**Day1**

여러 dependencies 를 설치했다. 프로젝트 폴더 내에 app 이란 이름의 폴더를 생성 하고 이곳에 가장 먼저 불러와 지는 .js 파일을 생성했다. 앱이 켜지면 처음에 루트 폴더에 있는 ‘index.js’ 파일을 불러오는데, 여기서 만들어 둔 app 폴더 내의 app.js를 불러오게 끔 하면 된다.

app/app.js

텍스트이(가) 표시된 사진

자동 생성된 설명

-AppRegistry : 리액트 네이티브를 실행하기 위한 api.  
([AppRegistry · React Native](https://reactnative.dev/docs/appregistry))

-createStore : redux 의 store를 만들어준다.  
([createStore | Redux](https://ko.redux.js.org/api/createstore))

-applyMiddleware : redux-promise 와 같은 미들웨어를 사용할 수 있게함.  
([applyMiddleware | Redux](https://ko.redux.js.org/api/applymiddleware))

-compose : 여러 store enhancer들을 순차적으로 적용할 수 있게함. Store enhancer란 여러 store creator 들을 조합해서 더욱 강화된 store creator를 반환하는 고차함수.  
([compose | Redux](https://ko.redux.js.org/api/compose))

-Provider : react-native 컴포넌트들이 store를 찾을 수 있게 패싱해줌.  
([Provider | React Redux (react-redux.js.org)](https://react-redux.js.org/api/provider))

createStoreWithMiddleware 로 스토어를 생성한 뒤, 전체 앱을 store를 prop으로 받는 Provider로 감싸준다. appRedux 는 jsx 컴포넌트를 리턴하므로 중괄호가 아닌 소괄호가 들어가야 한다. 오타란 말씀.

app/store/reducers/index.js

텍스트이(가) 표시된 사진

자동 생성된 설명

-combineReducers : 리듀싱 함수들을 받아 하나의 state 객체로 바꿔준다.  
([combineReducers | Redux](https://ko.redux.js.org/api/combinereducers))

-리듀싱 함수 : state 와 action 을 받아 새로운 state를 반환한다. 이 때, 리듀싱 함수는 같은 입력에 대해 같은 출력을 내보내는 순수 함수여야 한다.

app/app.js  
텍스트이(가) 표시된 사진

자동 생성된 설명

- window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_ : 크롬확장 프로그램에 작성된 함수인데, 리덕스 개발자 도구와 미들웨어를 사용하기 위함이다. 기존 compose 도 함께 사용한다.

**Day2**

텍스트이(가) 표시된 사진

자동 생성된 설명이와 같이 components 폴더를 새로 만들어 각각 화면을 만들어줄 디렉토리를 생성했다. 또 app 아래에 react navigation 을 설계할 routes.js를 만든다.

app/components/auth/index.js

텍스트이(가) 표시된 사진

자동 생성된 설명

각 화면들을 생성할 때 기본적으로 만들어 둔 폼이다.

* flex : [flex - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/CSS/flex)
* alignItems : 가로축, 중심축을 기준으로 정렬  
  ([align-items - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/CSS/align-items))
* justifyContent : 세로축, 중심축을 기준으로 정렬  
  ([justify-content - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content))

app/routes.js

텍스트이(가) 표시된 사진

자동 생성된 설명

로그인 여부에 따라 스택 네비게이터로 로그인 화면을 보여줄지, 탭 네비게이터가 있는 메인화면을 보여줄지 결정한다. 당장은 Boolean 으로 네비게이터 구조를 변환 시키지만, 추후엔 백엔드로 로그인 여부에 따라 변환 시키기로 함.

* AuthStack&MainScreenTab : 각각 stack navigator 와 tab navigator를 받아옴.
* RootNavigator : export를 하는 이유는, 루트 폴더의 index.js 에서 import 할 것 이기 때문이다. boolean 변수에 따라 true 일때는 Main화면을, false 일때는 로그인 화면을 보여준다.
* AppTabComponent : 탭 네비게이터로 만든 메인화면이다.

authLogo.js

텍스트이(가) 표시된 사진

자동 생성된 설명

로고 이미지 컴퍼넌트를 만드는 파일이다.

[Image Style Props · React Native](https://reactnative.dev/docs/image-style-props)

좀 특이하게도 resizeMode는 props로도 존재해서, style 바깥으로 빼놔도 상관 없다.

authForm.js

텍스트이(가) 표시된 사진

자동 생성된 설명잘은 모르겠지만 View의 props를 전부 날려야 index.js 에서 hello 텍스트가 출력되었다. index.js에서 ScrollView의 style을 style로 주면 상관 없었지만, contentContainerStyle 로 주자 hello가 출력되지 않아 authForm.js 에서의 View 스타일을 전부 날려주었다.

[react native - style Vs contentContainerStyle in ScrollView? - Stack Overflow](https://stackoverflow.com/questions/52892304/style-vs-contentcontainerstyle-in-scrollview)

알아보니 style은 컨테이너에 스타일을 주는거고, 후자는 안에 있는 컨텐츠에 스타일을 주는거란다.

같은텍스트이(가) 표시된 사진

자동 생성된 설명 파일 렌더함수에 추가한 TextInput 이다. placeholder라는 props는 보통 텍스트를 입력하기 전 지시사항 같은 것을 전달하는데 사용된다.

input.js

텍스트이(가) 표시된 사진

자동 생성된 설명위의 TextInput을 **재사용 가능한 컴퍼넌트** 형태로 구현했다. 이 때, authForm.js 에서 사용한 여러 props들을 input.js가 알고 있어야 하므로 props란 이름으로 props를 전달하여 (react에서 함수 스타일의 첫번째 인자는 항상 props이다. 이름은 임의로 정해도 된다) 스프레드 오퍼레이터를 활용해 모든 props를 props로 넣는다.

authForm.js

텍스트이(가) 표시된 사진

자동 생성된 설명이제 로그인 화면을 구성할 건데, 로그인 화면에서 한 버튼을 누르면 회원가입 페이지로 이동하고, 회원가입 페이지에서 같은 버튼을 누르면 다시 로그인 페이지로 넘어가도록 할 것이다.

-type : 화면의 type  
-action : 버튼을 눌렀을 때의 액션.  
-actioinMode : 나도 잘 모르겠지만 아마 액션 설명글 같은 것 아닐까 싶음.  
-hasErrors : 에러 발생시 에러 메시지 출력 하려는 boolean  
-form : Input 컴포넌트에 props 로 전달할 것들이다. value는 입력 값, rules는 텍스트가 올바른 폼으로 작성 됐나 확인, valid는 rules에 맞춰 true false 반환하는 boolean.

당연하게도 Input 컴포넌트에 value와 type같은 props를 넣었지만 TextInput의 props는 아니다.

텍스트이(가) 표시된 사진

자동 생성된 설명input.js

앞의 authForm.js에서 온 props중 type에 따라 분기되는 스위치 문이다. 이렇게 하면 form의 종류에 따라 다른 style의 컴포넌트를 내보낼 수 있다.

authForm.js

텍스트이(가) 표시된 사진

자동 생성된 설명

[[react-native]TextInput 속성 - onChange, onChangeText 잘 사용하기 (velog.io)](https://velog.io/@swanious/react-nativeTextInput-%EC%86%8D%EC%84%B1-onChange-onChangeText-%EC%9E%98-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0)

onChangeText라는 props으로, state의 value를 받는 value props를 updateInput이라는 콜백함수를 통해 업데이트 해준다.

