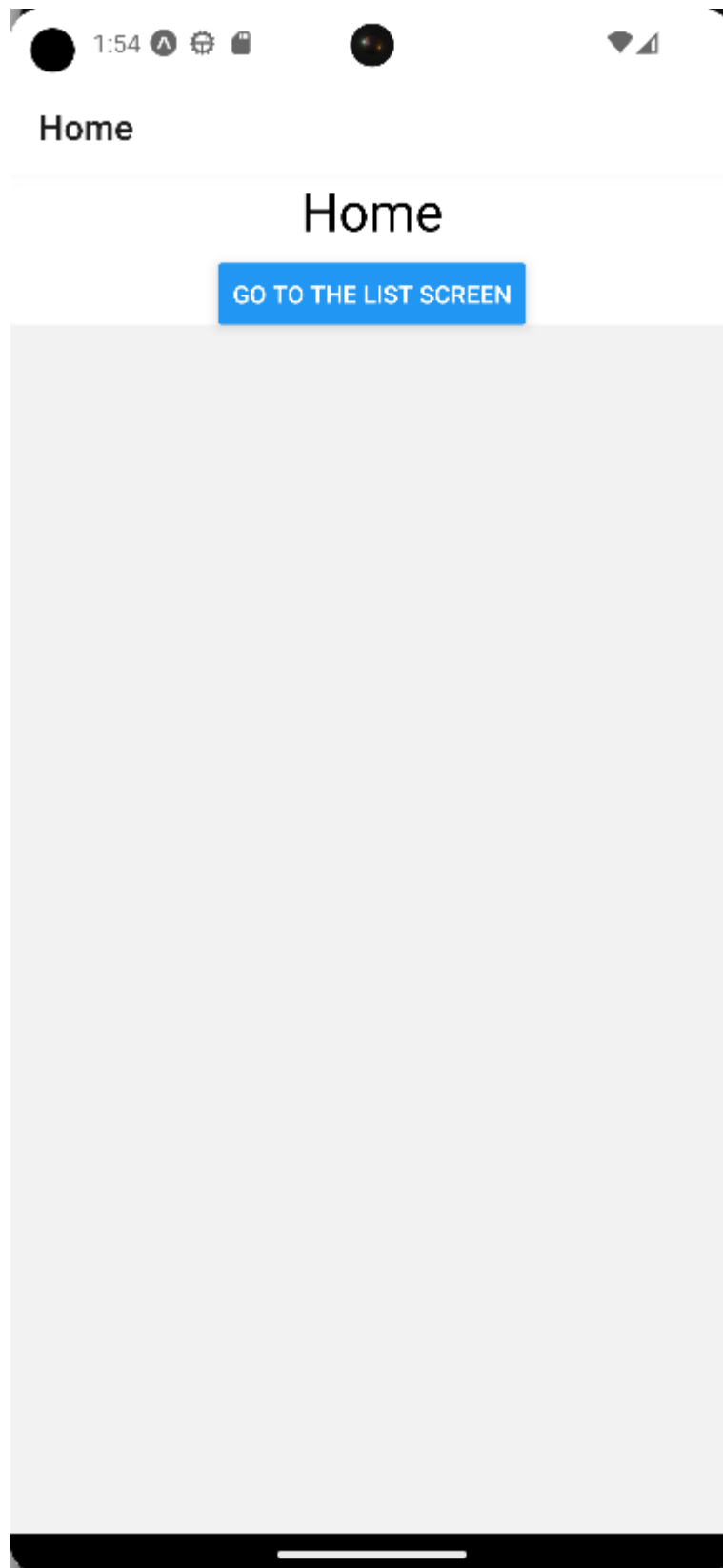


화면 배경색 수정하기

지금까지 사용한 화면은 배경색이 지정되지 않아 회색으로 나타났다. 이제 화면색을 지정해 보자.

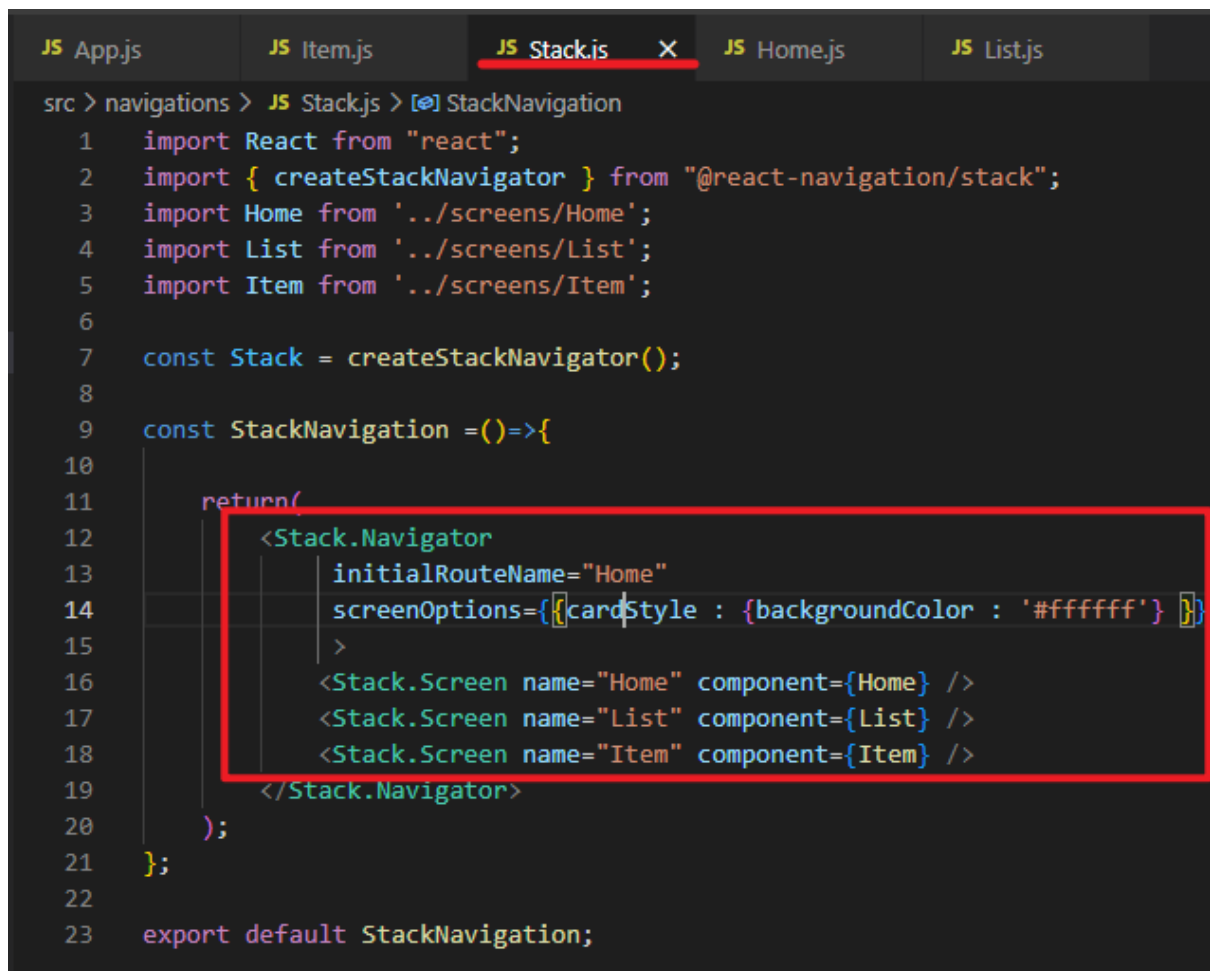


```
JS App.js JS Item.js JS Stack.js JS Home.js X
src > screens > JS Home.js > ...
1  import React from "react";
2  import { Button } from "react-native";
3  import styled from "styled-components";
4
5  const Container = styled.View`
6    background-color : #ffffff;
7    align-items: center;
8  `;
9  const StyledText = styled.Text`
10   font-size : 30px;
11   margin-bottom : 10px;
12 `;
13
14 const Home = ( { navigation } )=>{
15
```



내비게이션은 화면 전체를 차지하고 있지만 화면으로 사용된 컴포넌트의 영역이 전체를 차지하고 있지 않아서 나타나는 문제이다. 화면 컴포넌트가 전체 영역을 차지하도록 스타일에

flex : 1을 설정하면 문제를 해결할 수 있다. 하지만 상황에 따라 화면 전체를 차지하지 못하는 경우도 있는데, 이런 상황에서는 리액트 네비게이션의 설정을 수정하여 해결할 수 있다.



```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > StackNavigation
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = () => {
10
11    return(
12      <Stack.Navigator
13        initialRouteName="Home"
14        screenOptions={{cardStyle : {backgroundColor : '#ffffff'}} >
15      <Stack.Screen name="Home" component={Home} />
16      <Stack.Screen name="List" component={List} />
17      <Stack.Screen name="Item" component={Item} />
18    </Stack.Navigator>
19  );
20
21  };
22
23  export default StackNavigation;
```



cardStyle을 이용하면 스택 내비게이션의 화면 배경색을 수정할 수 있다. 화면의 배경색은 일반적으로 동일하게 사용하므로, 화면마다 설정하기보다 Navigator 컴포넌트의

screenOptions에 설정해서 화면 전체에 적용되도록 하는 것이 편하다.

만약 특정 화면만 배경색을 다르게 하고 싶다면 Screen 컴포넌트에서 설정하거나 화면으로 사용되는 컴포넌트에서 직접 배경색을 지정하면 된다.

헤더 수정하기

스택 네비게이션의 헤더(header)는 뒤로 가기 버튼을 제공하거나 타이틀(title)을 통해 현재 화면을 알려주는 역할을 한다. 이번에는 리액트 네비게이션에서 제공하는 다양한 속성들을 이용해 스택 네비게이션의 헤더를 변경하는 방법에 대해 알아보자.

