

day52-rn-chatapp1

☰ 태그	
📅 날짜	@2022년 12월 13일

그림 p.302 9-1

우리가 일상 생활 속에서 많이 사용하는 채팅 애플리케이션에는 다양한 기능이 들어있다. 다양한 기능을 모두 개발하면 좋겠지만, 채팅을 동작시키기 위한 기능만 구현해보자.

- 로그인/ 회원가입 : 이메일과 비밀번호를 이용한 로그인 회원가입
- 프로필 : 나의 정보 확인 및 변경
- 채널 생성 : 채널 생성 기능
- 채널 목록 : 생성된 채널들의 목록 조회
- 채널 : 실시간으로 메시지를 송수신하는 독립된 공간

프로젝트 준비

아래 명령어로 프로젝트를 생성하자

- `expo init react-native-simple-chat`

내비게이션

지금 만들어볼 채팅 애플리케이션은 크게 세 부분으로 나눌 수 있다. 로그인 등 인증하는 화면, 채팅방의 목록 등을 확인할 수 있는 화면, 메시지를 주고받는 화면.

화면은 유기적으로 연결되어 있으며 화면 간 이동이 잦다. 여기서 내비게이션 기능을 이용해서 채팅 애플리케이션의 화면을 구성하자.

- `npm install @react-navigation/native`

리액트 내비게이션 라이브러리 사용에 필요한 추가 라이브러리를 설치하자

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

프로젝트는 아래 그림과 같이 6개의 화면으로 이루어져 있다. 스택 내비게이션과 탭 내비게이션을 활용하여 화면 구조를 만들 예정이므로 추가로 필요한 라이브러리를 설치한다.

- npm install @react-navigation/stack @react-navigation/bottom-tabs

p.303 그림

라이브러리

리액트 내비게이션 외에도 몇 가지 라이브러리를 사용할 예정이다. 먼저 스타일 작성을 위한 스타일드 컴포넌트 라이브러리와 타입 확인을 위한 prop-types 라이브러리를 설치하자.

- npm install styled-components prop-types —force

위에 라이브러리 외에도 프로젝트를 진행하면서 다른 추가적인 라이브러리를 사용할 예정이다.

- expo-image-picker : <https://bit.ly/expo-imagepicker>

expo-image-picker 라이브러리는 기기의 사진이나 영상을 가져올 수 있도록 시스템 UI에 접근할 수 있는 기능을 제공한다.

이번엔 기기의 사진을 선택해 사용자의 사진을 설정 혹은 변경하기 위해 사용할 예정이다.

- moment : <https://momentjs.com/>

라이브러리는 시간을 다양한 형태로 변경하는 등 시간과 관련된 많은 기능을 제공하는 라이브러리로, 날짜와 관련된 라이브러리 중 가장 널리 알려져 있고 많이 사용되고 있다. 여기서는 타임 스탬프를 사용자가 보기 편한 형태로 변경하기 위해 사용한다.

- react-native-keyboard-aware-scroll-view : <https://bit.ly/keyboard-scroll>

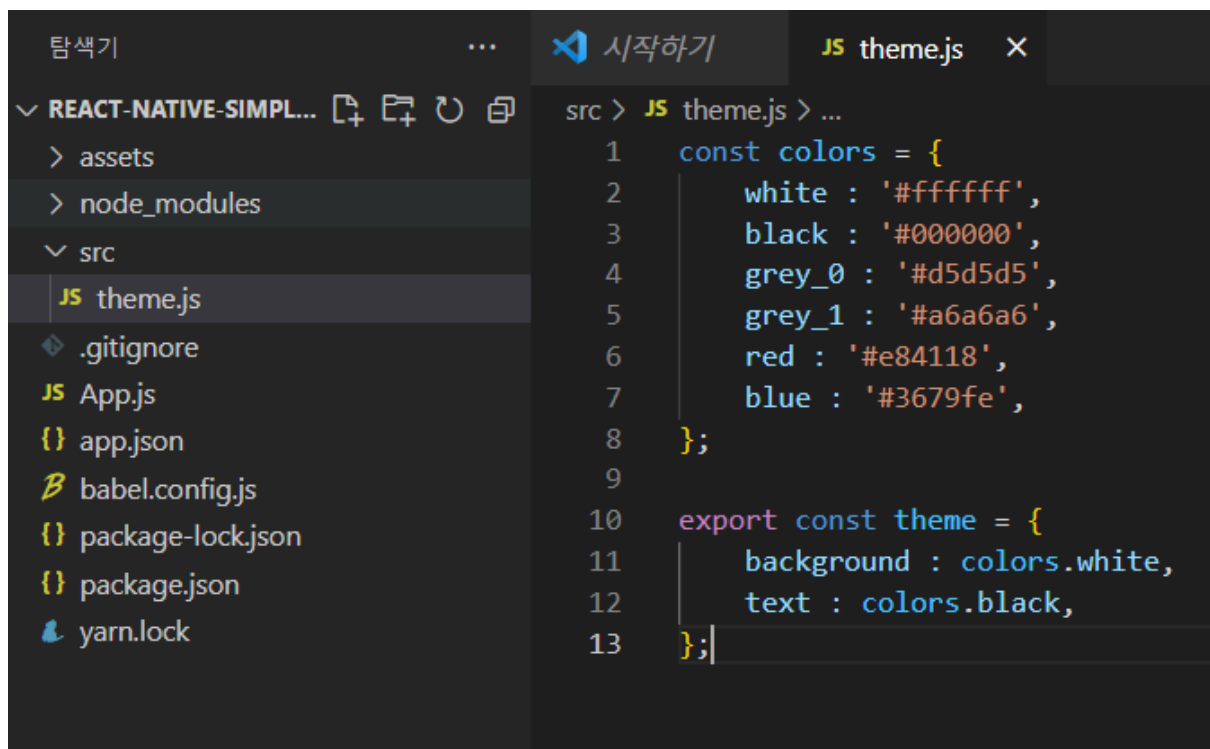
키보드가 화면을 가리면서 생기는 불편한 점을 해결하기 위해 사용되는 라이브러리이다.

- react-native-gifted-chat : <https://bit.ly/gifted-chat>

메시지를 주고받는 채팅 화면을 쉽게 구현할 수 있도록 도와주는 라이브러리이다.

프로젝트 파일 구조

모든 소스 파일을 관리할 src폴더를 생성한 후 폴더 안에 theme.js파일을 생성하고 아래 처럼 작성하자.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the project structure with folders like assets, node_modules, and src. The src folder is expanded, showing the theme.js file. The code editor shows the content of theme.js, which defines a set of colors and a theme object.

```
src > JS theme.js > ...
1  const colors = {
2    white : '#ffffff',
3    black : '#000000',
4    grey_0 : '#d5d5d5',
5    grey_1 : '#a6a6a6',
6    red : '#e84118',
7    blue : '#3679fe',
8  };
9
10 export const theme = {
11   background : colors.white,
12   text : colors.black,
13 };
```

위 코드는 공통으로 사용할 색을 정의하고 배경색과 글자의 색을 미리 정의했다. 이번에는 최상위 컴포넌트가 될 App 컴포넌트를 src폴더 안에 작성하자.

```
src > JS App.js > ...
1  import React from "react";
2  import { StatusBar } from "expo-status-bar";
3  import { ThemeProvider } from "styled-components";
4  import {theme} from './theme';
5
6  const App = ()=>{
7    return(
8      <ThemeProvider theme={theme}>
9        <StatusBar barStyle="dark-content" />
10      </ThemeProvider>
11    );
12  };
13
14  export default App;
```

스타일드 컴포넌트의 ThemeProvider 컴포넌트를 사용해 스타일드 컴포넌트에서 정의된 theme을 사용할 수 있도록 작성했다.

루트 디렉터리에 있는 App.js파일을 수정해서 src밑에 app 컴포넌트가 메인이 되도록 설정하자.

```
JS App.js > [?] default
1  import App from "./src/App";
2
3  export default App;
4
5
```

src폴더 밑에 각 역할에 맞는 파일을 관리할 폴더를 생성하자

- components : 컴포넌트 파일 관리
- contexts : Context API 파일 관리

- navigations : 내비게이션 파일 관리
- screens : 화면 파일 관리
- utils : 프로젝트에서 이용할 기타 기능 관리

파이어베이스

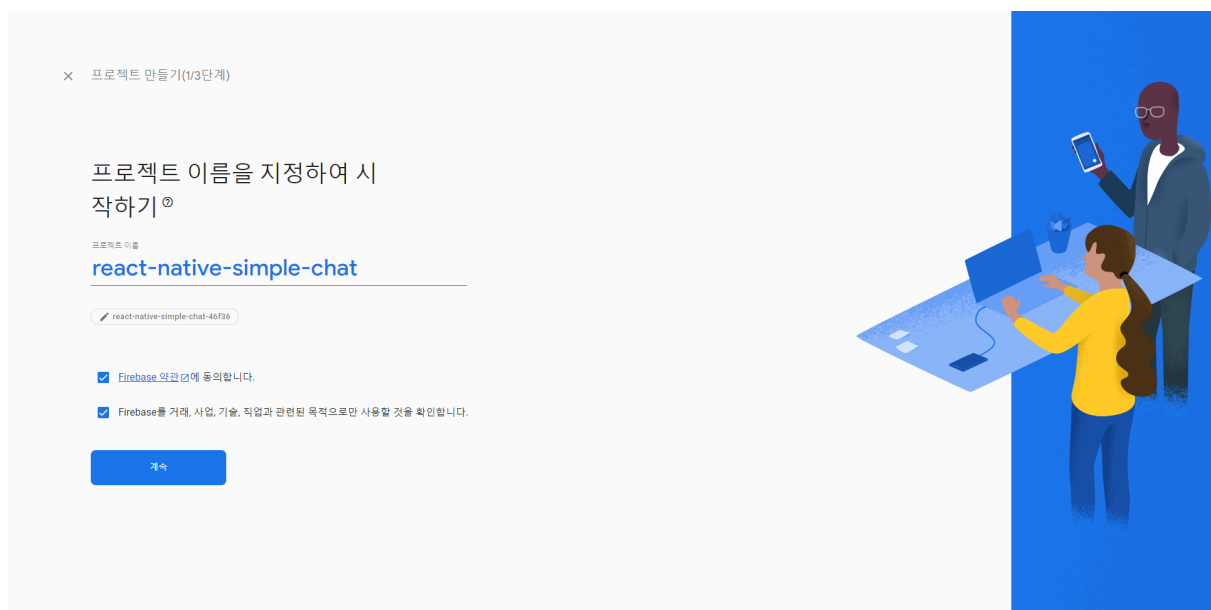
Firebase는 인증(authentication), 데이터베이스(database) 등의 다양한 기능을 제공하는 개발 플랫폼이다.

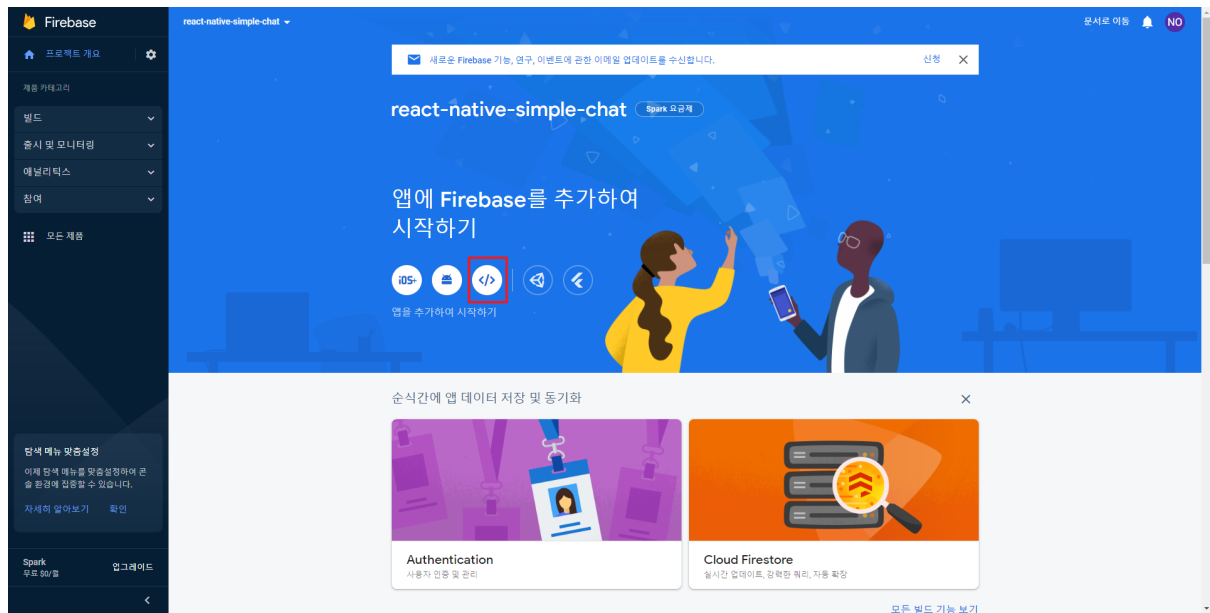
파이어베이스가 제공하는 기능을 이용하면 대부분의 서비스에서 필요한 서버와 데이터베이스를 직접 구축하지 않아도 개발이 가능하다는 장점이 있다.

별도의 서버 구축 없이 파이어베이스를 이용해 프로젝트를 진행하자.