입력시에텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명 에러가 있으면 안되니까 일단 hasErrors를 false로 해주고, state의 form 객체를 카피해와서 첫번째 인자인 name으로 다시 form 내부의 객체를 접근해서 그 안에서 value를 불러온다. 때문에 updateInput 함수에 전달되는 name 인자는 form 내부 객체들과 같은 이름을 가져야한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

첫번째 함수는 화면 타입이 login 이 아니면, 즉 로그인 화면이 아닐 때 Input 컴포넌트를 하나 불러오는 함수이다. 두번째 함수는 hasErrors가 true 일 때 경고 메시지를 띄워주는 함수이다.  
routes.js

텍스트이(가) 표시된 사진

자동 생성된 설명

[Navigation prop reference | React Navigation](https://reactnavigation.org/docs/navigation-prop)

링크를 참고하면, screen 컴포넌트는 navigation 이라는 props를 제공해준다고 한다.

auth/index.js

텍스트이(가) 표시된 사진

자동 생성된 설명

routes.js에서 메인화면 컴포넌트와 로그인 화면 컴포넌트를 묶어놨기 때문에 (정확한 이유는 잘 모르겠다) goWithoutLogin 함수와 같이 navigate할 수 있다. navigate 가 뭔지는 위 링크 참조. AuthForm은 authForm.js 컴포넌트 인데, goWithoutLogin이 함수임에도 중괄호 없이 부른다는 점이 좀 신기했다. 이유는 역시 모르겠다.

authForm.js

텍스트이(가) 표시된 사진

자동 생성된 설명 onPress props로 부모한테 받아온 goWithoutLogin()을 사용하고 있다. 전달을 괄호 없이 변수? 같은 형태로 줬는데 사용은 또 괄호를 붙여서 함수임을 명시해주는 것이 신기하다.

좀더 알아보니 어차피 goWithoutLogin이 받는건 함수이고, 그냥 표현법에 따른 차이일 뿐이다.

이제 화면 type 따라 다른 화면을 보여주는 버튼을 만들려고 한다. 그러려면 먼저 현재 state의 type을 받아야 하고, 그 type에 따라 첫번째 버튼의 title인 action과 두번째 버튼인 actionMode 필드를 바꿔주면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 로그인 버튼을 구현할 건데, 고려해야 하는게 꽤 많다. 이메일 주소와 비밀번호가 입력이 됐는지 체크해줘야 하고, 형식에 맞는지도 체크해야 한다. 비밀번호 규칙 같은 것도 체크해줘야 한다. 또 회원가입 같은 경우 비밀번호 입력란과 확인란에 입력된 내용이 같은지도 봐야한다.

텍스트이(가) 표시된 사진

자동 생성된 설명rules 필드를 채워넣었다.

-isRequired: 반드시 필요하는 뜻  
-isEmail: 형식에 맞는지.  
-minLength: 비밀번호 최소 길이  
-confirmPassword: ‘password’를 초기값으로 준 것은 password 객체에 접근하여 그 객체의 rule을 이용하려고 하는 목적이다.

텍스트이(가) 표시된 사진

자동 생성된 설명validationRules.js는 입력값인 value, 객체의 rules, 그리고 state의 form을 받아 valid 여부를 판단한다. 결국 state의 form 필드 안 각 객체들의 valid 를 업데이트 해주려고 하는 거니까, 기존 updateInput 함수를 이용하기로 한다.

validationRules.js

텍스트이(가) 표시된 사진

자동 생성된 설명입력값을 받아 공백이 아니면 true를 return 해주는 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

입력값을 받아 js 내장함수인 toLocaleLowerCase와 test를 이용해 전부 소문자로 바꾼 뒤 expression의 형식과 맞는지 test 하여 true나 false를 반환하는 함수이다.



앞서 계획했던 confirmPassword의 구현 방식이다. Validation 함수의 인자인 rules는, authForm.js의 state 필드 중 form의 특정 객체의 rules 이고, 따라서 이 스위치문에서 rule이 confirmPassword일 때 rules.confirmPassword는 ‘password’이다. 즉 form[password]가 되는 것이고, .value를 붙였으니 결국 회원가입 화면에서 비밀번호 입력란에 입력한 입력값이 되는 것이다. 즉 이 validateConfirmPassword 함수는 비밀번호 재입력 란에 입력한 값과, 비밀번호 입력란에 입력한 값을 받아오는 것이다.

**Day3**텍스트이(가) 표시된 사진

자동 생성된 설명로그인, 등록 버튼 즉 첫번째 버튼의 이벤트 핸들러 함수이다.  
-isFormValid : 해당 함수가 콜백 됐을 때, 입력된 내용들이 형식에 맞는지 state의 valid 필드를 받아오는 변수이다. 뒤에 for 문에서 이메일, 비밀번호, 비밀번호 재확인 입력 내용이 형식에 맞는지 체크한다.  
-submittedForm : 지금은 백엔드가 없으니 올바르게 state가 rules 아래 잘 업데이트 되는지 확인하는 용도의 객체이다. value를 받아 입력 내용을 출력해준다.

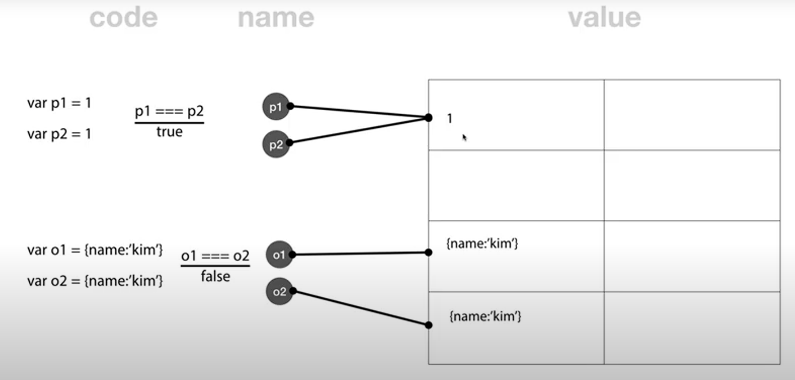
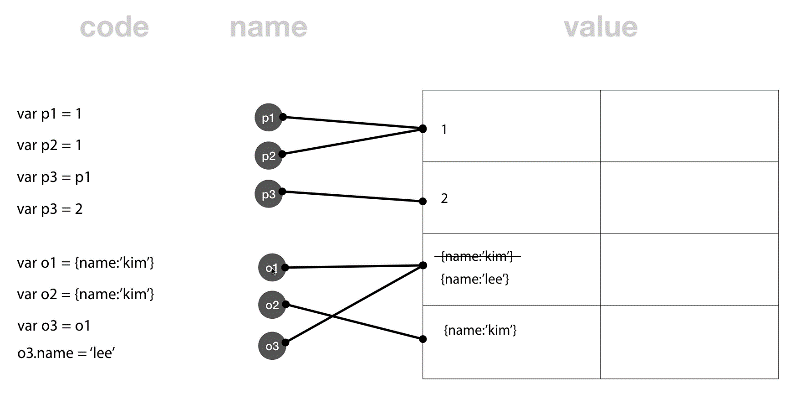
여기서 다시금 리덕스의 로직을 되짚어 볼 필요가 있어 조금 되돌아갔다.

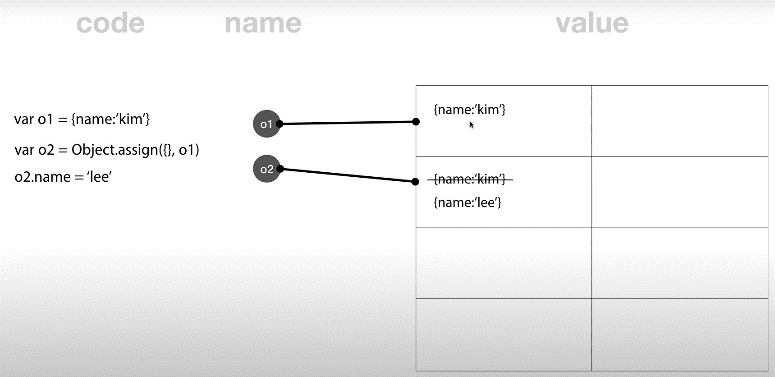
Redux의 docs를 읽어보는 것으로 시작했으나, 결국 관리해야하는 state의 양이 많아질수록 복잡도가 올라가기 때문에, 따로 관리 부분을 떼어내는게 redux가 하는일이다, 정도로 이해했다. 그러나 이것만으로는 여전히 redux를 왜 써야하는지에 대한 의문이 떠나지 않았으나, 배경지식이 너무나도 얕아 유튜브 강의 등의 힘을 빌려 조금이라도 이해해보고 넘어가기로 결정하였다.

