

day52-rn-chatapp2

☰ 태그	
📅 날짜	@2022년 12월 13일

로그인 화면에서 이메일을 입력받는 Input 컴포넌트의 return KeyType을 next로 설정하고 비밀번호를 입력받는 Input 컴포넌트는 done으로 설정했다. 이번에는 useRef를 이용해 이메일을 입력받는 Input 컴포넌트에서 키보드의 next버튼을 클릭하면 비밀번호를 입력하는 Input 컴포넌트로 포커스가 이동되는 기능을 추가하겠다.

```
JS Login.js ×
src > screens > JS Login.js > [e] Login
1  import React, {useState, useRef} 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     const passwordRef = useRef();
18
19     return(
20       <Container>
21         <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
22         <Input
23           label="Email"
24           value={email}
25           onChangeText={text => setEmail(text)}
26           onSubmitEditing={() => passwordRef.current.focus()}
27           placeholder="Email"
28           returnKeyType="next"
29         />
30         <Input
31           label="Password"
32           value={password}
33           onChangeText={text => setEmail(text)}
34           onSubmitEditing={() => {}}
35           placeholder="Password"
36           returnKeyType="done"
37           ref={passwordRef}
38         />
39       </Container>
40     );
41   };
42
43  export default Login;
```

useRef를 이용하여 passwordRef를 만들고 비밀번호를 입력하는 Input 컴포넌트의 ref로 지정했다. 이메일을 입력하는 Input 컴포넌트의 onSubmitEditing 함수를 passwordRef를 이용해서 비밀번호를 입력하는 Input 컴포넌트로 포커스가 이동되도록 수정했다.

이제 Input 컴포넌트에 전달된 ref를 이용해 TextInput 컴포넌트의 ref로 지정해야한다. 하지만 ref는 key처럼 리액트에서 특별히 관리되기 때문에 자식 컴포넌트의 props로 전달되지 않는다. 이런 상황에서 forwardRef함수를 이용하면 ref를 전달받을 수 있다.

```
JS Login.js JS Input.js X
src > components > JS Input.js > [🔍] Label
1  import React, {useState, forwardRef} from "react";
2  import styled from "styled-components";
3  import PropTypes from 'prop-types';
4
```

```
JS Login.js JS Input.js X
src > components > JS Input.js > [🔍] Label
25  border-radius : 4px;
26  ;
27  const Input = forwardRef(
28  (
29    {
30      label,
31      value,
32      onChangeText,
33      onSubmitEditing,
34      onBlur,
35      placeholder,
36      isPassword,
37      returnKeyType,
38      maxLength,
39    },
40    ref
41  ) => {
42    const [isFocused, setIsFocused] = useState(false);
43
44    > return( ...
68    );
69  }
70 );
71
72 Input.defaultProps = {
73   onBlur : () => {},
74 };
75
```

비밀번호를 입력하는 Input 컴포넌트로 포커스 이동이 잘 되는지 확인해보자.

키보드 감추기

뭔가 입력하는 곳에서 입력 도중 다른 곳을 터치하면 키보드가 사라지는데, 이는 사용자 편의를 위한 일반적인 애플리케이션의 동작 방식이다. 그리고 입력받는 컴포넌트의 위치에 따라 키보드가 내용을 가리고 있다면 스크롤을 통해 입력되는 모습을 사용자가 확인할 수 있도록 하는 것이 좋다.

입력 도중 다른 곳을 터치하면 키보드가 사라지는 기능과 키보드가 입력받는 컴포넌트를 가리지 않도록 하는 방법에 대해 알아보자.

리액트에서 제공하는 기능으로 다른 영역을 터치했을 때 키보드를 감추는 기능을 만들기 위해서는

TouchableWithoutFeedback 컴포넌트와 KeyboardAPI를 이용한다.

TouchableWithoutFeedback 컴포넌트는 클릭에 대해 상호 작용은 하지만 스타일 속성이 없고 반드시 하나의 자식 컴포넌트를 가져야 하는 특징이 있다. Keyboard API는 리액트 네이티브에서 제공하는 키보드 관련 api로 키보드 상태에 따른 이벤트 등록에 많이 사용되며, Keyboard API에서 제공하는 dismiss 함수는 활성화된 키보드를 닫는 기능이다.

아래처럼

- `import {TouchableWithoutFeedback, Keyboard} from 'react-native-gesture-handler';`

사용 시

A screenshot of a console error message. The text is: "ERROR AppLoading threw an unexpected error when loading: TypeError: undefined is not an object (evaluating '_reactNativeGestureHandler.Keyboard.dismiss')". The word "ERROR" is in a red box, and "TypeError" is in a yellow box.

위같은 에러가 생길 수 있다. 이럴 경우

- `import {TouchableWithoutFeedback, Keyboard} from 'react-native';`

로 변경하면 된다.

```
JS Login.js  X  JS Input.js
src > screens > JS Login.js > [e] Login
4 import {images} from '../utils/images'
5 import { TouchableWithoutFeedback, Keyboard } from 'react-native-gesture-handler';
6
7 > const Container = styled.View` ...
13 `;
14
15 const Login = ({navigation})=>{
16   const [email, setEmail] = useState('');
17   const [password, setPassword] = useState('');
18   const passwordRef = useRef();
19
20   return(
21     <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
22       <Container>
23         <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
24         <Input
25 >           label="Email" ...
31         />
32         <Input
33 >           label="Password" ...
39           ref={passwordRef}
40         />
41       </Container>
42     </TouchableWithoutFeedback>
43   )
}
```

TouchableWithoutFeedback 컴포넌트와 Keyboard API를 이용해서 만든 화면을 확인해 보면 입력 도중 다른 영역을 터치할 경우 키보드가 사라지는 것을 볼 수 있다. 하지만 위치에 따라 키보드가 Input 컴포넌트를 가리는 문제는 해결하지 못한다.

react-native-keyboard-aware-scroll-view 라이브러리를 이용하면 이런 고민을 쉽게 해결할 수 있다. react-native-keyboard-aware-scroll-view 라이브러리는 포커스가 있는 TextInput 컴포넌트의 위치로 자동 스크롤되는 기능 등 Input 컴포넌트에 필요한 기능들을 제공한다.


아래 명령어로 라이브러리를 설치하고 로그인 화면에 적용해보자.

- npm install react-native-keyboard-aware-scroll-view - -force

```
JS Login.js • JS Input.js
src > screens > JS Login.js > ...
4 import {images} from '../utils/images'
5 import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
6
7 > const Container = styled.View` ...
13 `;
14 const Login = ({navigation})=>{
15   const [email, setEmail] = useState('');
16   const [password, setPassword] = useState('');
17   const passwordRef = useRef();
18
19   return(
20     <KeyboardAwareScrollView
21       contentContainerStyle={{ flex : 1}}
22       extraScrollHeight={20}
23     >
24       <Container>
25         <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
26         <Input
27           label="Email" ...
28         />
29         <Input
30           label="Password" ...
31           ref={passwordRef}
32         />
33       </Container>
34     </KeyboardAwareScrollView>
35   )
36 }
```

react-native-keyboard-aware-scroll-view 라이브러리에서 제공하는 KeyboardAwareScrollView 컴포넌트를 로그인 화면에 적용하면, 입력 도중 다른 영역을 터치했을 때 키보드가 사라질 뿐만 아니라 포커스를 얻은 TextInput 컴포넌트의 위치에 맞춰 스크롤이 이동하는 것을 확인할 수 있다. 스크롤되는 위치를 조정하고 싶은 경우 extraScrollHeight의 값을 조절해서 원하는 위치로 스크롤 되도록 설정할 수 있다.

Login

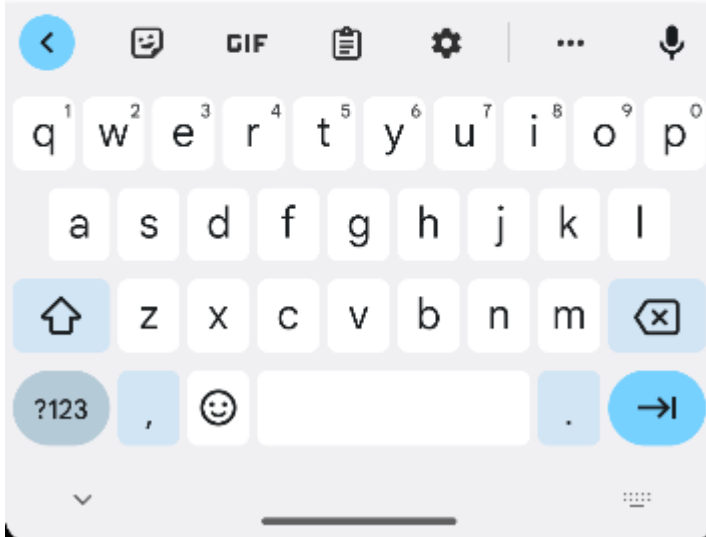


Email

Hongs@hongs.com

Password

Password



오류 메시지

Input 컴포넌트에 입력되는 값이 올바른 형태로 입력되었는지 확인하고, 잘못된 값이 입력되면 오류 메시지를 보여주는 기능을 만들어보자. 먼저 오류 메시지에서 사용할 색을 theme.js 파일에 정의하고 진행하자.

```
JS Login.js JS theme.js X JS Input.js
src > JS theme.js > [x] theme
1  import { color } from "react-native-reanimated";
2
3  const colors = {
4    white: '#ffffff',
5    black: '#000000',
6    grey_0: '#d5d5d5',
7    grey_1: '#a6a6a6',
8    red: '#e84118',
9    blue: '#3679fe',
10   };
11
12  export const theme = {
13    background: colors.white,
14    text: colors.black,
15    errorText : color.red,
16
17    imageBackground : colors.grey_0,
18    label : colors.grey_1,
19    inputPlaceholder : colors.grey_1,
20    inputBorder : colors.grey_1,
21  };
22
```

실수로 color.xx로 정의하게 되면

```
import { color } from "react-native-reanimated";
```

import 에서 위 구문을 삭제해야한다. 안그러면 오류 발생.

색의 정의가 완료되면 utils 폴더에 common.js파일을 생성하고 올바른 이메일 형식인지 확인하는 함수와 입력된 문자열에서 공백을 모두 제거하는 함수를 만들자.


```
JS Login.js JS theme.js JS common.js X JS Input.js
src > utils > JS common.js > removeWhitespace
1 export const validateEmail = email => {
2   const regex = /^[0-9?A-z0-9?]+(\.)?[0-9?A-z0-9?]+@[0-9?A-z]+\.[A-z]{2}?.?[A-z]{0,3}$/;
3   return regex.test(email);
4 };
5
6 export const removeWhitespace = text => {
7   const regex = /\s/g;
8   return text.replace(regex, '');
9 }
```

이제 로그인 화면에서 준비된 함수들을 이용해 입력되는 값이 올바른 이메일 형식인지 확인하고 알맞은 오류 메시지가 렌더링되도록 수정하자.

```
JS Login.js X JS theme.js JS common.js JS Input.js
src > screens > JS Login.js > Container
6 import { validateEmail, removeWhitespace } from '../utils/common';
7
8 const Container = styled.View`
9   flex : 1;
10  justify-content: center;
11  align-items: center;
12  background-color: ${({theme})=>theme.background};
13  padding : 20px;
14 `;
```

```
JS Login.js X JS theme.js JS common.js JS Input.js
src > screens > JS Login.js > Container
24
25 const Login = ({navigation})=>{
26   const [email, setEmail] = useState('');
27   const [password, setPassword] = useState('');
28   const passwordRef = useRef();
29   const [errorMessage, setErrorMessage] = useState('');
30
31   const _handleEmailChange = email => {
32     const changedEmail = removeWhitespace(email);
33     setEmail(changedEmail);
34     setErrorMessage(
35       validateEmail(changedEmail) ? '' : 'Please verity your email.'
36     );
37   };
38   const _handlePasswordChange = password => {
39     setPassword(removeWhitespace(password));
40   };
41 }
```

```
JS Login.js X JS theme.js JS common.js JS Input.js
src > screens > JS Login.js > [🔍] Login
40   };
41
42   return(
43     <KeyboardAwareScrollView
44       contentContainerStyle={{ flex : 1}}
45       extraScrollHeight={20}
46     >
47       <Container>
48         <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
49         <Input
50           label="Email"
51           value={email}
52           onChangeText={_handleEmailChange}
53           onSubmitEditing={()=> passwordRef.current.focus()}
54           placeholder="Email"
55           returnKeyType="next"
56         />
57         <Input
58           label="Password"
59           value={password}
60           onChangeText={_handlePasswordChange}
61           onSubmitEditing={()=> {}}
62           placeholder="Password"
63           returnKeyType="done"
64           ref={passwordRef}
65         />
66         <ErrorText>{errorMessage}</ErrorText>
67       </Container>
68     </KeyboardAwareScrollView>
69   )
70 }
71 export default Login;
```

이메일에는 공백이 존재하지 않으므로 email의 값이 변경될 때마다 공백을 제거하도록 수정하고, validateEmail 함수를 이용해 공백이 제거된 이메일이 올바른 형식인지 검사했다. 마지막으로 검사 결과에 따라 오류 메시지가 나타나도록 로그인 화면을 수정했다.

비밀번호도 공백을 허용하지 않기 위해 공백을 제거하는 코드가 추가 되었다.

Login



Email

hongsi@naver.

Password

Password

Please verity your email.



Login



Email

hongsi@naver.com

Password

Password



Button 컴포넌트

로그인 버튼등으로 활용될 Button 컴포넌트 만들어보자.

theme.js

```
JS theme.js X
src > JS theme.js > ...
1
2
3   const colors = {
4     white: '#ffffff',
5     black: '#000000',
6     grey_0: '#d5d5d5',
7     grey_1: '#a6a6a6',
8     red: '#e84118',
9     blue: '#3679fe',
10  };
11
12  export const theme = {
13    background: colors.white,
14    text: colors.black,
15    errorText : colors.red,
16
17    imageBackground : colors.grey_0,
18    label : colors.grey_1,
19    inputPlaceholder : colors.grey_1,
20    inputBorder : colors.grey_1,
21
22    buttonBackground : colors.blue,
23    buttonText : colors.white,
24    buttonUnfilledTitle : colors.blue,
25  };
26
```

버튼의 색은 로고의 색과 동일한 색을 사용하도록 작성했거, 내부가 채워지지 않은 버튼은 버튼의 타이틀 색을 다르게 사용하기 위한 값을 정의했다. 이제 정의된 색을 이용해 button 컴포넌트를 만들어보자

```
JS theme.js JS Button.js X
src > components > JS Button.js > ...
1  import React from "react";
2  import styled from "styled-components";
3  import PropTypes from 'prop-types';
4
5  const TRANSPARENT = 'transparent';
6
7  const Container = styled.TouchableOpacity`
8    background-color : ${({theme, isFilled}) => isFilled ? theme.buttonBackground : TRANSPARENT};
9    align-items : center;
10   border-radius : 4px;
11   width : 100%;
12   padding : 10px;
13 `;
14 const Title = styled.Text`
15   height : 30px;
16   line-height : 30px;
17   font-size : 16px;
18   color : ${({theme, isFilled})=> isFilled ? theme.buttonTitle : theme.buttonUnfilledTitle};
19 `;
20 const Button = ({containerStyle, title, onPress, isFilled}) => {
21   return (
22     <Container style = {containerStyle} onPress={onPress} isFilled={isFilled}>
23       <Title isFilled={isFilled}>{title}</Title>
24     </Container>
25   );
26 };
27 Button.defaultProps = {
28   isFilled : true,
29 };
30 Button.propTypes = {
31   containerStyle : PropTypes.object,
32   title : PropTypes.string,
33   onPress : PropTypes.func.isRequired,
34   isFilled : PropTypes.bool,
35 };
36 export default Button;
```

props로 전달된 isFilled의 값에 따라 버튼 내부를 채우거나 투명하게 처리하는 Button 컴포넌트를 만들었다. isFilled의 기본값을 true로 지정해서 색이 채워진 상태가 기본 상태로 되도록 하고, 버튼 내부가 채워지지 않았을 경우 props로 전달된 title의 색이 변경되도록 작성했다. 사용되는 곳에 따라 버튼의 스타일을 수정하기 위해 containerStyle을 props로 전달 받아 적용하도록 작성했다. Button 컴포넌트의 작성이 완료되면 componets폴더의 index.js파일을 아래 처럼 수정하자.