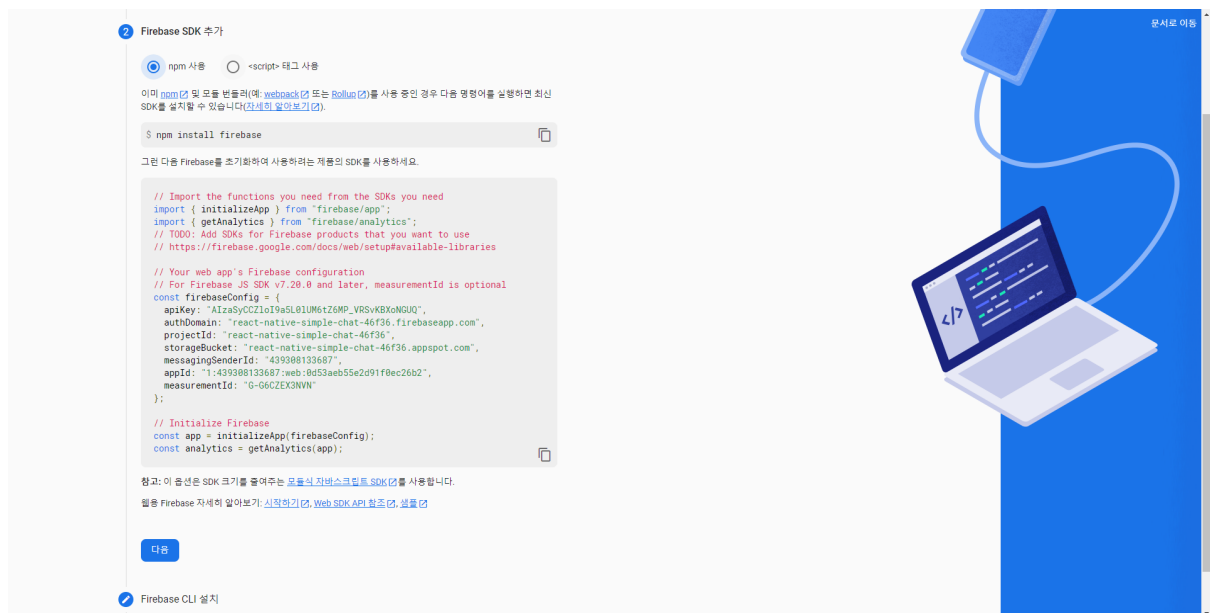
- 파이어베이스 콘솔 : <https://console.firebase.google.com/>

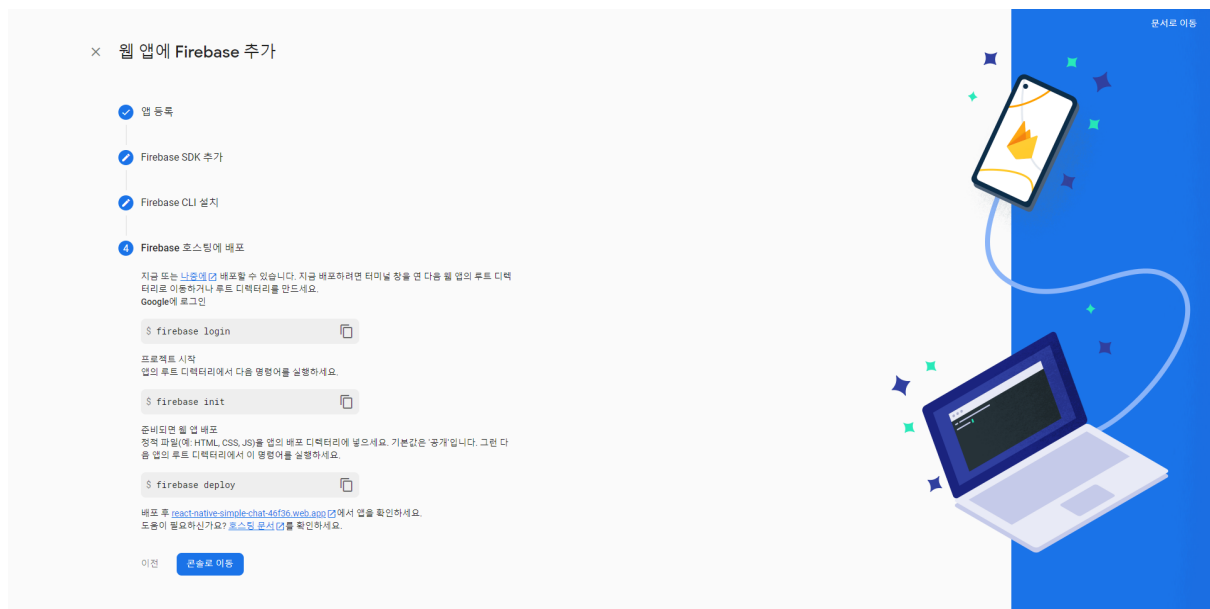
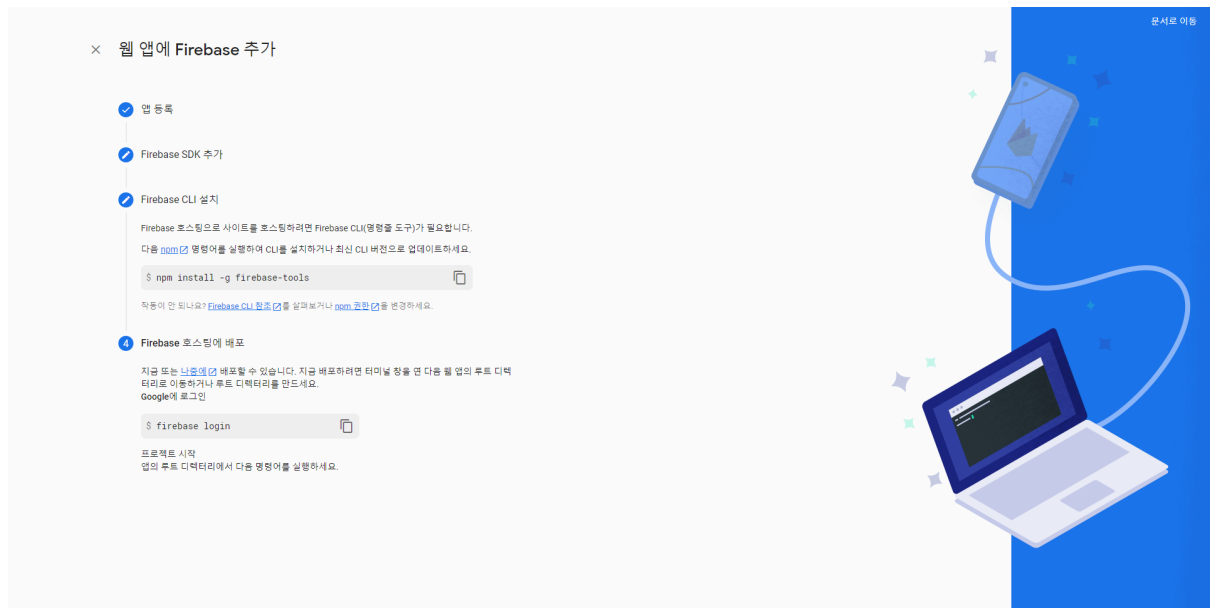
프로젝트 생성이 완료되고 프로젝트 화면으로 이동하면 사용할 플랫폼을 선택해서 앱을 추가해야 한다. “프로젝트 개요” 혹은 “프로젝트 설정 → 일반 → 내앱”에서 “웹”을 선택하고 앱을 추가하자. 앱을 추가하는 과정에서 입력해야하는 앱의 닉네임은 편의상 지정하는 내부용 식별자이므로 여러분이 지정하고 싶은 이름을 입력하면 된다.





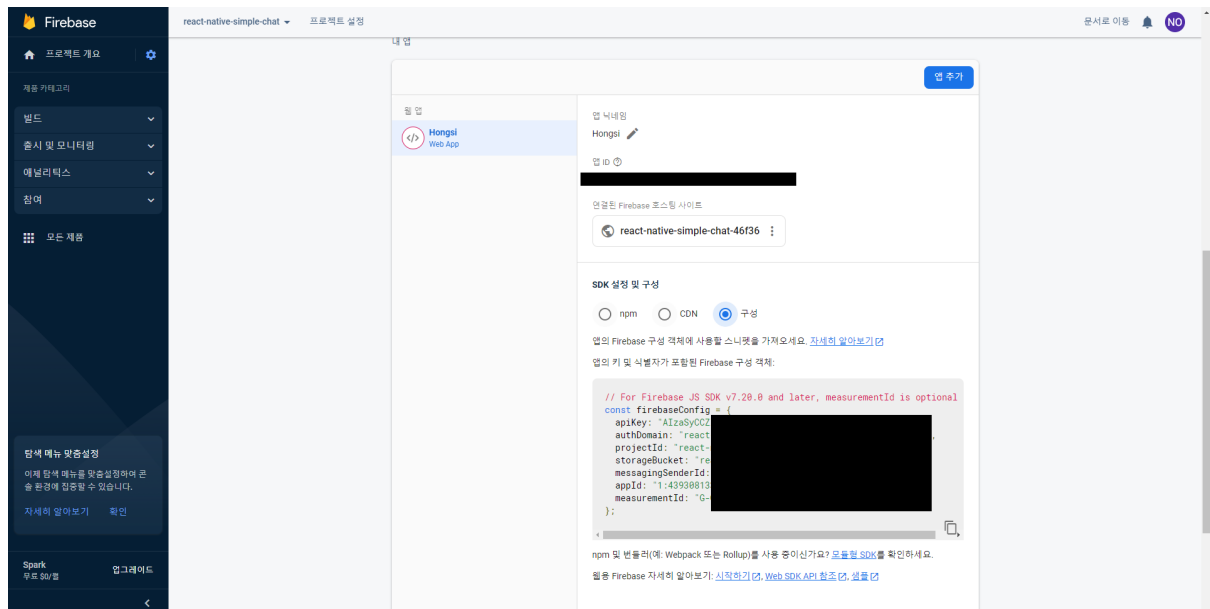
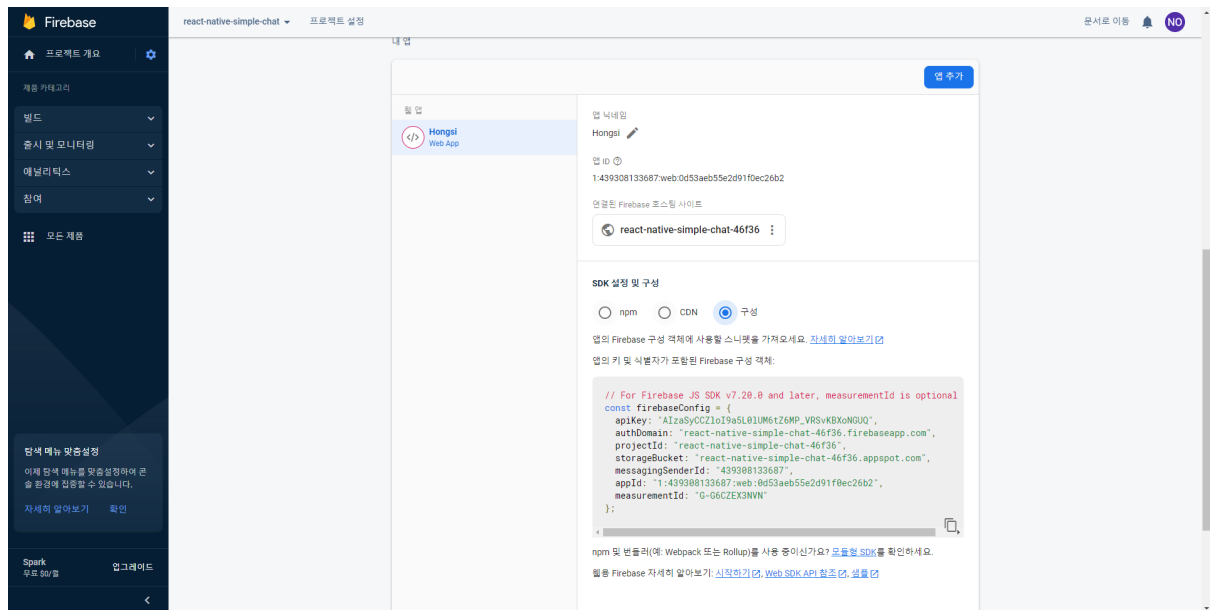
앱 추가가 완료된 후 “프로젝트 설정 → 일반 → 내 앱” 에서 “Firebase SDK snippet”을 확인하면 파이어베이스를 사용하기 위한 설정값을 확인할 수 있다.