타이틀 수정하기

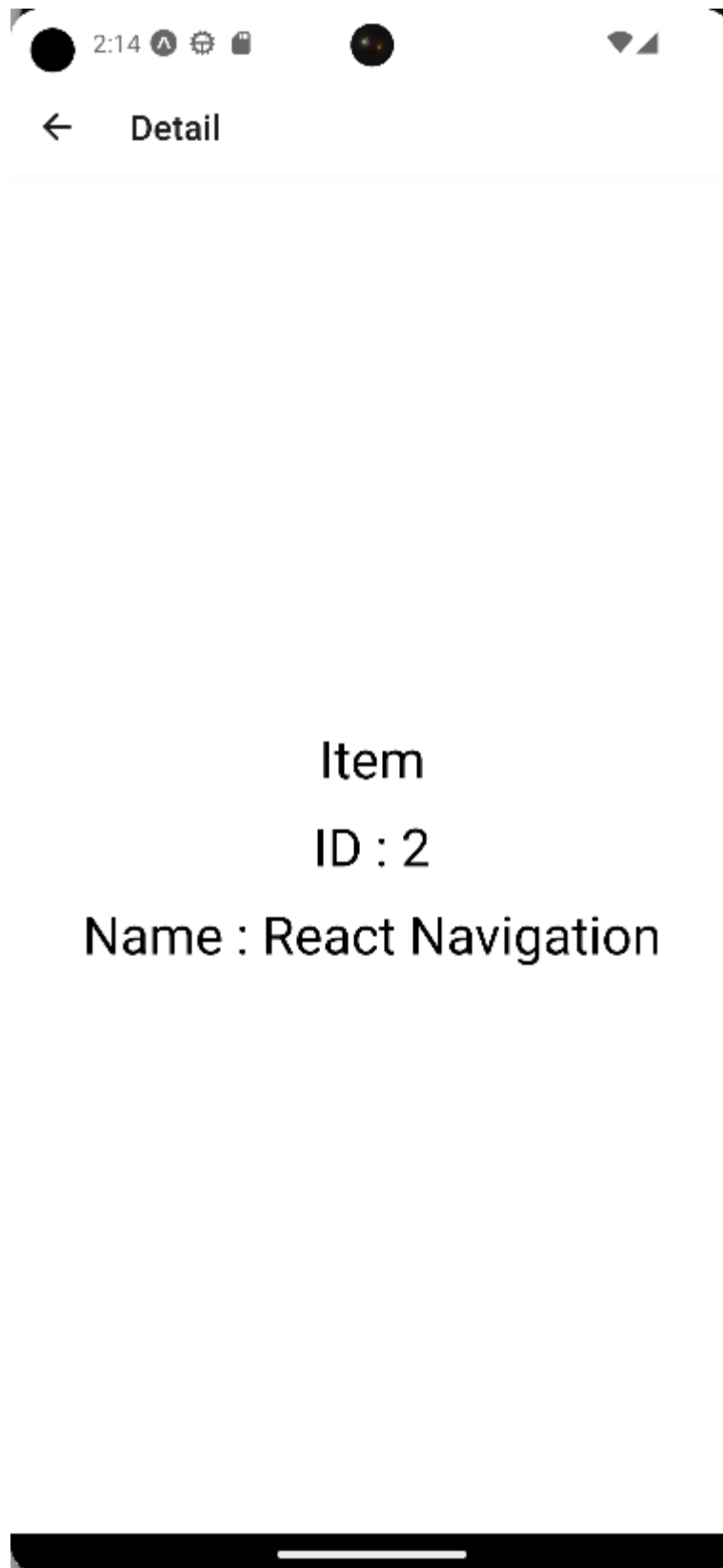
헤더의 타이틀은 Screen 컴포넌트의 name 속성을 기본값으로 사용한다. 헤더의 타이틀을 변경하는 가장 쉬운 방법은 name을 원하는 값으로 수정하는 것이다.


```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > [?] default
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = ()=>{
10
11    return(
12      <Stack.Navigator
13        initialRouteName="Home"
14        screenOptions={{cardStyle : {backgroundColor : '#ffffff'}} }}
15      >
16        <Stack.Screen name="Home" component={Home} />
17        <Stack.Screen name="List" component={List} />
18        <Stack.Screen name="Detail" component={Item} />
19      </Stack.Navigator>
20    );
21  };
22
23  export default StackNavigation;
```

Item 화면을 나타내는 Screen 컴포넌트의 name 속성을 Detail로 변경했다. name 의 값이 변경되었으므로 Item 화면으로 이동할 때 navigate 함수에 전달하는 첫 번째 파라미터값도 변경되어야 한다.

```
JS App.js JS Item.js JS Stack.js JS Home.js JS List.js X
src > screens > JS List.js > [🔍] List
1  import React from "react";
2  import { Button } from "react-native";
3  import styled from "styled-components/native";
4
5  const Container = styled.View`
6    flex : 1;
7    justify-content : center;
8    align-items : center;
9  `;
10 const StyledText = styled.Text`
11   font-size : 30px;
12   margin-bottom : 10px;
13 `;
14 const items = [
15   { _id: 1, name: 'React Native'},
16   { _id: 2, name: 'React Navigation'},
17   { _id: 3, name: 'Hongsi'},
18 ];
19
20 const List = ({navigation}) => {
21   const _onPress = item => {
22     navigation.navigate('Detail', {id : item._id, name: item.name});
23   };
24   return(
25     <Container>
26       <StyledText>List</StyledText>
27       {items.map(item => (
28         <Button
29           key={item._id}
30           title={item.name}
31           onPress={() => _onPress(item)}
32         />
33       ))}
34     </Container>
35   );
36 };
37
```

name 속성을 변경하는 것은 간편하지만, name 속성을 이용하는 곳을 찾아다니며 모두 수정해야 한다는 단점이 있다.



name 속성을 변경했을 때의 단점을 피하면서 화면 타이틀을 변경하는 방법은 headerTitle 속성을 이용하는 것이다.

```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > [x] default
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = ()=>{
10
11    return(
12      <Stack.Navigator
13        initialRouteName="Home"
14        screenOptions={{cardStyle : {backgroundColor : '#ffffff'} }}
15      >
16        <Stack.Screen name="Home" component={Home} />
17        <Stack.Screen
18          name="List"
19          component={List}
20          options={{headerTitle : 'List Screen' }}
21        />
22        <Stack.Screen name="Detail" component={Item} />
23      </Stack.Navigator>
24    );
25  };
26
27  export default StackNavigation;
```

List 화면의 타이틀을 List Screen으로 변경했다.

이렇게 개별 화면 설정을 수정하고 싶은 경우 Screen 컴포넌트의 options를 이용하면 된다.

모든 화면에서 같은 타이틀이 나타나도록 수정하고 싶다면 Navigator 컴포넌트의 screen Options 속성에 headerTitle을 지정하면 된다.

스타일 수정하기

이번에는 헤더의 스타일을 수정하는 방법에 대해 알아보자

헤더의 스타일을 수정하는 속성은 헤더의 배경색 등을 수정하는 headerStyle과 헤더의 타이틀 컴포넌트의 스타일을 수정하는 headerTitleStyle이 있다.

headerStyle과 headerTitleStyle 속성을 이용해서 모든 화면의 헤더 스타일을 변경해보자.

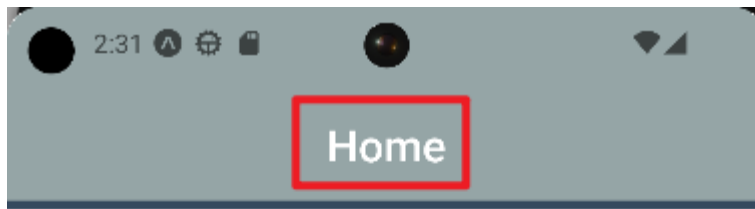
```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > StackNavigation
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = ()=>{
10
11    return(
12      <Stack.Navigator
13        initialRouteName="Home"
14        screenOptions={{
15          cardStyle : {backgroundColor : '#ffffff'},
16          headerStyle : {
17            height : 110,
18            backgroundColor : '#95a5a6',
19            borderBottomWidth : 5,
20            borderBottomColor : '#34495e',
21          },
22          headerTitleStyle : { color : '#ffffff', fontSize : 24},
23        }}
24      >
25        <Stack.Screen name="Home" component={Home} />
26        <Stack.Screen
27          name="List"
28          component={List}
29          options={{headerTitle : 'List Screen' }}
30        />
31        <Stack.Screen name="Detail" component={Item} />
32      </Stack.Navigator>
33    );
34  };
35
36  export default StackNavigation;
```

headerStyle을 이용해 헤더의 스타일을 변경하고, headerTitleStyle을 이용해 타이틀의 스타일을 변경했다.

스타일은 잘 적용되었지만, 안드로이드에서는 IOS와 달리 타이틀이 중앙으로 정렬되지 않은 것을 볼 수 있다.

타이틀 정렬을 두 플랫폼에서 동일하게 하려면 headerTitleAlign 속성을 이용해야 한다. headerTitleAlign에는 left와 center 두가지 값 중 한 가지만 설정할 수 있고 ios는 center, 안드로이드는 left가 기본값으로 설정되어 있다.

```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > [x] StackNavigation
1  import React from "react";
2  import { createStackNavigator } from "@react-navigation/stack";
3  import Home from '../screens/Home';
4  import List from '../screens/List';
5  import Item from '../screens/Item';
6
7  const Stack = createStackNavigator();
8
9  const StackNavigation = () => {
10
11    return(
12      <Stack.Navigator
13        initialRouteName="Home"
14        screenOptions={{
15          cardStyle : {backgroundColor : '#ffffff'},
16          headerStyle : {
17            height : 110,
18            backgroundColor : '#95a5a6',
19            borderBottomWidth : 5,
20            borderBottomColor : '#34495e',
21          },
22          headerTitleStyle : { color : '#ffffff', fontSize : 24},
23          headerTitleAlign : 'center',
24        }}
25      >
26      <Stack.Screen name="Home" component={Home} />
27      <Stack.Screen
```



Home

GO TO THE LIST SCREEN



타이틀 컴포넌트 변경

만약 타이틀에 문자열이 아닌 다른 것을 렌더링하고 싶다면 어떻게 해야 할까. 헤더의 타이틀을 변경하기 위해 문자열을 지정했던 `headerTitle` 속성에 컴포넌트를 반환하는 함수를 지정하면 타이틀 컴포넌트를 반환하는 컴포넌트로 변경할 수 있다. `headerTitle`에 함수가 설정되면 해당 함수의 파라미터로 `style`과 `tintColor`등이 포함된 객체가 전달된다. 그중 `style`은 `headerTitleStyle`에 설정된 값이고, `tintColor`는 `headerTintColor`에 지정된 값이 전달된다.

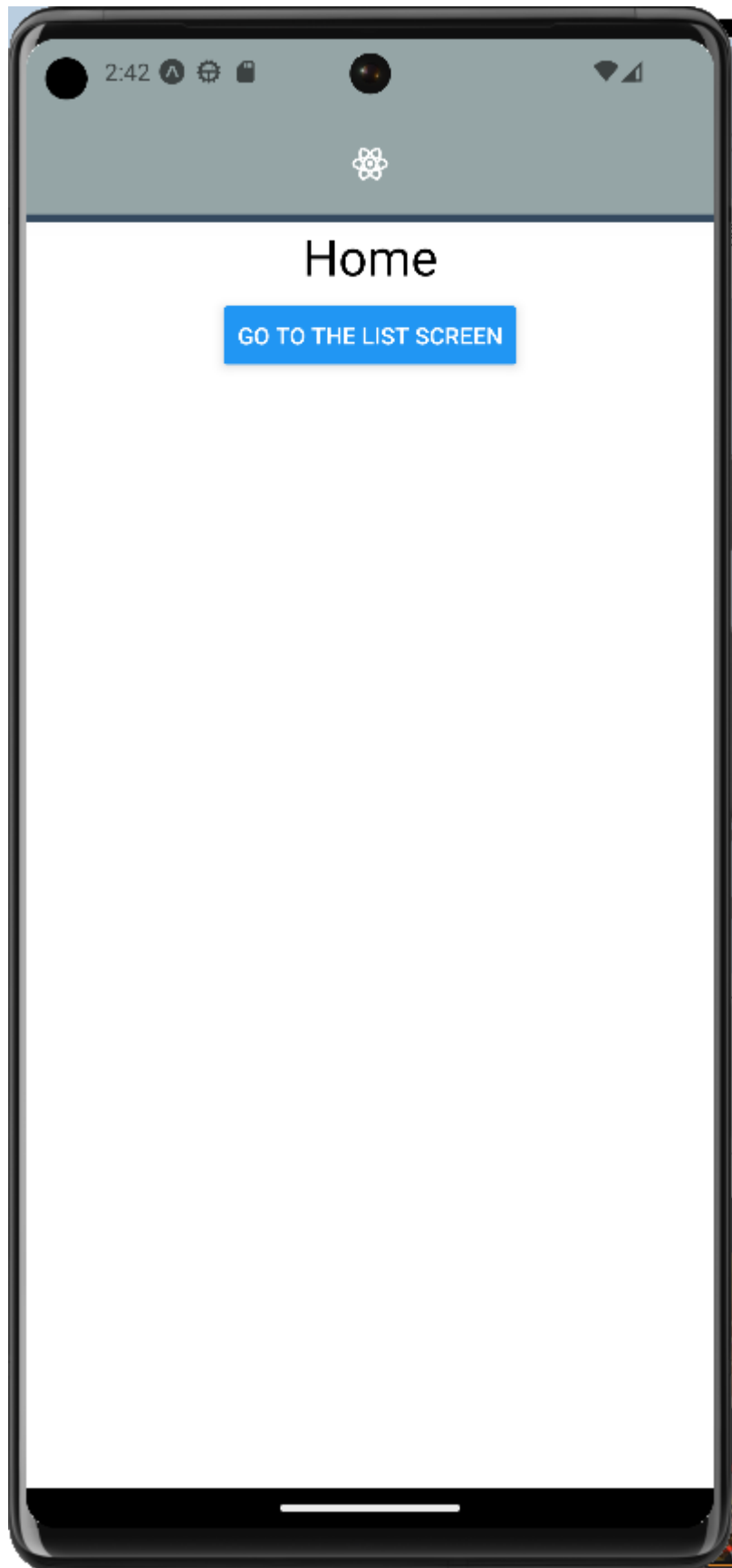
이번에는 `headerTitle`에 컴포넌트를 반환하는 함수를 지정해서 타이틀 컴포넌트를 교체해보자.


```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > StackNavigation > headerTitle
1 import React from "react";
2 import { createStackNavigator } from "@react-navigation/stack";
3 import Home from '../screens/Home';
4 import List from '../screens/List';
5 import Item from '../screens/Item';
6 import { MaterialCommunityIcons } from '@expo/vector-icons';
7
8 const Stack = createStackNavigator();
9
10 const StackNavigation = ()=>{
11
12   return(
13     <Stack.Navigator
14       initialRouteName="Home"
15       screenOptions={{
16         cardStyle : {backgroundColor : '#ffffff'},
17         headerStyle : {
18           height : 110,
19           backgroundColor : '#95a5a6',
20           borderBottomWidth : 5,
21           borderBottomColor : '#34495e',
22         },
23       headerTitleStyle : { color : '#ffffff', fontSize : 24},
24       headerTitleAlign : 'center',
25       headerTitle : ({style}) => (
26         <MaterialCommunityIcons name = "react" style={style} />
27       ),
28     }}
29   >
30     <Stack.Screen name="Home" component={Home} />
31     <Stack.Screen
32       name="List"
33       component={List}
34       options={{headerTitle : 'List Screen' }}
35     />
36   </Stack.Navigator>
37 )
38 }
```