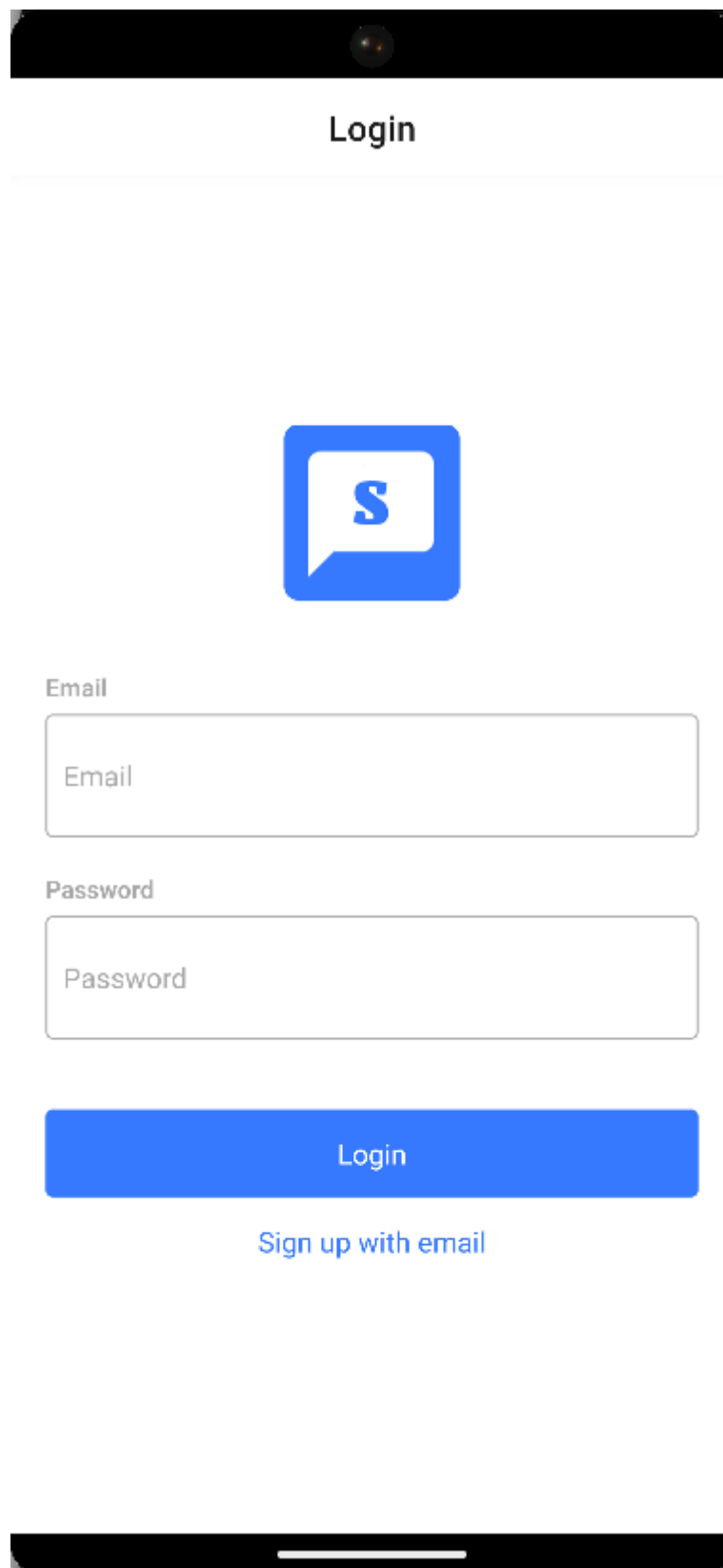
```
JS theme.js JS Button.js JS index.js X
src > components > JS index.js
1  import Image from './Image';
2  import Input from './Input';
3  import Button from './Button';
4
5  export {Image, Input, Button};|
```

이제 로그인 화면에서 Button 컴포넌트를 사용해보자.

```
JS theme.js JS Button.js JS index.js JS Login.js X
src > screens > JS Login.js > [🔍] Login
30
31   const _handleEmailChange = email => {
32     const changedEmail = removeWhitespace(email);
33     setEmail(changedEmail);
34     setErrorMessage(
35       validateEmail(changedEmail) ? '' : 'Please verify your email.'
36     );
37   };
38   const _handlePasswordChange = password => {
39     setPassword(removeWhitespace(password));
40   };
41   const _handleLoginButtonPress = () => {};
42   return (
43     <KeyboardAwareScrollView
44       contentContainerStyle={{ flex : 1}}
45       extraScrollHeight={20}
46     >
47       <Container>
48         <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
49         <Input
50 >   label="Email" ...
56   />
57         <Input
58           label="Password"
59           value={password}
60           onChangeText={_handlePasswordChange}
61           onSubmitEditing={_handleLoginButtonPress}
62           placeholder="Password"
63           returnKeyType="done"
64           ref={passwordRef}
65         />
66         <ErrorText {errorMessage} />
67         <Button title="Login" onPress={_handleLoginButtonPress} />
68         <Button
69           title="Sign up with email"
70           onPress={() => navigation.navigate('Signup')}
71           isFilled={false}
72         />
73       </Container>
74     </KeyboardAwareScrollView>
75   )
76 }
77 export default Login;
```

Button 컴포넌트를 사용해서 로그인 버튼과 회원가입 화면으로 이동하는 버튼을 만들었다. 로그인 버튼을 클릭했을 때 해야 하는 작업과 비밀번호를 입력받는 Input 컴포넌트의

onSubmitEditing 함수가 하는 역할이 같으므로 동일한 작업이 수행되도록 수정했다.



The image shows a mobile application login screen. At the top, there is a black header bar with a small circular icon in the center. Below the header, the word "Login" is centered in a bold, black font. A horizontal line separates the title from the main content area. In the center of the screen is a blue square icon with a white speech bubble and a blue letter "S" inside. Below the icon, there are two input fields. The first is labeled "Email" and contains the placeholder text "Email". The second is labeled "Password" and contains the placeholder text "Password". Below these fields is a blue button with the text "Login" in white. At the bottom of the form, there is a link that says "Sign up with email" in blue text. The entire screen is framed by a black border at the top and bottom, with a white horizontal line at the very bottom, suggesting a mobile device interface.

이번에는 이메일과 비밀번호가 입력되지 않으면 Button 컴포넌트가 동작하지 않도록 수정하자. Button 컴포넌트의 onPress에 전달하는 함수에서 버튼의 클릭 가능 여부를 확인하는 방법이 있지만, 사용자에게 버튼 동작 여부를 시각적으로 명확하게 알릴 수 있다.

```
JS theme.js JS Button.js X JS index.js JS Login.js
src > components > JS Button.js > default
1 import React from "react";
2 import styled from "styled-components";
3 import PropTypes from 'prop-types';
4
5 const TRANSPARENT = 'transparent';
6
7 const Container = styled.TouchableOpacity`
8   background-color : ${({theme, isFilled}) => isFilled ? theme.buttonBackground : TRANSPARENT};
9   align-items : center;
10  border-radius : 4px;
11  width : 100%;
12  padding : 10px;
13  opacity : ${({disabled})=> (disabled ? 0.5 : 1)};
14 `;
15
16 const Title = styled.Text`
17   height : 30px;
18   line-height : 30px;
19   font-size : 16px;
20   color : ${({theme, isFilled})=> isFilled ? theme.buttonTitle : theme.buttonUnfilledTitle};
21 `;
22
23 const Button = ({containerStyle, title, onPress, isFilled}) => {
24   return (
25     <Container style = {containerStyle} onPress={onPress} isFilled={isFilled} disabled={disabled}>
26       <Title isFilled={isFilled}>{title}</Title>
27     </Container>
28   );
29 };
30
31 Button.defaultProps = {
32   isFilled : true,
33 };
34
35 Button.propTypes = {
36   containerStyle : PropTypes.object,
37   title : PropTypes.string,
38   onPress : PropTypes.func.isRequired,
39   isFilled : PropTypes.bool,
40   disabled : PropTypes.bool,
41 };
42
43 export default Button;
```


Button 컴포넌트에서 props를 통해 전달되는 disabled의 값에 따라 버튼 스타일이 변경되도록 수정했다. Button 컴포넌트를 구성하는 TouchableOpacity 컴포넌트에 disabled속성을 전달하면 값에 따라 클릭 등의 상호 작용이 동작하지 않기 때문에 disabled 값을 props로 전달하는 것으로 버튼 비활성화 기능을 추가했다. 이제 로그인 화면에서 입력되는 값을 확인하고 버튼의 활성화 여부를 결정하도록 코드를 수정하자.

```
JS Login.js X
src > screens > JS Login.js > [0] Login
24
25 const Login = ({navigation})=>{
26   const [email, setEmail] = useState('');
27   const [password, setPassword] = useState('');
28   const passwordRef = useRef();
29   const [errorMessage, setErrorMessage] = useState('');
30   const [disabled, setDisabled] = useState(true);
31
32   useEffect(() => {
33     setDisabled(!(email && password && !errorMessage));
34   }, [email, password, errorMessage])
35
36   const _handleEmailChange = email => { ...
42   };
43   const _handlePasswordChange = password => { ...
45   };
46   const _handleLoginButtonPress = () => {};
47   return(
48     <KeyboardAwareScrollView
49       contentContainerStyle={{ flex : 1}}
50       extraScrollHeight={20}
51     >
52       <Container>
53         <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
54         <Input
55           label="Email" ...
61         />
62         <Input
63           label="Password"
64           value={password}
65           onChangeText={_handlePasswordChange}
66           onSubmitEditing={_handleLoginButtonPress}
67           placeholder="Password"
68           returnKeyType="done"
69           ref={passwordRef}
70           isPassword
71         />
72         <ErrorText>{errorMessage}</ErrorText>
73         <Button title="Login" onPress={_handleLoginButtonPress} disabled={disabled} />
74         <Button
75           title="Sign up with email"

```

useState를 사용해 버튼의 활성화 상태를 관리하는 disabled를 생성하고 useEffect를 이용해 email,password,errorMessage의 상태가 변할 때마다 조건에 맞게 disabled의 상태가 변경되도록 작성했다. 로그인 버튼은 이메일과 비밀번호가 입력되어 있고, 오류 메시지가 없는 상태에서만 활성화되어야 한다. 마지막으로 로그인 버튼의 Button 컴포넌트에 disabled를 전달해서 값에 따라 버튼의 활성화 여부가 결정되도록 작성했다.

Login



Email

krs1994@naver.com


Password

Password

Login

[Sign up with email](#)

Login



Email

hrs1994@naver.com

Password

.....

Login

[Sign up with email](#)

입력 값에 따라 버튼의 활성화 여부가 달라진다.

헤더 수정하기

현재 화면은 스택 내비게이션을 사용해서 로그인 화면과 회원가입 화면에 모두 헤더가 있다. 애플리케이션의 첫 화면인 로그인 화면에는 헤더가 굳이 필요하지 않으므로 헤더를 감추자

```
JS Login.js JS AuthStack.js X
src > navigations > JS AuthStack.js > default
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   return(
11     <Stack.Navigator
12       initialRouteName='Login'
13       screenOptions={{
14         headerTitleAlign: 'center',
15         cardStyle: {backgroundColor:theme.backgroundColor},
16       }}>
17       <Stack.Screen name="Login" component={Login} options={{headerShown: false}} />
18       <Stack.Screen name="Signup" component={Signup} />
19     </Stack.Navigator>
20   )
21 }
22
23
24 export default AuthStack;
```

로그인 화면에서는 헤더가 렌더링 되지 않도록 headerShown을 이용해 헤더를 감췄다.

회원가입 화면의 헤더에 나타나는 뒤로 가기 버튼의 타이틀을 감추고, 버튼과 타이틀의 색이 일치되도록 수정했다.

헤더에서 사용할 색을 theme.js 파일에 정의하자.

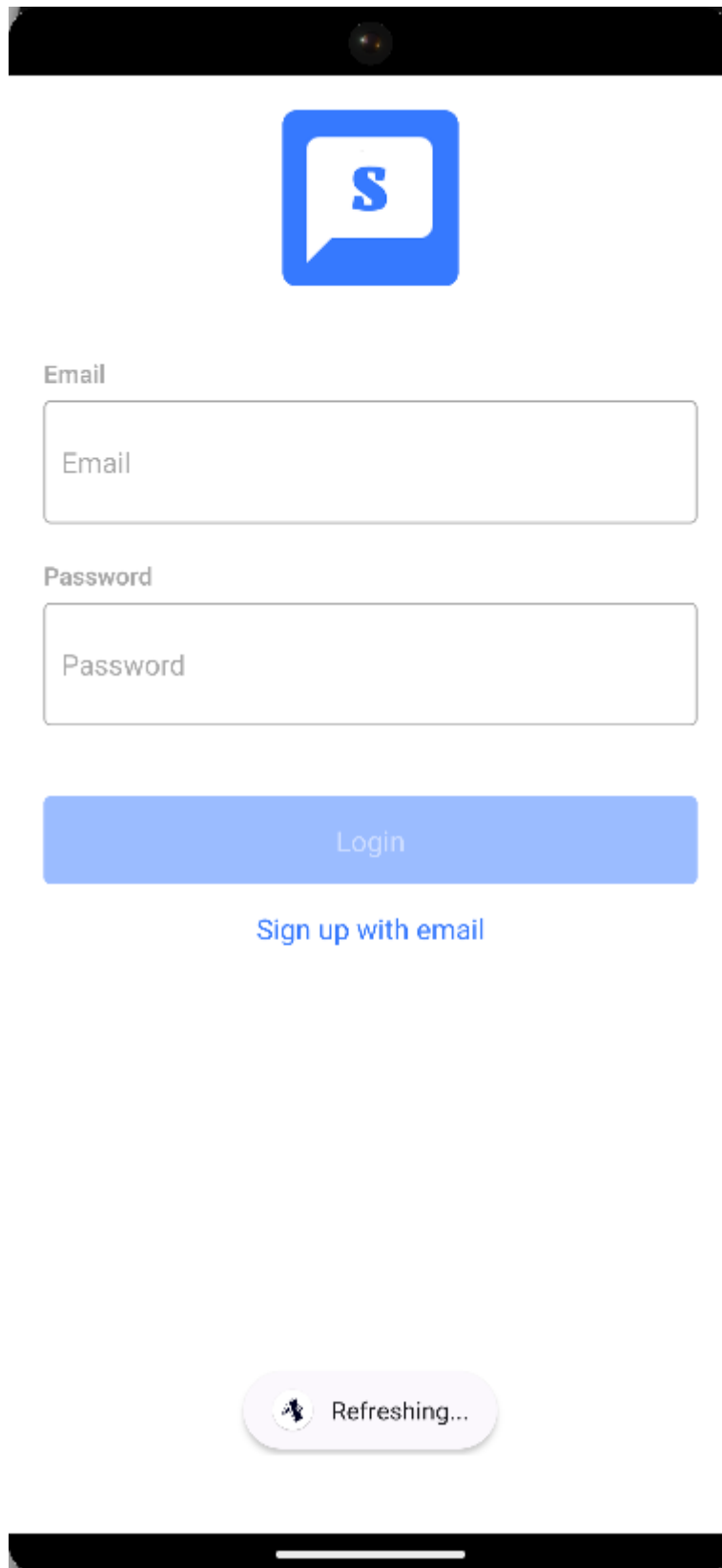
```
JS Login.js JS AuthStack.js JS theme.js X
src > JS theme.js > [⌘] theme
1
2
3   const colors = {
4     white: 'ffffff',
5     black: '000000',
6     grey_0: 'd5d5d5',
7     grey_1: 'a6a6a6',
8     red: 'e84118',
9     blue: '3679fe',
10  };
11
12  export const theme = {
13    background: colors.white,
14    text: colors.black,
15    errorText : colors.red,
16
17    imageBackground : colors.grey_0,
18    label : colors.grey_1,
19    inputPlaceholder : colors.grey_1,
20    inputBorder : colors.grey_1,
21
22    buttonBackground : colors.blue,
23    buttonTitle : colors.white,
24    buttonUnfilledTitle : colors.blue,
25
26    headerTintColor : colors.black,
27  };
28
```

헤더에서 사용할 색을 정의한 후 AuthStack 내비게이션을 아래 처럼 수정하자.

```
JS Login.js JS AuthStack.js X JS theme.js
src > navigations > JS AuthStack.js > 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  return(
11    <Stack.Navigator
12      initialRouteName='Login'
13      screenOptions={{
14        headerTitleAlign: 'center',
15        cardStyle: {backgroundColor:theme.backgroundColor},
16        headerTintColor : theme.headerTintColor,
17      }}>
18
19      <Stack.Screen name="Login" component={Login} options={{headerShown: false}} />
20      <Stack.Screen name="Signup" component={Signup} options={{headerBackTitleVisible : false}} />
21    </Stack.Navigator>
22  )
23 }
24
25 export default AuthStack;
```

노치 디자인 대응

내비게이션의 헤더를 감추면 노치 디자인에 대한 문제가 발생할 수 있다. 로그인 화면에서 스타일드 컴포넌트를 이용해 정의된 Container 컴포넌트의 스타일에서 "justify-content: center"를 삭제해보면 로고가 노치 디자인에 의해 가려지는 것을 확인할 수 있다.