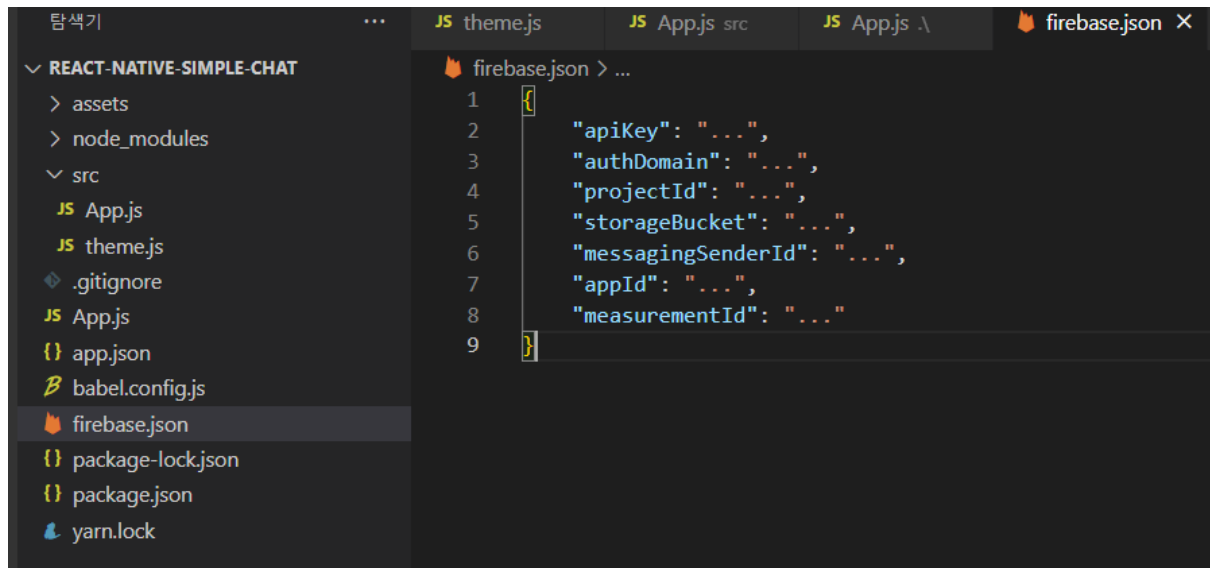
앱 IDhelp_outline

1:439308133687:web:0d53aeb55e2d91f0ec26b2

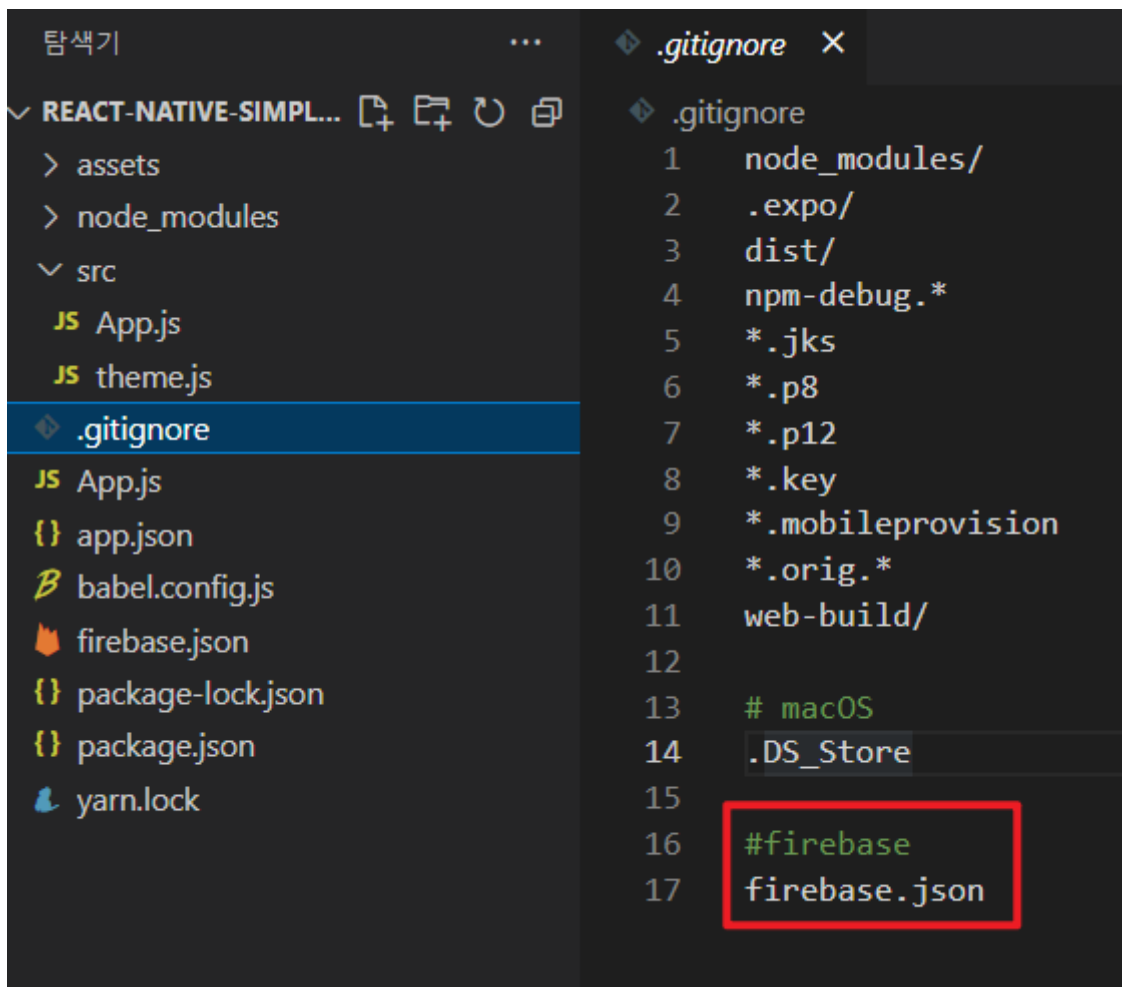


이 값들은 외부에서 노출되면 안 되는 중요한 값들이므로 잘 관리해야 한다. 이제 프로젝트 루트 디렉터리에 firebase.json 파일을 생성하고 위 코드를 아래와 같은 형태로 복사하고 .gitignore 파일에 firebase.json을 추가하자.

firebase.json



.gitignore



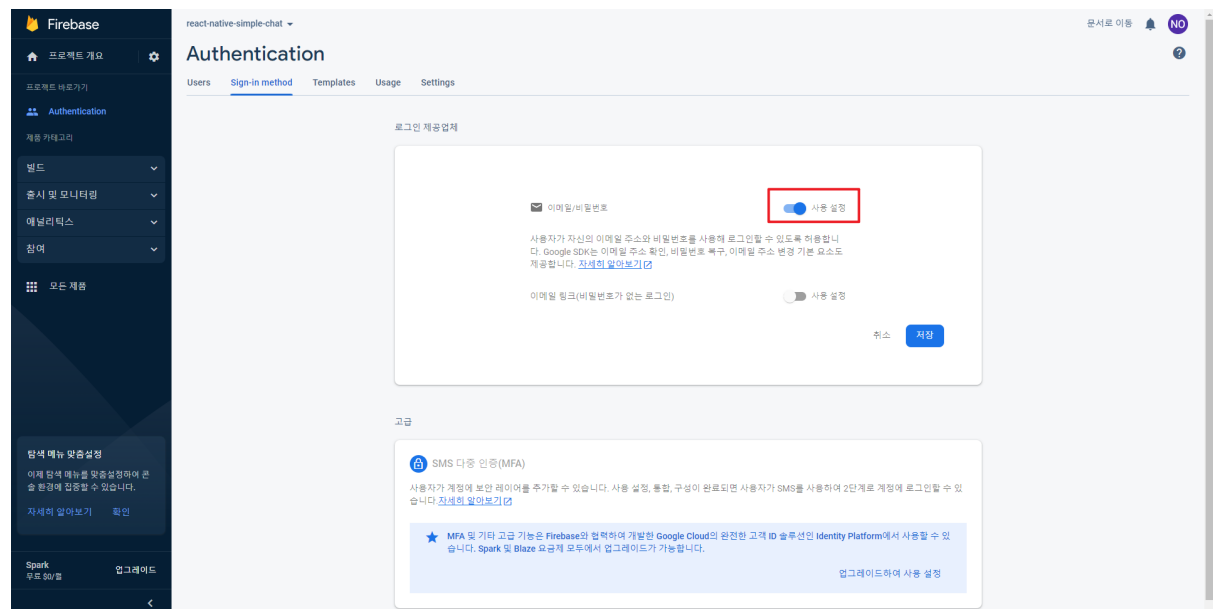
.gitignore : 깃허브에 올리면 안되는 부분을 설정

파이어베이스에서는 다양한 기능을 제공하지만, 리액트 네이티브에 조금 더 집중하기 위해 파이어베이스의 모든 내용이나 사용하는 기능에 대한 설명은 생략한다.

채팅 애플리케이션을 만들기 위해 알아야 할 인증, 데이터베이스, 스토리지(storage)기능에 대해서만 간략하게 다루고 넘어가도록 하자.

인증

파이어베이스의 인증 기능에서는 다양한 인증 방법을 제공한다. 우리는 이메일과 비밀번호를 이용하여 인증할 수 있는 기능을 만들 예정이므로 “이메일/비밀번호” 부분만 활성화하고 진행한다.

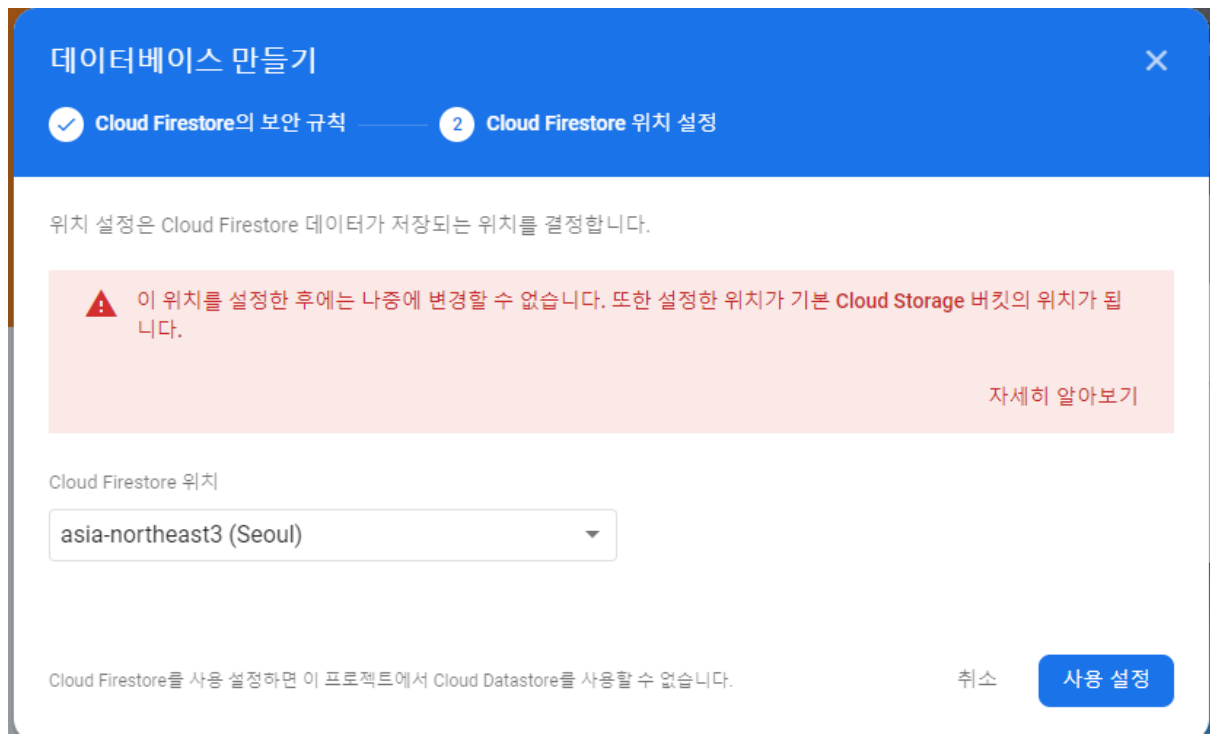
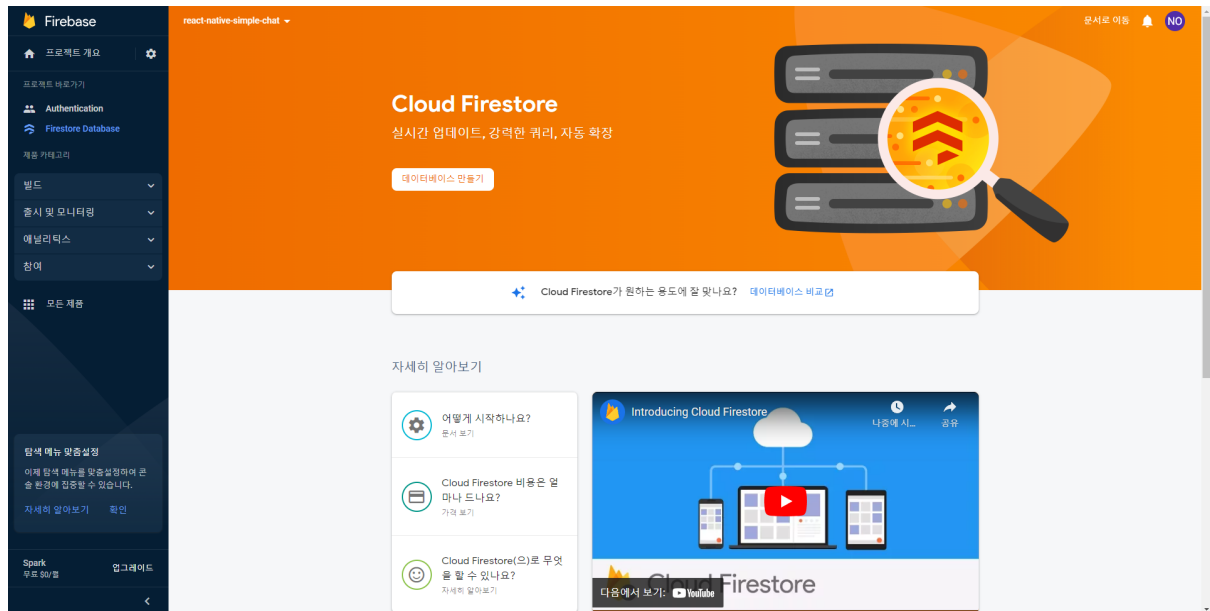


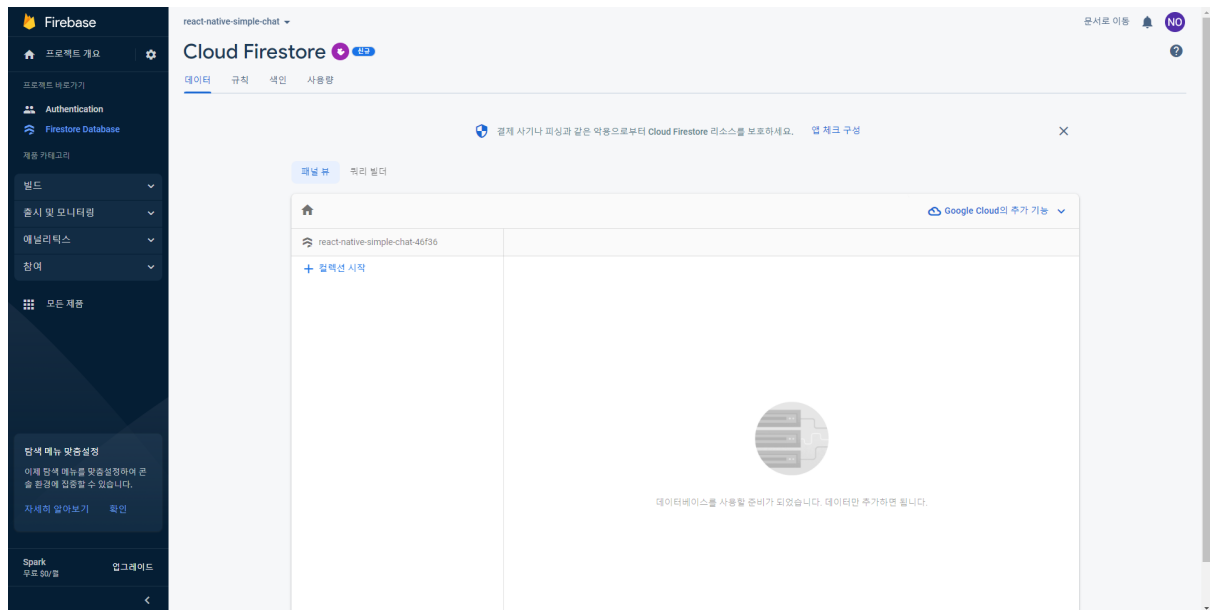
데이터베이스

생성되는 채널과 각 채널에서 발생하는 메시지를, 파이어베이스의 데이터베이스를 이용하여 관리한다. 데이터베이스의 경우 파이어스토어 (cloud Firestore)와 실시간 데이터베이스 (realtime database) 두 가지 종류를 제공하는데, 우리는 파이어스토어를 이용할 것이다. 데이터베이스를 사용하려면 데이터베이스 메뉴에서 데이터베이스 만들기를 진행해야 한다. 데이터베이스 만들기의 첫 번째 단계인 보안 규칙은 뒤에서 조금 더 다루고, 두 번째 단계인 위치를 선택하는 화면에서는 서비스하는 지역과 가장 가까운 지역을 선택하는 것이 좋다.

- 파이어베이스 위치 : <https://bit.ly/location-firebase>

한국에서 진행하기 때문에 서울인 asia-northeast3를 선택하자.