함수의 파라미터로 전달되는 style을 이용하여 headerTitleStyle에 지정한 스타일과 동일한 스타일이 적용되도록 작성했다. 반환되는 컴포넌트는 vector-icons에서 제공하는 컴포넌트를 이용해서 리액트 로고가 렌더링 되도록 작성했다.

※ vector-icons는 expo 프로젝트에서 기본적으로 설치되는 라이브러리이다. 많이 사용되는 아이콘이 있으며, 다음 링크에서 사용 가능한 아이콘을 확인하고 검색 할 수 있다.

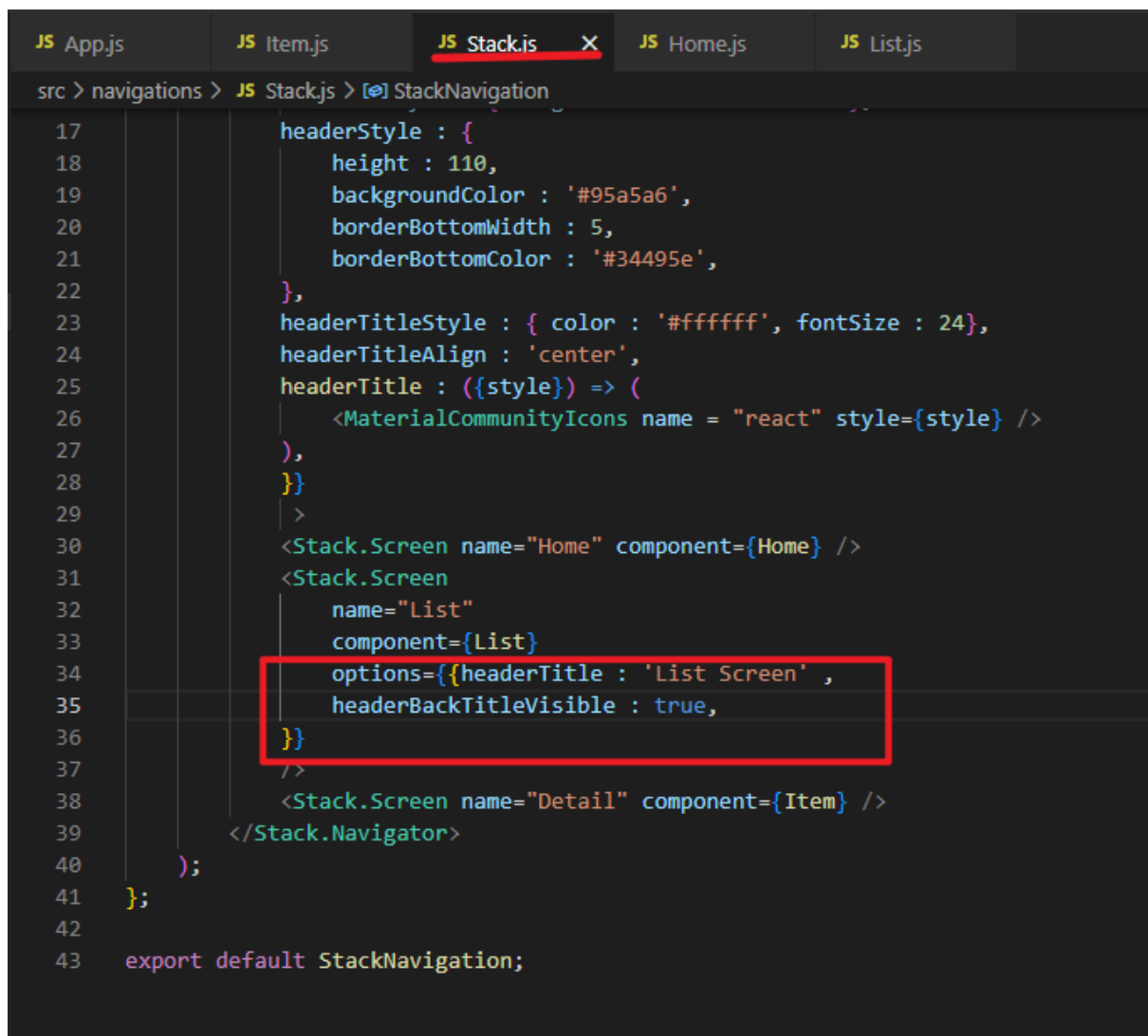
vector-icons directory : <https://icons.expo.fyi/>



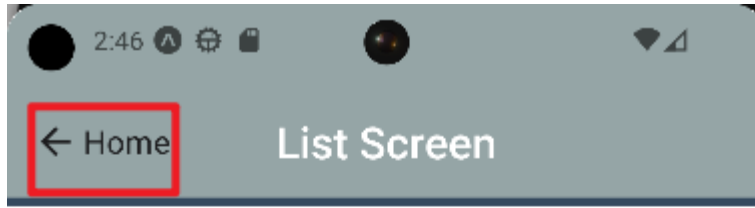
버튼 수정하기

스택 네비게이션에서 화면을 이동하면 헤더 왼쪽에 이전 화면으로 이동하는 뒤로가기 버튼이 나타난다. 만약 첫 화면처럼 이전 화면이 없는 경우에는 버튼이 생기지 않는다. 뒤로가기 버튼을 이용하는 방법 외엔 화면 왼쪽 끝에서 오른쪽 방향으로 스와이프 제스처를 통해 이전 화면으로 돌아가는 방법도 있다. 이번에는 헤더 왼쪽에 있는 버튼을 수정하는 방법에 대해 알아보자.

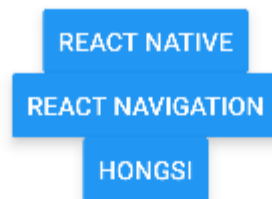
헤더 왼쪽에 뒤로 가기 버튼이 생성되는 것은 동일하지만, ios와 안드로이드의 모습에는 차이가 있다. 안드로이드는 버튼의 타이틀을 보여주지 않고, ios는 이전 화면의 타이틀을 버튼의 타이틀로 보여주는 것이 가장 눈에 띄는 차이점이다. `headerBackTitleVisible`을 이용하면 두 플랫폼의 버튼 타이틀 렌더링 여부를 동일하게 설정할 수 있다.



```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > StackNavigation
17 headerStyle : {
18   height : 110,
19   backgroundColor : '#95a5a6',
20   borderBottomWidth : 5,
21   borderBottomColor : '#34495e',
22 },
23 headerTitleStyle : { color : '#ffffff', fontSize : 24},
24 headerTitleAlign : 'center',
25 headerTitle : ({style}) => (
26   <MaterialCommunityIcons name = "react" style={style} />
27 ),
28 },
29 >
30 <Stack.Screen name="Home" component={Home} />
31 <Stack.Screen
32   name="List"
33   component={List}
34   options={{headerTitle : 'List Screen' ,
35             headerBackTitleVisible : true,
36           }}
37 />
38 <Stack.Screen name="Detail" component={Item} />
39 </Stack.Navigator>
40 );
41 };
42
43 export default StackNavigation;
```



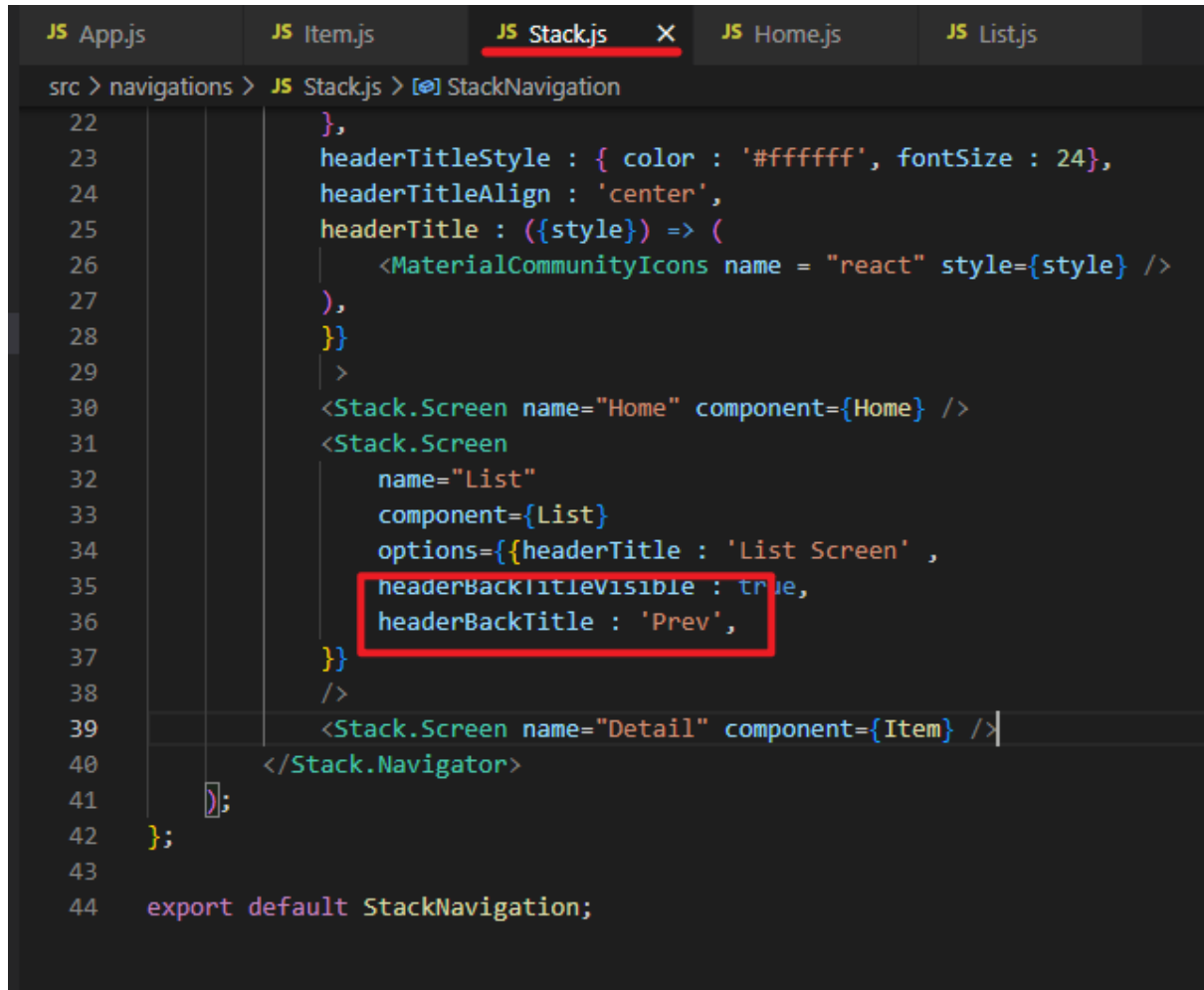
List



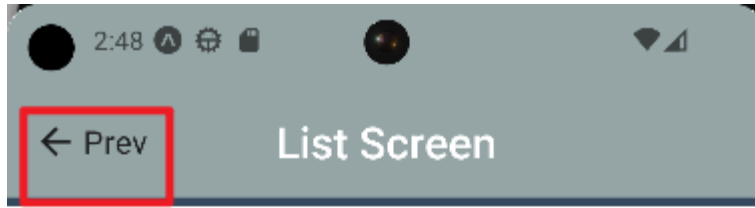
결과를 보면 안드로이드에도 버튼의 타이틀이 나타나는 것을 알 수 있다.

headerBackTitleVisible값을 변경하면서 버튼의 타이틀이 동일하게 나타나는지, 나타나지 않는지 확인해보자.