우리가 알고 있는 `SafeAreaView` 컴포넌트를 이용하는 방법 외에도 노치 디자인에 대응하기 위해 스타일에 설정해야 하는 `padding` 값을 얻는 방법이 있다. 리액트 네비게이션 라이브러리를 설치하는 과정에서 추가로 함께 설치한 `react-native-safe-area-context` 라이브러리가

제공하는 `useSafeAreaInsets` Hook 함수를 이용하면 된다. `useSafeAreaInsets` 함수를 이용해 로그인 화면을 아래 코드처럼 수정해보자.

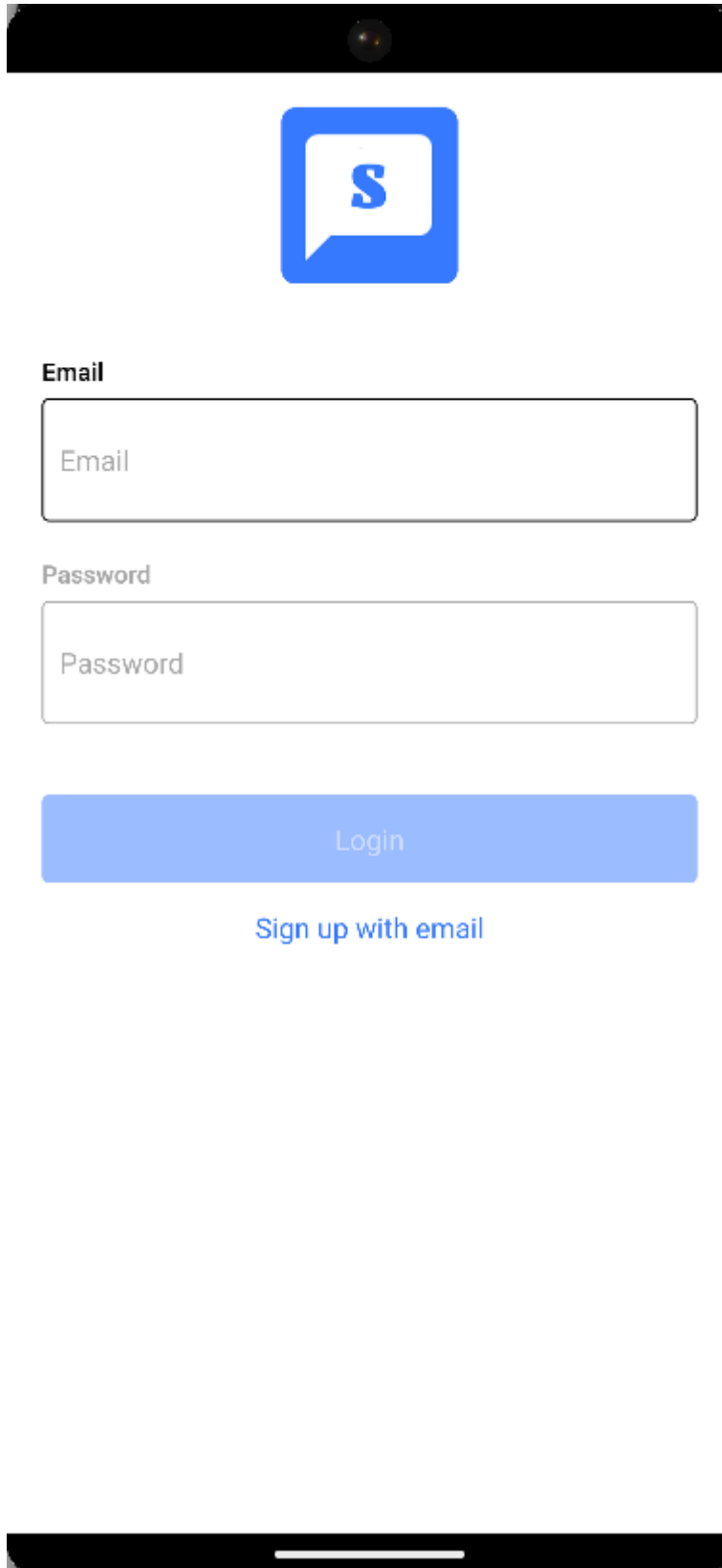
```
JS Login.js x JS index.js JS Signup.js JS AuthStack.js JS theme.js
src > screens > JS Login.js > [ErrorText]
2 import styled from 'styled-components';
3 import {Image, Input, Button} from '../components';
4 import {images} from '../utils/images';
5 import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
6 import { validateEmail, removeWhitespace } from '../utils/common';
7 import { useSafeAreaInsets } from 'react-native-safe-area-context';
8
9 const Container = styled.View`
10   flex : 1;
11   align-items: center;
12   background-color: ${({theme})=>theme.background};
13   padding : 0 20px;
14   padding-top : ${({ insets : {top} }) => top+20}px;
15   padding-bottom : ${({ insets : {bottom} }) => bottom}px;
16 `;
```

```
const insets = useSafeAreaInsets();
useEffect(() => {
  setDisabled(![email && password && !errorMessage]);
}, [email, password, errorMessage])
```

```

return(
  <KeyboardAwareScrollView
    contentContainerStyle={{ flex : 1}}
    extraScrollHeight={20}
  >
    <Container insets={insets}>
      <Image url={images.logo} imageStyle={{ borderRadius : 8 }}/>
      <Input
        label="Email" ...
      />
      <Input
        label="Password"
        value={password}
        onChangeText={_handlePasswordChange}
        onSubmitEditing={_handleLoginButtonPress}
        placeholder="Password"
        returnKeyType="done"
        ref={passwordRef}
        isPassword
      />
      <ErrorText>{errorMessage}</ErrorText>
      <Button title="Login" onPress={_handleLoginButtonPress} disabled={disabled} />
      <Button
        title="Sign up with email"
        onPress={()=> navigation.navigate('Signup')}
        isFilled={false}
      />
    </Container>
  </KeyboardAwareScrollView>
)

```



A mobile app login screen mockup. At the top is a black header bar with a small circular camera icon in the center. Below the header is a blue square icon with a white speech bubble and a blue letter 'S'. Underneath the icon are two text input fields. The first field is labeled 'Email' and the second is labeled 'Password'. Below these fields is a blue rectangular button with the text 'Login'. Under the button is a blue link that says 'Sign up with email'. At the bottom is a black footer bar with a white horizontal line in the center.

Email

Password

Login

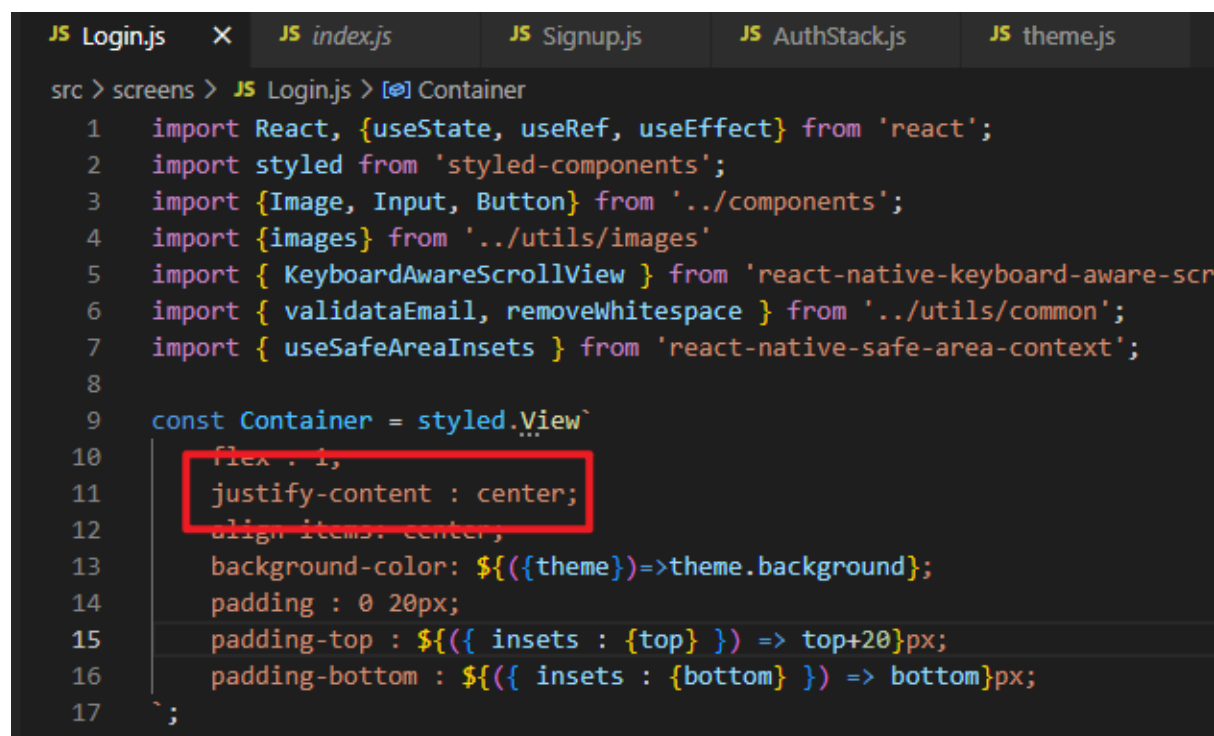
[Sign up with email](#)

나같은 경우 설정을 해도 노치에 딱 붙어있게 렌더링이 되어 top에 +20 px를 더 넣어주었다.

노치 디자인을 해결하기 위해 padding의 top과 bottom의 값을 useSafeAreaInsets 함수가 알려주는 값만큼 설정하고, 양 옆은 디자인에 맞게 20px로 설정했다. useSafeAreaInsets 함수의 장점은 IOS뿐 아니라 안드로이드에서도 적용 가능한 padding값을 전달한다.

useSafeAreaInsets를 사용하면 조금 더 세밀하게, 원하는 곳에 원하는 만큼만 padding을 설정해서 노치 디자인 문제를 해결할 수 있다는 장점이 있다.

삭제했던 justify-content : center를 다시 작성하자.



```
JS Login.js X JS index.js JS Signup.js JS AuthStack.js JS theme.js
src > screens > JS Login.js > Container
1 import React, {useState, useRef, useEffect} from 'react';
2 import styled from 'styled-components';
3 import {Image, Input, Button} from '../components';
4 import {images} from '../utils/images';
5 import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
6 import { validateEmail, removeWhitespace } from '../utils/common';
7 import { useSafeAreaInsets } from 'react-native-safe-area-context';
8
9 const Container = styled.View`
10   flex : 1,
11   justify-content : center;
12   align-items : center;
13   background-color: ${({theme})=>theme.background};
14   padding : 0 20px;
15   padding-top : ${({ insets : {top} }) => top+20}px;
16   padding-bottom : ${({ insets : {bottom} }) => bottom}px;
17 `;
```

회원가입 화면

회원가입 화면은 로그인 화면 제작 과정에서 만든 컴포넌트를 재사용하면 굉장히 쉽고 빠르게 만들 수 있다.

먼저 회원가입 화면에서 사용자의 사진을 원형으로 렌더링하기 위해 Image 컴포넌트에서 props를 통해 전달되는 값에 따라 이미지가 원형으로 렌더링 되도록 수정한다.

```
JS Login.js JS Image.js X JS Signup.js JS AuthStack.js JS theme.js
src > components > JS Image.js > ...
1  import React from "react";
2  import styled from "styled-components";
3  import PropTypes from 'prop-types';
4
5  const Container = styled.View`
6    align-self = center;
7    margin-bottom : 30px;
8  `;
9
10 const StyledImage = styled.Image`
11   background-color : ${({theme}) => theme.imageBackground};
12   width : 100px;
13   height : 100px;
14   border-radius : ${({rounded}) => (rounded ? 50 : 0)}px;
15 `;
16
17 const Image = ({url, imageStyle, rounded}) => {
18   return (
19     <Container>
20       <StyledImage source={{ uri : url}} style={imageStyle} rounded={rounded}/>
21     </Container>
22   );
23 };
24
25 Image.defaultProps={
26   rounded : false,
27 };
28
29 Image.propTypes = {
30   uri : PropTypes.string,
31   imageStyle : PropTypes.object,
32   rounded : PropTypes.bool,
33 };
34
35 export default Image;
```

props를 통해 rounded값을 전달 받아 이미지를 원형으로 렌더링할 수 있도록 수정했다.
회원가입 화면을 만들어보자.

```
JS Login.js JS Image.js JS Signup.js X JS common.js JS AuthStack.js JS theme.js
src > screens > JS Signup.js > [E] ErrorText
1  import React, {useState, useRef, useEffect} from 'react';
2  import styled from 'styled-components';
3  import {Image, Input, Button} from '../components'
4  import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
5  import { validateEmail, removeWhitespace } from '../utils/common';
6
7  const Container = styled.View`
8    flex: 1;
9    justify-content: center;
10   align-items: center;
11   background-color: ${({theme})=>theme.background};
12   padding : 0 20px;
13 `;
14
15  const ErrorText = styled.Text`
16    align-items : flex-start;
17    width : 100%;
18    height : 20px;
19    margin-bottom : 10px;
20    line-height : 20px;
21    color : ${({theme})=> theme.ErrorText};
22 `;
23
24  const Signup = ()=>{
25    const [name, setName] = useState('');
26    const [email, setEmail] = useState('');
27    const [password, setPassword] = useState('');
28    const [passwordConfirm, setPasswordConfirm] = useState('');
29    const [errorMessage, setErrorMessage] = useState('');
30    const [disabled, setDisabled] = useState(true);
31
32    const emailRef = useRef();
33    const passwordRef = useRef();
34    const passwordConfirmRef = useRef();
```

```

useEffect(() => {
  let _errorMessage = '';
  if (!name) {
    _errorMessage = 'Please enter your name.';
  } else if (!validateEmail(email)) {
    _errorMessage = 'Please verify your email.';
  } else if (password.length < 6) {
    _errorMessage = 'The password must contain 6 characters at least';
  } else if (password !== passwordConfirm) {
    _errorMessage = 'Passwords need to match.';
  } else {
    _errorMessage = '';
  }
  setErrorMessage(_errorMessage);
}, [name, email, password, passwordConfirm]);

useEffect(() => {
  setDisabled(
    !(name && email && password && passwordConfirm && !errorMessage)
  );
}, [name, email, password, passwordConfirm, errorMessage])

const _handleSignupButtonPress = () => {};