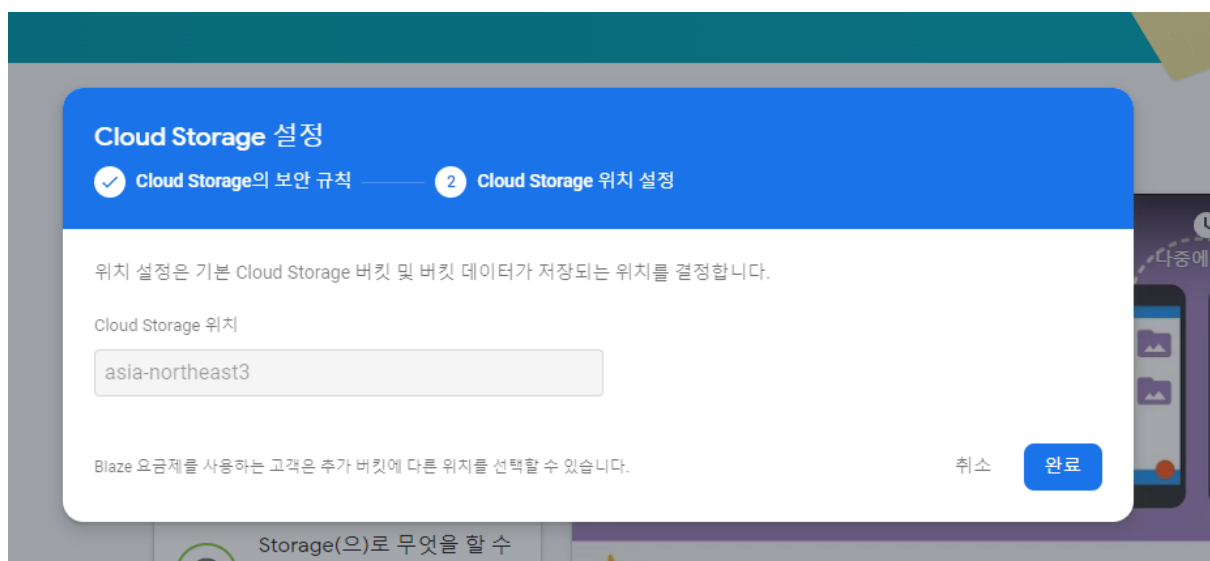




스토리지

스토리지는 서버 코드 없이 사용자의 사진, 동영상 등을 저장할 수 있는 기능을 쉽게 개발할 수 있도록 기능을 제공한다.

스토리지를 이용해서 채팅 애플리케이션에 가입한 사용자의 사진을 저장하고 가져오는 기능을 만들거다. 스토리지 메뉴에서 시작하기 버튼을 클릭해 스토리지를 사용하기 위한 설정을 진행하자. 데이터베이스와 마찬가지로 보안 규칙을 설정하는 화면이 나오고, 위치를 설정하는 화면이 나온다. 만약 책의 순서대로 데이터베이스에서 위치를 설정했다면, 데이터베이스에서 설정한 위치와 같은 위치로 나타난다.



라이브러리 설치

파이어베이스를 사용하기 위해서는 라이브러리 설치가 필요하다.

- 파이어베이스 라이브러리 : <https://www.npmjs.com/package/firebase>

아래 명령어로 파이어베이스 라이브러리를 설치하자.

- `expo install firebase`

또는

- `npm install --save firebase --force`

utils 폴더 아래에 `firebase.js` 파일을 생성하고 아래처럼 작성하자.

```
src > utils > JS firebase.js > ...
1  import * as firebase from 'firebase';
2  import config from '../../firebase.json';
3
4  const app = firebase.initializeApp(config);
5
```

앱 아이콘과 로딩 화면

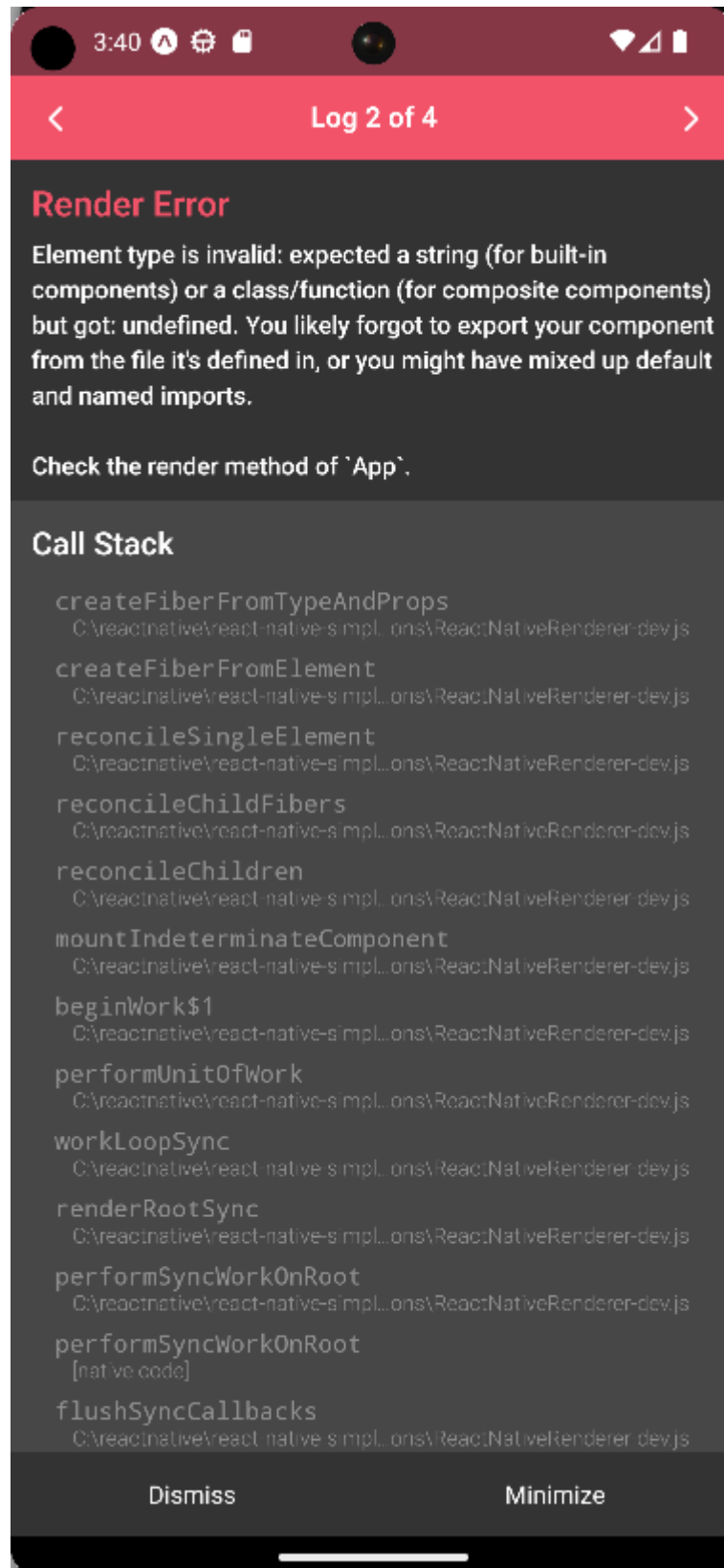
프로젝트의 기능과 화면 개발에 앞서 앱의 아이콘과 로딩 화면을 먼저 변경하자. 앱의 아이콘으로 사용할 1024 x 1024 크기의 `icon.png` 파일과 로딩 화면으로 사용할 1242x2436 크기의 `splash.png` 파일을 생성해서 `assets` 폴더에 자동으로 생성된 파일들과 교체하고 `App` 컴포넌트를 다음과 같이 수정한다.

```
src > JS App.js > [App] > [App] _loadAssets
1  import React, {useState} from "react";
2  import { StatusBar, Image } from "expo-status-bar";
3  import {AppLoading } from 'expo';
4  import {Asset} from 'expo-asset';
5  import * as Font from 'expo-font';
6  import { ThemeProvider } from "styled-components";
7  import {theme} from './theme';
8
9  const cacheImages = images => {
10   return images.map(image => {
11     if (typeof image === 'string') {
12       return Image.prefetch(image);
13     }else{
14       return Asset.fromModule(image).downloadAsync();
15     }
16   });
17 };
18 const cacheFonts = fonts => {
19   return fonts.map(font => Font.loadAsync(font));
20 };
21
22 const App = ()=>{
23   const [isReady, setIsReady] = useState(false);
24
25   const _loadAssets = async ()=> {
26     const imageAssets = cacheImages([require('../assets/splash.png')]);
27     const fontAssets = cacheFonts([]);
28
29     await Promise.all([...imageAssets, ...fontAssets]);
30   };
31
32   return isReady ? (
33     <ThemeProvider theme={theme}>
34       <StatusBar barStyle="dark-content" />
35     </ThemeProvider>
36   ) : (
37     <AppLoading
38       startAsync={_loadAssets}
39       onFinish={() => setIsReady(true)}
40       onError = {console.warn}
41     />
42   );
43 };
44
45 export default App;
```

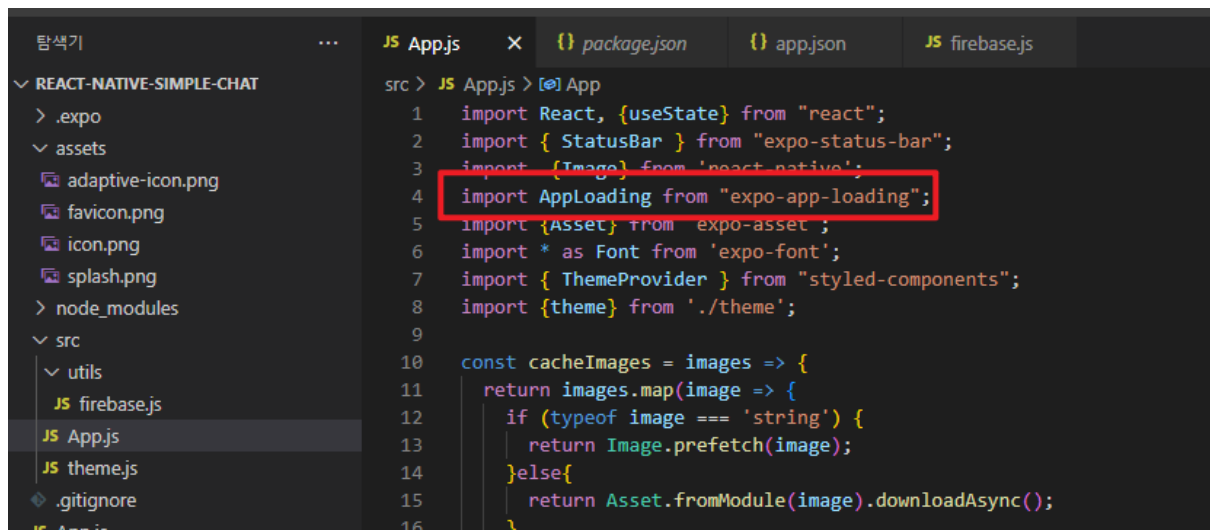
앞으로 프로젝트에서 사용할 이미지와 font를 미리 불러와서 사용할 수 있도록 cacheImages와 cacheFonts 함수를 작성하고 이를 이용해 _loadAssets 함수를 구성했다. 이미지나 폰트를 미리 불러오면 애플리케이션을 사용하는 환경에 따라 이미지나 폰트가 느리게 적용되는 문제를 개선할 수 있다.

애플리케이션은 미리 불러와야 하는 항목들을 모두 불러오고 화면이 렌더링되도록 AppLoading컴포넌트의 startAsync에 _loadAssets 함수를 지정하고, 완료되었을 때 isReady 상태를 변경해서 화면이 렌더링되도록 작성했다.

로딩 화면에서 기기의 크기에 따라 주변에 흰색 바탕이 보이는 것을 방지하기 위해 로딩 화면의 배경색을 로딩 화면 이미지의 배경색과 동일하게 변경하자.



에러가 난다.



```
src > JS App.js > App
1  import React, {useState} from "react";
2  import { StatusBar } from "expo-status-bar";
3  import { Image } from "react-native";
4  import AppLoading from "expo-app-loading";
5  import { Asset } from "expo-asset";
6  import * as Font from "expo-font";
7  import { ThemeProvider } from "styled-components";
8  import { theme } from "../theme";
9
10 const cacheImages = images => {
11   return images.map(image => {
12     if (typeof image === 'string') {
13       return Image.prefetch(image);
14     } else {
15       return Asset.fromModule(image).downloadAsync();
16     }
17   });
18 }
```

앱로딩 문제였다. 고치니 바로 정상적으로 작동한다.

expo install expo-app-loading

인증화면

이번에는 파이어베이스의 인증 기능을 이용해서 로그인 화면과 회원가입 화면을 만들어보자.

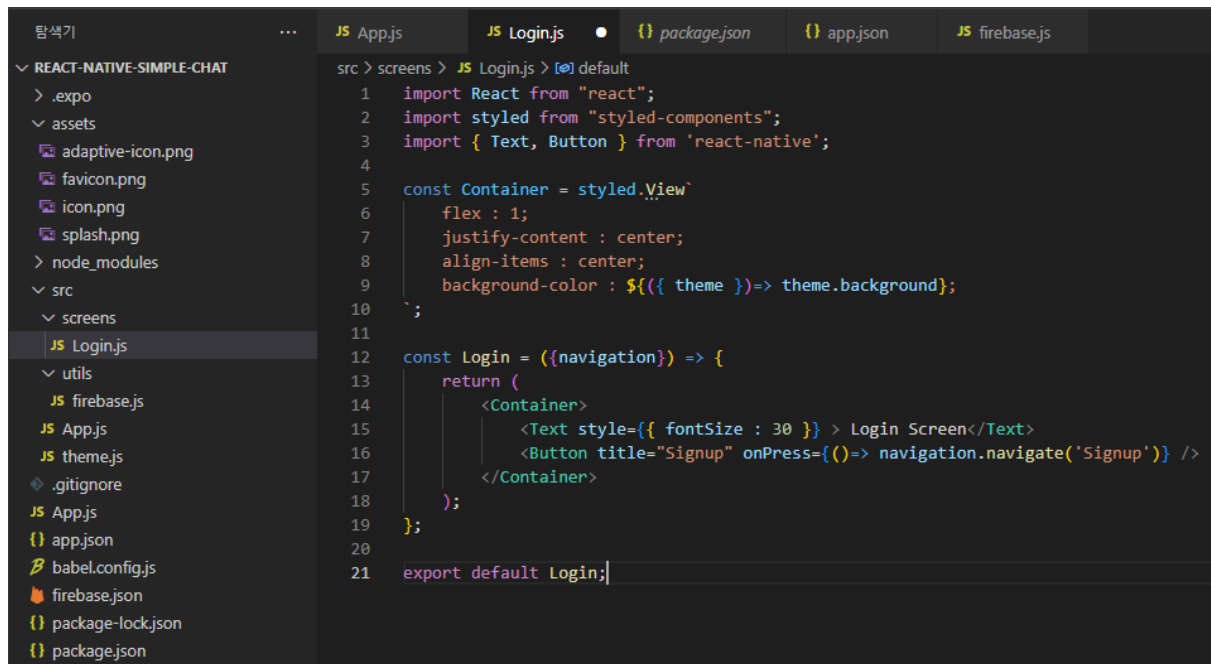
그림 p315

인증을 위해 이메일과 비밀번호가 필요하므로 로그인 및 회원가입 화면에서는 이메일과 비밀번호를 필수로 입력받고, 회원가입 시 사용자가 서비스에서 사용할 이름과 프로필 사진을 받도록 화면을 구성했다.