버튼의 타이틀은 나타나게 하지만, 이전 화면의 이름이 아닌 다른 값을 이용하고 싶은 경우 `headerBackTitle`을 이용한다.



```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > StackNavigation
22     },
23     headerTitleStyle : { color : '#ffffff', fontSize : 24},
24     headerTitleAlign : 'center',
25     headerTitle : ({style}) => (
26       <MaterialCommunityIcons name = "react" style={style} />
27     ),
28   }}
29   >
30   <Stack.Screen name="Home" component={Home} />
31   <Stack.Screen
32     name="List"
33     component={List}
34     options={{headerTitle : 'List Screen' ,
35              headerBackTitleVisible : true,
36              headerBackTitle : 'Prev',
37            }}
38   />
39   <Stack.Screen name="Detail" component={Item} />
40 </Stack.Navigator>
41 );
42 };
43
44 export default StackNavigation;
```



List

REACT NATIVE

REACT NAVIGATION

headerBackTitle에 빈 문자열을 입력하면 버튼 타이틀에 back이라는 문자열이 나타난다. 버튼의 타이틀에 빈 값을 주고 싶은 경우 headerBackTitle이 아니라 headerBackTitleVisible을 이용해서 타이틀이 보이지 않도록 해야한다는 것을 주의하자.

버튼 스타일 수정하기

버튼의 타이틀도 헤더의 타이틀처럼 우리가 원하는 스타일을 지정할 수 있다.

headerBackTitleStyle을 이용하면 글자의 색뿐만 아니라 글자 크기 등 다양한 스타일을 지정할 수 있지만 버튼의 타이틀에만 적용된다. 버튼의 타이틀과 이미지의 색을 동일하게 변경하려면 headerTintColor를 이용해야 한다.

headerTintColor에 지정된 색은 버튼뿐만 아니라 헤더의 타이틀에도 적용되지만, headerTitleStyle 혹은 headerBackTitleStyle이 우선순위가 높으므로 headerTintColor에 설정한 색으로 나타나게 하고 싶다면 다른 스타일과 겹치지 않도록 작성하는 것이 중요하다.

```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > StackNavigation
22     },
23     headerTitleStyle : { color : 'ffffff', fontSize : 24},
24     headerTitleAlign : 'center',
25     headerTitle : ({style}) => (
26       <MaterialCommunityIcons name = "react" style={style} />
27     ),
28   }}
29   >
30   <Stack.Screen name="Home" component={Home} />
31   <Stack.Screen
32     name="List"
33     component={List}
34     options={{headerTitle : 'List Screen' ,
35       headerBackTitleVisible : true,
36       headerBackTitle : 'Prev',
37       headerTitleStyle : {fontSize : 24},
38       headerTintColor : '#e74c3c',
39     }}
40   />
41   <Stack.Screen name="Detail" component={Item} />
42 </Stack.Navigator>
43 );
44 ];
45
46 export default StackNavigation;
```

headerTintColor 의 값을 설정하고 headerTitleStyle을 재정의하여 헤더의 타이틀에도 함께 적용되도록 수정했다.

버튼 컴포넌트 변경

두 플랫폼의 뒤로가기 버튼 아이콘에 동일한 아이콘이 렌더링 되도록 변경하려면 어떻게 해야할까 headerBackImage에 컴포넌트를 반환하는 함수를 전달해서 두 플랫폼이 동일한 이미지를 렌더링하도록 변경할 수 있다.

```
JS App.js  X  JS Item.js  JS Stack.js  X  JS Home.js  JS List.js
src > navigations > JS Stack.js > StackNavigation
7  import { Platform } from "react-native";
8
9  const Stack = createStackNavigator();
10
11  const StackNavigation = ()=>{
12
13    return(
14      <Stack.Navigator
15 >      initialRouteName="Home" ...
29    >>
30      <Stack.Screen name="Home" component={Home} />
31      <Stack.Screen
32        name="List"
33        component={List}
34        options={{headerTitle : 'List Screen' ,
35                  headerBackTitleVisible : true,
36                  headerBackTitle : 'Prev',
37                  headerTitleStyle : {fontSize : 24},
38                  headerTintColor : '#e74c3c',
40
41                  headerBackImage : ({tintColor})=> {
42                    const style = {
43                      marginRight : 5,
44                      marginLeft : Platform.OS === 'ios' ? 11 : 0,
45                    };
46                    return(
47                      <MaterialCommunityIcons
48                        name="keyboard-backspace"
49                        size={30}
50                        color={tintColor}
51                        style={style}
52                      />
53                    );
54  };
```

headerBackImage의 함수 파라미터에 전달되는 tintColor 값을 이용해 아이콘의 색을 지정하고, 두 플랫폼의 버튼 위치를 동일하게 만들기 위해 플랫폼에 따라 스타일을 다르게 적용했다.

뒤로 가기 버튼의 이미지가 아니라 헤더의 왼쪽 버튼 전체를 변경하고 싶다면 headerLeft에 컴포넌트를 반환하는 함수를 지정한다. 이와 동일한 방법으로 headerRight에 컴포넌트를 반환하는 함수를 지정하면 헤더의 오른쪽에 원하는 컴포넌트를 렌더링할 수 있다.


```
JS App.js JS Item.js X JS Stack.js JS Home.js JS List.js
src > screens > JS Item.js > [⌕] Item
1  import React, {useLayoutEffect} from "react";
2  import styled from "styled-components/native";
3  import {MaterialCommunityIcons} from '@expo/vector-icons';
4
5
6  > const Container = styled.View` ...
10 `;
11 > const StyledText = styled.Text` ...
14 `;
15
16 const Item = ({ navigation, route }) => {
17   useLayoutEffect(() => {
18     navigation.setOptions({
19       headerBackTitleVisible : false,
20       headerTintColor : '#ffffff',
21       headerLeft : ({onPress, tint_color}) => {
22         return(
23           <MaterialCommunityIcons
24             name="keyboard-backspace"
25             size = {30}
26             style={{marginLeft : 11}}
27             color={tint_color}
28             onPress={onPress}
29           />
30         );
31       },
32       headerRight : ({tint_color}) => {
33         <MaterialCommunityIcons
34           name = "home-variant"
35           size={30}
36           style={{marginRight: 11}}
37           color={tint_color}
38           onPress={()=>navigation.popToTop()}
39         />
40       },
41     });
42   }, []);
43   return(
```

useLayoutEffect Hook은 useEffect Hook 과 사용법이 동일하며 거의 같은 방식으로 동작한다. 주요 차이점은 컴포넌트가 업데이트된 직후에 화면이 렌더링 되기 전에 실행된다는 것이다. 이 특징 때문에 화면을 렌더링하기 전에 변경할 부분이 있거나 수치 등을 측정해야 하는 상황에서 많이 사용된다.

headerLeft 함수의 파라미터에는 다양한 값들이 전달되는데, 그 중 onPress는 뒤로 가기 버튼 기능이 전달된다. 화면의 왼쪽 버튼을 변경하면서 전혀 다른 기능을 설정하는 경우에는 필요 없지만, 뒤로 가기 버튼의 기능을 그대로 기용하고 싶은 경우 유용하게 사용할 수 있다.

headerRight 함수의 파라미터에는 tintColor만 전달되므로 onPress에 원하는 행동을 정의해줘야 한다. navigation에서 제공하는 다양한 함수 중 popToTop 함수는 현재 쌓여 있는 모든 화면을 내보내고 첫 화면으로 돌아가는 기능이다. 오른쪽 버튼의 onPress 이벤트는 popToTop 함수를 이용해서 첫 화면으로 돌아가도록 작성했다.



Item

ID : 1

Name : React Native



헤더 감추기

화면 종류나 프로젝트 기획에 따라 헤더를 감춰야 하는 상황이 있다. 이런 상황에서 이용할 수 있는 설정은 `headerMode`나 `headerShown`이다.

`headerMode`는 `Navigator`컴포넌트의 속성으로 헤더를 렌더링하는 방법을 설정하는 속성이다

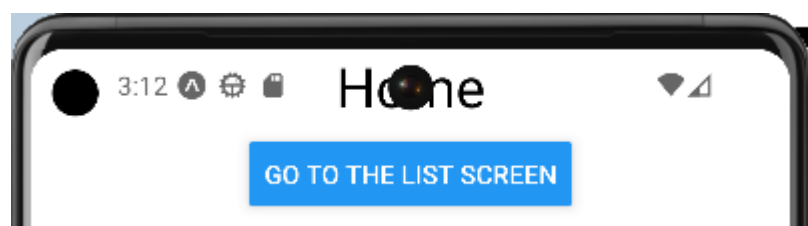
- `float` : 헤더가 상단에 유지되며 하나의 헤더를 사용한다.
- `screen` : 각 화면마다 헤더를 가지며 화면 변경과 함께 나타나거나 사라진다.
- `none` : 헤더가 렌더링되지 않는다

`float`은 ios에서 볼 수 있는 동작 방식이고, `screen`은 안드로이드에서 일반적으로 볼 수 있는 동작 방식이다. `headerMode`를 `none`으로 설정하면 자식 컴포넌트로 존재하는 모든 화면에서 헤더가 보이지 않는다

`headerShown` 은 화면 옵션으로, `Navigator` 컴포넌트의 `screenOptions`에 설정하면 전체 화면의 헤더가 보이지 않도록 설정할 수 있다.