```



```

return(
  <KeyboardAwareScrollView
    contentContainerStyle={{flex : 1}}
    extraScrollHeight={20}
  >
    <Container>
      <Image rounded />
      <input
        label="Name"
        value={name}
        onChangeText={text => setName(text)}
        onSubmitEditing={() => {
          setName(name.trim());
          emailRef.current.focus();
        }}
        onBlur={() => setName(name.trim())}
        placeholder="Name"
        returnKeyType="text"
      />
      <Input
        ref={emailRef}
        label="Email"
        value={email}
        onChangeText={text => setEmail(removeWhitespace(text))}
        onSubmitEditing={() => passwordRef.current.focus()}
        placeholder="Email"
        returnKeyType="next"
      />
      <Input
        ref={passwordRef}
        label="Password"
        value={password}
        onChangeText={text => setPassword(removeWhitespace(text))}
        onSubmitEditing={() => passwordConfirmRef.current.focus()}
        placeholder="Password"
        returnKeyType="next"
        isPassword
      />
    </Container>
  </KeyboardAwareScrollView>
)

```

```


      <Input
        ref={passwordConfirmRef}
        label="Password Confirm"
        value={passwordConfirm}
        onChangeText={text => setPasswordConfirm(removeWhitespace(text))}
        onSubmitEditing={_handleSignupButtonPress}
        placeholder="Password"
        returnKeyType="done"
        isPassword
      />
      <ErrorText>{errorMessage}</ErrorText>
      <Button
        title="Signup"
        onPress={_handleSignupButtonPress}
        disabled={disabled}
      />
    </Container>
  </KeyboardAwareScrollView>
);
};

export default Signup;

```

회원가입 화면은 사용자에게 입력받아야 하는 내용이 많아진 것을 제외하면 로그인 화면과 거의 같은 모습이다. 입력받아야 하는 값이 많아진 만큼 유효성 검사와 오류 메시지의 종류가 많아지므로 useEffect를 이용해 관련된 값이 변할 때마다 적절한 오류 메시지가 렌더링 되도록 작성했다.

← Signup



Name

Name

Email

Email

Password

Password

Password Confirm

Password

Please enter your name.

Signup

화면이 잘 구성된 것처럼 보이지만 기기의 크기에 따라 화면의 위아래가 잘려서 보이는 문제가 있다. 또한, 아직 어떤 값도 입력하지 않았는데 오류 메시지가 렌더링되는 문제도 있다.

화면 스크롤

화면 크기에 따라 내용이 화면을 넘어가는 문제를 해결해보자.

KeyboardAwareScrollView 컴포넌트에 contentContainerStyle을 이용해서 “flex : 1” 스타일에 적용시키면서 발행한 문제이다. flex:1 스타일을 설정하면 컴포넌트가 차지하는 영역이 부모 컴포넌트 영역만큼으로 한정되고, 컴포넌트의 크기에 따라 화면을 넘어가서 스크롤이 생성되도록 flex:1을 삭제한다.

```
JS Signup.js X
src > screens > JS Signup.js > [E] ErrorText
1  import React, {useState, useRef, useEffect} from 'react';
2  import styled from 'styled-components';
3  import {Image, Input, Button} from '../components'
4  import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
5  import { validateEmail, removeWhitespace } from '../utils/common';
6
7  const Container = styled.View`
8    flex : 1;
9    justify-content: center;
10   align-items: center;
11   background-color: ${({theme})=>theme.background};
12   padding : 40px 20px;
13 `;
14
15 const ErrorText = styled.Text`
16   align-items : flex-start;
```

```

return(
  <KeyboardAwareScrollView extraScrollHeight={20} >
    <Container>
      <Image rounded />
      <Input
        label="Name"
        value={name}
        onChangeText={text => setName(text)}
        onSubmitEditing={() => {
          setName(name.trim());
          emailRef.current.focus();
        }}
        onBlur={() => setName(name.trim())}
        placeholder="Name"
        returnKeyType="text"
      />
      <Input
        ref={emailRef}
        label="Email"

```

문제가 되었던 `contentContainerStyle`을 삭제하고 화면의 위아래에 약간의 여유 공간을 두기 위해 `Container` 컴포넌트의 `padding`값을 수정했다.

오류 메시지

회원가입 화면에서 입력되는 값에 따라 오류 메시지의 변화가 많아 `useEffect`를 이용해 오류 메시지를 한곳에서 관리하도록 작성했다. 하지만 회원가입 화면이 처음 렌더링될 때도 오류 메시지가 나타나는 문제가 있다. 이것은 `useEffect`의 특성 때문에 컴포넌트가 마운트 될 때도 `useEffect`에 정의된 함수가 실행되면서 나타나는 문제이다. 이것은 `useEffect`를 응용하여 해결할 수 있다.

```
JS Signup.js x
src > screens > JS Signup.js > [0] Signup > useEffect() callback
22  `;
23
24  const Signup = () => {
25    const [name, setName] = useState('');
26    const [email, setEmail] = useState('');
27    const [password, setPassword] = useState('');
28    const [passwordConfirm, setPasswordConfirm] = useState('');
29    const [errorMessage, setErrorMessage] = useState('');
30    const [disabled, setDisabled] = useState(true);
31
32    const emailRef = useRef();
33    const passwordRef = useRef();
34    const passwordConfirmRef = useRef();
35    const didMountRef = useRef();
36
37    useEffect(() => {
38      if(didMountRef.current) {
39        let _errorMessage = '';
40        if (!name) {
41          _errorMessage = 'Please enter your name.';
42        } else if (!validateEmail(email)) {
43          _errorMessage = 'Please verify your email.';
44        } else if (password.length < 6) {
45          _errorMessage = 'The password must contain 6 characters at least';
46        } else if (password !== passwordConfirm) {
47          _errorMessage = 'Passwords need to match.';
48        } else {
49          _errorMessage = '';
50        }
51        setErrorMessage(_errorMessage);
52      } else {
53        didMountRef.current = true;
54      }
55      return [name, email, password, passwordConfirm];
56    }, [name, email, password, passwordConfirm]);
57  };
58}
```

useRef 함수를 이용해서 생성된 didMountRef에 어떤 값도 대입하지 않다가, 컴포넌트가 마운트되었을 때 실행되는 useEffect 함수에서 didMountRef에 값을 대입하는 방법으로 컴포넌트의 마운트 여부를 확인하도록 수정했다.

※ 클래스형 컴포넌트를 사용하면 해결할 수 있는 문제이다. 하지만 공부한 책에서는 클래스형 컴포넌트를 다루지 않는다. 또한 함수형 컴포넌트 사용이 권장되므로, 되도록 함수형 컴포넌트에서 해결하는 방법을 찾는 것이 좋다.

사진 입력받기

회원가입 화면에서 사용자에게 사진을 입력받는 기능을 만들어보자. 먼저 사진이 선택되지 않았을 때 보여줄 기본 이미지를 스토리지에 업로드하고 규칙을 수정해서 로그인하지 않아도 접근이 가능하도록 하자. 파일명을 photo.png로 업로드하고 규칙을 아래처럼 수정하자.

```

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

```

이미지 업로드와 규칙 수정이 완료되면 로고 이미지와 마찬가지로 utils 폴더에 있는 images.js파일에 기본이미지의 url을 추가한다.

```

JS Signup.js  JS images.js  X
src > utils > JS images.js > ...
1  const prefix =
2    'https://firebasestorage.googleapis.com/v0/b/react-native-simp
3
4  export const images = {
5    logo : `${prefix}/logo.png?alt=media`,
6    photo : `${prefix}/photo.png?alt=media`,
7  };

```

```

import { validateEmail, removeWhitespace } from "../utils/common";
import { images } from '../utils/images'

> const Container = styled.View` ...
`;

> const ErrorText = styled.Text` ...
`;

const Signup = () => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [passwordConfirm, setPasswordConfirm] = useState('');
  const [errorMessage, setErrorMessage] = useState('');
  const [disabled, setDisabled] = useState(true);
  const [photoUrl, setPhotoUrl] = useState(images.photo);

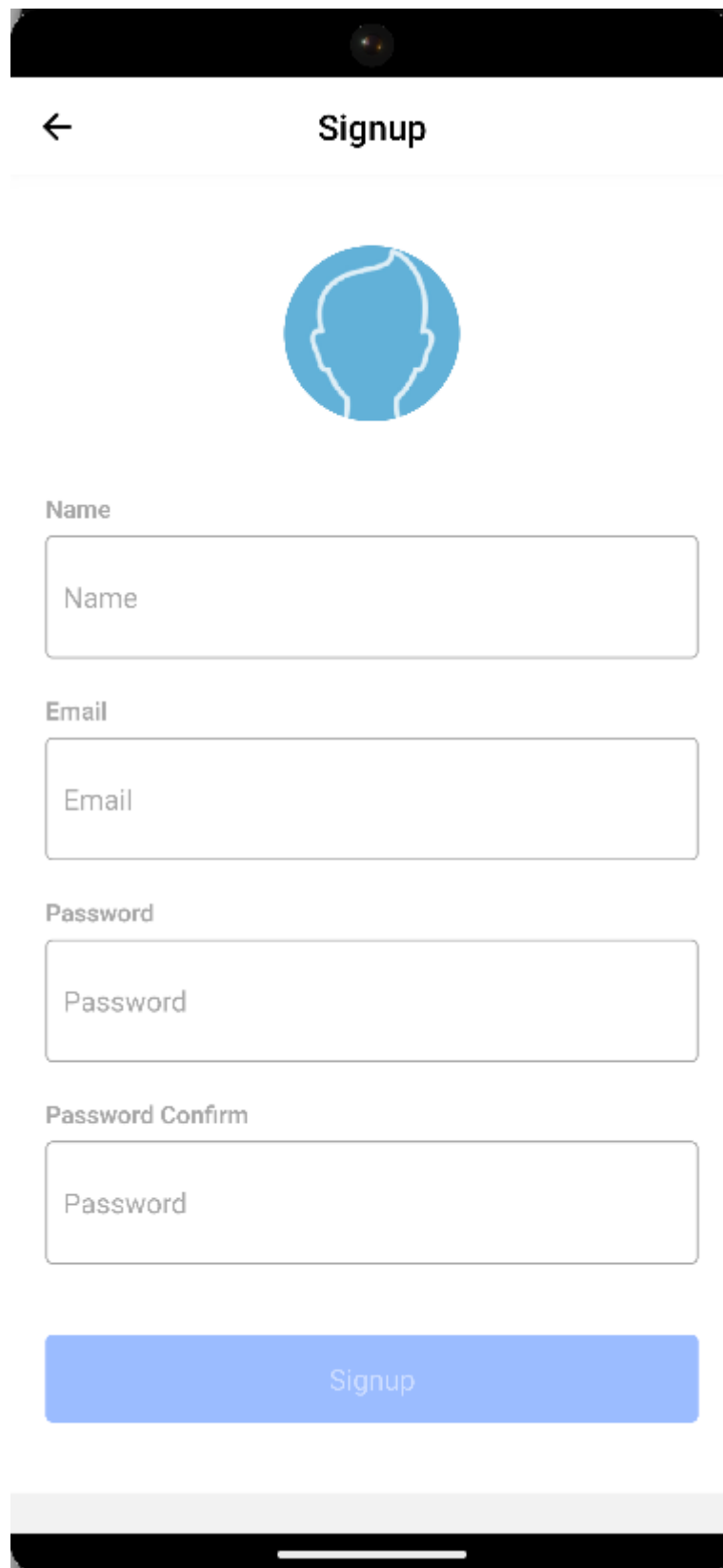
```

```


return(
  <KeyboardAwareScrollView extraScrollHeight={20}>
    <Container>
      <Image rounded url={photoUrl}/>
      <input
        label="name"
        value={name}
        onChangeText={text => setName(text)}
        onSubmitEditing={() => {
          setName(name.trim());
          emailRef.current.focus();
        }}

```

로고 이미지와 마찬가지로 쿼리 스트링의 token 부분은 제외하고 기본 이미지의 주소를 추가했다. 회원가입 화면에서 추가된 기본 이미지를 이용하도록 수정하자.



← Signup



Name

Name

Email

Email

Password

Password

Password Confirm

Password

Signup

Image 컴포넌트에 버튼을 추가하여 기기의 사진을 이용하는 기능을 만들자. 추가되는 버튼에서 사용할 색을 theme.js 파일에 정의하자.

```
JS Signup.js JS theme.js X JS images.js
src > JS theme.js > [⌘] theme
1
2
3   const colors = {
4     white: '#ffffff',
5     black: '#000000',
6     grey_0: '#d5d5d5',
7     grey_1: '#a6a6a6',
8     red: '#e84118',
9     blue: '#3679fe',
10  };
11
12  export const theme = {
13    background: colors.white,
14    text: colors.black,
15    errorText : colors.red,
16
17    imageBackground : colors.grey_0,
18    imageButtonBackground : colors.grey_1,
19    imageButtonIcon : colors.white,
20    label : colors.grey_1,
21    inputPlaceholder : colors.grey_1,
22    inputBorder : colors.grey_1,
23
24    buttonBackground : colors.blue,
25    buttonText : colors.white,
26    buttonUnfilledTitle : colors.blue,
27
28    headerTintColor : colors.black,
29  };
30
```

색의 정의가 완료되면 Image 컴포넌트를 다음과 같이 수정하자.