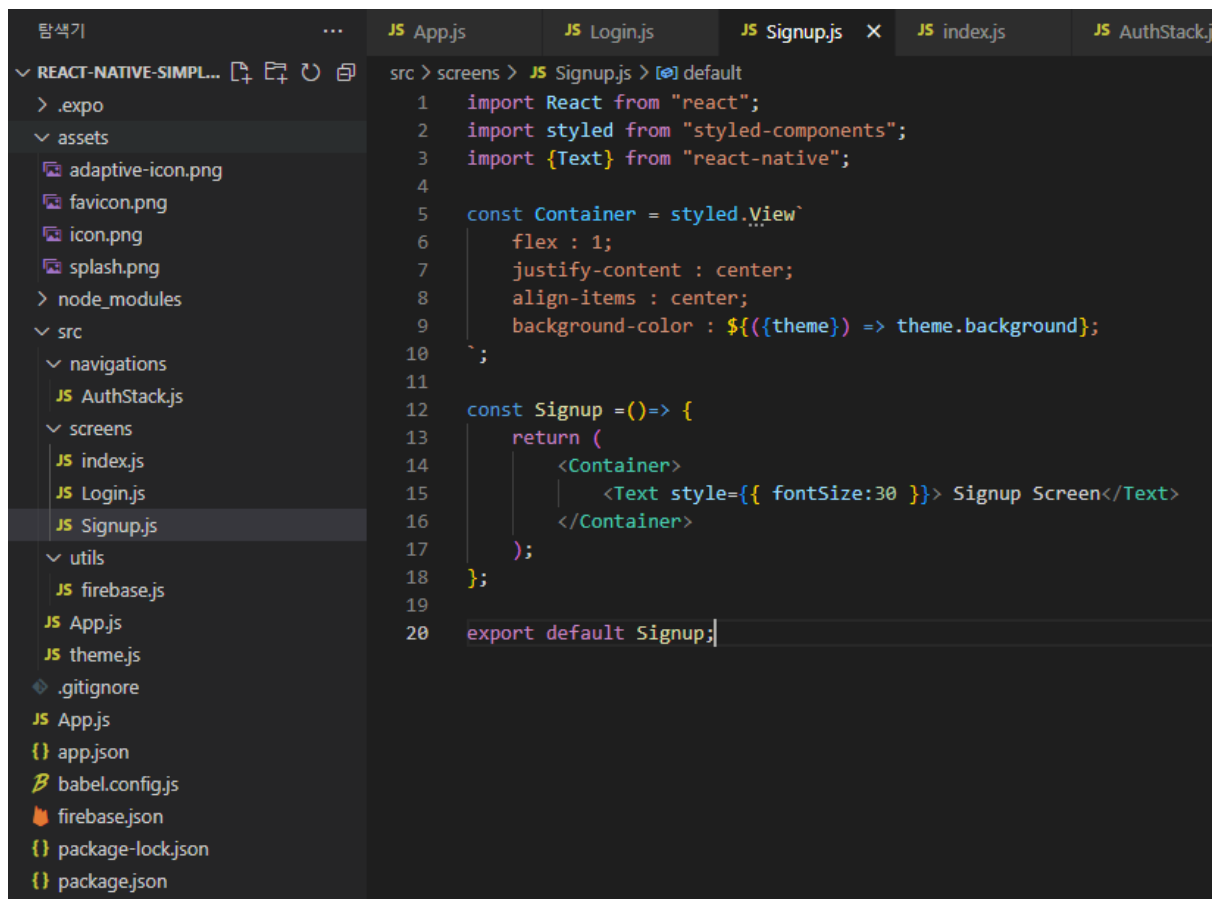
내비게이션

먼저 로그인 화면과 회원가입 화면으로 사용할 컴포넌트를 screens 폴더 밑에 만든다.

그후 로그인 화면을 아래처럼 작성한다.

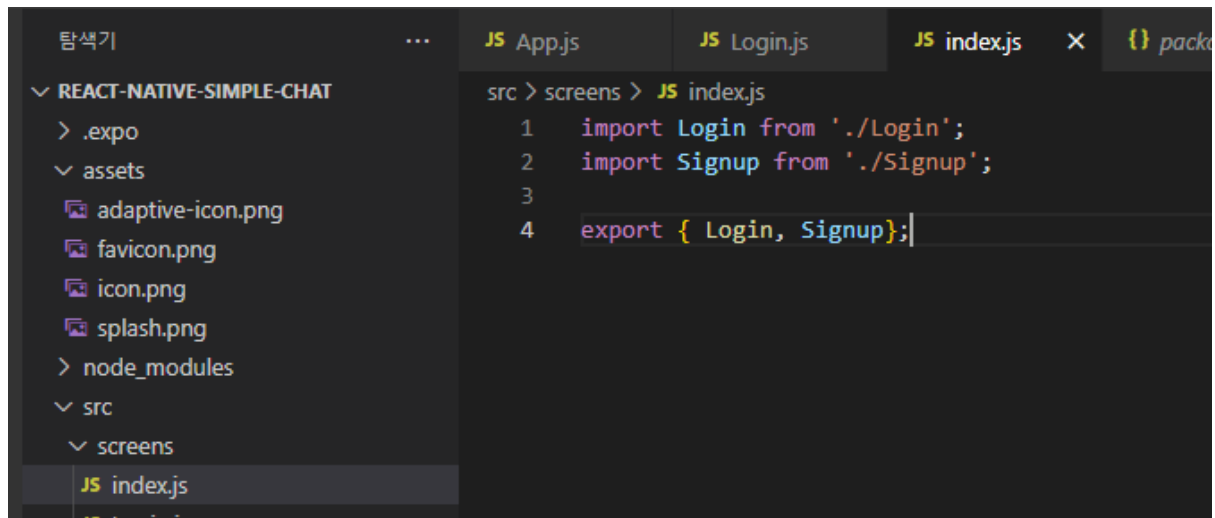


```
src > screens > JS Login.js > [🔍] default
1  import React from "react";
2  import styled from "styled-components";
3  import { Text, Button } from 'react-native';
4
5  const Container = styled.View`
6    flex : 1;
7    justify-content : center;
8    align-items : center;
9    background-color : ${({ theme })=> theme.background};
10 `;
11
12 const Login = ({navigation}) => {
13   return (
14     <Container>
15       <Text style={{ fontSize : 30 }} > Login Screen</Text>
16       <Button title="Signup" onPress={()=> navigation.navigate('Signup')} />
17     </Container>
18   );
19 };
20
21 export default Login;
```



```
src > screens > JS Signup.js > [🔍] default
1  import React from "react";
2  import styled from "styled-components";
3  import {Text} from "react-native";
4
5  const Container = styled.View`
6    flex : 1;
7    justify-content : center;
8    align-items : center;
9    background-color : ${({theme}) => theme.background};
10 `;
11
12 const Signup = ()=> {
13   return (
14     <Container>
15       <Text style={{ fontSize:30 }}> Signup Screen</Text>
16     </Container>
17   );
18 };
19
20 export default Signup;
```

회원가입 화면도 로그인 화면과 마찬가지로 화면을 확인할 수 있는 텍스트가 있는 간단한 화면으로 구성했다. 두 화면이 완성되면 screens폴더에 index.js파일을 생성하고 다음과 같이 작성한다.



이제 화면 준비가 완료되었으므로 내비게이션 파일을 작성해보자.

내비게이션 파일을 관리하는 `navigations` 폴더 아래에 스택 내비게이션을 이용해서 아래처럼 작성해보자.

첫 화면을 로그인 화면으로 하고 로그인 화면과 회원가입을 가진 내비게이션을 만들었다. 스타일드 컴포넌트에서 제공하는 `ThemeContext`와 `useContext` Hook 함수를 `theme`을 받아오고, 내비게이션 화면의 배경색 `theme`에 정의된 배경색으로 설정했다. 마지막으로 헤더의 타이틀 위치를 안드로이드와 ios에서 동일한 위치에 렌더링하기 위해 `headerTitleAlign`의 값을 `center`로 설정했다.

`authStack` 내비게이션 작성이 완료되면 `navigations` 폴더 안에 `index.js` 파일을 생성하고 아래처럼 작성하자.

```
JS Signup.js U    JS index.js ...\screens U    JS AuthStack.js U X    JS index.js ...\navigations U

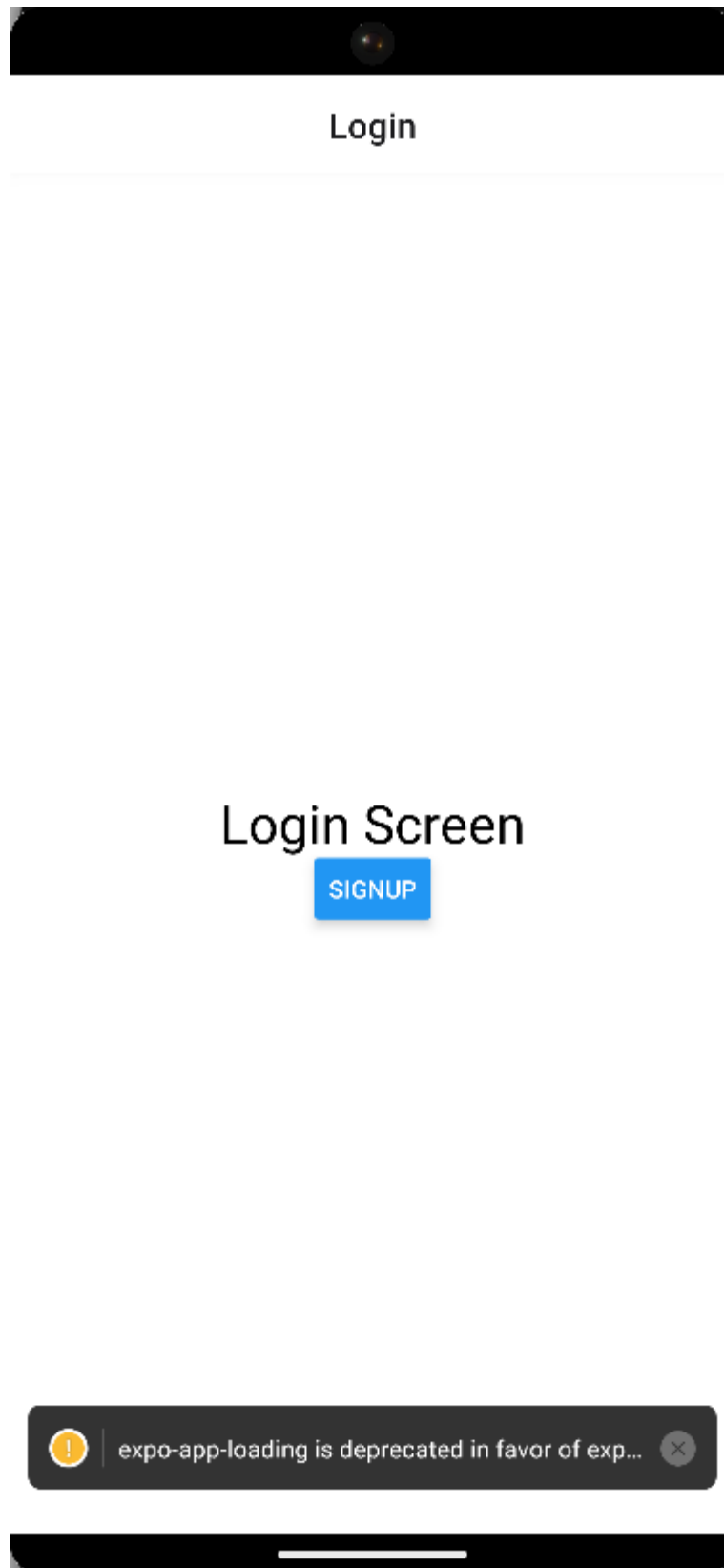
src > navigations > JS AuthStack.js > [X] AuthStack
1  import React, {useContext} from "react";
2  import { ThemeContext } from "styled-components";
3  import { createStackNavigator } from "@react-navigation/stack";
4  import { Login, Signup } from '../screens';
5
6  const Stack = createStackNavigator();
7
8  const AuthStack = () => {
9      const theme = useContext(ThemeContext);
10
11      return(
12          <Stack.Navigator
13              initialRouteName="Login"
14              screenOptions={{
15                  headerTitleAlign : "center",
16                  cardStyle : { backgroundColor : theme.background},
17              }}
18          >
19              <Stack.Screen name="Login" component={Login} />
20              <Stack.Screen name="Signup" component={Signup} />
21          </Stack.Navigator>
22      );
23  };
24
25  export default AuthStack;
```

```
탐색기    ...    JS index.js    X

v REACT-NATIVE-SIMPLE-CHAT
  > .expo
  v assets
    adaptive-icon.png
    favicon.png
    icon.png
    splash.png
  > node_modules
  v src
    v navigations
      JS AuthStack.js
      JS index.js
    v screens
      JS index.js
      JS Login.js
      JS Signup.js

src > navigations > JS index.js > [X] default
1  import React from "react";
2  import { NavigationContainer } from "@react-navigation/native";
3  import AuthStack from "../AuthStack";
4
5  const Navigation = () => {
6      return(
7          <NavigationContainer>
8              <AuthStack />
9          </NavigationContainer>
10      );
11  };
12
13
14  export default Navigation;
```

NavigationContainer 컴포넌트를 사용하고 자식 컴포넌트로 AuthStack 내비게이션을 사용했다. 이제 작성된 내비게이션이 렌더링 되도록 App 컴포넌트를 수정하자.