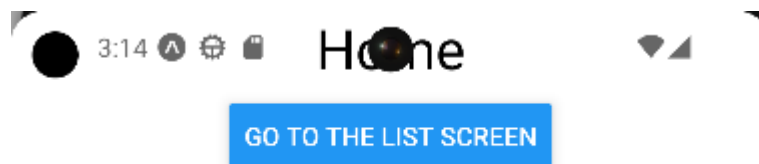
일단 첫 화면인 Home화면에서만 헤더가 보이지 않도록 수정해보자.

```
JS App.js JS Item.js JS Stack.js X JS Home.js JS List.js
src > navigations > JS Stack.js > [0] StackNavigation
7 import { Platform } from "react-native";
8
9 const Stack = createStackNavigator();
10
11 const StackNavigation = ()=>{
12
13   return(
14     <Stack.Navigator
15 >   initialRouteName="Home" ...
29   >>
30   >
31   <Stack.Screen |
32     name="Home"
33     component={Home}
34     options={{headerShown : false}}
35   />
36   <Stack.Screen
37     name="List"
38     component={List}
39     options={{headerTitle : 'List Screen' ,
40             headerBackTitleVisible : true,
41             headerBackTitle : 'Prev',
42             headerTitleStyle : {fontSize : 24},
43             headerTintColor : '#e74c3c',
44
45             headerBackImage : ({tintColor})=> {
46               const style = {
47                 marginRight : 5,
```



헤더가 사라지면서 노치 디자인 문제로 화면의 일부가 가려지는 문제를 해결하기 위해 Home화면을 다음과 같이 수정하자.

```
JS App.js JS Item.js JS Stack.js JS Home.js X JS List.js
src > screens > JS Home.js > [e] StyledText
1 import React from "react";
2 import { Button } from "react-native";
3 import styled from "styled-components";
4
5 const Container = styled.SafeAreaView`
6   background-color : #ffffff;
7   align-items: center;
8 `;
9
10 const StyledText = styled.Text`
11   font-size : 30px;
12   margin-bottom : 10px;
13 `;
14
15 const Home = ( { navigation } )=>{
```



적용해도 변하지 않았다...

탭 내비게이션

탭 내비게이션은 보통 화면 위나 아래에 위치하며, 탭 버튼을 누르면 버튼과 연결된 화면으로 이동하는 방식으로 동작한다. 많이 사용되는 카카오톡, 왓츠앱 등의 채팅 애플리케이션에서 쉽게 확인할 수 있다. 그 외에도 인스타그램, 유튜브 등 다수의 애플리케이션에서 탭 내비게이션을 사용하여 화면을 구성한다.

그림p.284

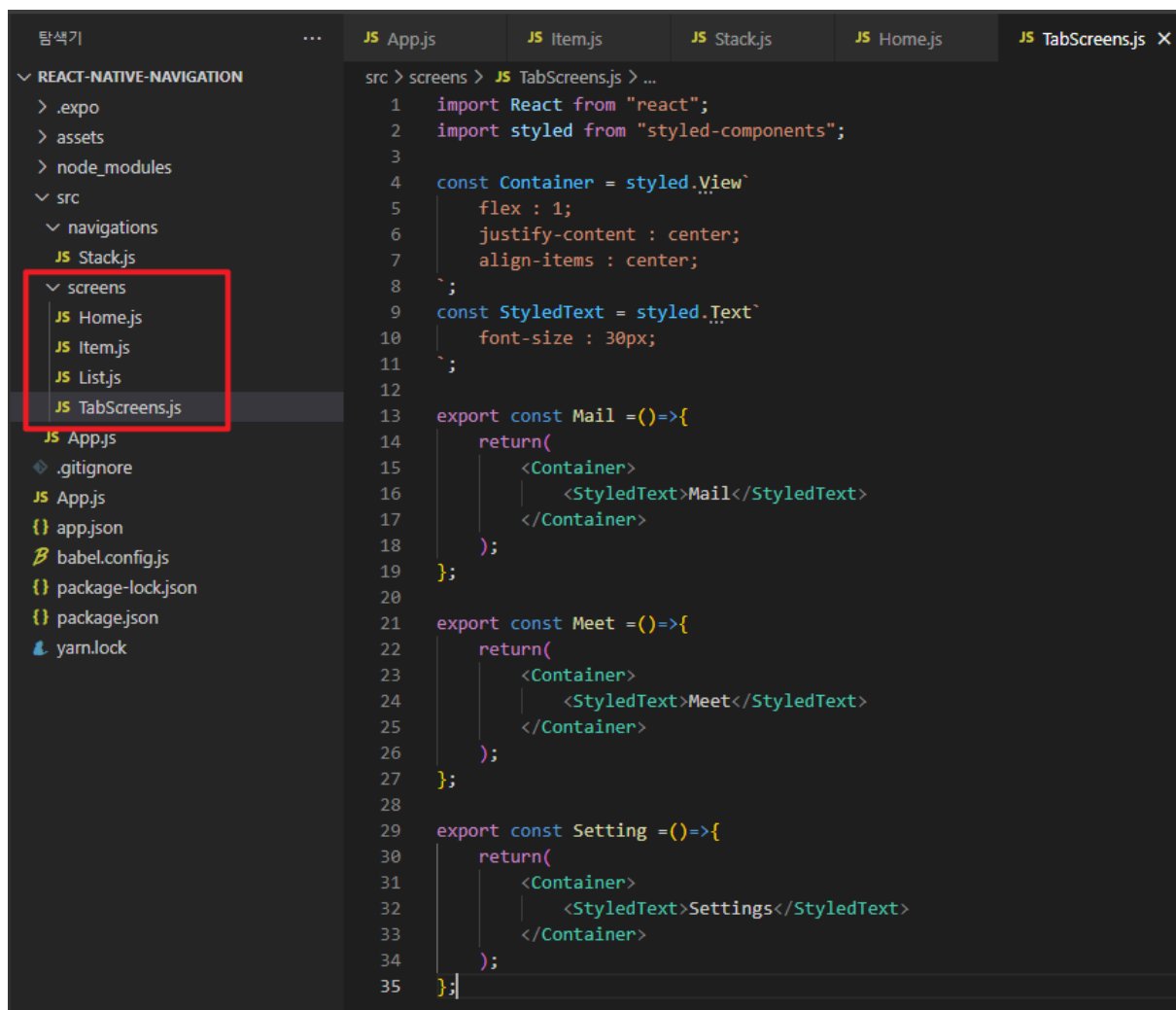
리액트 내비게이션의 탭 내비게이션을 이용하여 화면을 구성하는 방법에 대해 알아보자. 먼저 탭 내비게이션을 이용하기 위한 추가 라이브러리를 설치하자

- `npm install @react-navigation/bottom-tabs —force`

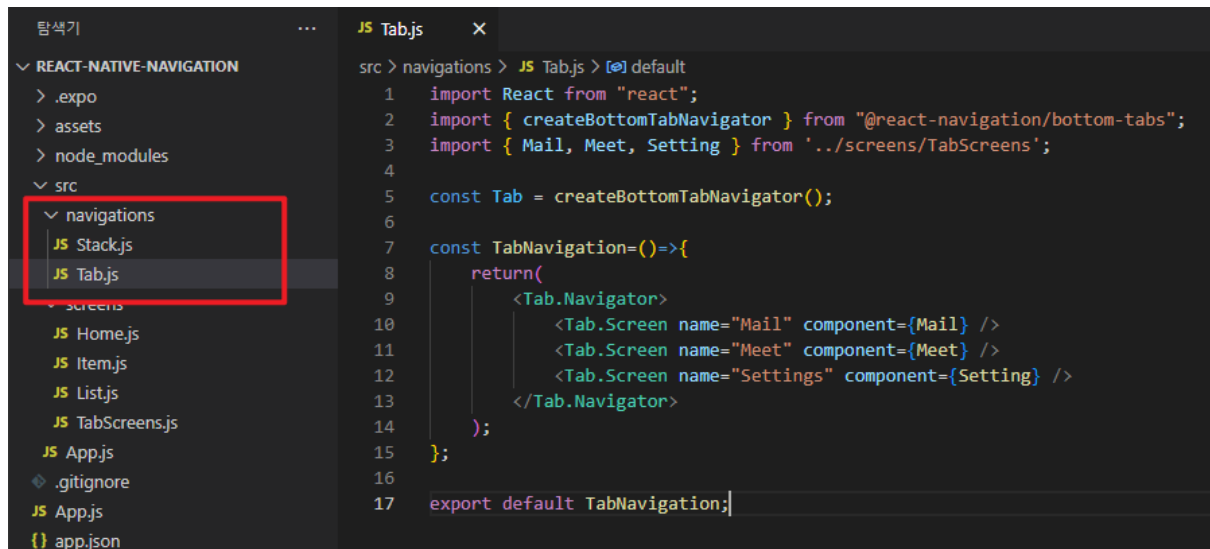
화면 구성

이번에는 3개의 버튼과 해당 버튼에 연결된 화면으로 구성된 탭 내비게이션을 만들어보자. src폴더 밑에 있는 screens 폴더 내에 TabScreen.js 파일을 만들고 화면으로 사용할 컴포넌트를 아래 처럼 작성하자.

※탭 내비게이션에서는 화면에서 특별한 작업을 하지 않을 예정이므로 하나의 파일에 모든 화면을 함께 작성한다. 스택 내비게이션처럼 화면 각각의 파일을 따로 만들어도 상관 없다.



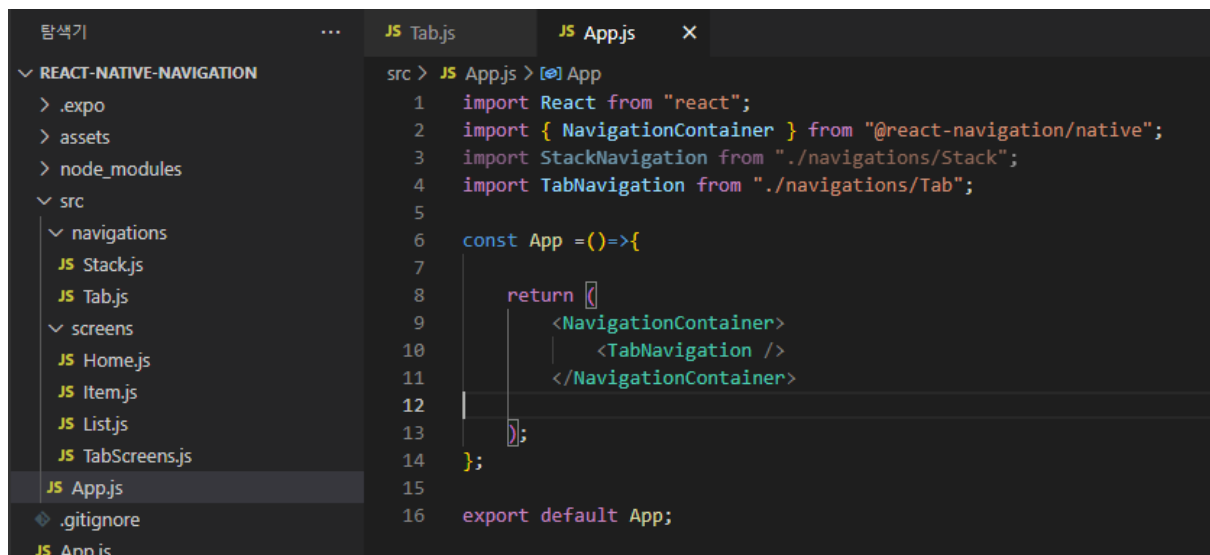
현재 화면을 확인할 수 있는 텍스트가 나타나는 간단한 컴포넌트를 3개 만들었다. 이제 생성된 컴포넌트를 이용해서 탭 내비게이션을 만들어보자.



```
src > navigations > JS Tab.js > [🔍] default
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Setting } from "../screens/TabScreens";
4
5  const Tab = createBottomTabNavigator();
6
7  const TabNavigation=()=>{
8    return(
9      <Tab.Navigator>
10        <Tab.Screen name="Mail" component={Mail} />
11        <Tab.Screen name="Meet" component={Meet} />
12        <Tab.Screen name="Settings" component={Setting} />
13      </Tab.Navigator>
14    );
15  };
16
17  export default TabNavigation;
```