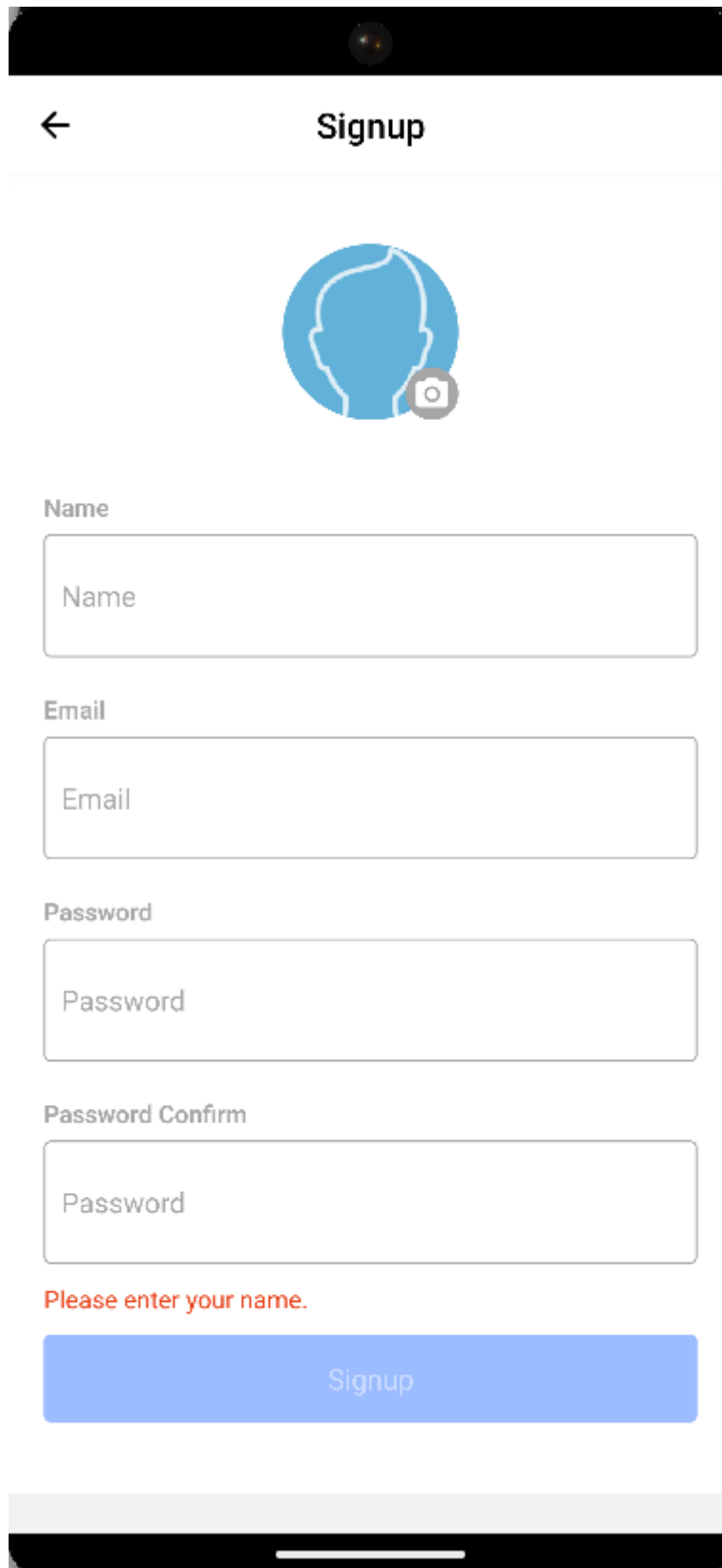
```
JS Signup.js JS theme.js JS images.js JS Image.js X
src > components > JS Image.js > ...
import PropTypes from 'prop-types';
4 import {MaterialIcons} from '@expo/vector-icons'
5
6 const Container = styled.View`
7   align-self = center;
8   margin-bottom : 30px;
9 `;
10
11 const StyledImage = styled.Image`
12   background-color : ${({theme}) => theme.imageBackground};
13   width : 100px;
14   height : 100px;
15   border-radius : ${({rounded}) => (rounded ? 50 : 0)}px;
16 `;
17
18 const ButtonContainer = styled.TouchableOpacity`
19   background-color : ${({theme}) => theme.imageButtonBackground};
20   position : absolute;
21   bottom : 0;
22   right : 0;
23   width : 30px;
24   height : 30px;
25   border-radius : 15px;
26   justify-content : center;
27   align-items : center;
28 `;
29
30 const ButtonIcon = styled(MaterialIcons).attrs({
31   name : 'photo-camera',
32   size : 22,
33 })`
34   color : ${({theme}) => theme.imageButtonIcon};
35 `;
36
37 const PhotoButton = ({onPress}) => {
38   return(
39     <ButtonContainer onPress={onPress}>
40       <ButtonIcon />
41     </ButtonContainer>
42   );
43 };
44
```

```

45 const Image = ({url, imageStyle, rounded, showButton}) => {
46   return (
47     <Container>
48       <StyledImage source={{ uri : url}} style={imageStyle} rounded={rounded}/>
49       {showButton && <PhotoButton/>}
50     </Container>
51   );
52 };
53
54 Image.defaultProps={
55   rounded : false,
56   showButton : false,
57 };
58
59 Image.propTypes = {
60   uri : PropTypes.string,
61   imageStyle : PropTypes.object,
62   rounded : PropTypes.bool,
63   showButton : PropTypes.bool,
64 };
65
66 export default Image;


```

Image 컴포넌트에서 사진 변경하기 버튼으로 사용할 PhotoButton 컴포넌트를 만들었다. 추가된 버튼은 Image 컴포넌트의 props로 전달되는 showButton의 값에 따라 렌더링 여부가 결정되도록 작성했다. 이제 회원가입 화면에서 Image 컴포넌트에 버튼이 렌더링 되도록 수정하자.



A mobile app interface for a signup screen. At the top is a black header bar with a camera icon in the center. Below the header is a white navigation bar with a back arrow on the left and the text "Signup" in the center. The main content area is white and contains a blue circular profile picture placeholder with a camera icon in the bottom right corner. Below the profile picture are four form fields, each with a label above it: "Name", "Email", "Password", and "Password Confirm". Each field contains a placeholder text with the same label. Below the "Password Confirm" field is a red error message: "Please enter your name.". At the bottom of the form is a blue button with the text "Signup". The screen is framed by a black border at the top and bottom, with a white home indicator bar at the very bottom.

Signup



Name

Name

Email

Email

Password

Password

Password Confirm

Password

Please enter your name.

Signup

이제 버튼을 클릭하면 기기의 사진첩에 접근해서 선택된 사진의 정보를 가져오는 기능을 추가해보자. 아래 명령어를 통해 expo-image-picker 라이브러리를 설치하고 사진첩 접근 기능을 구현하자.

- expo install expo-image-picker

설치가 완료되면 Image 컴포넌트에서 설치된 라이브러리를 사용해 기기의 사진첩에 접근하는 기능을 추가하자.

```
JS Signup.js JS theme.js JS images.js JS Image.js X
src > components > JS Image.js > ...
1  import React, {useEffect} from "react";
2  import {Platform, Alert} from 'react-native';
3  import * as ImagePicker from 'expo-image-picker';
4  import * as Permissions from 'expo-permissions';
5  import styled from "styled-components";
6  import PropTypes from 'prop-types';
7  import {MaterialIcons} from '@expo/vector-icons'
8
9  const Container = styled.View`
10    align-self = center;
11    margin-bottom : 30px;
12 `;
13
14  const StyledImage = styled.Image`
15    background-color : ${({theme}) => theme.imageBackground};
16    width : 100px;
17    height : 100px;
18    border-radius : ${({rounded}) => (rounded ? 50 : 0)}px;
19 `;
20
21  const ButtonContainer = styled.TouchableOpacity`
22    background-color : ${({theme}) => theme.imageButtonBackground};
23    position : absolute;
24    bottom : 0;
25    right : 0;
26    width : 30px;
27    height : 30px;
28    border-radius : 15px;
29    justify-content : center;
30    align-items : center;
31 `;
32
33  const ButtonIcon = styled(MaterialIcons).attrs({
34    name : 'photo-camera',
35    size : 22,
36  })`
37    color : ${({theme}) => theme.imageButtonIcon};
38 `;
39
```

```

40  ✓ const PhotoButton = ({onPress}) => {
41  ✓    return(
42  ✓      <ButtonContainer onPress={onPress}>
43  ✓        <ButtonIcon />
44  ✓      </ButtonContainer>
45  ✓    );
46  ✓  };
47
48  ✓ const Image = ({url, imageStyle, rounded, showButton, onChangeImage}) => {
49  ✓    useEffect(()=> {
50  ✓      (async ()=> {
51  ✓        try{
52  ✓          if (Platform.OS === 'ios'){
53  ✓            const { status } = await Permissions.askAsync(
54  ✓              Permissions.CAMERA_ROLL
55  ✓            );
56  ✓            if ( status !== 'granted'){
57  ✓              Alert.alert(
58  ✓                'Photo Permission',
59  ✓                'Please turn on the camera roll permissions.'
60  ✓              );
61  ✓            }
62  ✓          }
63  ✓        }catch (e) {
64  ✓          Alert.alert('Photo Permission Error', e.message);
65  ✓        }
66  ✓      })();
67  ✓    },[]);

```

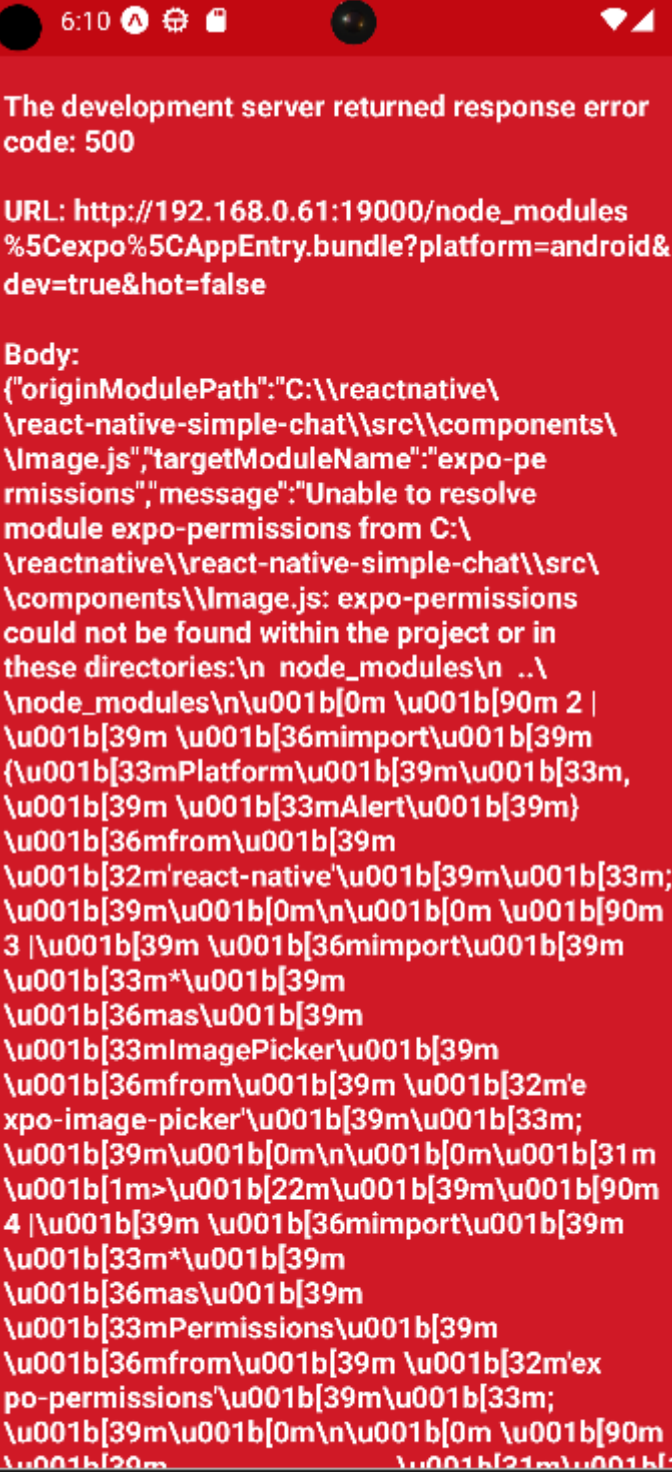
```

68     const _handleEditButton = async () =>{
69         try{
70             const result = await ImagePicker.launchImageLibraryAsync({
71                 mediaTypes : ImagePicker.MediaTypeOptions.Image,
72                 allowsEditing : true,
73                 aspect : [1,1],
74                 quality : 1,
75             });
76             if (!result.cancelled) {
77                 onChangeImage(result.uri);
78             }
79         }catch (e){
80             Alert.alert('Photo Error', e.message);
81         }
82     };
83     return (
84         <Container>
85             <StyledImage source={{ uri : url}} style={imageStyle} rounded={rounded}/>
86             {showButton && <PhotoButton onPress={_handleEditButton}/>}
87         </Container>
88     );
89 };
90
91 Image.defaultProps={
92     rounded : false,
93     showButton : false,
94     onChangeImage : ()=>{},
95 };
96
97 Image.propTypes = {
98     uri : PropTypes.string,
99     imageStyle : PropTypes.object,
100     rounded : PropTypes.bool,
101     showButton : PropTypes.bool,
102     onChangeImage : PropTypes.func,
103 };
104
105 export default Image;

```

IOS에서는 사진첩에 접근하기 위해 사용자에게 권한을 요청하는 과정이 필요하므로 권한을 요청하는 부분이 추가되었다. 안드로이드는 특별한 설정 없이 사진에 접근할 수 있기 때문에 ios에서만 동작하도록 작성했다.

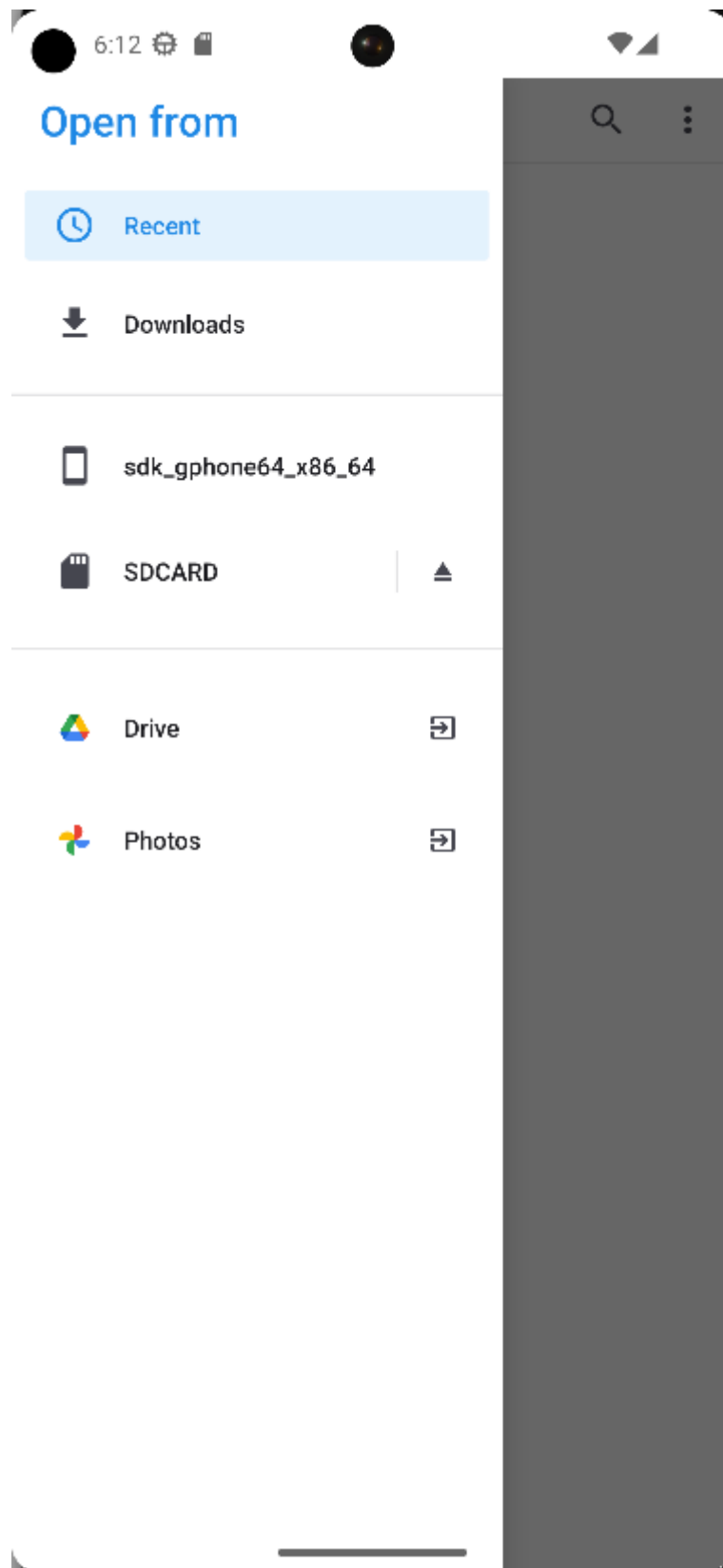
※ Expo 프로젝트에서 안드로이드는 기본적으로 모든 권한을 포함하고 있어 특별한 설정이 필요 없다.



에러 발생 permissions 모듈을 설치를 안해서 그런것 같다.