생활코딩의 튜토리얼을 들으려 했는데, 먼저 js에 대한 학습과 immutability 에 대한 선수지식을 요구했다. 그래서 먼저 immutability가 뭔지 강의를 들어보기로 했다.

자바스크립트엔 두가지 크게 두가지 데이터 타입이 존재한다텍스트이(가) 표시된 사진

자동 생성된 설명.

이렇듯 원시 데이터와 객체 데이터는 불변함에 있어 다른 특성을 보인다. 객체들 끼리 같은 내용의 property 를 가진다 해도 서로 다른 객체로 취급하는 것이다.  
  
  
var o1 = {name:'kim'}  
var o2 = {name:'kim'}  
console.log(o1 === o2);  
=> false  
이렇듯 o1과 o3는 같은 value를 가리키고 있었는데 o3의 name property를 ‘lee’로 변경하자 o1역시 바뀐 value를 가리키고 있다.

이런식으로 객체를 복제해서 변경하면 o1이 변경되지 않는다. 그러나 객체 내에 또 객체가 들어있는 경우에는, 객체 내에 있는게 더 이상 원시 데이터가 아니기 때문에 다음과 같은 경우엔 불변하지 않다.

var o1 = {score:[1,2]}  
var o2 = Object.assign({}, o1);  
o2.score.push(3);  
console.log(o1.score)  
=> [1,2,3]이다. o1이 가리키는 객체를 복사했으나, 그 객체 내에 있는건 또 다른 객체기 때문이다.

var o1 = {score:[1,2]}  
var o2 = Object.assign({}, o1);  
o2.score = o2.score.concat(); // 배열을 복사합니다.  
o2.score.push(3);  
console.log(o1.score);  
=> [1,2]이다. 이렇게 배열을 복사해오거나  
var o1 = {score:[1,2]}  
Object.freeze(o1);  
Object.freeze(o1.score);  
o1.score.push(3);  
console.log(o1.score);  
=> 이렇게 value를 변경 불가능하게 얼려버리는 방법이 있다.

강의에서 덧붙이길, react는 이런 불변함 특성을 이용한 프레임 워크라고 한다. 렌더링을 할 때, 이전 state와 현재 state를 변경 불가능하게 해둬서 ‘===’와 같은 간단한 연산자로 비교할 수 있게 해놓았다는 것이다. 처음 react에 대한 얘기를 들을 땐 그냥 그렇구나 싶었는데, 이제와서 보니 어느정도 의도를 알 것 같기도하다. 위에서도 redux 의 목표가 여러 state처리를 따로 떼어놓고, 내가 알기론 store라는 react 와 서로 알 수 없는 영역으로 분리시켜서 한다는 건데, 여기서도 복합적인 state처리를 처리 되기 전/후를 불변하게 해서 좀더 파악하기 쉽게 함이 목적 아닐까 싶다.

Redux에 대한 강의도 조금 봤는데, 추상적인 부분이 필요한지라 앞부분만 듣고 넘어가기로 하였다. 필요하면 나중에 더 들어볼 예정이다. 일단 내가 예상한 부분이 어느정도 맞아떨어진다는 생각이 들었다. 여러 앱을 사용할 땐 수많은 화면변경이 일어나고, 개발자는 이런상황을 예상하여 렌더링을 위한 state 업데이트 요소들을 이것저것 많이 만들어 놔야한다. 그런데 이러한것들이 너무 많아지면 관리가 힘들기 때문에, store라는 외부와 철저히 분리된 (immutability를 왜 들으라 했는지 이해가 됐다) 공간 내에서만 state 갱신을 하고, 이를 단 하나의 state로 묶어 다시 react로 보내서 갱신을 하기 때문에 “Predictable” 즉 예측 가능한 개발 구성이 가능하다는 것이다. 이제 다시 rn으로 넘어가겠다.

user\_actions.js

텍스트이(가) 표시된 사진

자동 생성된 설명types.js에 만들어둔 타입을 불러온다. 따로 만들어놓고 불러오는 이유는 아마 뒤의 user\_reducer.js 에서도 같은 타입을 불러와 쓰기 때문일 것이다.

-action : 데이터를 store로 전달하는 함수이다. 전달 된 데이터들은 reducer를 통해 내부적으로 업데이트 된다. 형식은 그림과 같이 반드시 type 과 전달할 내용인 payload를 property로 두는게 국룰이라고 한다.

-action creator : 그림과 같이 단순히 action들을 반환하는 파일이다.

보는것과 같이 이 action creator에선 두 가지 액션이 보이는데, 하나는 sign in 즉 로그인, 두번째는 sign up 즉 회원가입이다.

user\_reducer.js

텍스트이(가) 표시된 사진

자동 생성된 설명reducer들을 모아놓은 파일이다. user\_actions.js의 action이 인자로 전달되는 것을 볼 수 있는데, 이 액션의 타입에 따라 분기해서 state를 업데이트 하는 모습을 볼 수 있다. false와 or 처리를 해서 아무것도 입력되지 않았을 때는 email, token에 false를 반환하려는 것을 볼 수 있다. 이 때 false가 앞에 오면 안되는데, 예를 들어 아무것도 입력되지 않은 상황이면 payload.email 이 undefined를 반환할 텐데 false가 앞에 있다면 false || undefined 와 같은 상황이 된다. || 연산자는 기본적으로 왼쪽 value가 바뀔 수 없다면 오른쪽 value를 반환 하므로, 항상 undefined만 반환 할 것이다.

authForm.js

텍스트이(가) 표시된 사진

자동 생성된 설명

authForm.js, 즉 AuthForm 컴포넌트에 이제 action 들을 store로 전달하고 스토어 내부에서 업데이트 된 state를 받아 갱신하여 주는 작업을 한다. export 하는게 AuthForm 클래스가 아닌 connect 함수인 것이 먼저 눈에 띄는데, AuthForm 컴포넌트를 store와 이어 준다는 것, 즉 한마디로 redux의 허락을 받고(?) state 갱신을 하겠다는 뜻이 된다.

-mapStateToProps : store 가 업데이트 될 때 마다 이 함수가 불린다. 여기를 null 로 두면 이 컴포넌트가 더 이상 store를 구독하지 않는다고 한다. docs 설명에 따르면 return 되는건 stateProps라는 객체이고, connect 된 컴포넌트에 props로 넘겨진다고 한다. state.User는 mapStateToProps 함수가 받아오는 store내부의 state의 User이고, 필드로 선언된 User는 AuthForm 컴포넌트로 넘겨지는 props 이다.

-mapDispatchToProps : 원래는 다음과 같이 사용한다.  
이렇듯 각 필드는 액션을 스토어로 dispatch하는 함수여야 하고, 이 역시 dispatchProps라는 props로 연결된 컴포넌트에 넘겨진다.

텍스트이(가) 표시된 사진

자동 생성된 설명bindActionCreators라는 함수로 여러 액션을 묶어서 dispatch 한다.

이렇게 store를 통한 state 업데이트를 하게 하는 컴포넌트와 기존 AuthForm 컴포넌트를 묶어 export 하는것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명submitUser 함수의 일부분이다. 원래 submitUser함수는 첫번째 버튼을 누를 때 콜백 되어, form의 각 필드 별 value를 출력해 줬었다. 이제는 action creator에서 import 한 두 action 에 출력 됐던 객체를 전달한다. 그러면 액션에서 store로 전달할 부분인 payload가 각각 email과 password를 전달받고, 위의 bindActionCreators에 의해 묶여서 한꺼번에 dispatch 된다. store의 ruducer로 전달된 payload 에 의해서 store내부의 state는 업데이트 되고, mapStateToProps로 다시 rn scope로 전달된다.