로그인 화면

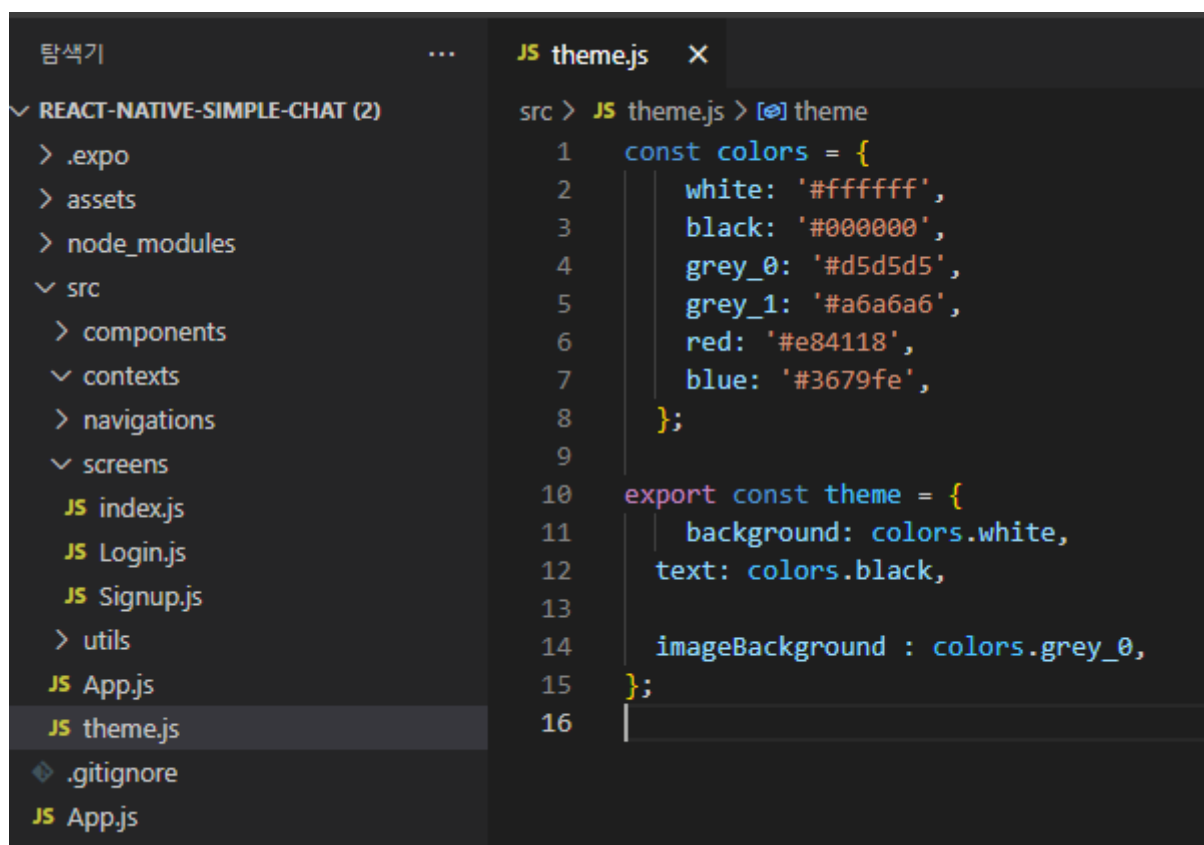
로그인 화면에서 사용자의 이메일과 비밀번호를 입력받을 수 있도록 화면을 만들어보자. 로그인 화면에서는 로고를 렌더링하는 컴포넌트와 사용자의 입력을 받는 컴포넌트, 크리고 클릭과 그에 따른 이벤트가 발생하는 컴포넌트가 필요하다.

image컴포넌트

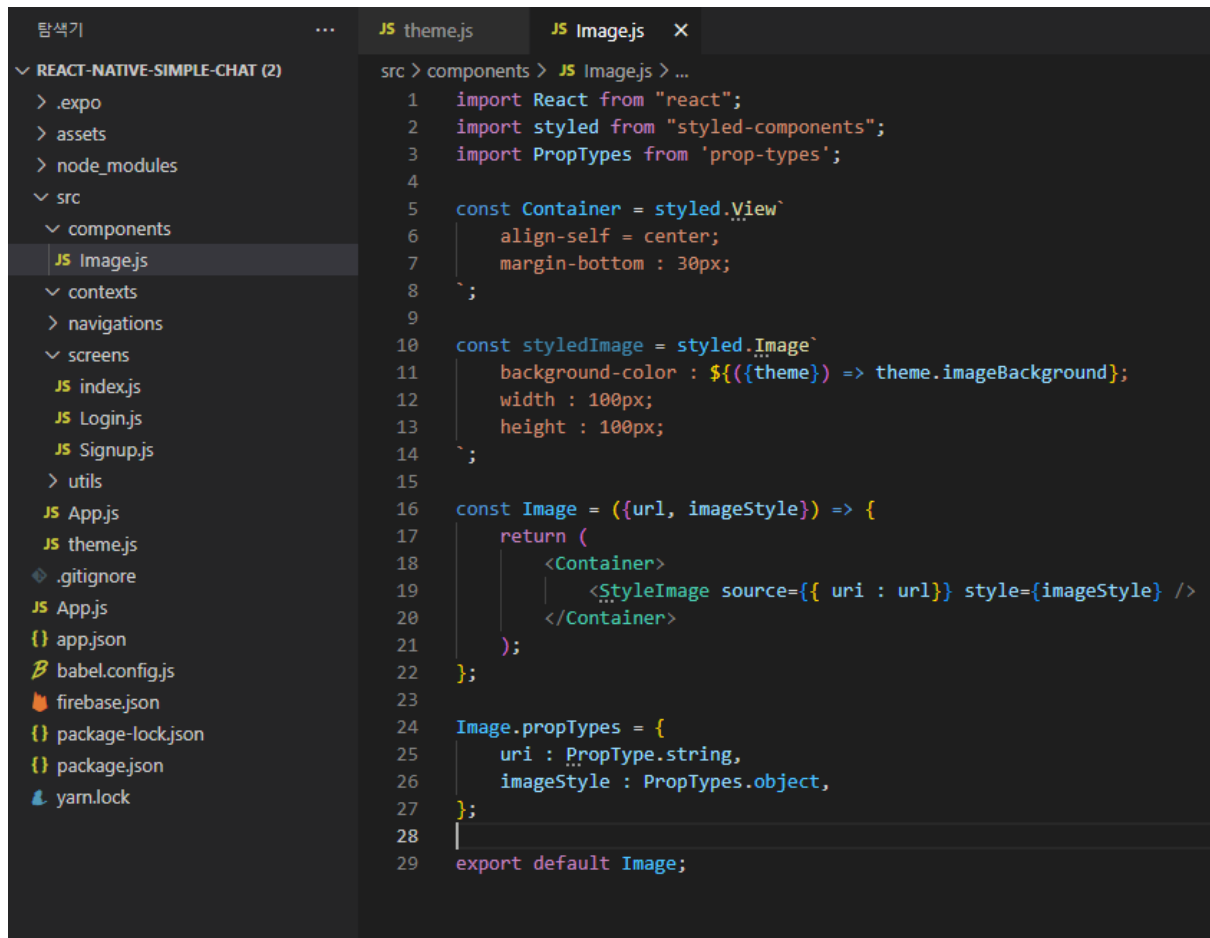
먼저 url 을 전달받아 원격에 있는 이미지를 렌더링하는 image 컴포넌트를 만들자.

로그인 화면에서는 image 컴포넌트를 이용해 애플리케이션의 로고를 렌더링한다.

우선 Image컴포넌트의 배경색으로 사용할 값을 theme.js파일에 정의하고 진행한다.

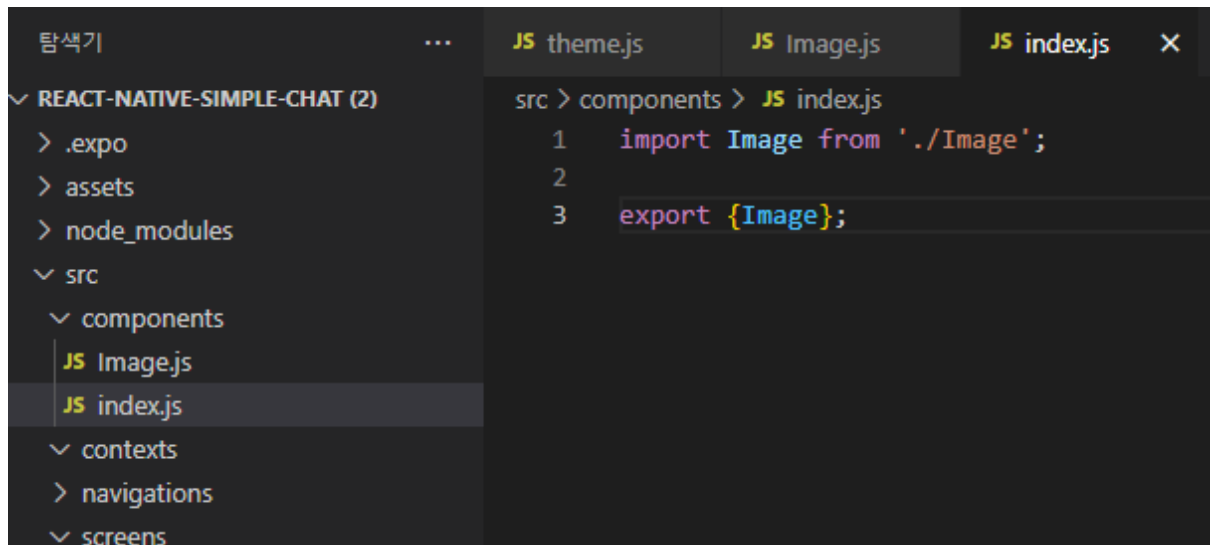


이제 Image 컴포넌트를 작성할 Image.js파일을 components폴더 안에 만들고 아래처럼 작성하자.

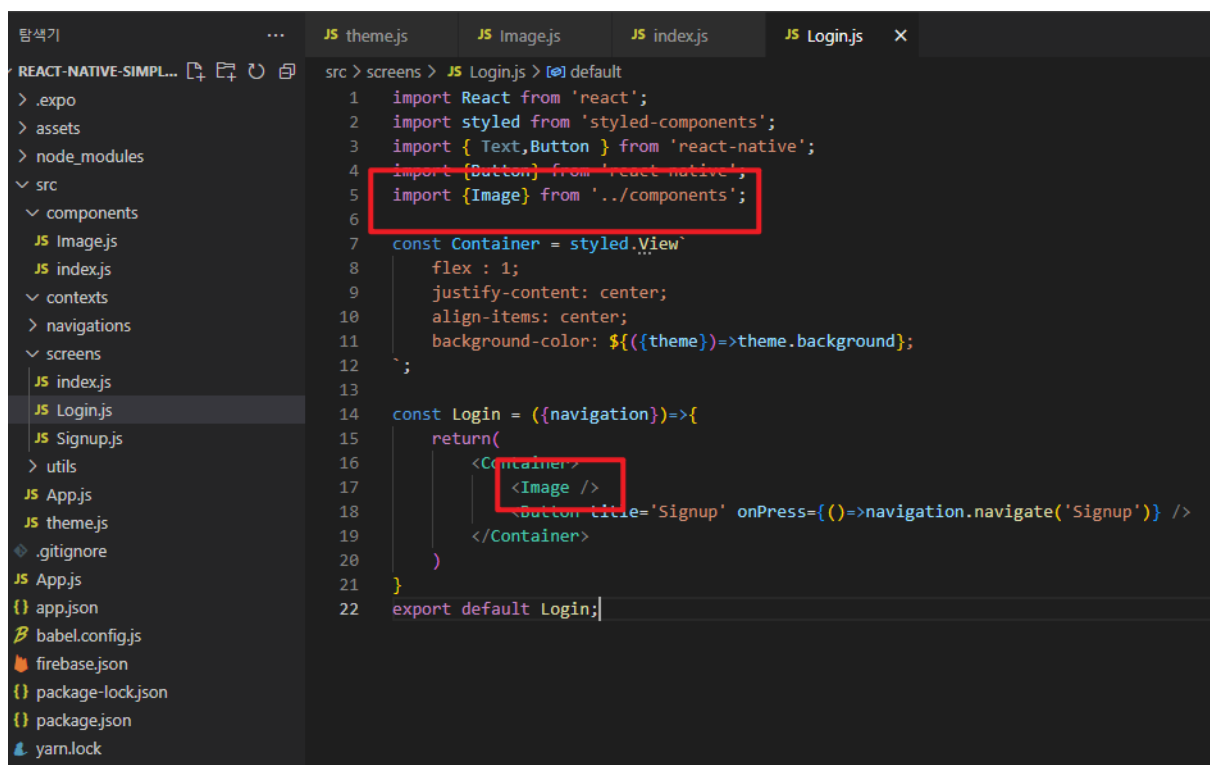


props로 전달되는 url을 렌더링하고 imageStyle을 전달받아 컴포넌트의 스타일을 수정할 수 있는 Image 컴포넌트를 만들었다.

Image 컴포넌트 작성이 완료되면 components 폴더에 index.js 파일을 생성하고 다음과 같이 작성한다.



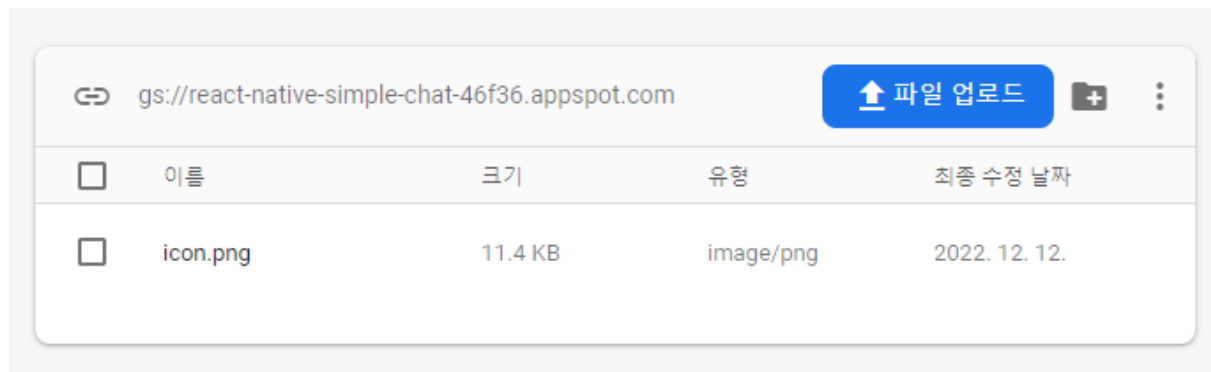
이제 image 컴포넌트를 사용해서 Login화면을 수정해보자.



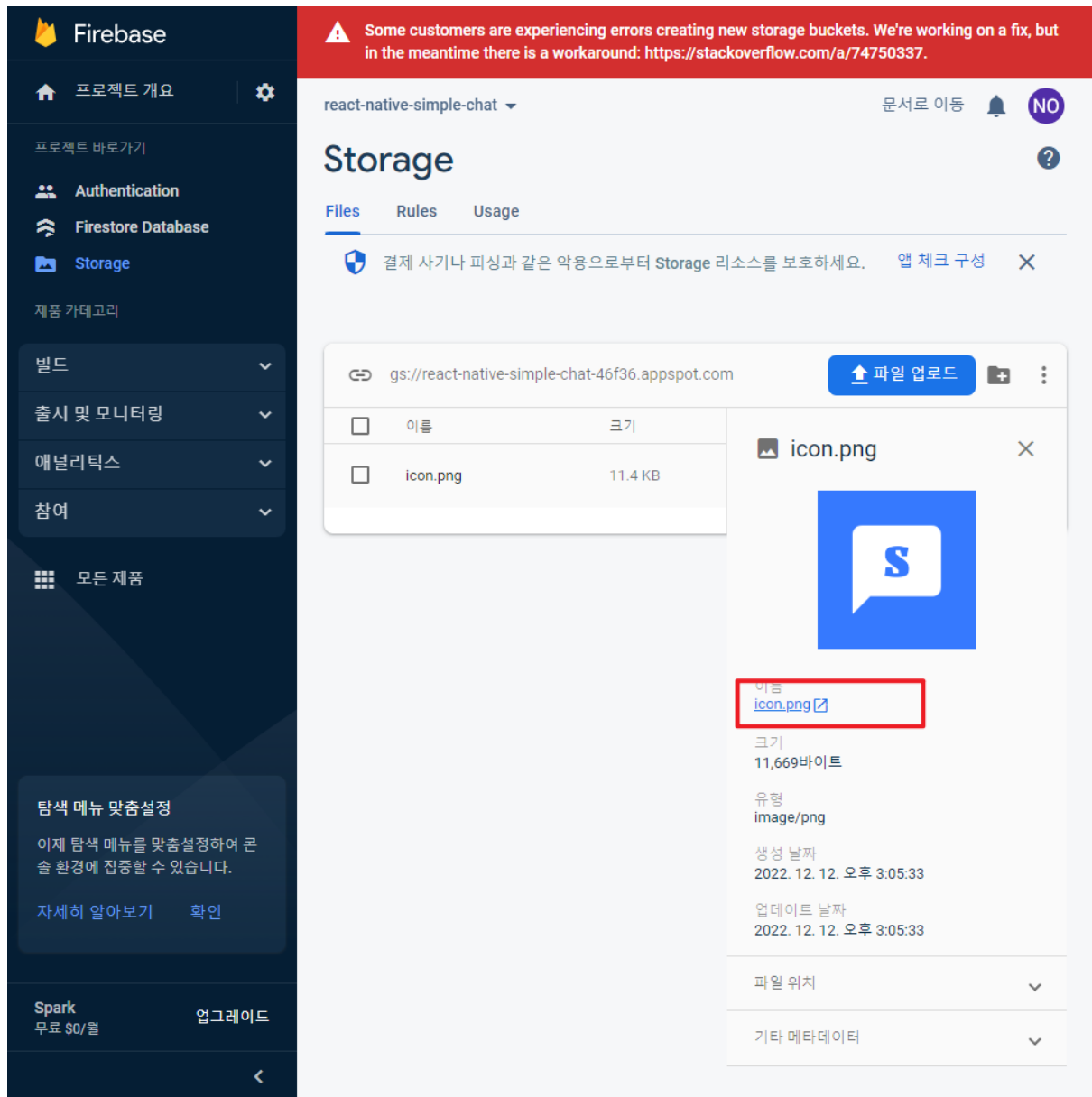
로고적용하기

애플리케이션의 로고를 파이어베이스 스토리지에 업로드하고 로그인 화면에서 사용하도록 만들어보자. 앞에서 작성한 Image 컴포넌트 크기인 100x100 보다 큰 사이즈로 로고 이미지를 준비하고, 파일을 파이어베이스의 스토리지에 업로드하자.