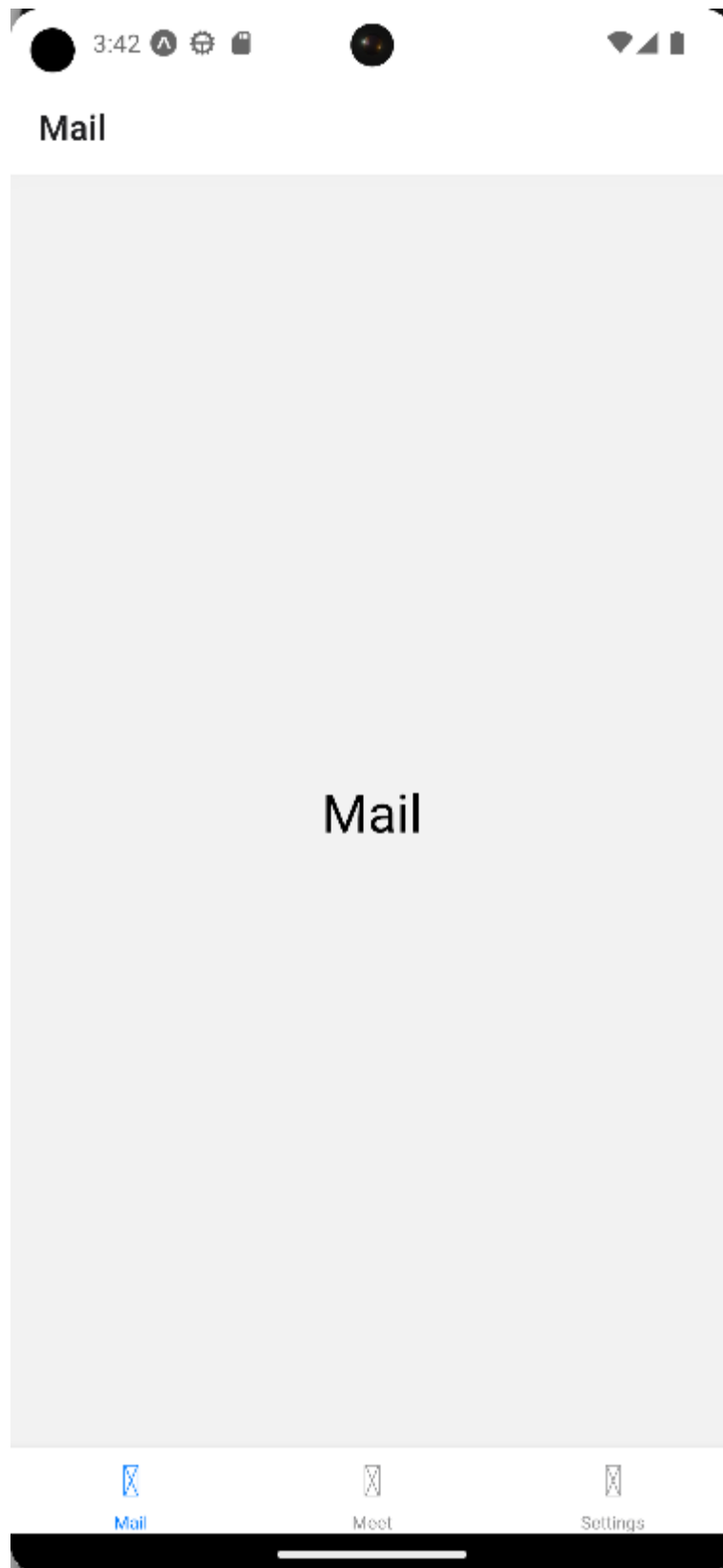
createBottomTabNavigator 함수를 이용해 탭 내비게이션을 생성했다. 탭 내비게이션에도 스택 내비게이션과 동일하게 Navigator 컴포넌트, Screen 컴포넌트가 있다. 앞에서 만든 컴포넌트들을 Screen 컴포넌트의 component로 지정해 화면으로 사용하고 Navigator 컴포넌트로 감싸준다.

이제 App컴포넌트에서 사용해보자.

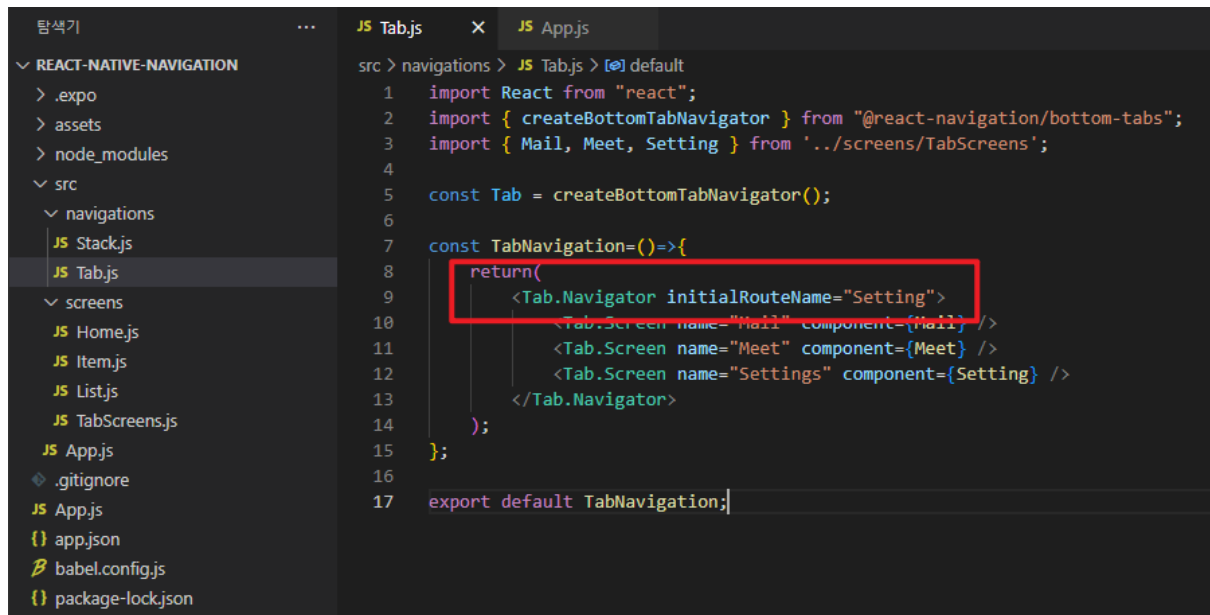


```
src > JS App.js > [🔍] App
1  import React from "react";
2  import { NavigationContainer } from "@react-navigation/native";
3  import StackNavigation from "../navigations/Stack";
4  import TabNavigation from "../navigations/Tab";
5
6  const App =()=>{
7
8    return [
9      <NavigationContainer>
10        <TabNavigation />
11      </NavigationContainer>
12    ];
13  };
14
15
16  export default App;
```

결과를 보면 화면 하단에 3개 버튼이 놓인 탭 바가 있고, 탭의 버튼을 클릭할 때마다 화면이 변경되는 것을 확인 할 수 있다.



탭 바에 있는 버튼 순서는 Navigator 컴포넌트의 자식으로 있는 Screen 컴포넌트의 순서와 동일하며 첫 번째 자식 컴포넌트를 첫 화면으로 사용한다. 탭 버튼의 순서는 변경하지 않고 렌더링 되는 첫 번째 화면을 변경하고 싶은 경우 `initialRouteName` 속성을 이용한다.



```
src > navigations > JS Tab.js > default
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Setting } from "../screens/TabScreens";
4
5  const Tab = createBottomTabNavigator();
6
7  const TabNavigation={()=>{
8    return(
9      <Tab.Navigator initialRouteName="Setting">
10        <Tab.Screen name="Mail" component={Mail} />
11        <Tab.Screen name="Meet" component={Meet} />
12        <Tab.Screen name="Settings" component={Setting} />
13      </Tab.Navigator>
14    );
15  };
16
17  export default TabNavigation;
```

탭 바 수정하기

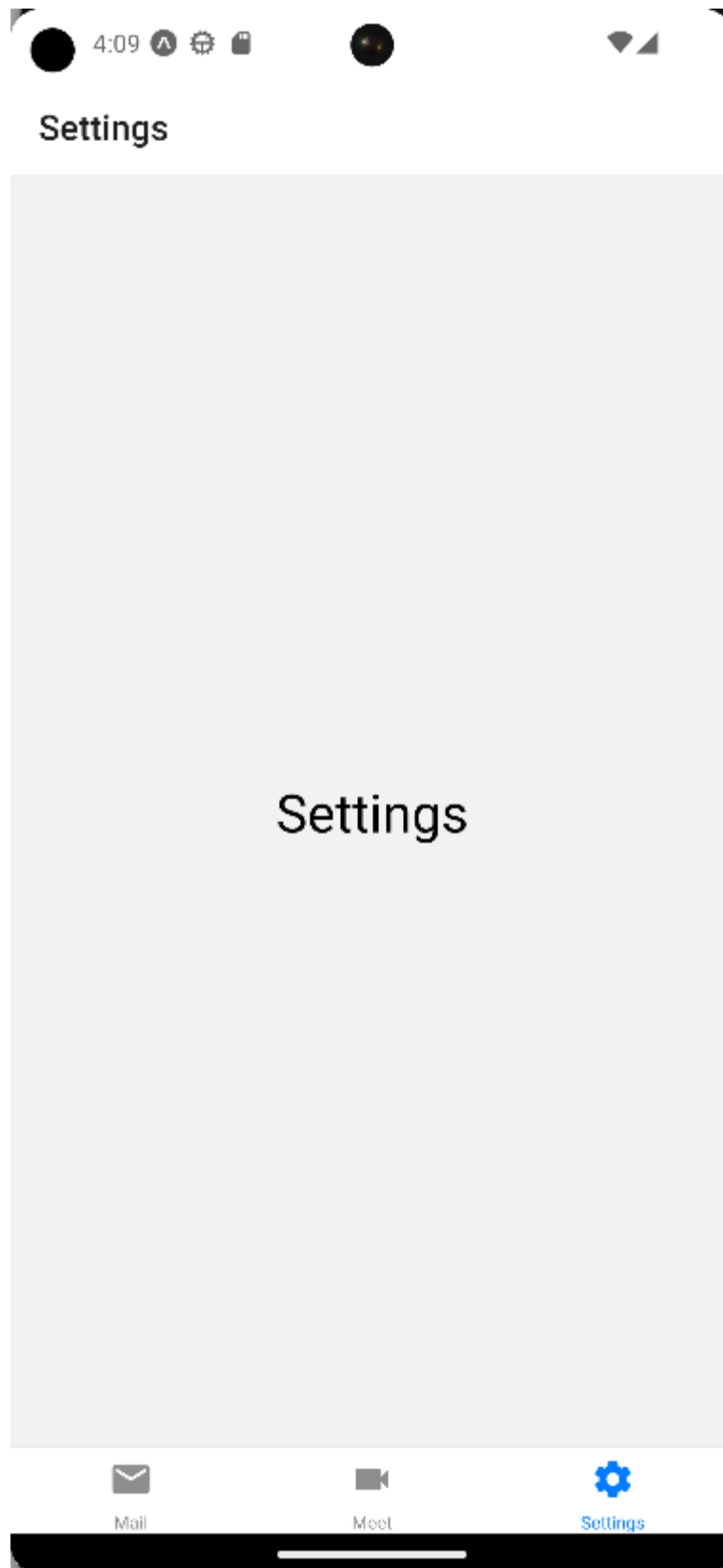
탭 바와 탭 버튼을 수정하는 방법에 대해 알아보자.