**Day4**

서버에 데이터를 요청하는 방식에는 두가지가 있다고 한다.  
1. Get : url 방식으로 요청한다. url끝에 “?”를 붙인다음 변수명1=값1&변수명2=값2 와 같은 형식이다. 기본적으로 유저에게 노출되기 때문에 보안상 무방한 데이터를 get방식으로 요청한다.  
2. Post : 데이터 전달을 요청하는 시점에 http request body에 데이터를 인자로 추가 해줘서 길이 제한이 없다고 한다. 또 유저가 어떠한 데이터가 오가는지 알 수 없다. 비밀번호와 같은 보안상 중요한 데이터를 다룰 때 사용한다고 한다.

이제 firebase를 이용해서 백엔드를 구현할 건데, 먼저 axios라는 서버에 데이터 요청을 하는 툴을 이용해서 백엔드가 어떻게 작동하는지 살펴보려 한다. axios는 서버에 데이터를 promise 객체로 요청하고 받아온다고 한다.

그런데 진행하던 도중 궁금증이 생겼다. 도대체 이 axios라는 녀석은 왜 필요한거고, 서버에 데이터를 요청한다는 것은 무슨 의미인가? 검색 해본 결과 react 자체에 서버-클라이언트 간 통신api 가 내장되어 있지 않아서 axios와 같은 것을 불러다 쓴다는 것은 알았다. 그런데 이 서버라는 개념이 나에겐 좀 두루뭉실한 것 같다. node.js 도 서버아닌가? Firebase에 요청하는 것 말고도 내부 서버에 요청하는 일도 있지 않은가? 일단 기초부터 조금씩 알아보자.

-HTTP : 서버와 클라이언트가 통신하는 규칙 이라고 한다. 데이터를 주고받는 방식이라고 알고 있으면 되겠다.  
-RESTful API : 요청이 http를 통해 관리되는 클라이언트-서버 아키텍처 이며, 이 외에도 여러가지 기준, 원칙을 가진 API를 의미한다. 더 읽어봐도 모르겠으므로 여기까지 하겠다.  
-promise : 데이터 요청을 비동기 식으로 처리 가능한 객체이다. 이렇게 하면 요청 응답이 올 때까지 마냥 기다리지 않고 일부라도 화면에 표시된다고 한다.  
-axios : 그래서 axios 가 뭐냐? Promise를 사용하고, http 방식의 비동기 통신 라이브러리 이다.  
-런타임 : 프로그래밍 언어가 구동되는 환경. Js는 웹브라우저에서, 또 node.js 에서 구동되는데, 이 때 브라우저와 node.js를 런타임이라 할 수 있다.  
-node.js : 무슨 언어의 일종인 줄 알았는데, npm 으로 디펜던시 관리와 같은 cli 를 입력하는 것 역시 node.js 위에서 돌리는 경우이고, 아무튼 그냥 런타임의 일종이라고 한다.  
-require vs import : 두 키워드 전부 외부 라이브러리를 불러오는 코드라 한다. 전자는 nodejs에서 사용하는 node 표준의 (CommonJs) 모듈화 방법이며, 후자는 es6부터 자바스크립트 자체적으로 지원되기 시작한 모듈화 방법이다.  
([CommonJS (velog.io)](https://velog.io/@leobit/CommonJS))

아이디어는 대충 이런듯 하다. user\_actions.js, 즉 action creator는 컴포넌트에서 불려서, store로 action이 넘겨지는데, 이 때 payload 에 담는 데이터를 axios를 통해 먼저 firebase에 request 한 뒤, 요청이 무사히 끝났다면 payload에 정상적으로 입력한 값이 담길 테니 reducer를 통해 그 값으로 state 갱신을 하는 것이다. 한마디로 store에 넘기는 payload에 데이터를 담기전에 파이어베이스에 데이터를 넘겨봐서 잘 넘겨졌는지, 안됐는지 검사해본다는 것이다.

user\_actions.js

텍스트이(가) 표시된 사진

자동 생성된 설명request 라는 axios 객체를 만들어 준다. http 방식을 쓰고 있는데, 자세한 config는 공식 깃허브에서 찾아볼 수 있었다. headers 객체에 들어가는 건 http 규칙으로, content-type 역시 entity header라고 따로 정해져 있었다. Firebase 공식문서에서 request하는데 필요한 정보들을 모두 넣어준 형태이다. 작동 메커니즘은 위에 기술한 것과 같다.

user\_reducer.js

파이어베이스를텍스트이(가) 표시된 사진

자동 생성된 설명 통과(?)하고 온 response는, 공식문서에 따라 각각 localId 등으로 온다.

로그인 역시 같은 방법으로 구현하며, 작동 되는 것을 확인했다. 이제 로컬 저장소에 로그인 정보를 저장 해놓고 앱을 껐다 켜도 자동으로 로그인이 되는 것을 구현해 보겠다.

**지금까지 완전히 착각하고 있었다는 것을 알았다.** state갱신은 그저 updateInput 함수로 입력 될 때마다 업데이트 되는 것이고, redux가 맡은 영역은 action, 즉 로그인과 회원가입이라는 액션을 스토어한테 props로 넘겨주고, 연결된 컴포넌트의 state까지 같이 받아서 업데이트 한 뒤 그걸 mapStateToProps 함수로 다시 연결된 컴포넌트에 props 형태로 돌려주는 것 이였다. 그렇게 함으로써 입력될 때마다 rn 의 state 갱신, 즉 입력할 때마다의 인풋들을 렌더링해서 보여 줄 수 있으며 동시에 입력한 값들을 확인 받는 작업 즉 로그인이나 회원가입 같은 액션을 처리하고 그걸 다시 props로 받아볼 수 있는 것이다! 내가 입력한 값이 아닌, 진짜 서버에 저장된 데이터들을 받아 로컬 저장소에 저장해놓고 필요할 때 꺼내쓸 수 있게 된 것이다. 진짜 비동기 방식의 처리 라는게 무엇인지 살짝 맛본듯 하다. 이렇게 작동한다면, rn의 진짜 state 갱신에 대한 걱정 없이 필요한곳에 넣고싶은 action을 넣어 처리하고 싶은것도 다 처리할 수 있는것이다.

authForm.js

텍스트이(가) 표시된 사진

자동 생성된 설명형식에 맞으면, 액션을 호출하는 부분이다. 여기에 이제 콜백함수를 넣어둔다. 액션이 호출되고, 리듀싱 함수까지 동작하고 스토어 내부의 state가 이 컴포넌트에 props로 전달 된 후에 작동한다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명mapStateToProps를 통해 스토어 내부 ‘state가 props로 전달 됐다. userId가 파이어베이스에 등록 되어 있으면, setTokens라는 함수에 auth 객체를 전달하고 메인화면으로 이동한다. auth 객체를 전달 했기 때문에, 아마 setTokens 에서 로컬 저장소에 저장해놓고 앱을 껐다 키면 자동 로그인이 되게끔 할것이다.

misc.js

텍스트이(가) 표시된 사진

자동 생성된 설명

-async : 자바스크립트의 기본 함수로, 함수를 async로 선언하면 await 키워드가 비동기 코드를 호출할 때 까지 아무것도 실행하지 않는 것이다. 로컬 저장소에 등록된 회원정보를 저장하는 것은 매우 중요한 일이기 때문에 이렇게 비동기로 처리하는 모양이다. 아무튼 저장이 무사히 완료되면, 메인화면으로 가는 콜백함수를 호출한다. 콜백함수를 인자로 넣어 promise 객체 내에서 이런 방식으로 활용하는게 인상적이다.

auth/index.js

텍스트이(가) 표시된 사진

자동 생성된 설명해당 컴포넌트가 렌더링 된 후에 호출되는 함수이다.

getTokens는 misc.js에 만든 저장소에서 값을 읽어오는 함수이다.