- expo install expo-permissions

위 명령어로 설치 후 실행



화면을 확인하면 권한을 요청하는 창이 메시지가 서비스에 적절하지 않은 것을 확인할 수 있다. 권한을 요청할 때 나타나는 창이 메시지는 app.json 파일을 수정해서 변경할 수 있지만, 배포를 위한 빌드 과정을 거쳐야 적용되므로 개발 단계에서 확인이 어렵다.

사진 변경 버튼을 클릭하면 호출되는 함수에서 기기의 사진에 접근하기 위해 호출되는 라이브러리 함수는 다음과 같은 값들을 포함한 객체를 파라미터로 전달받는다.

- mediaTypes : 조회하는 자료의 타입
- allowsEditing : 이미지 선택 후 편집 단계 진행 여부
- aspect : 안드로이드 전용 옵션으로 이미지 편집 시 사각형의 비율([x, y])
- quality : 0~1 사이의 값을 받으며 압축 품질을 의미 (1: 최대 품질)

조회하는 자료는 사진으로 설정하고, 크기가 다양한 사진을 편집하기 위해 편집 단계를 진행하도록 설정하자. IOS는 정사각형이므로 안드로이드에서도 1:1 비율로 편집되도록 설정했고 품질은 최대로 설정했다.

기기의 사진에 접근하는 함수는 결과를 반환하는데, 반환된 결과의 cancelled값을 통해 선택여부를 확인할 수 있다. 만약 사용자가 사진을 선택했다면 반환된 결과의 uri를 통해 선택된 사진의 주소를 알 수 있다.

그림 p.365

결과에 따라 선택된 사진의 uri를 props로 전달된 onChangeImage 함수에 파라미터로 전달하며 호출하도록 작성했다.

회원가입 화면을 수정해서 선택된 사진이 Image 컴포넌트에 렌더링 되도록 수정하자.

```
JS Signup.js x
src > screens > JS Signup.js > [0] Signup

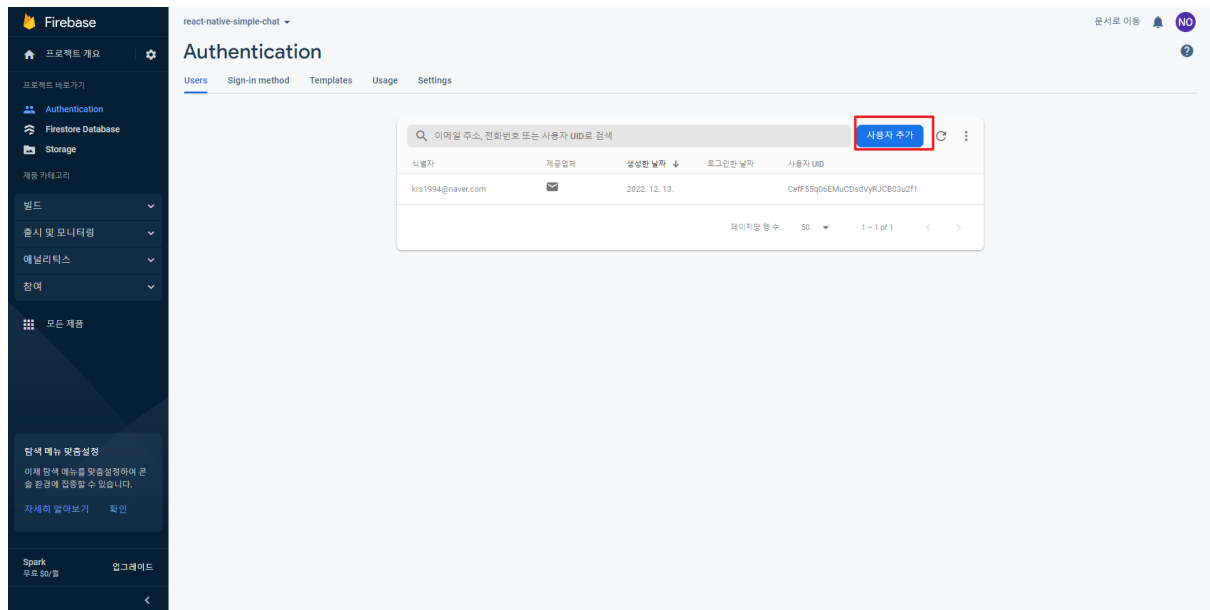
57     }
58     }, [name,email,password,passwordConfirm]);
59
60     useEffect(()=>{
61         setDisabled(
62             !(name && email && password && passwordConfirm && !errorMessage)
63         );
64     },[name,email,password,passwordConfirm,errorMessage])
65
66     const _handleSignupButtonPress = ()=>{};
67
68     return(
69         <KeyboardAwareScrollView extraScrollHeight={20} >
70             <Container>
71                 <Image
72                     rounded url={photoUrl}
73                     showButton
74                     onChangeImage={url => setPhotoUrl(url)} />
75                 <Input
76                     label="Name"
77                     value={name}
78                     onChangeText={text => setName(text)}
79                     onSubmitEditing={() => {
80                         setName(name.trim());
81                         emailRef.current.focus();
82                     }}
83                     onBlur={() => setName(name.trim())}
```

로그인과 회원가입

파이어베이스의 인증을 이용해 로그인 기능과 회원가입 기능을 만들어보자

로그인

아직 생성된 사용자가 없으므로 파이어베이스 콘솔에서 사용자를 추가하고 로그인 기능을 만들자.



이메일과 비밀번호를 이용해서 인증받는 함수는 `signInWithEmailAndPassword` 이다. 함수의 이름은 같지만, 그만큼 역할을 명확하게 알 수 있다는 것이 장점이다. 이제 사용자 추가가 완료되면 `utils` 폴더 밑에 있는 `firebase.js` 파일에 아래 코드와 같이 로그인 하는 함수를 만들자.

```

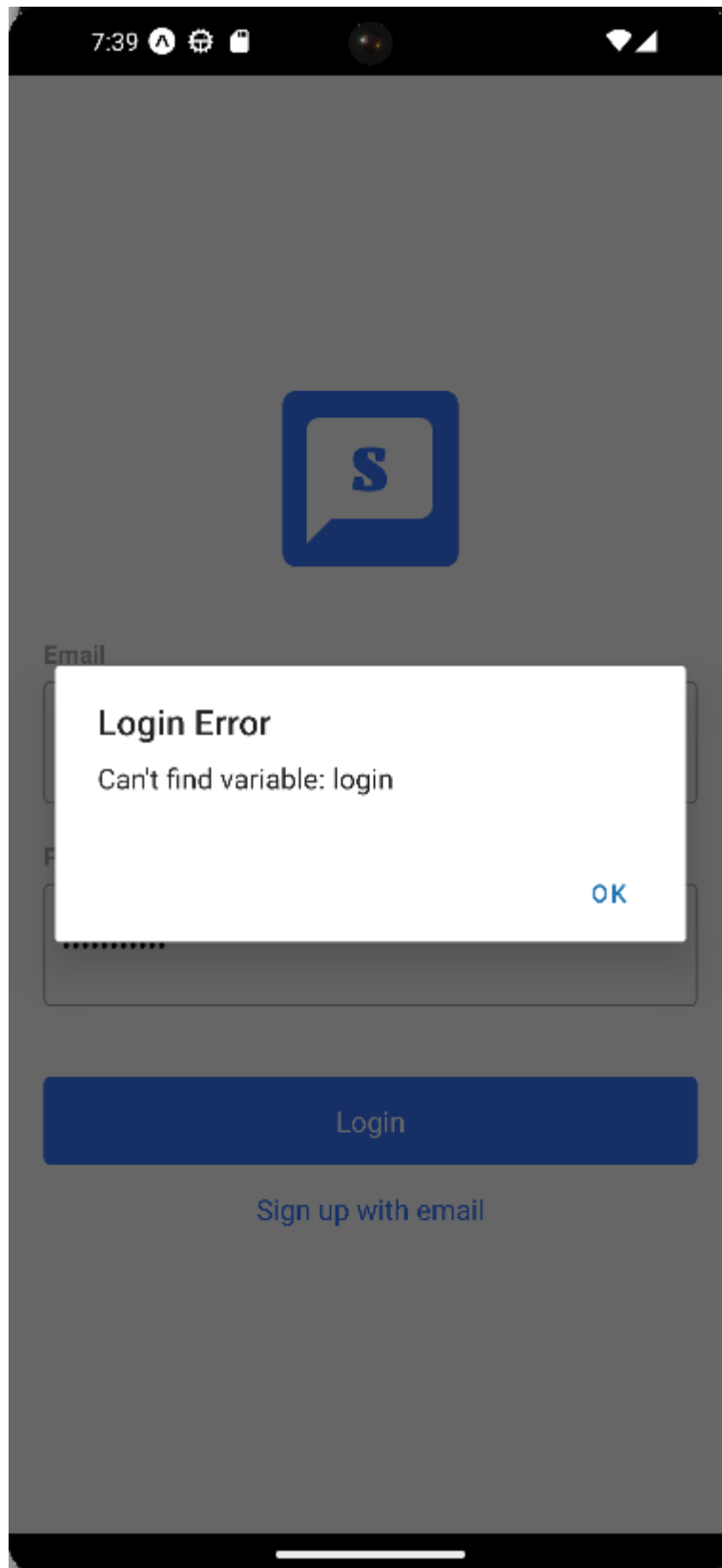
JS Signup.js  JS firebase.js X
src > utils > JS firebase.js > ...
1  import * as firebase from 'firebase';
2  import config from '../..//firebase.json';
3
4  const app = firebase.initializeApp(config);
5
6  const Auth = app.auth();
7
8  export const login = async ({email, password}) => {
9    const {user} = await Auth.signInWithEmailAndPassword(email, password);
10   return user;
11 };

```

작성된 함수를 이용해서 로그인 화면을 아래 코드와 같이 수정하자.

```
JS Signup.js JS firebase.js JS Login.js X
src > screens > JS Login.js > [e] Login
8  import {Alert} from 'react-native';
9  import {login} from '../utils/firebase';
10
11 > const Container = styled.View` ...
19 `;
20
21 > const ErrorText = styled.Text` ...
28 `;
29
30 const Login = ({navigation})=>{
31   const insets = useSafeAreaInsets();
32   const [email, setEmail] = useState('');
33   const [password, setPassword] = useState('');
34   const passwordRef = useRef();
35   const [errorMessage, setErrorMessage] = useState('');
36   const [disabled, setDisabled] = useState(true);
37
38   const _handleLoginButtonPress = async () => {
39     try {
40       const user = await login ({email, password});
41       Alert.alert('Login Success', user.email);
42     }catch (e){
43       Alert.alert('Login Error', e.message);
44     }
45   };

```



계속 에러가 나고 해결되지 않는다. 일단 넘어가야겠다.
firebase의 버전이 올라가면서 명령어가 많이 변했다.

JS firebase.js X JS Login.js

src > utils > JS firebase.js > [⌘] signup

```
1 import { initializeApp } from 'firebase/app';
2 import {
3   getAuth,
4   signInWithEmailAndPassword,
5   createUserWithEmailAndPassword,
6   signOut,
7   updateProfile,
8 } from 'firebase/auth';
9 import { getFirestore, collection, doc, setDoc } from 'firebase/firestore';
10 import { getDownloadURL, getStorage, ref, uploadBytes } from 'firebase/storage';
11 import config from '../../firebase.json';
12 export const app = initializeApp(config);
13 const auth = getAuth(app);
14 export const signin = async ({ email, password }) => {
15   const { user } = await signInWithEmailAndPassword(auth, email, password);
16   return user;
17 };
18 export const signup = async ({ name, email, password, photoUrl }) => {
19   const { user } = await createUserWithEmailAndPassword(auth, email, password);
20   const photoURL = await uploadImage(photoUrl);
21   await updateProfile(auth.currentUser, { displayName: name, photoURL });
22   return user;
23 };
24 const uploadImage = async uri => {
25   if (uri.startsWith('https')) {
26     return uri;
27   }
28   const response = await fetch(uri);
29   const blob = await response.blob();
30   const { uid } = auth.currentUser;
31   const storage = getStorage(app);
32   const storageRef = ref(storage, `/profile/${uid}/photo.png`);
33   await uploadBytes(storageRef, blob, {
34     contentType: 'image/png',
35   });
36   return await getDownloadURL(storageRef);
37 };
38
```



```

JS firebase.js  JS Login.js  ●
src > screens > JS Login.js > ...
1  import React,{useState,useRef,useEffect} from 'react';
2  import styled from 'styled-components';
3  import { Image ,Input,Button} from '../components';
4  import { images } from '../utils/images';
5  import { TouchableWithoutFeedback ,Keyboard } from 'react-native';
6  import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
7  import { validateEmail, removeWhitespace } from '../utils/common';
8  import { useSafeAreaInsets } from 'react-native-safe-area-context';
9  import { Alert } from 'react-native';
10 import { signin } from '../utils/firebase';
11
12 const Container = styled.View`
13   flex : 1;
14   justify-content: center;
15   align-items: center;
16   padding: 0 20px;
17   background-color: ${({theme})=>theme.background};
18   padding: 20px;
19 `;
20 const ErrorText = styled.Text`
21   align-items: flex-start;
22   width: 100%;
23   height: 20px;
24   margin-bottom: 10px;
25   line-height: 20px;
26   color: ${({ theme }) => theme.errorText};
27 `;

```

```

JS firebase.js  JS Login.js  ●
src > screens > JS Login.js > ...
28  const Login = ({navigation})=>{
29      const [email,setEmail] = useState('');
30      const [password, setPassword] = useState('');
31      const passwordRef = useRef();
32      //에러변수 담기
33      const [errorMessage, setErrorMessage] = useState('');
34      const insets = useSafeAreaInsets();
35      const [disabled,setDisabled] = useState(true);
36
37      useEffect(() => {
38          setDisabled(!(email && password && !errorMessage));
39      }, [email, password, errorMessage]);
40
41      //이메일을 입력 체크 함수
42      const _handleEmailChange = email => {
43          //공백 체크
44          const changedEmail = removeWhitespace(email);
45          //set으로 담기
46          setEmail(changedEmail);
47          //에러체크하기
48          setErrorMessage(
49              validateEmail(changedEmail) ? '' : 'Please verify your email.'
50          );
51      }
52      //패스워드 공백체크
53      const _handlePasswordChange = password => {
54          setPassword(removeWhitespace(password));
55      };