버튼 아이콘 설정하기

탭 네비게이션의 기본 설정에는 탭 버튼 아이콘이 지정되어 있지 않아 조금 행한 모습을 볼 수 있다. 탭 버튼에 아이콘을 렌더링 하는 방법은 `tabBarIcon`을 이용하는 것이다. 스택 네비게이션에서 타이틀 컴포넌트를 변경하거나 헤더의 버튼 컴포넌트를 변경했던 것처럼 `tabBarIcon`에 컴포넌트에 반환하는 함수를 지정하면 버튼의 아이콘이 들어갈 자리에 해당 컴포넌트를 렌더링한다. `tabBarIcon`에 설정된 함수에는 `color`, `size`, `focused` 값을 포함한 객체가 파라미터로 전달된다는 특징이 있다.

```
src > navigations > JS Tabjs > TabNavigation
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Settings } from "../screens/TabScreens";
4  import { MaterialCommunityIcons } from '@expo/vector-icons';
5
6
7  const TabIcon = ({name, size, color})=>{
8    return <MaterialCommunityIcons name={name} size={size} color={color} />;
9  };
10
11  const Tab = createBottomTabNavigator();
12
13  const TabNavigation=()=>{
14    return(
15      <Tab.Navigator initialRouteName="Setting">
16        <Tab.Screen
17          name="Mail"
18          component={Mail}
19          options = {{
20            tabBarIcon: props => TabIcon({...props, name: 'email'}),
21          }}
22        />
23        <Tab.Screen
24          name="Meet"
25          component={Meet}
26          options = {{
27            tabBarIcon: props => TabIcon({...props, name: 'video'}),
28          }}
29        />
30        <Tab.Screen
31          name="Settings"
32          component={Settings}
33          options = {{
34            tabBarIcon: props => TabIcon({...props, name: 'cog'}),
35          }}
36        />
37      </Tab.Navigator>
38    );
39  };
40
```

책에는 settings로 되어있지만 버전이 안맞아서 인지 아이콘이 나오지 않아 cog으로 바꾸니 정상적으로 아이콘이 나왔다.



화면을 구성하는 Screen 컴포넌트마다 `tabBarIcon`에 `MaterialCommunityIcons` 컴포넌트를 반환하는 함수를 지정했다. 반환되는 컴포넌트의 색과 크기는 `tabBarIcon`에 지정된 함수의 파라미터로 전달되는 `color`와 `size`를 이용해서 설정했다.

만약 Screen 컴포넌트마다 탭 버튼 아이콘을 지정하지 않고 한곳에서 모든 버튼의 아이콘을 관리하고 싶은 경우 Navigator 컴포넌트의 screenOptions 속성을 사용해서 관리할 수 있다.

```
JS Tab.js  X  JS App.js
src > navigations > JS Tab.js > [🔍] TabNavigation
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Settings } from '../screens/TabScreens';
4  import { MaterialCommunityIcons } from '@expo/vector-icons';
5
6
7  const TabIcon = ({name, size, color})=>{
8    return <MaterialCommunityIcons name={name} size={size} color={color} />;
9  };
10
11  const Tab = createBottomTabNavigator();
12
13  const TabNavigation=()=>{
14    return(
15      <Tab.Navigator initialRouteName="Setting"
16        screenOptions={({route})=> ({
17          tabBarIcon : props => {
18            let name = '';
19            if (route.name === 'Mail') name = 'email';
20            else if (route.name === 'Meet')name = 'video';
21            else name = 'cog';
22            return TabIcon({ ...props, name});
23          },
24        })
25      >
26
27      <Tab.Screen
28        name="Mail"
29        component={Mail}
30        options = {{
31          tabBarIcon: props => TabIcon({...props, name: 'email'}),
```

screenOptions에 객체를 반환하는 함수를 설정하고 함수로 전달되는 route를 이용한다. Screen 컴포넌트의 name 속성을 값으로 갖는 route의 name값에 따라 렌더링되는 아이콘이 변경되도록 작성했다.

screen 컴포넌트마다 options에 tabBarIcon을 설정하는 방법과 Navigator 컴포넌트에서 screenOptions를 이용하는 방법과 결과는 동일하다.

라벨 수정하기

버튼 아이콘 아래에 렌더링 되는 라벨은 Screen 컴포넌트의 name값을 기본값으로 사용한다. 탭 버튼의 라벨은 tabBarLabel을 이용해서 변경할 수 있다. 탭 네비게이션의 Screen 컴포넌트에서 tabBarLabel값을 설정하여 탭 버튼의 라벨을 변경해보자.

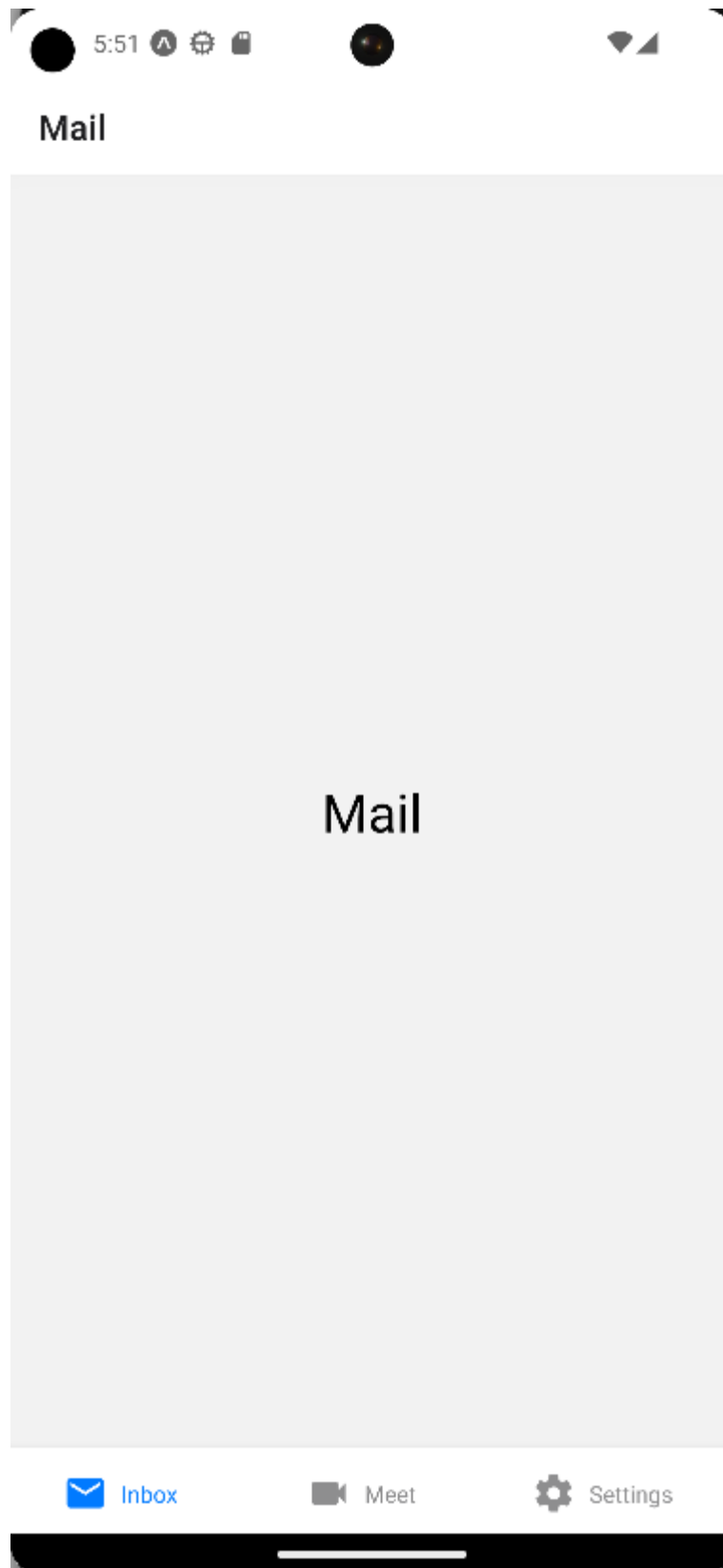
```
JS Tab.js  X  JS App.js
src > navigations > JS Tab.js > [🔍] TabNavigation > <function> > tabBarIcon
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Settings } from '../screens/TabScreens';
4  import {MaterialCommunityIcons} from '@expo/vector-icons';
5
6
7  const TabIcon = ({name, size, color})=>{
8    return <MaterialCommunityIcons name={name} size={size} color={color} />;
9  };
10
11  const Tab = createBottomTabNavigator();
12
13  const TabNavigation=()=>{
14    return(
15      <Tab.Navigator initialRouteName="Setting"
16        screenOptions={({route})=> ({
17          tabBarIcon : props => {
18            let name = '';
19            if (route.name === 'Mail') name = 'email';
20            else if (route.name === 'Meet') name = 'video';
21            else name = 'cog';
22            return TabIcon({ ...props, name});
23          },
24        })
25      >
26
27      <Tab.Screen
28        name="Mail"
29        component={Mail}
30        options = {{
31          tabBarLabel : 'Inbox',
32          tabBarIcon: props => TabIcon({...props, name: 'email'}),
33        }}
34      />
35      <Tab.Screen
36        name="Meet"
```

라벨을 버튼 아이콘의 아래가 아닌 아이콘 옆에 렌더링 되도록 변경하고 싶으면 labelPosition의 값을 변경해서 조정할 수 있다. labelPosition은 below-icon과 beside-icon 두 값만 설정할 수 있으며, beside-icon으로 설정하면 아이콘 오른쪽에 라벨이 렌더링된다.

```
JS Tab.js  X  JS App.js
src > navigations > JS Tab.js > [🔍] TabNavigation
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Settings } from '../screens/TabScreens';
4  import {MaterialCommunityIcons} from '@expo/vector-icons';
5
6
7  const TabIcon = ({name, size, color})=>{
8    return <MaterialCommunityIcons name={name} size={size} color={color} />;
9  };
10
11 const Tab = createBottomTabNavigator();
12
13 const TabNavigation=()=>{
14   return(
15     <Tab.Navigator
16       initialRouteName="Setting"
17       tabBarOptions={{ labelPosition : 'beside-icon'}}
18       // screenOptions=(({route})=> ({
19       //   tabBarIcon : props => {
20       //     let name = '';
21       //     if (route.name === 'Mail') name = 'email';
22       //     else if (route.name === 'Meet')name = 'video';
23       //     else name = 'cog';
24       //     return TabIcon({ ...props, name});
25       //   },
26       // })}
27   >
28
29     <Tab.Screen
30       name="Mail" |
31       component={Mail}
32       options = {{
33         tabBarLabel : 'Inbox',
34         tabBarIcon: props => TabIcon({...props, name: 'email'}),
35       }}
36     />
37     <Tab.Screen
```

주석된 부분이 풀려있으면 서로 영향이 있는건지 제대로 렌더링 되지 않아서 주석을 해 놓았다.

보통 라벨의 위치는 모든 버튼에 동일하게 적용되므로 Navigator 컴포넌트의 tabBarOptions를 이용해서 모든 버튼에 적용되도록 작성했다.