이제 저장해놓은 refresh token을 앱이 새로 켜질 때, 즉 로그인 화면이 렌더링 될 때, 다시 말해 getTokens함수가 호출 될 때 firebase에 넘겨서 매칭되는 id를 받아다가 자동으로 메인화면으로 이동하게끔 할 것이다. access token은 일반적으로 수명이 있다. 수명이 다되면 더 이상 api는 해당 토큰으로 정보를 가져와 주지 않는데, 이전에 발급받은 refresh token으로 새로운 access token을 발급받는 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명action creators에 새로운 액션을 만들었다. refresh token을 인자로 받아 파이어 베이스에 넘겨주는 것이다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

해당 액션의 리듀싱 함수의 일부이다. response로 받은 키워드들이 이름이 조금씩 다른 것이 눈에 띈다. 여기서는 새로운 refresh\_token도 발급받는다.

순서는 다음과 같이 해야 할 것이다. 먼저 처음 로그인 화면이 렌더링 될 때, getTokens 함수가 호출되어 로컬 저장소에서 token 값들을 읽어온다. 무사히 읽어오고, 그 값이 비어있지 않다면, autoSignIn 액션을 호출해서 firebase에 refresh token을 넘겨준다. 무사히 response를 받고, 등록된 유저임이 확인돼서 새로운 access token을 발급 받았다면 메인화면으로 넘어간다.

정보를텍스트이(가) 표시된 사진

자동 생성된 설명 로컬에서 읽어오고 그 뒤의 과정을 콜백함수로 넣어준다.

이텍스트이(가) 표시된 사진

자동 생성된 설명 때, 로컬에 저장된 정보는 이와 같은 배열로 저장되어 있음을 명심한다.

설명은텍스트이(가) 표시된 사진

자동 생성된 설명 위에 다 되어있다.

**Day5**

구성할 화면은 다음과 같다.

Stack Navigator

-Stack Screen : 로그인 화면

-Stack Screen (Tab Navigator Nested) : 메인화면

-Tab Screen (Stack Navigator Nested) : Diary

-Stack Screen : Diary

-Stack Screen : Diary Docu

-Tab Screen (Stack Navigator Nested) : News

코드를 짜던 중 몇가지 사실을 알아냈다. 먼저 navigator의 option을 줄 때 screen components 들의 navigator 중 단 하나에 대해서만 적용된다는 것이다. 즉 예를들어 메인화면인 tab navigator에 haederShown: false 라는 옵션을 주고 싶다고 하면, 해당 tab navigator에서 옵션을 줘야 한다는 것이다. 당연한 말 같지만, 상위 navigator인 stack navigator에 headerShown: false를 줘봤자 하위 navigator에는 적용되지 않고, 반대로 stack navigator인 diary 화면과 news 화면에 옵션을 줘도 상위 navigator에는 적용되지 않는다는 것이다. 당연하지만, 그래도 잘 숙지해둬야겠다. 또 하나는 알고는 있었지만 이놈의 react navigation이 변동이 워낙 잦고 그만큼 deprecate도 잘 일어나는데다가 심지어 docs도 좀 애매하게 되어있어서 써먹기가 힘들다는 것이다. 당장은 style 자체엔 너무 힘을 주지 말아야겠다. 아무튼 위와 같은 구성을 하기 위해 routes.js 에 몇가지 navigator를 추가했다.

텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

main 화면의 navigator인데, component로 navigator를 중첩해서 넣어놨다.

텍스트이(가) 표시된 사진

자동 생성된 설명메인 화면 컴포넌트다. 탭 바에 아이콘을 불러오고, 이것저것 옵션을 줬다. tabBarIcon라는 property는 focused, color, size를 react.node 로 반환해주는 함수이다.

([JSX.Element vs ReactNode vs ReactElement의 차이 (tistory.com)](https://simsimjae.tistory.com/426))  
위가 관련 링크인데, TabBarIcon 함수에 focused를 인자로 가져올 때 중괄호를 감싸길래 왜그러지 싶어서 찾아봤다.  
([컴포넌트에 함수 전달하기 – React (reactjs.org)](https://ko.reactjs.org/docs/faq-functions.html))

일단 알아본 바로는, 중괄호로 감싸는건 route 와 focused 둘 다 객체 안의 property이기 때문인듯 하다. 무슨말이냐면, 예를들어

onChangeText = {value => this.updateInput("email", value)}

에서는 인자를 아무것도 감싸지 않는다. 이런 경우는 value가 단순히 변수이기 때문인듯 하다.

그러나 route 와 focused는 각각 props 객체안의 property와 tabBarIcon의 property인데, 이들을 인자로 전달할 때는 중괄호로 감싸줘야하는 것 같다. 솔직히 하면할수록 궁금증과 의문만이 계속 남는다. 언제한번 날잡고 props와 state 에 대한 개념공부와, es6 문법을 조금 익혀봐야겠다.

기본적으로텍스트이(가) 표시된 사진

자동 생성된 설명 props나 state나 둘다 화면이 렌더링 된 후에 존재하는 것들이라서, 이렇게 렌더링 되기 전에 불리는 함수 같은 경우, 즉 screenOptions와 같이 렌더링되기 전에 불리는 함수내에서 불리는 함수 같은 경우엔 (말이 참 어렵네) this.props 대신 react navigation이 자체적으로 넘겨주는 props인 route를 사용하면 된다. 스크린의 name 값을 가지는 route.name을 인자로 전달받은 TabBarIcon함수는 이에 따라 아이콘 사이즈를 늘렸다 줄였다 한다.

**DAY6**

이제 서버에서 diary 화면에서 볼 내용들을 가져오는 action creator와 reducer를 만들 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

firebase에서 .json 파일의 데이터를 가져오는 새로운 액션이다. 여기쯤에서 스프레드 연산자가 뭔지 다시 생각해볼 필요가 있는데, 일반적으로 es6에서 …연산자 라고 하면 배열 병합에서 많이 볼 수 있다. 즉 diaryData라는 배열에 response.data[key] 배열을 병합시키는 것이다. 디버거를 쓸 수가 없어서 콘솔로 띄워서 파이어 베이스 데이터를 앱으로 잘 가져왔는지 확인하려 했는데, 계속 undefined가 떠서 한참 뻘짓하다 확인해보니 action에서 받아온 데이터를 배열로 받아 놓기만 하고 return을 안해줬다. 당장 위 사진만 봐도 return이 없다…

하여튼 이제 diary 스크린을 꾸며줄 것이다. 즉 dairy stack navigator를 꾸며준다는 것. 구성은 다음과 같이 한다.  
텍스트이(가) 표시된 사진

자동 생성된 설명ScrollView에 일기 리스트를 채워넣고, touchableOpacity는 커스텀 버튼으로 일기를 작성할 때 누르는 버튼이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

position 에 대해선 다음 링크를 참조하자.  
([position - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/CSS/position))

위에서 Dimensions를 rn에서 import하여 screenWidth와 screenHeight를 가져왔는데, 폰 마다 화면이 다르기 때문에 이렇게 했다. onPress에서는 navigation props의 navigate 함수를 불러와 DiaryDocu라는 이름을 가진 스크린으로 넘어가게 했고, props로 새 작성인지 아닌지를 판단하는 newDiary를 가진 객체를 넘겨준다. resizeMode는 contain으로 줘서 컨테이너 안에 들어갈 때 비율을 유지하게 한다.  
([object-fit - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/CSS/object-fit))

[JSX 소개 – React (reactjs.org)](https://ko.reactjs.org/docs/introducing-jsx.html)

위 링크는 자꾸 중괄호 넣는게 헷갈려서 넣어놨다.

텍스트이(가) 표시된 사진

자동 생성된 설명diary 화면에서 보여줄 내용이다. 인자로는 props로 받아오는 Diaries 배열을 받을거고, 아래 링크에 따라 map 을 호출하여 요소 값인 item과 index를 넣어줬다. key는 react가 어떤 항목을 변경할지 식별하는데 도움을 준다고 한다. 난 여기에 index를 넣어줬다. 그런데 index는 변경될 수 있는 값이므로 보통 추천되는 방법은 아니라고 한다. 또 중요한게, **이 함수와 같이 () => () 로 선언된 함수는 () 안의 값 자체를 return한다는 의미라고 한다.**

([리스트와 Key – React (reactjs.org)](https://ko.reactjs.org/docs/lists-and-keys.html))

([Array.prototype.map() - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map))

텍스트이(가) 표시된 사진

자동 생성된 설명view안에 이렇게 채워넣었다. Index를 출력해주고 이미지가 존재하면 이미지도 출력해주는 것.