```

```

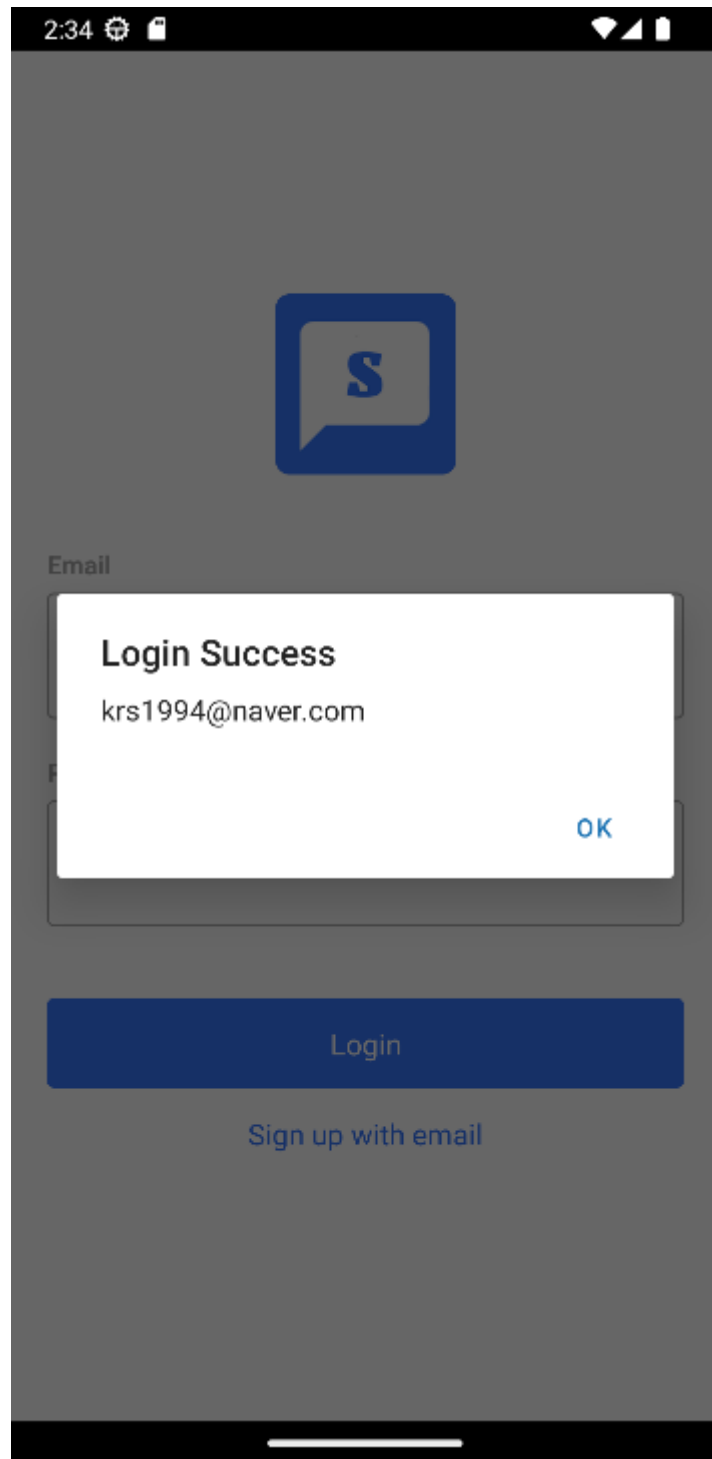
JS firebase.js  JS Login.js  ●
src > screens > JS Login.js > ...
56
57      //버튼누르면 연결될 함수
58      const _handleLoginButtonPress = async () => {
59          try {
60
61              const user = await signin({ email, password });
62              Alert.alert('Login Success',user.email);
63          } catch (e) {
64              Alert.alert('Login Error', e.message);
65          } finally {
66
67          }
68      };

```

```
JS firebase.js × JS Login.js ●
src > screens > JS Login.js > [🔍] Login
70   return(
71     <KeyboardAwareScrollView
72       contentContainerStyle={{flex:1}}
73       extraScrollHeight={20}
74     >
75       <Container insets={insets}>
76         <Image url = {images.logo} imageStyle={{borderRadius:8}}/>
77         <Input
78           label="Email"
79           value={email}
80           onChangeText={_handleEmailChange}
81           onSubmitEditing={()=>{passwordRef.current.focus()}}
82           placeholder="Email"
83           returnKeyType="next"
84         />
85         <Input
86           ref={passwordRef}
87           label="Password"
88           value={password}
89           onChangeText={_handlePasswordChange}
90           onSubmitEditing={()=>{}}
91           placeholder="Password"
92           returnKeyType="done"
93           isPassword
94         />
95         { /* 에러변수 출력 */ }
96       <ErrorText>{errorMessage}</ErrorText>
97       <Button
98         title="Login"
99         onPress={_handleLoginButtonPress}
100        disabled = {disabled}
101      />
102    </Container>
103  )
```

```
JS firebase.js JS Login.js ●
src > screens > JS Login.js > [e] Login
104     <Button
105       title="Sign up with email"
106       onPress={() => navigation.navigate('Signup')}
107       isFilled={false}
108     />
109   </Container>
110
111
112
113   </KeyboardAwareScrollView>
114
115
116
117   )
118 }
119 export default Login;
```

위 코드로 변경하니 잘 해결되었다.



회원가입

회원가입 기능을 만들어보자. 파이어베이스에서 제공하는 함수 중 이메일과 비밀번호를 이용해서 사용자를 생성하는 함수는 `createUserWithEmailAndPassword` 이다. 이 함수를 이용해 `firebase.js`파일에서 회원가입 기능을 만들어보자

```

JS firebase.js X JS Signup.js JS Login.js
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
6  const Auth = app.auth();
7  export const login = async ({email, password}) => {
8    const {user} = await Auth.signInWithEmailAndPassword(email, password);
9    return user;
10 };
11
12 export const signup = async ({ email, password}) => {
13   const {user} = await Auth.createUserWithEmailAndPassword(email,password);
14   return user;
15 };

```

작성된 함수를 이용해 회원가입 화면을 아래와 같이 수정하자.

```

JS firebase.js JS Signup.js X JS Login.js
src > screens > JS Signup.js > ...
1  import React, {useState, useRef, useEffect} from 'react';
2  import styled from 'styled-components';
3  import {Image, Input, Button} from '../components'
4  import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
5  import { validateEmail, removeWhitespace } from '../utils/common';
6  import {images} from '../utils/images'
7  import { Alert } from 'react-native';
8

```

```

const _handleSignupButtonPress = async ()=>{
  try{
    const user = await signup ({email, password});
    console.log(user);
    Alert.alert('Signup Success', user.email);
  }catch(e) {
    Alert.alert('Signup Error', e.message);
  }
};

```

사용자는 정상적으로 추가되지만, 우리는 사용자의 사진과 이름을 이용하지 않고 이메일과 비밀번호만으로 사용자를 생성했다. 파이어베이스에서 제공하는 사용자 생성 함수는 이메일과 비밀번호만으로 파라미터로 받는데 어떻게 사용자의 사진과 이름을 지정할 수 있을까. signup 함수에서 반환되는 user객체를 보면 입력한 이메일과 함께 다음과 같은 내용을 확인할 수 있다.

그림p.370

email은 우리가 입력한 사용자의 이메일 주소고, uid는 사용자마다 갖고 있는 유일한 키 값으로 사용자를 식별하는 데 사용된다. 우리는 displayName에 사용자의 이름을, photoURL에 사용자 사진의 url을 입력해서 생성된 사용자의 정보를 추가할 수 있다.

사용자 이름은 문자열로 입력할 수 있지만, 사진은 약간의 변화가 필요하다. 라이브러리를 이용해서 받은 선택된 사진의 내용은 "file:///..."로 진행되는 값을 갖고 있어 바로 사용할 수 없다. 이 문제는 사용자에게 의해 선택된 사진을 스토리지에 업로드하고 업로드된 사진을 url을 이용하는 방법으로 해결할 수 있다. 이번에는 사진을 스토리지에 업로드하는 함수를 만들고 signup 함수를 수정해서 생성되는 사용자의 사진과 이름을 설정하도록 수정하자.

```

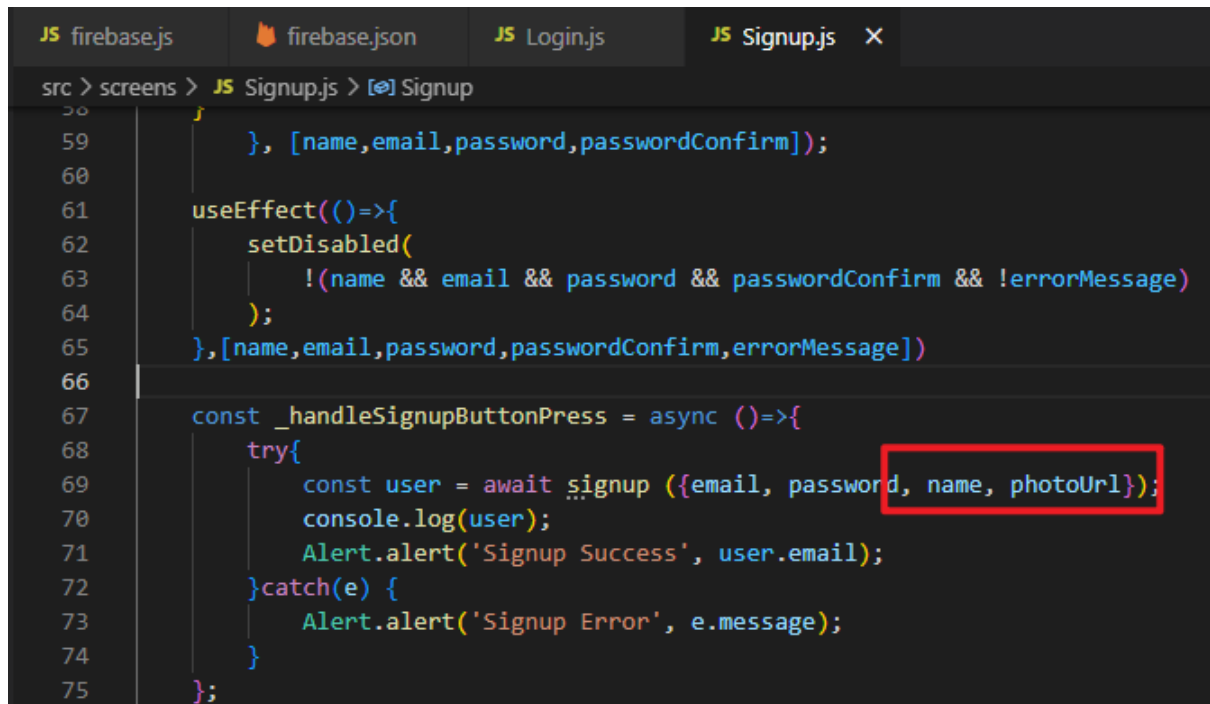
JS firebase.js X
src > utils > JS firebase.js > login
1  import * as firebase from 'firebase'
2  import config from '../..//firebase.json';
3
4  const app = firebase.initializeApp(config);
5  const Auth = app.auth();
6
7  const uploadImage = async uri => {
8      const blob = await new Promise((resolve, reject) =>{
9          const xhr = new XMLHttpRequest();
10         xhr.onload = function () {
11             resolve(xhr.response);
12         };
13         xhr.onerror = function (e) {
14             reject(new TypeError('Network request failed'));
15         };
16         xhr.responseType = 'blob';
17         xhr.open('GET',uri, true);
18         xhr.send(null);
19     });
20     const user = Auth.currentUser;
21     const ref = app.storage().ref(`/profile/${user.uid}/photo.png`);
22     const snapshot = await ref.put(blob, {contentType : 'image/png'});
23     blob.close();
24     return await snapshot.ref.getDownloadURL();
25 };
26 export const login = async ({email, password}) => {
27     const { user } = await Auth.signInWithEmailAndPassword(email, password);
28     return user;
29 };
30 export const signup = async ({ email, password, name, photoUrl}) => {
31     const { user } = await Auth.createUserWithEmailAndPassword(email,password);
32     const storageUrl = photoUrl.startsWith('https')
33       ? photoUrl
34       : await uploadImage(photoUrl);
35     await user.updateProfile({
36         displayName : name,
37         photoURL : storageUrl,
38     });
39     return user;
40 };

```

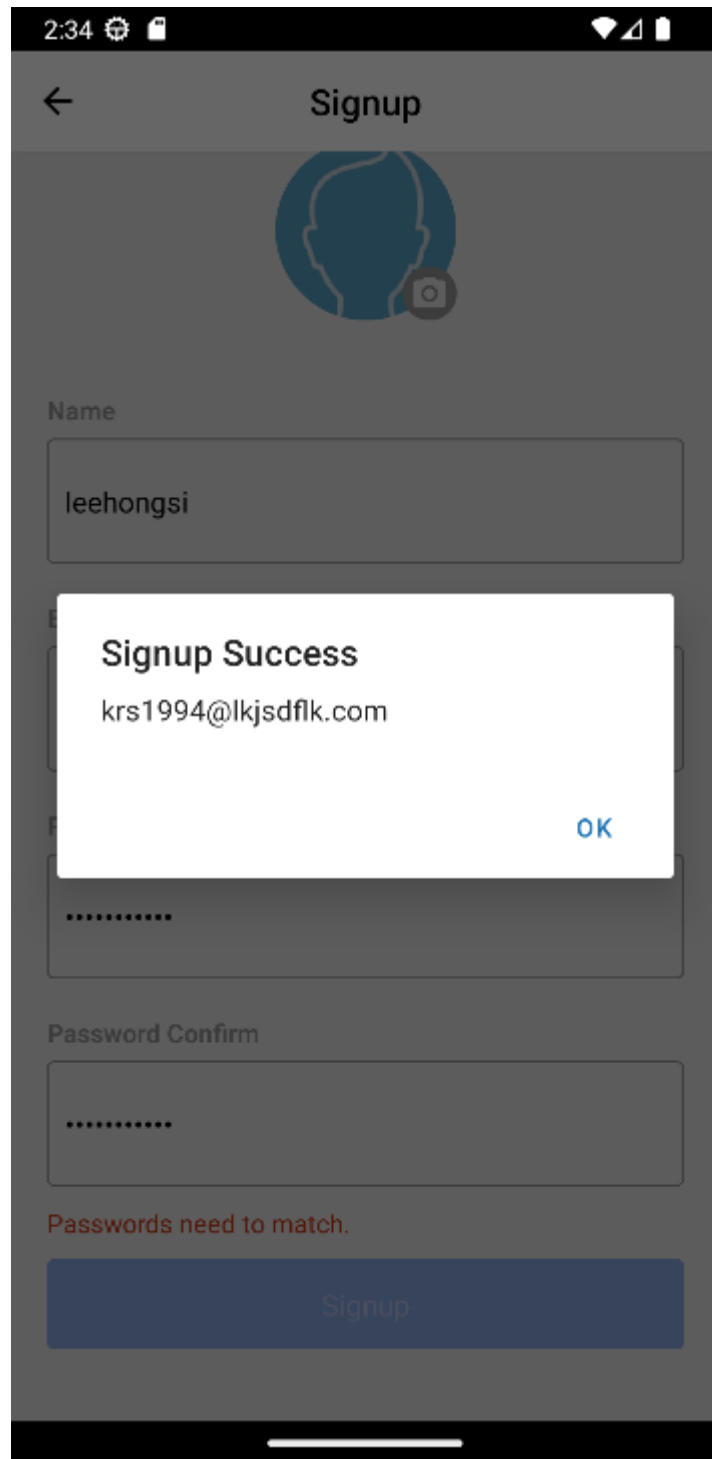
스토리지에 사진을 업로드하고 url을 반환하는 uploadImage함수를 만들었다. 스토리지에 업로드할 때, 현재 인증된 사용자의 정보를 담은 currentUser의 uid를 이용해 사진이 저장될 주소를 구분하도록 작성했다. 이렇게 사용자의 uid를 이용해서 파일이 저장되는 주소를 지정하면 규칙 수정을 통해 파일의 접근 권한을 설정하기 쉬울 뿐 아니라, 해당 사용자의 사진을 쉽게 찾을 수 있다는 장점이 있다.

signup 함수는 사용자의 이름과 선택된 사진 주소를 추가로 전달받도록 수정되었다. 사용자가 사진을 선택하지 않고 진행할 경우, 앞에서 스토리지에 업로드한 기본 이미지의 주소를 갖고 있으므로 업로드를 따로 진행하지 않도록 작성했다.