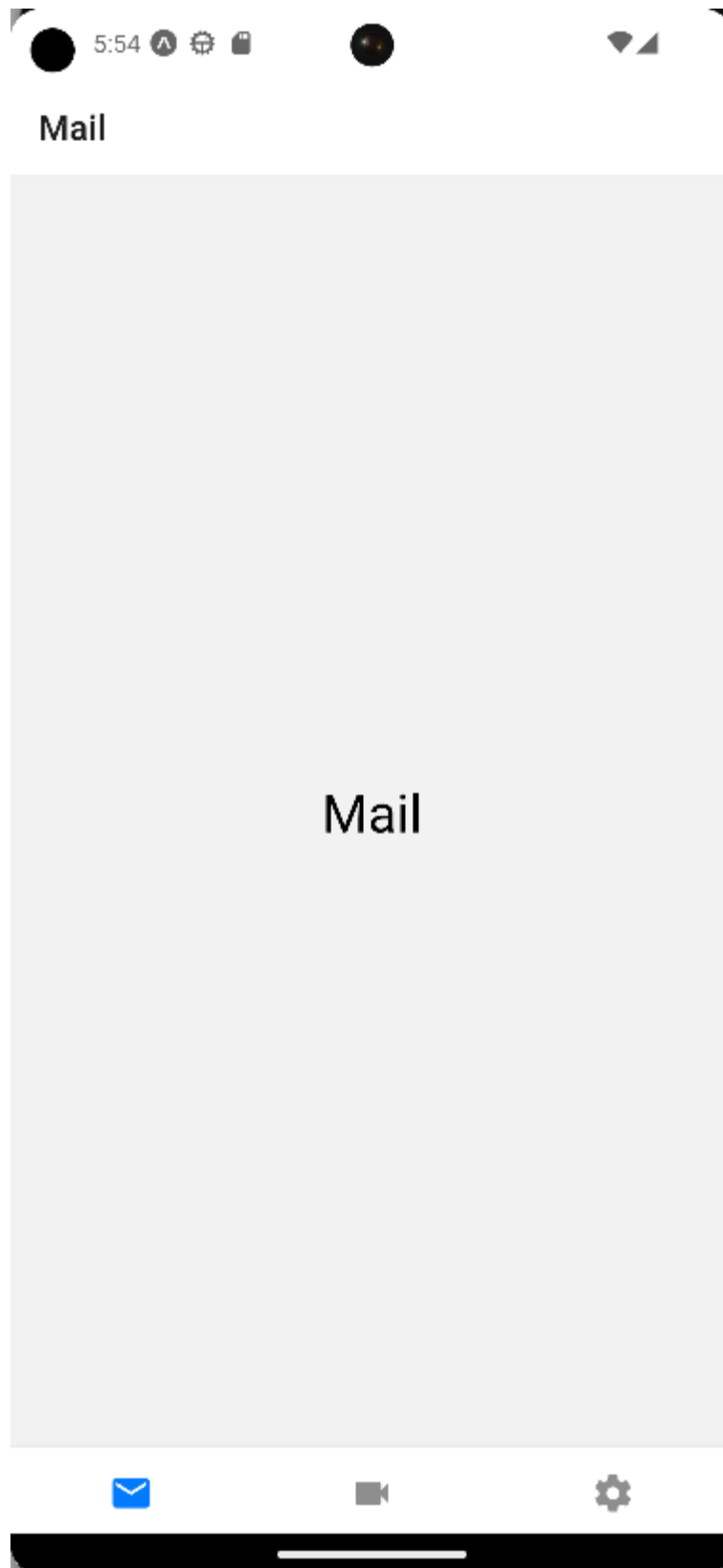


프로젝트의 기획이나 디자인에 따라 라벨을 렌더링하지 않고 아이콘만 사용하는 경우도 많다. `showLabel`을 이용하면 탭 바에서 라벨이 렌더링되지 않도록 설정할 수 있다.


```

JS Tab.js  JS App.js
src > navigations > JS Tab.js > [x] TabNavigation > showLabel
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Settings } from '../screens/TabScreens';
4  import { MaterialCommunityIcons } from '@expo/vector-icons';
5
6
7  const TabIcon = ({name, size, color})=>{
8    return <MaterialCommunityIcons name={name} size={size} color={color} />;
9  };
10
11  const Tab = createBottomTabNavigator();
12
13  const TabNavigation=()=>>{
14    return(
15      <Tab.Navigator
16        initialRouteName="Setting"
17        tabBarOptions={{ labelPosition : 'beside-icon', showLabel : false }}
18        // screenOptions={({route})=> ({
19          //   tabBarIcon : props => {
20            //     let name = '';
21            //     if (route.name === 'Mail') name = 'email';
22            //     else if (route.name === 'Meet') name = 'video';
23            //     else name = 'cog';
24            //     return TabIcon({ ...props, name});
25            //   },
26          //   })
27        >
28
29        <Tab.Screen
30          name="Mail"
31          component={Mail}
32          options = {{
33            tabBarLabel : 'Inbox',
34            tabBarIcon: props => TabIcon({...props, name: 'email'}),
35          }}
36        />
37        <Tab.Screen
38          name="Meet"

```

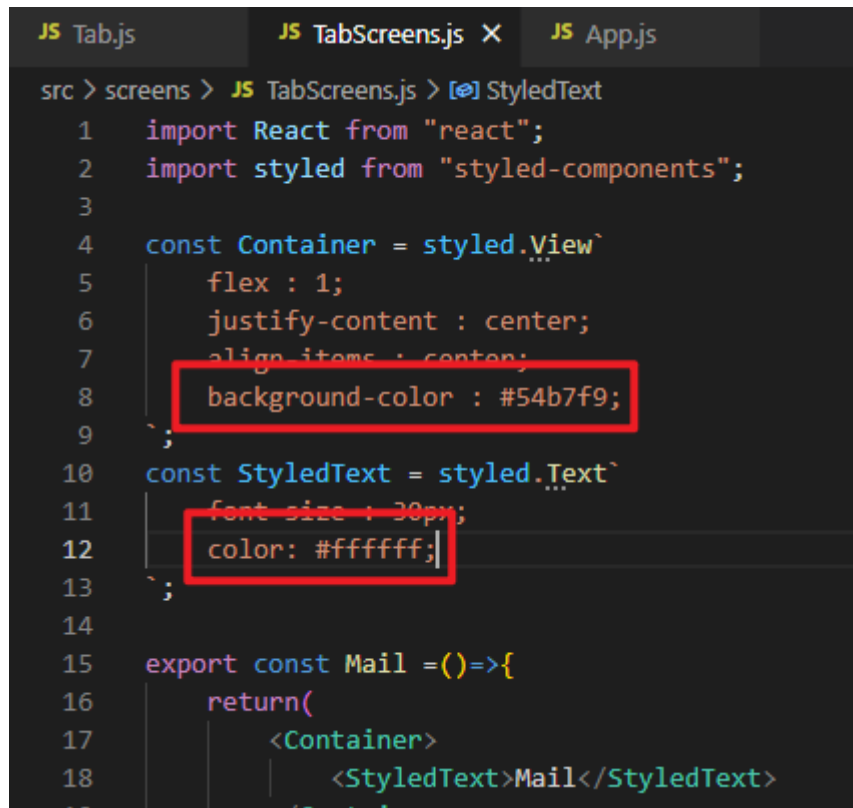


라벨 없이 버튼의 아이콘만 잘 나타난다!

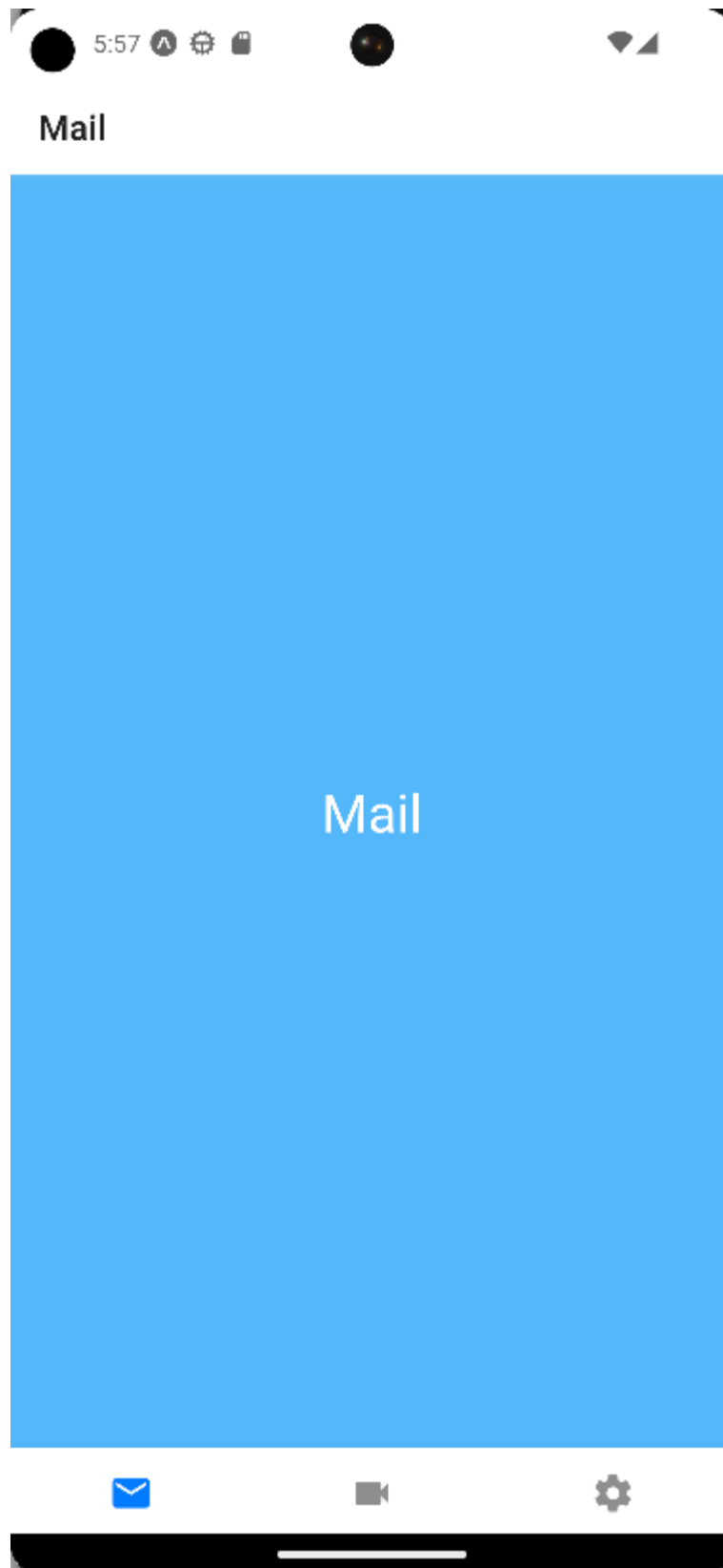
스타일링 수정하기

탭 네비게이션의 탭 바 배경색은 흰색이 기본값이다. 만약 화면의 배경색이 탭 바의 기본 색과 어울리지 않는다면 탭 바의 배경색 등을 수정해야한다. 이번에는 탭 바의 스타일을 수정하는 방법에 대해 알아보자

먼저 화면의 배경색을 아래처럼 변경하자.



```
JS Tab.js JS TabScreens.js X JS App.js
src > screens > JS TabScreens.js > [🔍] StyledText
1 import React from "react";
2 import styled from "styled-components";
3
4 const Container = styled.View`
5   flex : 1;
6   justify-content : center;
7   align-items : center;
8   background-color : #54b7f9;
9 `;
10 const StyledText = styled.Text`
11   font-size : 30px;
12   color: #ffffff;
13 `;
14
15 export const Mail =()=>{
16   return(
17     <Container>
18       <StyledText>Mail</StyledText>
19     </Container>
20   )
21 }
```



화면의 스타일 수정이 완료되면 탭 바의 스타일을 변경한다 탭바의 스타일은 `tabBarOptions` 속성에 `style` 값으로 스타일 객체를 설정하여 변경할 수 있다.

```
JS TabScreens.js JS Home.js JS Stack.js JS Tab.js X JS App.js
src > navigations > JS Tab.js > [x] TabNavigation
1  import React from "react";
2  import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
3  import { Mail, Meet, Settings } from '../screens/TabScreens';
4  import {MaterialCommunityIcons} from '@expo/vector-icons';
5
6
7  const TabIcon = ({name, size, color})=>{
8    return <MaterialCommunityIcons name={name} size={size} color={color} />;
9  };
10
11  const Tab = createBottomTabNavigator();
12
13  const TabNavigation=()=>{
14    return(
15      <Tab.Navigator
16
17 > // initialRouteName="Settings" ...
31 // }}}
32     screenOptions={{
33       headerStyle:{
34         height : 110,
35         backgroundColor : '#54b7f9',
36       },
37     },
38     tabBarStyle:{
39       backgroundColor : '#54b7f9',
40       borderTopColor : '#ffffff',
41       borderTopWidth : 2,
42     }
43   }}
44   tabBarOptions={{
45     labelPosition: 'beside-icon',
46     showLabel:false,
47     activeTintColor: '#000000',
48     inactiveTintColor: '#cfcfcf',
49   }}
50 >
51
```