텍스트이(가) 표시된 사진

자동 생성된 설명index와 img를 양끝에 나란히 배치한다.

[flex-direction - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/CSS/flex-direction)

[justify-content - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content)

텍스트이(가) 표시된 사진

자동 생성된 설명

똑같이 diary 스크린에 있는 본문 내용을 불러오는 부분인데, 문제가 있다.

이렇게텍스트이(가) 표시된 사진

자동 생성된 설명 본문이 길어서 touchableOpacity 영역을 넘어가는 것이다.

그래서텍스트이(가) 표시된 사진

자동 생성된 설명 text 다신 다른 컴포넌트를 쓴다.

이제 저 버튼을 터치하면 짧게 끊긴 본문내용도 볼 수 있도록 diary docu 스크린으로 넘어가게 만들어야 한다. 그런데 index.js에서 newDiary, diaryData, index 와 같은 필드들을 navigate를 통해 diaryDocu.js에 props로 넘겨줬다. 이제 diaryDocu.js는 받은 props를 써야하는데, 만약 render() 에서 state를 선언해주면 한번 렌더링 된 이후에 state가 초기화 될것이고, 그것을 받아온 props를 적용시켜 화면에 띄우려면 다시 렌더링 되어야한다. 그렇게 하면 아무래도 불편하니, 여기서는 constructor()를 쓰기로 한다. 아래 링크는 super에 대한 설명이다. 생성자에서 props를 쓰려면 super(props) 가 필요하다고 한다. 그리고 super() 자체도 this를 초기화 해야하기 때문에 반드시 필요한 함수이다.  
[super - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/super)  
[[번역] React ES6 — Class constructor에서의 super() | by UMI OH | Medium](https://medium.com/@umioh1109/react-es6-class-constructor%EC%97%90%EC%84%9C%EC%9D%98-super-9d53ba0611d9)

diaryDocu.js

텍스트이(가) 표시된 사진

자동 생성된 설명이와 같이 state를 바로 초기화시킨다.

Props를 가져왔으니 이제 이 화면에서 본문, 날짜, 인덱스, 이미지 등을 볼 수 있게하고, 새로운 일기일 경우엔 작성할 수 있게 해야겠다. View로 이런 화면들을 구성할건데, flex로 크기를 조정하려한다. 왜냐면 폰 모델마다 높이같은게 다르기 때문.

대충 화면 구성이 끝났으니, 이제 index 쪽부터 차근히 구현해보려 한다. 그런데 index의 경우 새로운 일기를 작성할 때, 현재 존재하는 일기들의 개수와 같을 것이다. 또 id 값의 경우 일기들이 0부터 하나씩 커지는데, 마지막으로 쓴 일기보다 id가 1 크면 될것이다. id가 존재하는 이유는 데이터베이스 상에서 일기들이 삭제될 수도 있는데, 이 때 인덱스로만 일기들을 구분하면 갑자기 인덱스가 달라져서 구분이 불가능해질 수 있기 때문이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명이제 날짜 인풋란이다. 당연히 newDiary인지 아닌지에 따라 분기되며, 아닌 경우에는 수정 불가능하다. 맞는 경우에는 표시할 내용인 value에 먼저 공란으로 초기화된 date 값을 넣고, 수정 됐을 경우 바꾸기로 한다. 따라서 onTextChange에 들어가는 함수는 onChangeInput 함수로, TextInput의 value를 인자로 넘겨서 또 다시 setState로 date값을 갱신하다. 말이 길게 됐는데, 결국엔 state의 date를 읽어와 보는거고, 그걸 콜백함수로 갱신하겠다는 것이다. 하다보니 새로운 일기를 작성하는 페이지에서 작성이 제대로 안됐는데, 멍청하게 onChangeInput 함수에서 setState를 할 때 diaryData 객체를 써주질 않았다.

아무튼 이제 firebase sdk를 이용해 이미지를 받아와서 띄워주기로 하겠다. 이미지를 가져오는 타이밍은 이전 화면에서 피드를 터치하고 새로운 화면이 렌더링 되기전, 즉 그 사이에 이루어져야 한다. 즉 constructor 안에서 그 작업이 이루어지면 맞겠다. 가져오는 조건은 이미 존재하는 일기이고 또 imagePath가 존재해야 한다. 또 새로운 일기를 작성하는 화면에선 이미지를 추가하고 업로드하는 버튼이 있어야겠다. 그래서 view 컴포넌트를 두개 줬고, 나란히 두려고 flexDirection을 row로 줬다. Flex는 각자 10:1로 뒀다. 아래는 이미지를 불러오는 부분까지 구현된 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

source에서 불러오는 uri는 다음과 같이 불러온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위에서 계획한대로 constructor 내부에서 만들어줬다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Image 태그 source에 uri를 주기위해 getDownloadURL을 이용했다.

로컬 저장소에서 이미지를 불러오려고 react-native-image-picker를 npm install 했는데…

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같은 에러를 마주쳤다. 해결방법은 간단히  
rd /s /q "node\_modules/react-native-text-truncate/.git"  
를 입력하면 됐다. 아래는 참고한 링크이다.  
([npm ERR! code EISGIT, Appears to be a git repo or submodule · Issue #73 · react-navigation/react-native-safe-area-view · GitHub](https://github.com/react-navigation/react-native-safe-area-view/issues/73))  
(["rm -rf" equivalent for Windows? | Newbedev](https://newbedev.com/rm-rf-equivalent-for-windows))

방금은 또 이런 에러를 만났다.  
(Warning: Failed prop type: Invalid props.style key `fontsize` supplied to `ForwardRef(TextInput)`.)

해결방법은 다음과 같다.  
([Getting "Failed prop type: Invalid props.style key `fontsize` supplied to `Text`." warning · Issue #28438 · facebook/react-native · GitHub](https://github.com/facebook/react-native/issues/28438))

이제 이미지를 불러오는 버튼을 구현하도록 하겠다. 이미지를 불러오는건 새로운 일기를 작성할 때 이다. 따라서 Image 칸 안에서 버튼 뷰는 다음과 같이 분기한다.

diaryDocu.js  
텍스트이(가) 표시된 사진

자동 생성된 설명

selectImage는 누르면 사진폴더를 불러온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그런데 사진이 렌더링 되는 조건, 그러니까 왼쪽에 있는 view에서 사진이 보이는 조건은 무엇인가? Imagepath가 비어있지 않을 때, 이미지 태그 안에서 uri는 state.image에서 불린다. 즉 image 필드를 채워주면 되는 것이다. selectImage()는 option엔 빈 객체를 넣고 response의 객체중 assets 이라는 배열에서 0번의 uri를 넣어줬다. 그러고 나서 imagepath에 값을 채워넣기 위해 새로 작성하는 일기의 id를 줬다.

이제 버튼 세개를 만든다. 스타일을 줬는데 오른쪽으로 정렬할 땐 justifyContent에 ‘flex-end’를 준다. 또 오른쪽으로 정렬을 했기에 paddingRight 역시 15로 준다.

또 ios의 키보드를 아래로 내릴 수 없는 점, 그리고 키보드가 컴포넌트들을 가려버리는 점 등을 해결하기 위해 keyboardavoidview라는 컴포넌트를 import했다. Os 별로 다르게 조건을 줄거라 platform 역시 import 했다. KeyboardAvoidView에서는 behavior라는 props가 있는데, 여기서 position은 컴포넌트들이 화면 위 넘어로 뚫고 가는 것, padding은 focus된 컴포넌트가 화면상에서 높은 위치로 이동하는 것, height는 키보드가 올라옴에 따라 컴포넌트 크기 비율이 조정되는 것이다.



또텍스트이(가) 표시된 사진

자동 생성된 설명 키보드를 위로 올리면 컴포넌트들끼리 씹는 현상은 이렇게 해결했다.