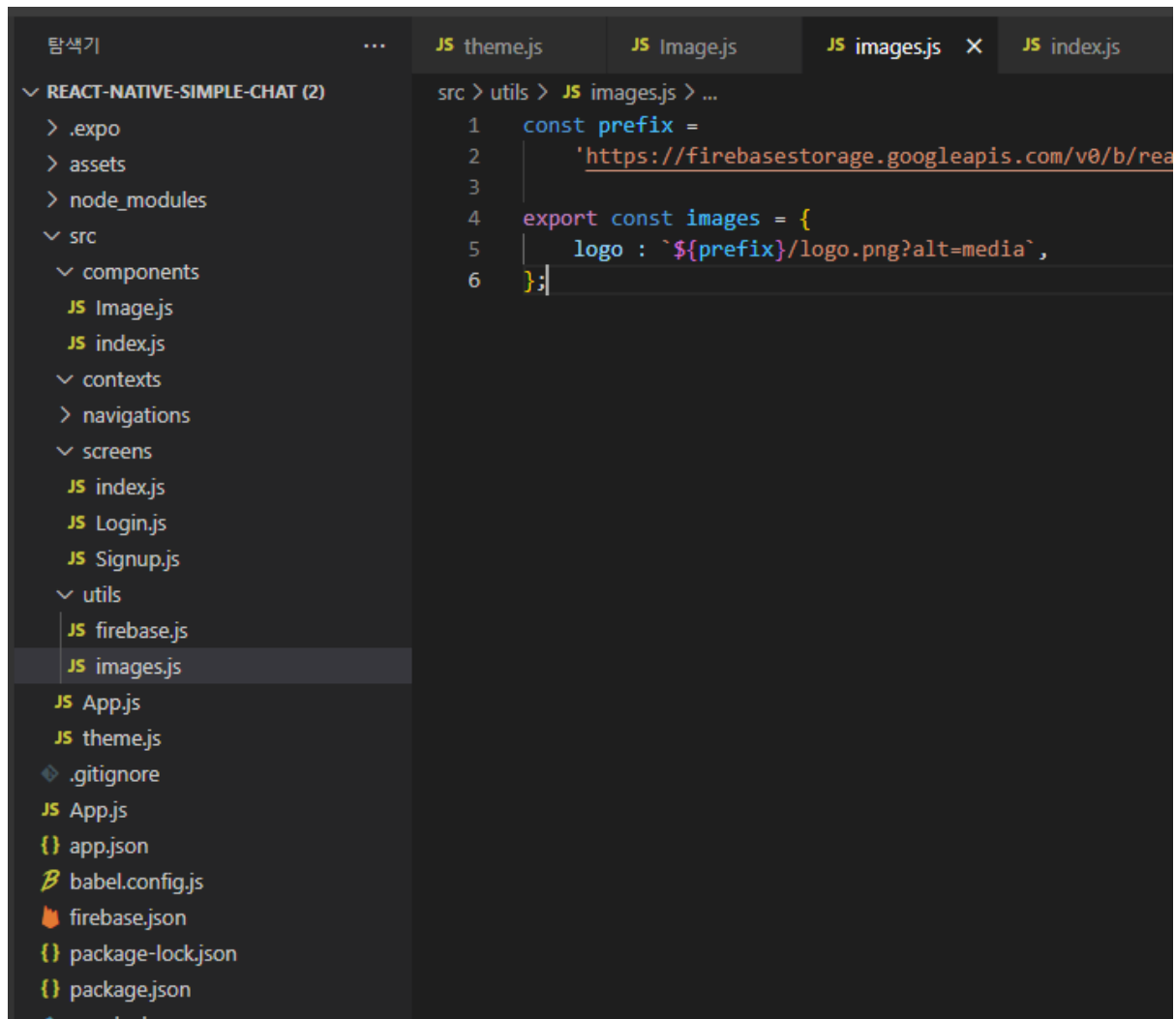
여기에선 200x200 사이즈의 이미지를 사용했다.



스토리지에 파일을 업로드하고 파일 정보에서 이름을 클릭하면 해당 파일의 url을 얻을 수 있다.



스토리지에 업로드된 이미지의 url을 관리하기 위해 utils 폴더 밑에 images.js 파일을 생성하고 앞에서 얻은 url을 이용해 다음과 같이 작성한다.



여기서 주의할 점은 앞의 코드처럼 복사된 주소의 쿼리스트링에서 token 부분을 제외하고 사용해야한다.

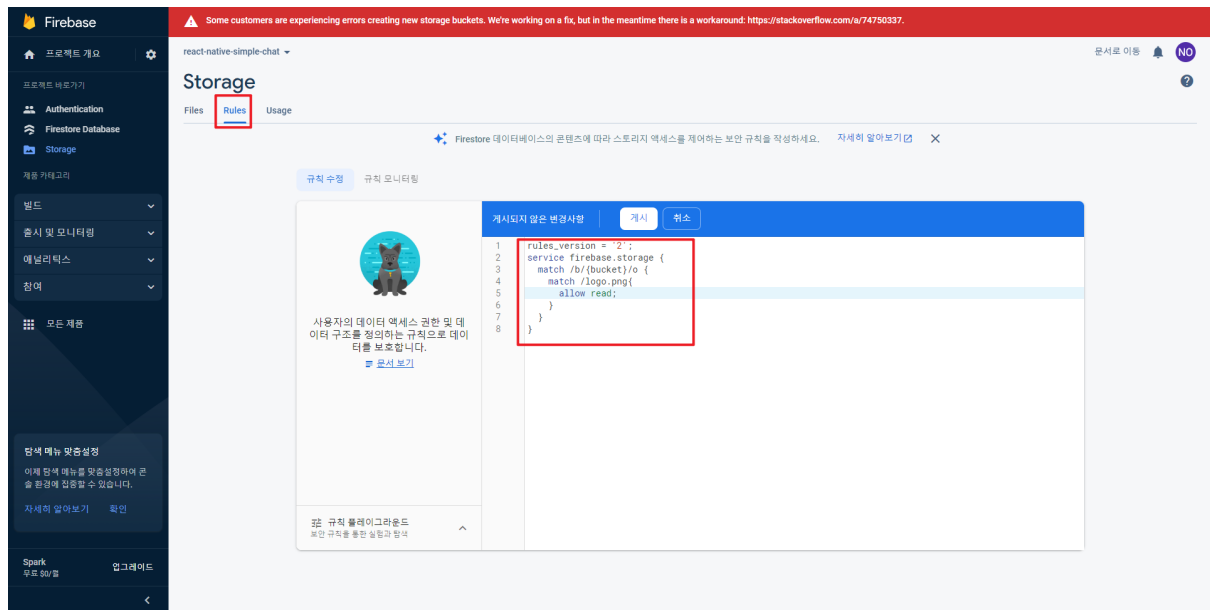
쿼리 스트링에 있는 token은 현재 로그인된 사용자에게 발급된 값입니다. 실제 사용할 때는 token이 변경될 뿐만 아니라, 로그인 화면에서는 아직 로그인 전이므로 token이 없는 상태로 접근이 가능해야 한다. 이제 로고 이미지도 로딩 과정에서 미리 불러오도록 App컴포넌트를 아래처럼 수정하자.


```
JS theme.js JS Image.js JS images.js JS App.js JS index.js JS Login.js X
src > screens > JS Login.js > default
1 import React from 'react';
2 import styled from 'styled-components';
3 import {Button} from 'react-native';
4 import {Image} from 'react-native';
5 import {images} from '../utils/images';
6
7 const Container = styled.View`
8   flex : 1;
9   justify-content: center;
10  align-items: center;
11  background-color: ${({theme})=>theme.background};
12 `;
13
14 const Login = ({navigation})=>{
15   return(
16     <Container>
17       <Image url={images.logo}/>
18       <Button title='Signup' onPress={()=>navigation.navigate('Signup')} />
19     </Container>
20   )
21 }
22 export default Login;
```

결과를 보면 업로드한 이미지가 나타나지 않고 다음과 같은 경고가 나타난다.

Error : Failed to load https~~

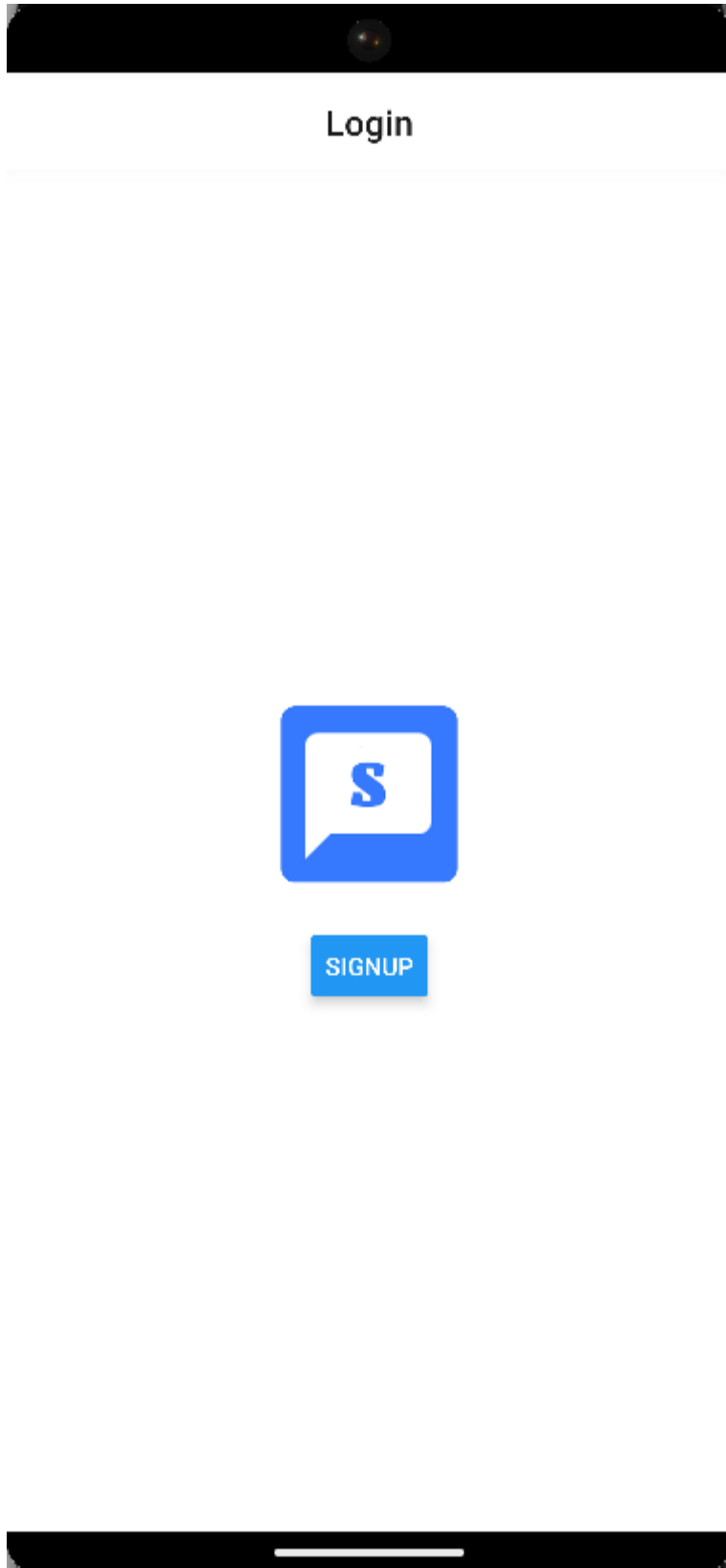
위 문제는 스토리지의 파일 접근 권한 문제로, 보안 규칙을 수정해서 해결해야 한다. 스토리지 메뉴의 “Rules” 탭에서 로그인하지 않아도 파일을 읽을 수 있도록 아래처럼 규칙을 수정하자.



```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /logo.png {
      allow read;
    }
  }
}
```

※ “규칙 플레이그라운드”를 이용하면 수정한 규칙을 적용하기 전에 원하는 조건으로 테스트를 진행할 수 있다.