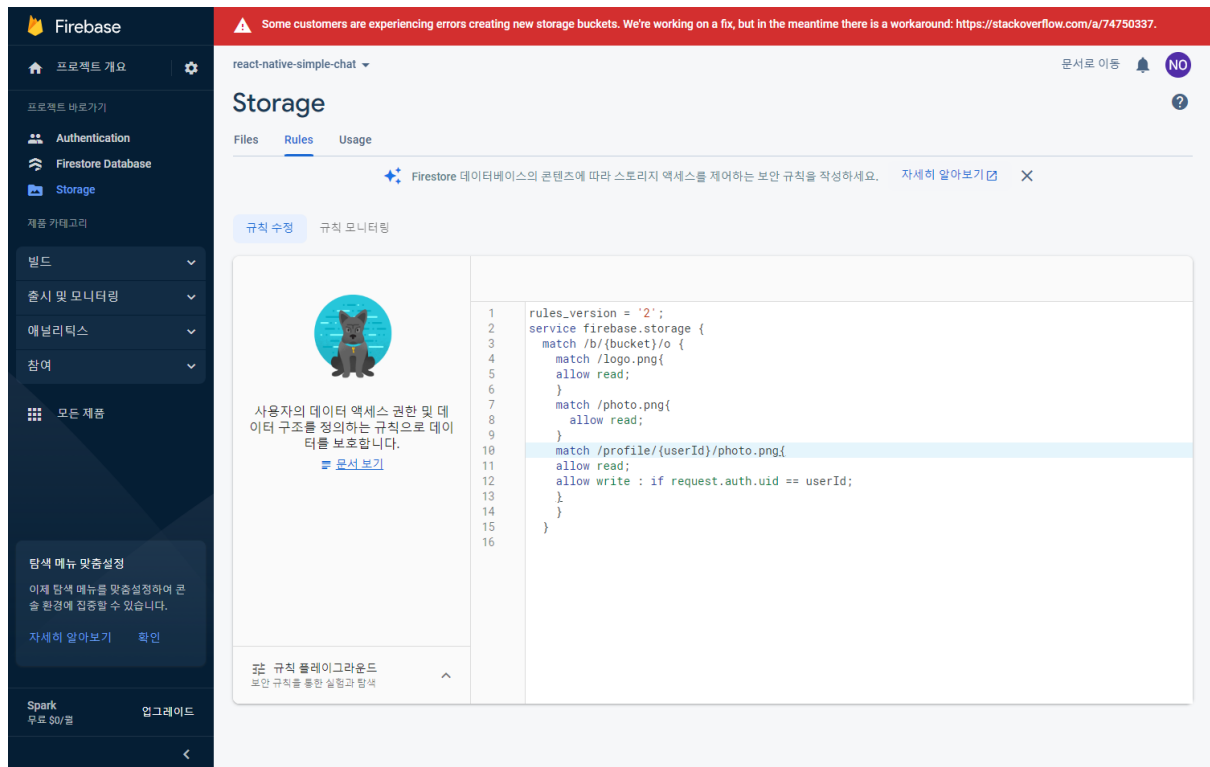
firebase.js 파일 수정이 완료되면 회원가입 화면을 아래 코드와 같이 수정하자.



```
JS firebase.js  firebase.json  JS Login.js  JS Signup.js X
src > screens > JS Signup.js > [🔍] Signup
58      },
59      }, [name, email, password, passwordConfirm]);
60
61      useEffect(() => {
62          setDisabled(
63              !(name && email && password && passwordConfirm && !errorMessage)
64          );
65      }, [name, email, password, passwordConfirm, errorMessage])
66
67      const _handleSignupButtonPress = async () => {
68          try {
69              const user = await signup ({email, password, name, photoUrl});
70              console.log(user);
71              Alert.alert('Signup Success', user.email);
72          } catch (e) {
73              Alert.alert('Signup Error', e.message);
74          }
75      };
```



회원가입 화면 수정이 완료되면 쓰기 권한은 사용자 본인만 가능하도록 하고 읽기 권한은 누구나 가능하도록 스토리지의 보안 규칙을 아래 코드와 같이 수정한다.



규칙 수정에서 “시뮬레이션 유형”을 변경하면서 다양한 상황을 만들 수 있고, 임의의 uid를 지정해서 인증된 사용자의 접근 여부를 테스트할 수 있으므로 다양한 상황을 테스트 해보자.

규칙 수정이 완료되면 회원가입 화면에서 선택한 사진과 입력된 이름이 생성되는 사용자의 정보에 추가된 것을 확인할 수 있다.

Spinner 컴포넌트

로그인 혹은 회원가입이 진행되는 동안 데이터를 수정하거나 버튼을 추가로 클릭하는 일이 발생하지 않도록 Spinner 컴포넌트를 만들어 사용자의 잘못된 입력이나 클릭을 방지하는 기능을 만들어보자. 먼저 Spinner 컴포넌트에서 사용할 색을 정의하고 진행한다.

```
JS firebase.js  firebase.json  JS Login.js  JS Signup.js  JS theme.js  X
src > JS theme.js > [🔍] theme
19   imageButtonIcon : colors.white,
20   label : colors.grey_1,
21   inputPlaceholder : colors.grey_1,
22   inputBorder : colors.grey_1,
23
24   buttonBackground : colors.blue,
25   buttonTitle : colors.white,
26   buttonUnfilledTitle : colors.blue,
27
28   headerTintColor : colors.black,
29
30   //spinner 컴포넌트 테마
31   spinnerBackground : colors.black,
32   spinnerIndicator : colors.white,
33 };
34
```

이제 components 폴더에 Spinner 컴포넌트를 만든다. Spinner컴포넌트를 리액트네이티브에서 제공하는 ActivityIndicator 컴포넌트를 이용해서 쉽게 만들 수 있다.

Spinner 컴포넌트는 화면 전체를 차지하면서 사용자가 다른 행동을 취할 수 없도록 다른 컴포넌트보다 위에 있게 작성했다. Spinner 컴포넌트 작성이 완료되면 components 폴더의 index.js파일을 아래 코드처럼 수정한다.

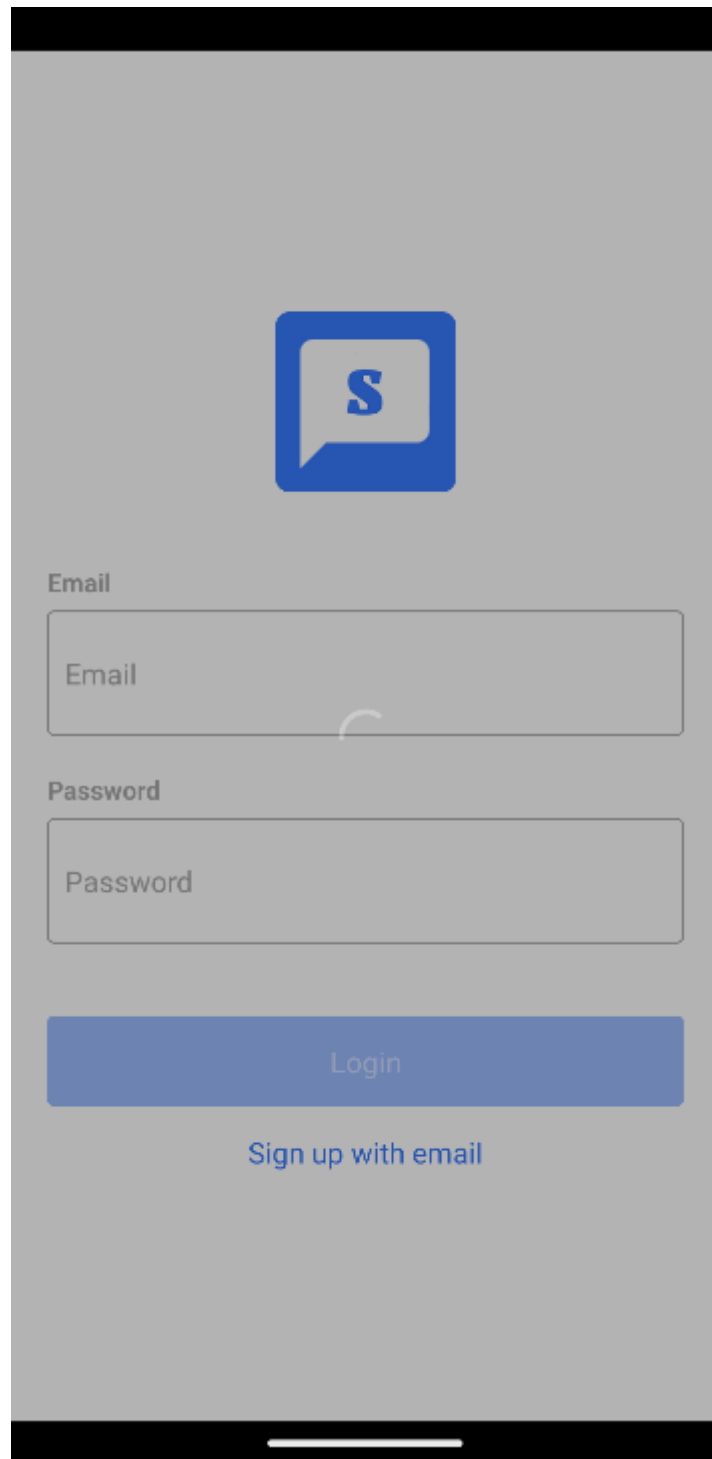
```
JS Spinner.js X JS index.js ...\components JS index.js ...\navigations JS Login.js
src > components > JS Spinner.js > [e] Container
1 import React, {useContext} from "react";
2 import { ActivityIndicator } from "react-native";
3 import styled, {ThemeContext} from "styled-components";
4
5 const Container = styled.View`
6   position : absolute;
7   z-index : 2;
8   opacity : 0.3;
9   width : 100%;
10  height : 100%;
11  justify-content : center;
12  background-color : ${({theme})=> theme.spinnerBackground};
13 `;
14
15 const Spinner =()=> {
16   const theme = useContext(ThemeContext);
17   return (
18     <Container>
19       <ActivityIndicator size={'large'} color={theme.spinnerIndicator} />
20     </Container>
21   );
22 };
23
24 export default Spinner;
```

```
JS Spinner.js JS index.js X
src > components > JS index.js
1 import Image from './Image';
2 import Input from './Input';
3 import Button from './Button';
4 import Spinner from './Spinner'
5
6 export {Image, Input, Button, Spinner};
```

Spinner 컴포넌트를 AuthStack 네비게이션의 하위 컴포넌트로 사용하면 네비게이션을 포함한 화면 전체를 차지할 수 없다. 네비게이션을 포함한 화면 전체를 감싸기 위해 navigations폴더의 index.js에서 AuthStack 네비게이션과 같은 위치에 Spinner 컴포넌트를 사용하자.

```
JS Spinner.js    JS index.js ...\components    JS index.js ...\navigations X    JS Login.js

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



Spinner 컴포넌트가 화면 전체를 차지하고 있다.

Spinner 컴포넌트는 로그인 버튼을 클릭했을 때나 회원가입 버튼을 클릭했을 때처럼 여러 화면에서 발생하는 특정 상황에서만 렌더링 되어야 한다. 이렇게 여러 화면에서 하나의 상태를 이용하기 위해 전역적으로 상태를 관리하는 방법으로는 Context API가 있다.

이번에는 Context API를 이용해 Spinner 컴포넌트의 렌더링 상태를 전역적으로 관리하도록 만들어보자.

먼저 contexts 폴더 안에 Progress.js 파일을 생성하고 아래 코드처럼 작성하자.

```
JS Progress.js ●
src > contexts > JS Progress.js > ...
1  import React, {useState, createContext} from "react";
2
3  const ProgressContext = createContext({
4    |   inProgress : false,
5    |   spinner : () => {},
6  });
7
8  const ProgressProvider = ({children}) => {
9    |   const [inProgress, setInProgress] = useState(false);
10   |   const spinner = {
11   |     |   start : () => setInProgress(true),
12   |     |   stop : () => setInProgress(false),
13   |     |   };
14   |   const value = {inProgress, spinner};
15   |   return (
16   |     |   <ProgressContext.Provider value={value}>
17   |     |     {children}
18   |     |   </ProgressContext.Provider>
19   |   );
20  };
21
22  export {ProgressContext, ProgressProvider};
```

createContext 함수를 이용해 Context를 생성하고, Provider 컴포넌트의 value에 Spinner 컴포넌트의 렌더링 상태를 관리할 inProgress 상태 변수와 상태를 변경할 수 있는 함수를 전달했다. 상태를 변경하는 함수는 사용자가 명확하게 렌더링 여부를 관리할 수 있도록 start 함수와 stop 함수를 만들어서 전달했다.

ProgressContext 작성이 완료되면 contexts 폴더에 index.js 파일을 생성하고 아래 코드처럼 작성한다.


```
JS Progress.js JS index.js X
src > contexts > JS index.js
1 import { ProgressContext, ProgressProvider } from "../Progress";
2
3 export { ProgressContext, ProgressProvider };
```

이제 App 컴포넌트에서 ProgressProvider 컴포넌트를 이용해 애플리케이션 전체를 감싸도록 수정하자.

```
JS Progress.js JS index.js JS App.js X
src > JS App.js > App
9 import Navigation from './navigations';
10 import {images} from './utils/images';
11 import { ProgressProvider } from './contexts';
12
13 > const cacheImages = images => { ...
21 };
22 > const cacheFonts = fonts => { ...
24 };
25
26
27 < const App = () => {
28   const [isReady, setIsReady] = useState(false);
29
30   > const _loadAssets = async() => { ...
35   }
36
37   < return isReady ? (
38     < ThemeProvider theme={theme} >
39     < ProgressProvider >
40     < StatusBar barStyle="dark-content" />
41     < Navigation />
42     < /ProgressProvider >
43   < /ThemeProvider >
44 }
```

이제 Spinner 컴포넌트가 ProgressContext의 inProgress 상태에 따라 렌더링되도록 navigations의 index.js파일을 아래 코드처럼 수정하자.

```
JS Progress.js JS index.js ...\contexts JS App.js JS index.js ...\navigations X
src > navigations > JS index.js > default
1 import React, {useContext} from 'react';
2 import { NavigationContainer } from '@react-navigation/native';
3 import AuthStack from '../AuthStack';
4 import Spinner from '../components/Spinner';
5 import { ProgressContext } from '../contexts';
6
7 const Navigation = ()=>{
8   const {inProgress} = useContext(ProgressContext);
9   return(
10     <NavigationContainer>
11       <AuthStack />
12       {inProgress && <Spinner />}
13     </NavigationContainer>
14   )
15 }
16
17 export default Navigation;
```

inProgress의 초깃값이 false이므로 Spinner 컴포넌트가 나타나지 않는다.

로그인 화면에서 로그인 버튼을 클릭했을 때 inProgress 상태를 변경하여 Spinner 컴포넌트가 렌더링 되도록 수정하자.

```
JS Login.js X
src > screens > JS Login.js > Login > _handleLoginButtonPress
1 import React, {useState, useRef, useEffect, useContext} from 'react';
2 import { ProgressContext } from '../contexts';
3 import styled from 'styled-components';
4 import {Image, Input, Button} from '../components';
5 import {images} from '../utils/images'
```

```
JS Login.js X
src > screens > JS Login.js > Login > _handleLoginButtonPress
32
33 const Login = ({navigation})=>{
34   const insets = useSafeAreaInsets();
35   const [email, setEmail] = useState('');
36   const [password, setPassword] = useState('');
37   const passwordRef = useRef();
38   const [errorMessage, setErrorMessage] = useState('');
39   const [disabled, setDisabled] = useState(true);
40   const {spinner} = useContext(ProgressContext);
41
42   > useEffect(() => { ...
44     }, [email, password, errorMessage])
45
46   > const _handleEmailChange = email => { ...
52     };
53   > const _handlePasswordChange = password => { ...
55     };
56
57   const _handleLoginButtonPress = async () => {
58     try {
59       spinner.start();
60       const user = await login({email, password});
61       Alert.alert('Login Success', user.email);
62     }catch(e){
63       Alert.alert('Login Error', e.message);
64     }finally{
65       spinner.stop();
66     }
67   };
68   return
```

login 함수를 호출하기 전에 Spinner 컴포넌트가 렌더링 되도록 상태를 변경하고, 성공 여부와 상관없이 작업이 완료되면 Spinner 컴포넌트가 렌더링 되지 않게 상태를 변경하도록 작성했다.

마지막으로 회원가입 화면에서도 signup 함수를 호출하기 전에 Spinner 컴포넌트가 렌더링 되도록 수정하자.