이제 아무데나 터치하면 키보드가 사라지게끔 하겠다. ios에서 쓸 기능이다. TouchableWithoutFeedback은 말그대로 아무곳이나 터치해도 이벤트가 발생한다. Keyboard API는 키보드 관련 api 이다. 또 안드로이드 에서는 tabBar를 키보드가 같이 띄워버렸는데, 이를 해결하기 위해 route.js에서 tabBar 옵션을 tabBarHideOnKeyboard를 true로 줬다.

**DAY7**

이제 파이어베이스 스토리지나 데이터베이스에 새로운 데이터를 입력하는 작업을 할것이다. 일단 실행 순서가 중요한데, 예를들어 어떤 파일을 업로드 하는 작업을 한다고 가정해보자. 파일을 올리는 작업을 만약에 파일 경로가 실제로 존재하는지 확인하는 작업보다 먼저 실행하면, 경로가 존재하는지 확인하는 작업 자체가 무의미해진다. 따라서 이러한 일련의 작업들이 확실하게 순서가 보장받을 수 있도록 async 와 await을 이용한 비동기 처리로 만들것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명isLoading이 true 인 동안에는 activityIndicator가 돌게 할 것이다.



변수이름은 반드시 data로 해야한다. Firebase 내부에서 정보들이 담긴 객체 이름이 data 이기 때문이다.



텍스트이(가) 표시된 사진

자동 생성된 설명database의 set함수이다. 넘겨줄 데이터와 경로를 인자로 넣어줬다. await으로 만든 이유는, db에 데이터를 넘겨서 중요한 imagePath와 같은 필드를 업데이트 한 뒤, 그게 끝나면 이미지를 업로드하게끔 로직을 짜려는 것이다. 예기치 않게 await이 실패하면 activityIndicator를 끄고 저장이 실패했다는 메시지를 띄워준다.

그런데 잠깐 생각해보고 가야할 부분이 있다. await 이 끝나면 이미지를 업로드하고, 각각 db와 storage에 새로운 일기가 등록될 것이다. 등록된 직후 다시 diary 화면으로 돌아갈 것인데, 여기서 문제가 발생한다. 지금까지 우리가 줄곧 사용하던 navigation props의 navigate함수는, 지금 보고있는 screen에서 route사이에 navigate하려는 screen의 name 이 존재하는지 확인하고, 없다면 최상단에 화면을 쌓고, 있다면 그곳으로 돌아가는 기능을 한다. 즉 우리가 만약 새로운 일기를 작성하고 그게 diary화면에 반영 되길 원하는데 navigate를 쓴다면 일기를 등록하기 전 화면으로 돌아가 버린다는 것이다. 즉 다음과 같다.

3: Diary 1 로 다시 돌아감!  
2: Diarydocu 와 같이 되어야 하는데 navigate를 쓰면 2: Diary Docu  
1: Diary 1: Diary

그래서 아예 새로운 화면을 만들어주는 push 함수를 쓰기로 한다!

([reactjs - React Navigation V2: Difference between navigation.push and navigation.navigate - Stack Overflow](https://stackoverflow.com/questions/51090135/react-navigation-v2-difference-between-navigation-push-and-navigation-navigate))  
([StackActions reference | React Navigation](https://reactnavigation.org/docs/stack-actions/))

위에 정말 훌륭하게 설명해놨다!

이렇게 화면끼리 state를 업데이트 해주는 경우엔 push를 쓴다는건 알았다. 이제 diary의 stack navigator를 고려해보자. docu->diary는 일단 push 로 넘긴다고 전제하자.

먼저 기존 일기를 터치하는 경우를 생각해보자. (1: diary 2:docu 3:diary -> 2: docu 4: diary -> 2: docu) 이렇게 되면 안된다. 그러니까 난 분명 제목이 바꾼 일기를 터치했는데, 막상 docu 페이지로 이동하고 나니 2번 상태 그대로인 화면이 나타나는 것이다. 따라서 이경우도 무조건 push를 써야겠다.

또 새로운 일기를 작성하는 경우도 생각해보자. 하다보니 머리가 아파서 그만뒀다. 일단은 이런 경우엔 무조건 push를 쓰는걸로 마음먹었다.

이제 정말 이미지 업로드를 구현해 보겠다. 이미지 업로드 역시 로컬 저장소에서 업로드 하려는 이미지를 무사히 가져오는지 등에 대한 비동기 처리가 필요하기 때문에 async로 만들어야 한다. 로컬에서 이미지를 request하여 response로 받고, 그것을 blob(binary large object) 으로 다시 반환해줘야 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명“완료”버튼의 이벤트 핸들러이다. Database의 함수 set으로 reference에 this.state.diaryData를 덮어 씌운다. 서버에 대한 write이 끝나면, set은 resolve를 한다. 이제 uploadImage에 이미지를 저장할 디렉토리를 넘겨준다. 만약 set이 write에 실패했다면, 에러 메시지를 띄워준다.

텍스트이(가) 표시된 사진

자동 생성된 설명fetch 함수는 자바스크립트 내장함수로, url을 인자로 받아서 Promise(response) 객체로 반환한다. Response 에는 blob() 라는 메서드가 있는데, Response 객체를 Blob객체로 resolve 해주는 (이역시 프로미스 형태이다) 메서드다. 이 작업 역시 무사히 끝나면 try, catch 문으로 await promise 인 uploadBytes가 잘 resolve 됐는지, 에러를 반환하는지에 따라 작업을 분기한다. resolve가 잘 되면 diary화면으로 push 하고, 에러가 발생하면 에러 메시지와 함께 저장에 실패했다고 알린다. 처음 if 문으로 이미지를 업로드 하는경우와 안하는 경우로 나눴는데, 안하는 경우엔 그냥 다시 diary 화면으로 push 한다. 아래는 이해에 참고한 링크이다. 꽤나 복잡하므로, 다시, 자주 읽어봐야할 것 같다…

([Fetch API - Web API | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/API/Fetch_API))  
([Using Fetch - Web API | MDN (mozilla.org)](https://developer.mozilla.org/ko/docs/Web/API/Fetch_API/Using_Fetch))  
([봐도 봐도 헷갈리는 resolve, reject (velog.io)](https://velog.io/@rejoelve/%EB%B4%90%EB%8F%84-%EB%B4%90%EB%8F%84-%ED%97%B7%EA%B0%88%EB%A6%AC%EB%8A%94-resolve-reject))  
([Response.blob() - Web APIs | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/API/Response/blob))  
([async와 await (javascript.info)](https://ko.javascript.info/async-await))

Diary/index.js

그냥 다이어리 리스트를 역순으로 출력하게 했다.

diaryDocu.js

텍스트이(가) 표시된 사진

자동 생성된 설명이러한 로딩 화면을 구성해 보았다.

이제 삭제버튼을 구현하겠다. 똑같이 reference를 만들고, db 와 storage 각각 remove와 deleteObject를 이용해서 삭제를 실행하면 된다. 당연한거지만 db의 경우 현재 보고있는 docu 화면의 id를 받아서 삭제경로를 지정할 것이다. 화면 상에서 id가 없을 수가 없으니 바로 await으로 remove를 요청하고, 끝나면 storage의 reference를 요청해서 이미지가 있는지 없는지 확인한다. 없어서 error 가 걸려도 그냥 diary 화면으로 push 하게끔 한다. remove는 콜백함수가 걸려있지 않기 때문에, 에러발생시 삭제 실패 경고문이 출력된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 마지막으로 업데이트 버튼을 구현한다. 입력 값을 state에 새로 저장하는 작업인데… 이거 어어디 해본 것 같다. 그렇다. createData 가 바로 그 기능을 한다. 다시 createData를 보자. 현재 보고있는 docu의 id 값을 읽고, 그 경로에 따라 현재 state.diaryData를 서버에 넘겨준다. 당연히 이미 있는 화면에서 이 함수가 실행되면 데이터는 아무것도 바뀌지 않는다. 그럼 남은 것은 textinput이 수정 가능한 상태가 되는것이다. 그 상태란 newDiary가 true일 때이고, 따라서 해야하는건 newDiary를 true로 갱신시켜주는 것 뿐이다.