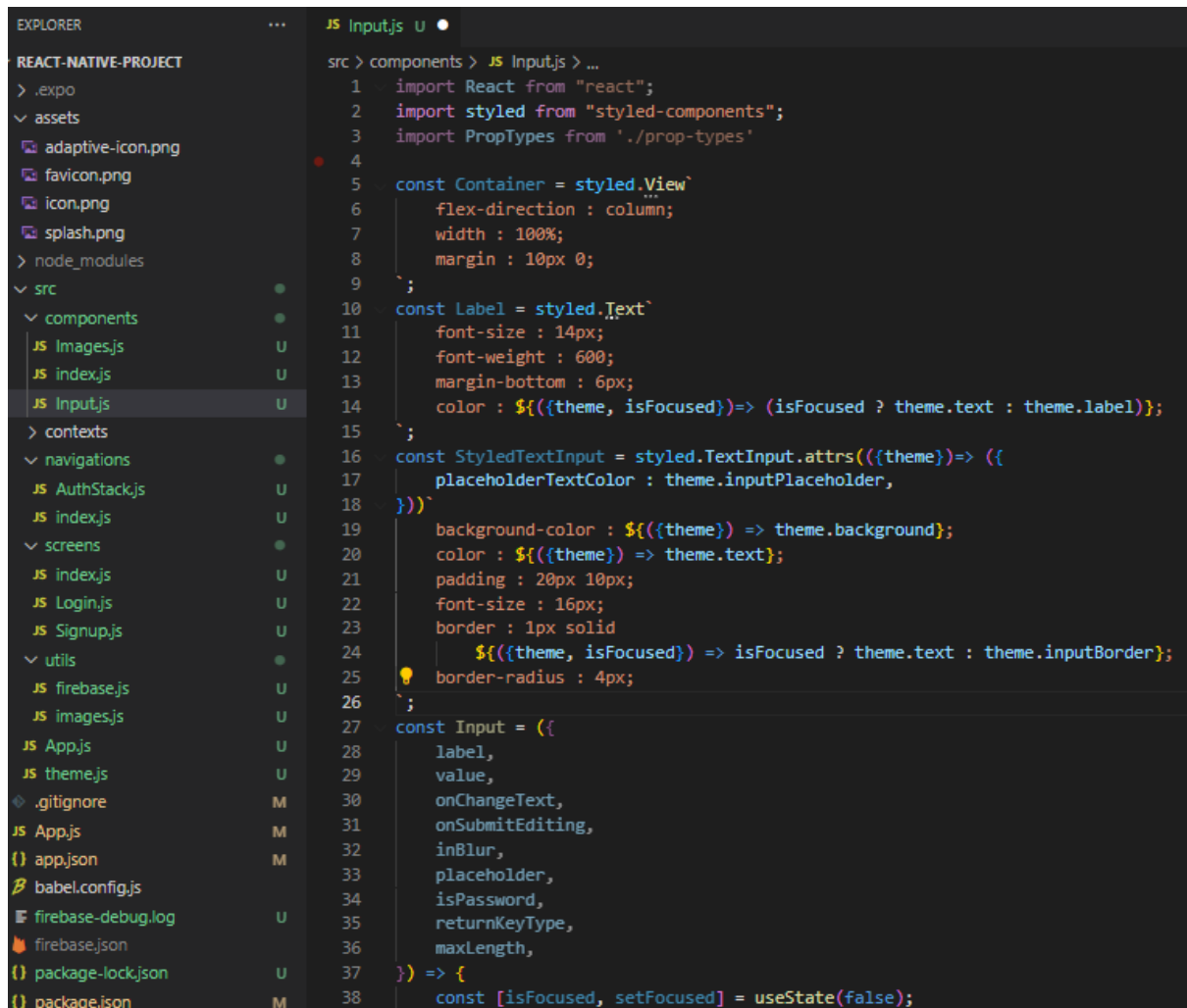
규칙을 적용시키고 화면을 다시 확인하면 스토리지에 업로드한 로고가 잘 나타나는 것을 볼 수 있다. 마지막으로 로그인 화면에서 Image 컴포넌트에 imageStyle을 전달해 렌더링되는 로고의 모습을 조금 변경하자.



input 컴포넌트

이번에는 아이디와 비밀번호를 입력받을 수 있도록 Input컴포넌트를 만들어보자. 먼저 Input컴포넌트에서 placeholder 등에 사용할 색을 정의하자.

동일한 색을 사용했지만 이후 유지보수를 위해 적용되는 부분을 명확하게 알 수 있도록 정의했다. 이제 components 폴더 밑에 Input.js 파일을 생성하고 Input 컴포넌트를 만들어보자.



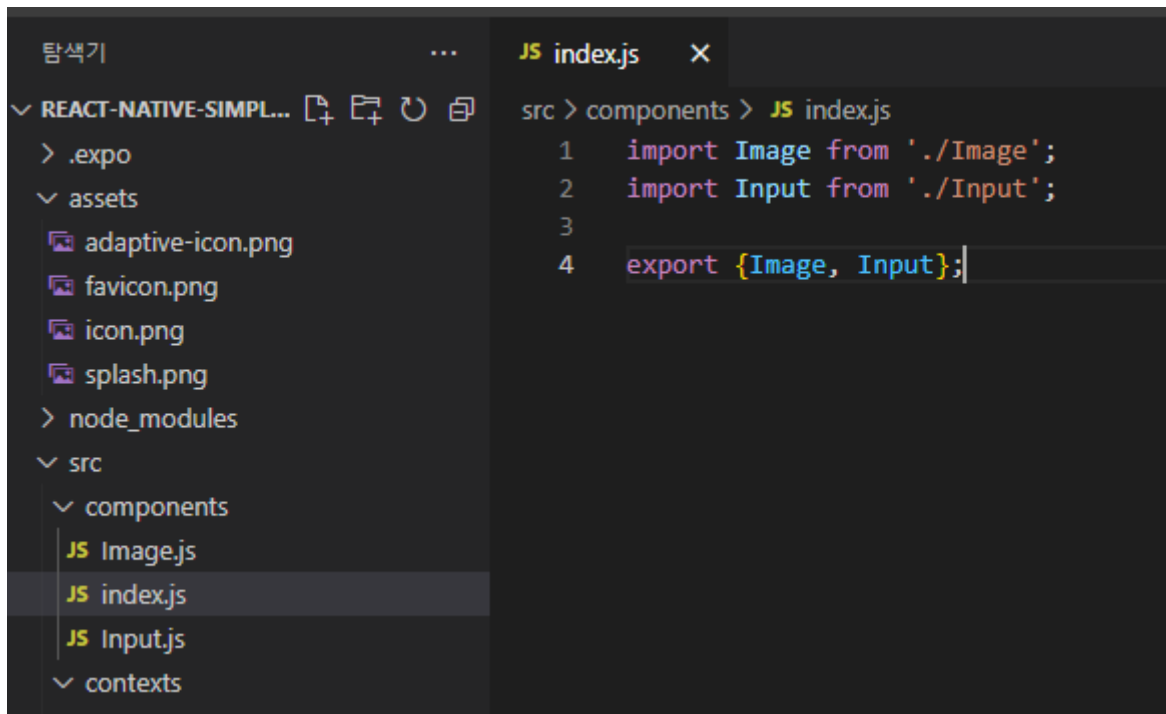
```
1 import React from "react";
2 import styled from "styled-components";
3 import PropTypes from './prop-types'
4
5 const Container = styled.View`
6   flex-direction : column;
7   width : 100%;
8   margin : 10px 0;
9 `;
10 const Label = styled.Text`
11   font-size : 14px;
12   font-weight : 600;
13   margin-bottom : 6px;
14   color : ${({theme, isFocused})=> (isFocused ? theme.text : theme.label)};
15 `;
16 const StyledTextInput = styled.TextInput.attrs(({theme})=> ({
17   placeholderTextColor : theme.inputPlaceholder,
18 })))`
19   background-color : ${({theme}) => theme.background};
20   color : ${({theme}) => theme.text};
21   padding : 20px 10px;
22   font-size : 16px;
23   border : 1px solid
24     ${({theme, isFocused}) => isFocused ? theme.text : theme.inputBorder};
25   border-radius : 4px;
26 `;
27 const Input = ({
28   label,
29   value,
30   onChangeText,
31   onSubmitEditing,
32   inBlur,
33   placeholder,
34   isPassword,
35   returnKeyType,
36   maxLength,
37 }) => {
38   const [isFocused, setFocused] = useState(false);
```

```
JS theme.js JS Input.js X JS Image.js JS images.js JS App.js JS
src > components > JS Input.js > [x] StyledTextInput
39
40   return(
41     <Container>
42       <Label isFocused={isFocused}>{label}</Label>
43       <StyledTextInput
44         isFocused={isFocused}
45         value={value}
46         onChangeText={onChangeText}
47         onSubmitEditing={onSubmitEditing}
48         onFocus={() => setIsFocused(true)}
49         onBlur={() => {
50           setIsFocused(false);
51           onBlur();
52         }}
53         placeholder={placeholder}
54         secureTextEntry={isPassword}
55         returnKeyType={returnKeyType}
56         maxLength={maxLength}
57         autoCapitalize="none"
58         autoComplete={false}
59         keyboardType="none" // IOS ONLY
60         underlineColorAndroid = "transparent" // Android only
61       />
62     </Container>
63   );
64 };
```

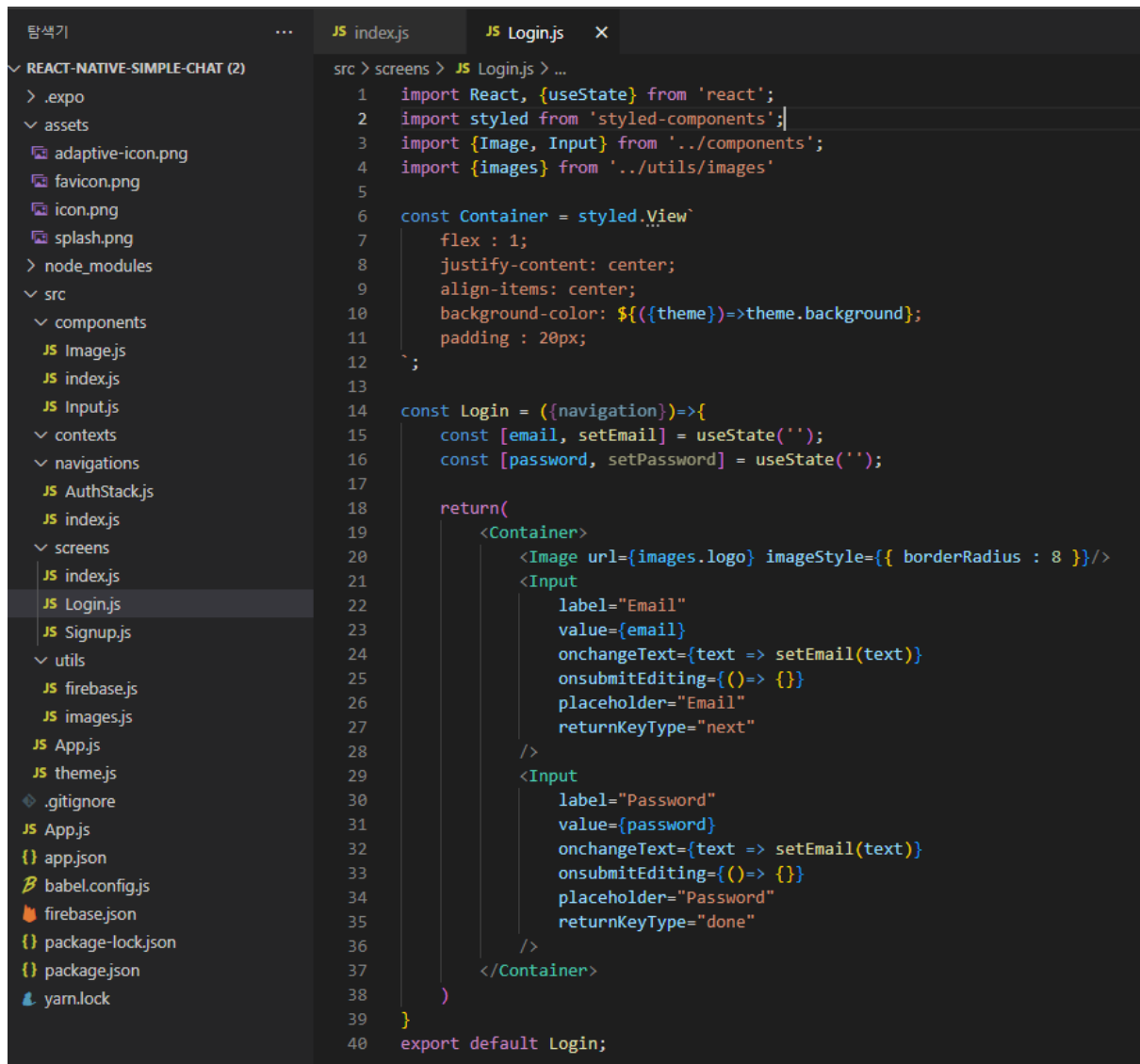


```
JS theme.js JS Input.js X JS Image.js JS images.js JS Ap
src > components > JS Input.js > ...
57         autoCapitalize="none"
58         autoComplete={false}
59         keyboardType="none" // IOS ONLY
60         underlineColorAndroid = "transparent" // An
61     />
62 </Container>
63 );
64 };
65
66 Input.defaultProps = {
67     onBlur : () => {},
68 };
69
70 Input.propTypes = {
71     label : PropTypes.string.isRequired,
72     value : PropTypes.string.isRequired,
73     onChangeText : PropTypes.func.isRequired,
74     onSubmitEditing : PropTypes.func.isRequired,
75     onBlur : PropTypes.func,
76     placeholder : PropTypes.string,
77     isPassword : PropTypes.bool,
78     returnKeyType : PropTypes.oneOf(['done', 'next']),
79     maxLength : PropTypes.number,
80 };
81
82 export default Input;
83
```

라벨을 TextInput 컴포넌트 위에 렌더링하고 포커스 여부에 따라 스타일이 변경되는 Input 컴포넌트를 만들었다. `secureTextEntry` 속성은 입력되는 문자를 감추는 기능으로 비밀번호를 입력하는 곳에서 많이 사용된다. Input 컴포넌트 작성이 완료되면 components 폴더의 index.js 컴포넌트를 아래 처럼 수정하자.




이제 사용자의 이메일과 비밀번호를 입력받을 수 있도록 Input 컴포넌트를 이용해 로그인 화면을 아래처럼 수정하자.



```
1 import React, {useState} from 'react';
2 import styled from 'styled-components';
3 import {Image, Input} from '../components';
4 import {images} from '../utils/images'
5
6 const Container = styled.View`
7   flex : 1;
8   justify-content: center;
9   align-items: center;
10  background-color: ${({theme})=>theme.background};
11  padding : 20px;
12 `;
13
14 const Login = ({navigation})=>{
15   const [email, setEmail] = useState('');
16   const [password, setPassword] = useState('');
17
18   return(
19     <Container>
20       <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
21       <Input
22         label="Email"
23         value={email}
24         onChangeText={text => setEmail(text)}
25         onSubmitEditing={()=> {}}
26         placeholder="Email"
27         returnKeyType="next"
28       />
29       <Input
30         label="Password"
31         value={password}
32         onChangeText={text => setEmail(text)}
33         onSubmitEditing={()=> {}}
34         placeholder="Password"
35         returnKeyType="done"
36       />
37     </Container>
38   )
39 }
40 export default Login;
```

입력되는 이메일과 비밀번호를 관리할 email과 password를 useState 함수로 생성하고 각 이메일과 비밀번호를 입력받는 Input 컴포넌트의 value로 지정했다. 비밀번호를 입력받는 Input 컴포넌트는 입력되는 값이 보이지 않도록 isPassword 속성을 추가했다.

Login



Email

Email

Password

Password