이제 정말 중요한 작업, 로그인에 기반한 앱 동작을 구현할 것이다. 유저에 따른 데이터 접근 권한을 설정하겠다는건데, 현재 데이터베이스는 이러한 유저 정보를 담고있지 않고있기 때문에 사용이 어렵다. 새로 쓰는 json은 유저식별자를 줘서, uid에 따라 데이터를 생성할 때 그 영역 안에서만 생성이 이루어지도록 하고, 가져와서 렌더링 할 때에도 마찬가지로 uid 에 따라 접근할 수 있도록 해야한다.

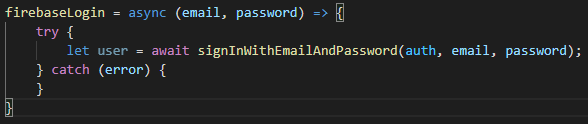
가장 먼저 로그인이 되어있지 않은 사람한테 메인화면 대신 보여줄 화면을 구성해 보겠다. 로그인이 되어있지 않은 화면에서 메인화면으로 넘어가는 건 ‘비회원 로그인’ 버튼이다. goWithoutLogin은 “AppTapComponent”, 즉 메인화면을 보여주는 diary의 index.js로 이동시켜준다. 따라서 index.js 에 이러한 로직을 구현하면 된다. 로그인 여부를 나타내는 boolean을 state로 두고 scrollview와 touchableopacity를 잠궈버리면 될 것이다. 따라서 렌더링 직후에 실행되는 componentDidMount 에서 로그인 여부, 즉 유효한 access token을 가지고 있는지를 따져서 만약 있다면 getDiaries 함수를 불러와서 해당 유저의 데이터를 서버에서 가져오는 액션을 해주고, 없다면 isAuth를 false로 해서 화면을 잠궈버린다. 그렇다면 남은건 access token을 어떻게 읽냐는 것이다. 일단은 자동로그인 할 때 썼던 것을 그대로 가져오기로 한다. 그렇게 하면 access token이 만료된 유저의 경우에도, (아마 자동로그인을 한 시점에서 새로 access token을 발급받고 로컬 저장소에 저장해서 상관 없는 얘기겠지만) autoSignIn으로 access token을 다시 발급받고 getDiaries를 무사히 실행하고 isAuth를 true로 돌릴 수 있게된다.

텍스트이(가) 표시된 사진

자동 생성된 설명렌더 함수안에 이런 조건을 넣어줬다.

getTokens는 로컬 저장소의 토큰들을 읽어 온다. 어디까지나 로그인 여부, 즉 access token을 받아오기 위함이다. 다시 파이어베이스에 요청하는게 아니고 로컬에서 꺼내오는 방법을 선택한것이다. 이것이 바로 redux의 엄청난 점!

이제 기존 axios로 데이터를 가져오던 액션을 전부 firebase sdk로 바꾸어 주려고 한다. axios로는 당장 어떤 유저가 데이터에 접근하는지 모르기 때문이기도 하고, 구현하려면 할 수 있겠지만 보안상 firebase 에서 제공해주는 authentication을 이용하는게 맞기 때문이다.



Signin 액션에서 firebaseLogin을 실행하면, 아래 함수로 파이어베이스에 성공적으로 로그인 정보가 전달되었고, 그 정보를 다시 받아오는 것까지 성공적으로 작동했다. 기존에 axios로 길게 promise 객체를 파이어베이스에 전달해서 response를 받아오던 과정보다 훠얼씬 간단했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**Day8**

firebase에서 성공적으로 입력한 email과 password 를 보내서 로그인 정보를 얻어내는 것을 확인했으니, 이제 받아온 정보를 바탕으로 database에 접근하여 데이터를 불러오도록 하자. Diary의 index.js에서 autoSignIn action으로 refresh token을 파이어베이스에 보내 access token을 다시 얻어오는 작업이 성공적으로 마무리 되면, setTokens로 로컬 저장소에 넣은 뒤 getDiaries 액션을 실행하여 db에서 데이터를 요청하는 작업을 한다. 이때 redux-thunk 미들웨어를 이용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

([[JavaScript] Pure Function(순수 함수) (velog.io)](https://velog.io/@nittre/JavaScriptFunction-Composition-Series-1.-Pure-Function))

기본적으로 redux를 사용할 때, action creator는 항상 액션 객체를 반환해야한다. 위는 내가 직접 axios로 response를 받아서 배열에 받아 payload에 넣어줬다. 따라서 onValue와 같은 firebase에서 제공해주는 함수를 직접 사용하려면 redux-thunk를 이용해야 하는 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 나중에 이 액션을 dispatch하면 정상적으로 작동한다. 아래는 redux-thunk에 대한 링크이다.

([2-1. redux-thunk · GitBook (vlpt.us)](https://redux-advanced.vlpt.us/2/01.html))

텍스트이(가) 표시된 사진

자동 생성된 설명추가로, 보안을 강화하기 위해 firebase 규칙을 설정했다.

이와 같이 읽기, 쓰기 권한을 Authentication에서 등록된 유저의 uid가 database의 폴더명 uid하고 일치해야 부여되게끔 한 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명stroage도 마찬가지로 규칙을 수정해 주었다.

storage역시 uid 에 따라 데이터에 접근해야하니, 해당하는 부분들도 바꿔줘야 한다. 이 역시 redux의 store에서 꺼내오면 된다! 아까 database접근도 알맞게 경로를 수정해 줬으니, 남은건 diaryDocu에 userId를 전달해줘서 이미지를 불러올 수 있게하면 된다. 물론 업로드도 마찬가지다.

그러고보니 파이어베이스에서 데이터를 가져올 때 null 인 경우도 가져와 버려서 쓸데없이 index를 카운트 하고 있는 모습이 발견됐다. 따라서 다음과같이 snapshot.val()이 null 이 아닐때만 배열에 값을 넣어주고 그 배열을 리턴했다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

CRUD 버튼들 전부 경로만 바꿔줘서 따로 캡쳐본은 올리지 않겠다.

이제 첫 화면을 추가할 차례이다. 아래 링크의 Interpolation 항목과 Using the native driver 항목을 참조하자.

([Animations · React Native](https://reactnative.dev/docs/animations#interpolation))

스타일은텍스트이(가) 표시된 사진

자동 생성된 설명 조금 잘렸지만, 중요한 부분이 아니므로 넘어갔다. 아무튼 이제 firebase에서 로그인이 되어있는지 아닌지를 직접 체크해주기 때문에, onAuthStateChnaged 를 이용하여 기존에 쓰던 isLoggedIn을 버리고 onComplete와 같이 분기하기로 한다. onLoad에서는 가장먼저 로딩화면 애니메이션을 띄워주기로 한다. 사소한 문제로 backbutton을 누르면 멈춰버린 로딩화면으로 돌아가는게 있었는데, 다음과 같이 수정한다.

자꾸 찾는것도 귀찮아서, react navigation의 header option 링크를 남긴다.

([Native Stack Navigator | React Navigation](https://reactnavigation.org/docs/native-stack-navigator/#options))

headerLeft:null로 diary, news 화면에서 로딩쪽으로 돌아가지 못하게 막아버렸다.

안드로이드의 경우 뒤 페이지로 아예가지 못하게 하는 방법도 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

화면을 떠나지 못하게 하는 이벤트의 기본액션을 취하라는 코드이다. Auth 화면에 적용되어있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

diary화면에서는 같은 방법을 쓰지 못하는데, 왜냐면 위와 같이 해버리면 뒤의 화면으로 이동하는 이벤트 핸들러가 모두 막혀버리기 때문이다.

([Preventing going back | React Navigation](https://reactnavigation.org/docs/preventing-going-back/))

([BackHandler · React Native](https://reactnative.dev/docs/backhandler))

이제 로그아웃 버튼을 구현한다. 로컬 저장소에 있는 토큰 데이터들을 전부 날려줄 필요가 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명